Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Estimating Models of Strategic Interaction in R

Brenton Kenkel
University of Rochester

Curtis S. Signorino
University of Rochester

Abstract

This article introduces new software, the **games** package, for the estimation of strategic statistical models in R. In these models, the probability distribution over outcomes corresponds to the equilibrium of an underlying game form. We review such models and provide derivations for one example, including discussion of alternative motivations for the stochastic component of the models. We introduce the basic functionality of the **games** package, such as how to estimate players' utilities for outcomes as a function of covariates. The software also includes functions for bootstrapping, plotting fitted values with their confidence intervals, performing non-nested model comparisons, and checking global convergence failures. We use the new software to replicate Leblang's (2003) analysis of speculative currency attacks.

Keywords: random utility models, structural estimation, game theory, econometrics, political science.

1. Introduction

The **games** package provides functions for estimation of statistical models of strategic interaction in the R language (R Development Core Team 2010). These are random-utility models of choices by multiple agents, each of whom conditions his or her actions on the likely decisions of the other players. In these models, the distribution of outcomes is determined by the equilibrium of the underlying game. The goal is to estimate how the players' utility for each possible outcomes varies as a function of observed variables. These models are appropriate for situations where the ultimate outcome following one actor's choice depends on actions taken by another, which are common in social science. In such cases, standard estimators like binary-choice regression or Heckman selection models will lead to incorrect inferences (Signorino 2002; Signorino and Yilmaz 2003). The **games** package implements models for both discrete interactions and ultimatum bargaining. The package also provides various post-estimation functions, including convergence checks, plotting of fitted values, and

non-nested model comparison tests.

Estimation of strategic statistical models was previously implemented in Gauss in the **Strat** package (Signorino 2003a). The **games** package provides new models and additional post-estimation functionality over that available in **Strat**, and it is implemented in the popular R language. Certain recursive strategic models can be estimated using standard software for probit or logistic regression (Bas, Signorino, and Walker 2007), but these two-step estimates are inefficient compared to full-information maximum likelihood. We are aware of no other software for full structural estimation of strategic statistical models.

Section 2 of this paper provides a brief introduction to strategic statistical models, including a derivation of the simplest example. Section 3 discusses the details of their implementation in the **games** package and provides a replication of Leblang (2003). The post-estimation functionality is covered in Section 4.

2. Strategic statistical models

Strategic statistical models were first introduced to analyze international crises and the outbreak of war (Signorino 1999). Signorino shows that standard techniques like logistic regression are inappropriate when applied to data generated by multi-agent strategic interactions, a point developed further by Signorino and Yilmaz (2003). Signorino (2003b) introduces a class of models that yield consistent estimates of players' utilities when the structure of interaction is known. These models have since been applied to data on U.S. congressional races (Carson 2003, 2005), currency markets (Leblang 2003), armed deterrence (Signorino and Tarar 2006), international economic sanctions (McLean and Whang 2010), territorial conflict (Carter 2010), and Latin American governmental crises (Helmke 2010).

Every strategic model is associated with a game form and solution concept. First, the structure of the interaction must be known: the number of players, the order in which they move, the number of actions each has available, and the possible outcomes. The purpose is to estimate players' utilities for each outcome, usually as a function of covariates, from data on observed outcomes of the game being played. This requires the introduction of a stochastic component, so that there is a non-degenerate probability distribution over outcomes for any given set of coefficients (i.e., those on the covariates describing players' utilities). In particular, we will specify where error enters the model and calculate the equilibrium outcome for each given set of parameters and stochastic shocks, using the appropriate solution concept for the assumed stochastic structure (see below). The probability of each outcome can then be obtained by assuming a distribution for the error terms.

The choice of stochastic structure is crucial for the estimation and interpretation of utility parameters. The **games** package implements methods for two cases:

Agent error Each player's utility over outcomes is fixed and common knowledge. However, there are perceptual or implementation errors that can lead to a player not choosing the action that maximizes her expected utility calculated in terms of the outcome payoffs. This can be represented as a shock α_{mj} to Player m's expected utility for taking action j, where the shock is realized immediately before m makes her action choice (and hence is unknown to the preceding players). We typically assume that each α_{mj} is drawn independently from a normal or logistic distribution. The solution concept under agent error is quantal response equilibrium (McKelvey and Palfrey 1998), wherein each player

anticipates the probability of "mistakes" by the others and adjusts her expectations accordingly.

Private information There is a different stochastic shock to each player's utility for each outcome. We will write this as π_{mk}, for Player m and outcome k. The key assumption is that each player fully knows her utility for each outcome, but only knows the distribution of the shocks to the other players' outcome utilities. The solution concept in this case is perfect Bayesian equilibrium: each player takes the action that gives her the highest expected utility, with respect to the realized shocks to her preferences and her expectations about the actions the other players will take. Whereas the distribution over outcomes was induced by the possibility of wrong decisions in the agent error case, now it comes from the fact that observationally indistinguishable players may have different privately known preferences. Except in the statistical ultimatum model, we will assume that each π_{mk} is drawn from a normal distribution.

We illustrate both of these stochastic structures in the `egame12` example below. Signorino (2003b) discusses these in greater depth, and concludes from Monte Carlo experiments that models of the two types do not yield appreciably different results when the underlying game form is relatively simple.

2.1. Illustration: The `egame12` model

The `egame12` model, with two players and three possible outcomes, is the simplest strategic model. The players are indexed $m = 1, 2$, and each has an action set $a_m = \{L, R\}$. The outcomes are indexed $Y = 1, 3, 4$. The structure of the interaction is as follows:

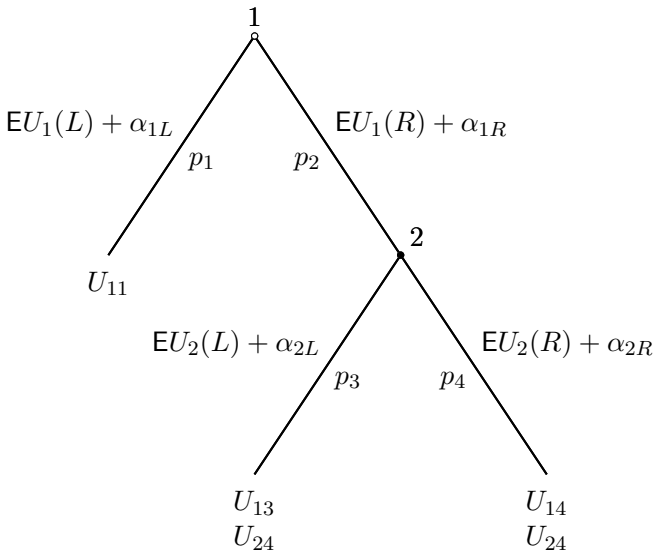
1. Player 1 chooses his action a_1. If $a_1 = L$, the game ends and the outcome is $Y = 1$. Otherwise, if $a_1 = R$, Player 2 gets to move.

2. Player 2 chooses her action a_2. If $a_2 = L$, the outcome is $Y = 3$; if $a_2 = R$, the outcome is $Y = 4$.

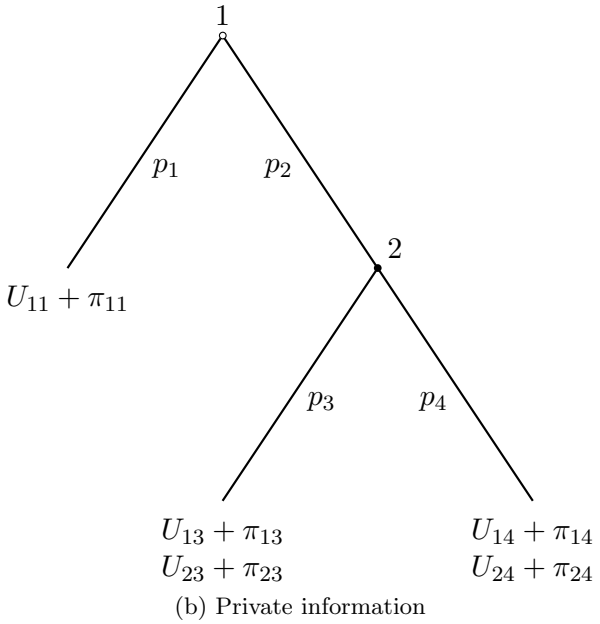
These are illustrated in the game trees in Figure 1. Let p_1 and p_2 denote the action probabilities for Player 1, where $p_1 = \mathsf{P}(a_1 = L)$ and $p_2 = \mathsf{P}(a_2 = R)$. Let Player 2's action probabilities, conditional on 2's move being reached, be $p_3 = \mathsf{P}(a_2 = L|a_1 = R)$ and $p_4 = \mathsf{P}(a_2 = R|a_1 = R)$.

Each player's utility depends on the outcome of the game; utility to Player m for outcome $k \in \{1, 3, 4\}$ is denoted U_{mk}. When estimating this game, we usually model each of these utilities as a linear function of known covariates, $U_{mk} = X_{mk}^{\top}\beta_{mk}$, with the goal of estimating the coefficients β_{mk}. Each player chooses her action $a \in \{L, R\}$ to maximize her expected utility, $\mathsf{E}U_m(a)$. Since Player 2 moves last, her expected utilities are simply $\mathsf{E}U_2(L) = U_{23}$ and $\mathsf{E}U_2(R) = U_{24}$. Similarly, since the action $a_1 = L$ is a game-ending move, Player 1's expected utility from this is $\mathsf{E}U_1(L) = U_{11}$. However, Player 1's expected utility from the action R depends on Player 2's choice, so we have $\mathsf{E}U_1(R) = p_3U_{13} + p_4U_{14}$.

We observe N plays of the game, each with realized outcome Y_i and associated regressors X_{mki} for each player m and outcome k. Our goal is to estimate $\beta = (\beta_{mk})_{m\in\{1,2\},k\in\{1,3,4\}}$, the set of coefficients describing players' utilities for particular outcomes, via maximum likelihood.



(a) Agent error



(b) Private information

Figure 1: Game trees for the `egame12` model.

Once we assume a stochastic structure (agent error or private information), we can calculate the observation-wise choice probabilities $p_{1i}, \ldots, p_{4i}$ for any β. The log-likelihood is then

$$\log \ell(\beta \mid X, Y) = \sum_{Y_i=1} \log p_{1i} + \sum_{Y_i=3} (\log p_{2i} + \log p_{3i}) + \sum_{Y_i=4} (\log p_{2i} + \log p_{4i}). \tag{1}$$

The calculation of these choice probabilities is the subject of the following subsections. For ease of exposition, observation subscripts are dropped from all terms in the following materials.

Agent error

Assume that each player receives a stochastic shock α_{mj} to her expected utility for choosing $j \in \{L, R\}$, where each α_{mj} is drawn independently from a normal distribution with mean 0 and variance σ^2.[1] To solve for the quantal response equilibrium of the game, we will proceed via backward induction, solving for Player 2's choice probabilities in order to find Player 1's. If Player 2's turn is reached, her choice determines the outcome for sure. We thus have $\mathsf{E}U_2(L) = U_{23} = x_{23}^\top \beta_{23}$ and $\mathsf{E}U_2(R) = U_{24} = x_{24}^\top \beta_{24}$. The *ex ante* probability that Player 2 chooses R is

$$\begin{aligned} p_4 &= \mathsf{P}[\mathsf{E}U_2(R) + \alpha_{2R} \geq \mathsf{E}U_2(L) + \alpha_{2L}] \\ &= \mathsf{P}[\alpha_{2L} - \alpha_{2R} \leq x_{24}^\top \beta_{24} - x_{23}^\top \beta_{23}] \\ &= \Phi\left(\frac{x_{24}^\top \beta_{24} - x_{23}^\top \beta_{23}}{\sigma\sqrt{2}}\right), \end{aligned} \tag{2}$$

where $\Phi(\cdot)$ is the standard normal CDF. Since Player 2 must choose from L or R, we have $p_3 = 1 - p_4$. We now can solve for Player 1's choice probabilities. If Player 1's action is L, the outcome is $Y = 1$ for certain. However, if he chooses R, the outcome is a lottery over outcomes 3 and 4, with probabilities p_3 and p_4 respectively. Since Player 1 also receives a shock to his expected utilities by action, his *ex ante* chance of choosing R is

$$\begin{aligned} p_2 &= \mathsf{P}[\mathsf{E}U_1(R) + \alpha_{1R} \geq \mathsf{E}U_1(L) + \alpha_{1L}] \\ &= \mathsf{P}[\alpha_{1L} - \alpha_{1R} \leq p_3 x_{13}^\top \beta_{13} + p_4 x_{14}^\top \beta_{14} - x_{11}^\top \beta_{11}] \\ &= \Phi\left(\frac{p_3 x_{13}^\top \beta_{13} + p_4 x_{14}^\top \beta_{14} - x_{11}^\top \beta_{11}}{\sigma\sqrt{2}}\right). \end{aligned} \tag{3}$$

Because Player 1 must choose L or R, we have $p_1 = 1 - p_2$. We can then estimate the agent error model for a given dataset by substituting (2) and (3) into the log-likehood function (1).

As in standard binary dependent variable models (e.g., GLMs with a logit or probit link), the statistical model is not identified with respect to the scale parameter σ, so it cannot be estimated (Signorino 1999; Lewis and Schultz 2003). The scale parameter is fixed to $\sigma = 1$ in all of the extensive-form models in **games**, so each estimated utility coefficient $\hat{\beta}_{mkj}$ can be interpreted as an estimate of the ratio β_{mjk}/σ. Alternatively, we allow for σ to be modeled as a function of covariates, in which case the regression coefficients for these variables can be estimated. For details, see the example in Section 3.3.

Private information

Assume there is an additive shock π_{mk} to each outcome utility U_{mk}, where each π_{mk} is drawn

[1] The same calculations as follow can be applied in the case of errors with logistic distributions.

independently from a normal distribution with mean 0 and variance σ^2. We will again proceed by backward induction, this time to solve for the perfect Bayesian equilibrium.

Player 2 will choose R if and only if $U_{24} + \pi_{24} \geq U_{23} + \pi_{23}$. The *ex ante* probability of Player 2 choosing R is therefore

$$\begin{aligned} p_4 &= \mathsf{P}[U_{24} + \pi_{24} \geq U_{23} + \pi_{23}] \\ &= \mathsf{P}[\pi_{23} - \pi_{24} \leq x_{24}^\top \beta_{24} - x_{23}^\top \beta_{23}] \\ &= \Phi\left(\frac{x_{24}^\top \beta_{24} - x_{23}^\top \beta_{23}}{\sigma\sqrt{2}}\right). \end{aligned} \tag{4}$$

As before, $p_3 = 1 - p_4$. In addition, notice that (4) is essentially the same as in (2), since Player 2's actions are decisive over outcomes. The same does not hold when we consider Player 1's choice probabilities. In particular, his expected utility for choosing R in the private-information case is

$$\mathsf{E}U_1(R) = p_3(x_{13}^\top \beta_{13} + \pi_{13}) + p_4(x_{14}^\top \beta_{14} + \pi_{14})$$

The *ex ante* probability of Player 1 selecting R is

$$\begin{aligned} p_2 &= \mathsf{P}[\mathsf{E}U_1(R) \geq \mathsf{E}U_1(L)] \\ &= \mathsf{P}[p_3(x_{13}^\top \beta_{13} + \pi_{13}) + p_4(x_{14}^\top \beta_{14} + \pi_{14}) \geq x_{11}^\top \beta_{11} + \pi_{11}] \\ &= \mathsf{P}[\pi_{11} - p_3\pi_{13} - p_4\pi_{14} \leq p_3 x_{13}^\top \beta_{13} + p_4 x_{14}^\top \beta_{14} - x_{11}^\top \beta_{11}] \\ &= \Phi\left(\frac{p_3 x_{13}^\top \beta_{13} + p_4 x_{14}^\top \beta_{14} - x_{11}^\top \beta_{11}}{\sigma\sqrt{1 + p_3^2 + p_4^2}}\right). \end{aligned} \tag{5}$$

In particular, the variance of Player 1's choice probabilities depends on p_3 and p_4, which was not the case under agent error. To estimate the private information model, as before, substitute the choice probability equations (4) and (5) into the log-likelihood (1). As in the agent error model, the scale parameter σ cannot be estimated, so it is fixed to 1 by the fitting functions.

2.2. The statistical ultimatum model

The ultimatum game is a workhorse model of bargaining in economics in political science, in which one player makes a "take it or leave it" offer to the other. The equilibrium size of an offer depends on the proposer's expectations about what will be accepted, which standard models like OLS fail to account for. To facilitate analysis of bargaining data, we implement the statistical ultimatum game of Ramsay and Signorino (2009) via the `ultimatum` function. The structure of the game is:

1. Player 1 makes an offer $x \in [0, Q]$ to Player 2.
2. Player 2 can accept or reject the offer.
   (a) If accepted, payoffs are $Q - x$ for Player 1 and x for Player 2.
   (b) If rejected, payoffs are $R_1 + \epsilon_1$ and $R_2 + \epsilon_2$ respectively, where ϵ_1, ϵ_2 are i.i.d. logistic variables with scale parameters s_1 and s_2. The reservations R_m are common knowledge, but the realized stochastic terms ϵ_m are privately known by the players.

In applications, the reservation values are modeled as a function of covariates, $R_{mi} = x_{mi}^{\top}\beta_m$, and the goal is to estimate β_1, β_2, s_1, and s_2. For example, experimental economists have investigated whether there are cross-cultural differences in play of the ultimatum game in lab settings; e.g., if Americans make systematically lower offers (see Botelho, Harrison, Hirsch, and Rutström 2005). To test this, one would include an indicator for nationality in the equations for the players' reservation values.

For a full derivation of the estimator and application to experimental bargaining data, see Ramsay and Signorino (2009).

3. Specification and estimation

In this section and those below, we replicate Leblang's (2003) analysis of speculative currency attacks to illustrate the package's functionality. The dataset is available in **games** as `leblang2003`.

```
R> data(leblang2003)
R> names(leblang2003)
```

```
 [1] "outcome"      "preelec"      "postelec"     "rightgov"     "unifgov"
 [6] "lreserves"    "realinterest" "lexports"     "capcont"      "overval"
[11] "creditgrow"   "service"      "USinterest"   "contagion"    "prioratt"
[16] "nation"       "month"        "year"
```

Each observation is a country observed in a particular year. The assumed data-generating process follows the `egame12` model, with two players and three potential outcomes. Player 1 is "the market," which decides whether or not to initiate a speculative attack on Player 2's (the country's) currency. If the market decides not to attack, the game ends. If there is an attack, the country decides whether to devalue the currency or defend its exchange-rate peg. The observed distribution of outcomes is:

```
R> table(leblang2003$outcome)
```

```
  no attack devaluation     defense
       7152          42          46
```

We assume that the market is strategic, incorporating its expectations of the country's response into its initial decision of whether to make a currency attack. The source of uncertainty is assumed to be private information about payoffs, which yields outcome probabilities given by equations (4) and (5). The market's utility for the three possible outcomes, and each country's utility for defending the currency, is assumed to be a linear function of observed covariates. For identification, the country's utility for devaluation is fixed to 0.[2] See the original paper or the help page for `leblang2003` for information on the covariates and specific assignments to each utility equation.

[2]In the original study, Leblang estimates a constant for Player 2's utility from devaluation and leaves a constant out of utility for defense. Our approach here yields substantively identical results.

3.1. Modeling player utilities

The typical use of a strategic model is to estimate the effect of observed factors on players' utility for each possible outcome. To avoid an overabundance of parameters and potential inefficiency, analysts will typically want to make some exclusion restrictions—i.e., to leave some regressors out of some utility equations.[3] This necessitates the use of multiple model formulas, which we handle via the **Formula** package (Zeileis and Croissant 2010). The variables to include in each utility are specified using the standard `formula` syntax, and each set is separated by a vertical bar (`|`). For example, in the `egame12` model, an analyst may want to use the specification

$$\begin{aligned} U_{11} &= \beta_{11,0} + \beta_{11,1} x_1 \\ U_{13} &= 0 \\ U_{14} &= \beta_{14,0} + \beta_{14,1} x_1 + \beta_{14,2} x_2 \\ U_{24} &= \beta_{24,0} + \beta_{24,2} x_2, \end{aligned}$$

where x_1 and x_2 are observed variables. The appropriate `Formula` syntax is `y ~ x1 | 0 | x1 + x2 | x2`.

In some of the more complex models, such as `egame123` with its eight utility equations, writing the model formulas manually may be daunting or prone to error. We provide two options to ease the process. First, users may specify the model formulas as a list; the fitting functions then use the internal function `checkFormulas` to convert it to the appropriate `Formula` object.

```
R> f1 <- list(u11 = y ~ x1, u13 = ~0, u14 = ~x1 + x2, u24 = ~x2)
R> games:::checkFormulas(f1)

y ~ x1 | 0 | x1 + x2 | x2
<environment: 0x21e7c10>
```

(Elements of the list need not be named; in fact, the names are ignored.) Second, the function `makeFormulas` provides interactive prompts for constructing the model formulas step by step. The user only needs to supply the name of the model he or she intends to fit and a character vector containing outcome descriptions. For the Leblang data, the appropriate call would look like `makeFormulas(egame12, outcomes = c("no attack", "devaluation", "defense"))`. The following menu will appear at the R console:

```
Equation for player 1's utility from no attack:

1: fix to 0
2: intercept only
3: regressors, no intercept
4: regressors with intercept

Selection:
```

[3] A necessary condition for identification in a strategic model is that no regressor, including the constant, appear in all of a player's utility equations for the outcomes reachable after her move (Lewis and Schultz 2003). The fitting functions and `makeFormulas` enforce this condition. One way to accomplish it is to fix each player's utility to 0 for one outcome. This comes without loss of generality, since Von Neumann–Morgenstern utilities are unique only up to a positive affine transformation.

If 3 or 4 is selected, the user will be prompted to enter a space-separated list of variables to include in the utility equation of interest. We use functions from **stringr** (Wickham 2010) in parsing the input. The same menu will then be displayed for player 1's utility from devaluation, player 1's utility from defense, and player 2's utility from defense. The final prompt will ask for the name of the variable (or variables; see Section 3.2 below) containing information on the observed outcomes. The function will then return the `Formula` specification corresponding to the given input, which can be supplied as the `formulas` argument of the appropriate fitting function.

3.2. Dependent variable specification

For most of the models included in the **games** package, there are a few different ways that the dependent variable might be stored in the dataset. For example, all of the following are plausible representations of the outcome variable in the Leblang data:

- Numeric indicators for the final outcome, where `1` means no currency attack, `2` means devaluation in response to an attack, and `3` means defense against an attack.
- Factor indicators for the final outcome, where the levels correspond to no attack, devaluation, and defense respectively.
- Binary variables representing each player's action. The first would be coded `0` when there is no attack and `1` when there is an attack. The second would be coded `0` when the targeted country devalues and `1` when it defends the currency peg.

The **games** package allows for all of these types of specifications. To use a numeric or factor indicator for the final outcome, the form of the specification is simply `y ~ .`, as in typical model formulas. To use binary indicators, the names of the indicators should be separated with `+` signs on the left-hand side, as in `y1 + y2 ~ .`. When using binary indicators, unobserved outcomes—in this case, the value of `y2` when `y1 == 0`—should *not* be coded as `NA`s, as this will typically result in their being removed from the dataset.

The method of specifying the dependent variable has no effect on the estimation results, as shown in the next example.

```
R> leblang2003$attack <- as.numeric(leblang2003$outcome != "no attack")
R> leblang2003$defend <- as.numeric(leblang2003$outcome == "defense")
R> flb <- outcome ~ capcont + lreserves + overval + creditgrow +
+     USinterest + service + contagion + prioratt - 1 | 1 | 1 |
+     unifgov + lexports + preelec + postelec + rightgov + realinterest +
+         capcont + lreserves
R> flb1 <- as.Formula(flb)
R> flb2 <- update(flb1, attack + defend ~ .)
R> leb1 <- egame12(flb1, data = leblang2003, link = "probit", type = "private")
R> leb2 <- egame12(flb2, data = leblang2003, link = "probit", type = "private")

R> all.equal(coef(leb1), coef(leb2), check.attributes = FALSE)

[1] TRUE
```

The only difference is in the construction of the names of the utility equations. When binary action indicators are used, the outcome names are inferred from the names of the action variables. When numeric or factor outcome variables are used, their values/levels are used as the outcome names.

```
R> cbind(leb1$equations, leb2$equations)
```

```
     [,1]               [,2]
[1,] "u1(no attack)"    "u1(~attack)"
[2,] "u1(devaluation)" "u1(attack,~defend)"
[3,] "u1(defense)"      "u1(attack,defend)"
[4,] "u2(defense)"      "u2(attack,defend)"
```

The methods for specifying the dependent variable differ slightly across models; see the help page of each fitting function for a list of allowable specifications.

3.3. Model fitting

Once the formula has been constructed, it is straightforward to fit a strategic model. All of the fitting functions contain the arguments `data`, `subset`, and `na.action`, which are used in the typical way to construct the model frame. In addition, the `method` argument is passed to `maxLik` (from the **maxLik** package; Toomet, Henningsen, with contributions from Spencer Graves, and Croissant 2010) to select an optimization routine, and other parameters to control the process (e.g., `reltol`, `iterlim`) can be passed as named arguments.

Each fitting function returns an object inheriting from two S3 classes. The first is the `"game"` class, for which most of the methods of interest are defined, including `print` and `summary`. The second is the name of the particular model that was fit; this is used by the `predict` methods. For the most part, the elements of a `"game"` object are the same as those of `"lm"` and `"glm"` objects (e.g., `coefficients`, `vcov`). Pertinent differences include:

- The `log.likelihood` element contains the vector of the n observationwise log-likelihoods evaluated at the parameter estimate, for use in non-nested model tests (see Section 4.2 below).
- The `y` element contains the outcome variable represented as a factor whose levels are the outcome names.
- The `link` and `type` elements store the link function and source of error respectively.
- The `equations` element contains the names of the utility equations and scale terms; this is used by `print.game` and `latexTable` to group the parameters estimated.

Fitted `ultimatum` models contain some additional elements, which are discussed below.

The nonparametric bootstrap is implemented as part of the fitting process via the `boot` argument of the model functions. To run the bootstrap on a model that has already been estimated, use `update` as in the next example. A status bar is printed by default, but it can be suppressed by setting `bootreport = FALSE`.

```
R> set.seed(42)
R> leb1 <- update(leb1, boot = 100)

Running bootstrap iterations...
========================================================================
```

Bootstrap results are stored in the `boot.matrix` element of the fitted model object. When a model has been bootstrapped, the default behavior of `summary.game` is to use the bootstrap results to calculate standard error estimates.

```
R> summary(leb1)

Call:
egame12(formulas = flb1, data = leblang2003, link = "probit",
    type = "private", boot = 100)

Coefficients:
                            Estimate Std. Error z value Pr(>|z|)
u1(no attack):capcont        -0.4525     0.3352   -1.35  0.17710
u1(no attack):lreserves       0.2292     0.0629    3.64  0.00027
u1(no attack):overval        -0.4413     0.1400   -3.15  0.00162
u1(no attack):creditgrow     -0.0648     0.0380   -1.71  0.08800
u1(no attack):USinterest     -0.0505     0.0507   -1.00  0.31902
u1(no attack):service        -0.0288     0.0401   -0.72  0.47291
u1(no attack):contagion      -0.1159     0.0435   -2.66  0.00773
u1(no attack):prioratt       -0.1218     0.0457   -2.66  0.00771
u1(devaluation):(Intercept)  -3.6648     0.3855   -9.51  < 2e-16
u1(defense):(Intercept)      -3.1385     0.4057   -7.74    1e-14
u2(defense):(Intercept)       0.4269     1.7442    0.24  0.80663
u2(defense):unifgov          -0.3568     0.3862   -0.92  0.35559
u2(defense):lexports         -0.1997     0.1891   -1.06  0.29085
u2(defense):preelec           1.6632     1.9215    0.87  0.38672
u2(defense):postelec          1.0623     0.9031    1.18  0.23947
u2(defense):rightgov         -0.9358     0.5176   -1.81  0.07058
u2(defense):realinterest      1.7955     1.0693    1.68  0.09312
u2(defense):capcont           0.0656     1.7098    0.04  0.96940
u2(defense):lreserves         0.3099     0.2046    1.51  0.12985

Standard errors estimated from bootstrap results

Log-likelihood: -482.02
AIC: 1002
No. observations: 7240
```

To see the normal-theory standard errors instead, supply the option `useboot = FALSE` to the `summary` call.

The other arguments for the fitting functions depend on whether the model is one of the discrete extensive form games or the statistical ultimatum game.

Extensive-form models

The stochastic structure of the extensive-form models is specified via the arguments `link` and `type`. The `link` argument is used to specify the distributional form of the error terms: `"probit"` for normal, `"logit"` for type I extreme value. The `type` argument specifies whether the source of randomness is `"agent"` error or `"private"` information. Normal errors must be used in the case of private information; if a model is specified with `link = "logit"` and `type = "private"`, a warning will be issued and a probit link will be enforced.

The error variance σ normally is not estimable on its own, as noted above in Section 2. This is no longer the case if σ is modeled as function of known covariates:

$$\sigma = \exp(\gamma_1 Z_1 + \gamma_2 Z_2 + \ldots + \gamma_k Z_k).$$

The argument `sdformula` is used to estimate γ for such a model. The formula should be one-sided, with nothing to the left of the `~`, as in the following example with the Leblang data.

```
R> leb3 <- egame12(outcome ~ lreserves + overval - 1 | 1 | 1 | preelec +
+     realinterest, sdformula = ~prioratt - 1, data = leblang2003,
+     link = "probit", type = "private")

R> summary(leb3)

Call:
egame12(formulas = outcome ~ lreserves + overval - 1 | 1 | 1 |
    preelec + realinterest, data = leblang2003, link = "probit",
    type = "private", sdformula = ~prioratt - 1)

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
u1(no attack):lreserves      0.2264     0.0464    4.88  1.1e-06
u1(no attack):overval       -0.4381     0.0836   -5.24  1.6e-07
u1(devaluation):(Intercept) -3.0137     0.2470  -12.20  < 2e-16
u1(defense):(Intercept)     -2.8410     0.2275  -12.49  < 2e-16
u2(defense):(Intercept)     -0.0190     0.1988   -0.10   0.9239
u2(defense):preelec          1.3690     0.6402    2.14   0.0325
u2(defense):realinterest     1.4943     0.5610    2.66   0.0077
log(sigma):prioratt          0.0427     0.0186    2.29   0.0221

Standard errors estimated from inverse Hessian

Log-likelihood: -495.37
AIC: 1006.7
No. observations: 7240
```

The equation for the scale parameter coefficients is `log(sigma)`; for models with a logit link, it would be `log(lambda)`. The positive coefficient on prior attacks indicates that outcomes are less predictable (since the stochastic terms are larger relative to the systematic components) for countries that have been victims of speculative currency attacks in the past. Note that it is also possible to estimate separate scale-term equations for each player, by using the argument `sdByPlayer = TRUE` and using an equation of the form `sdformula = scale1vars | scale2vars`.

The extensive-form models also allow for estimation of the error variance when the average payoffs are known to the analyst, such as in data from lab experiments. In this case, the payoffs can be specified with the `fixedUtils` argument. The only information needed from the model formula is the outcome variable, so the `formulas` argument can be written in the form `y ~ .` or `y ~ 1`. When the argument `fixedUtils` is used, the default behavior is to estimate a single common scale parameter, as in the next example.

```
R> lebfixed <- egame12(outcome ~ ., data = leblang2003, fixedUtils = c(1,
+     -1, 0, 1), link = "probit", type = "private")

R> summary(lebfixed)

Call:
egame12(formulas = outcome ~ ., data = leblang2003, link = "probit",
    type = "private", fixedUtils = c(1, -1, 0, 1))

Coefficients:
           Estimate Std. Error z value Pr(>|z|)
log(sigma)  -1.0513     0.0199   -52.9   <2e-16

Standard errors estimated from inverse Hessian

Fixed terms:
  u1(no attack) u1(devaluation)     u1(defense)     u2(defense)
              1              -1               0               1

Log-likelihood: -646.54
AIC: 1295.1
No. observations: 7240
```

Loosely speaking, the higher the estimated scale parameter relative to the utility values, the greater the role of uncertainty in each player's decisions. As before, `sdByPlayer` can be used to estimate a separate scale parameter for each player, and `sdformulas` to specify the scale term(s) as a function of covariates.

The `ultimatum` *model*

In the ultimatum model, each observation consists of the value of the offer made by Player 1 and whether Player 2 accepted it. By assumption, there is an exogenous upper bound on the size of the offer, which is specified via `maxOffer`. The lower bound is always 0. It is

important to be able to identify which offers were at one of these boundary points, since the log-likelihood of an observation depends on whether the offer was interior. If offers are stored as floating-point numbers, naive equality tests may misclassify some boundary observations as interior. To mitigate this, we use the argument `offertol` and code an offer x as meeting the lower bound if $x <$ `offertol` and the upper bound if $x >$ `maxOffer` $-$ `offertol`. Unless there are extremely slight differences between observed offers, on the order of 1×10^{-8}, the default value of `offertol` should suffice for most analyses.

The arguments `s1` and `s2` are for fixing the scale parameters of the stochastic component of the players' reservation values. If either of these is left unspecified, it is estimated. We recommend fixing `s2`, since attempts to estimate it often run into numerical stability issues (Ramsay and Signorino 2009).

The model formula for `ultimatum` should be written in the form `offer + accept ~ R1 | R2`, where `R1` and `R2` contain the variables for Player 1's and 2's reservation values respectively. Some researchers may only have access to data on offer size, but not whether the offer was accepted. For such datasets, run `ultimatum` with the argument `outcome = "offer"` and specify the model formula as `offer ~ R1 | R2`. Parameters for Player 2's reservation value are still estimable in this case, since the optimal offer for Player 1 depends on his or her expectations of the probability of acceptance. Even when acceptance data are available, the option `outcome = "offer"` may be useful for making formal comparisons of the statistical ultimatum model to OLS models of offer size, as in Ramsay and Signorino (2009). For more on model comparison, see Section 4.2 below.

We illustrate the statistical ultimatum game with data from a classroom experiment. Each `gender` variable is an indicator for whether the proposer (1) or receiver (2) is female. We investigate whether players' reservation values—the amount they get if the offer is rejected—is a function of their gender.

```
R> load("ultimatum2010.rda")

R> ult1 <- ultimatum(offer + accept ~ gender1 | gender2, maxOffer = 100,
+     data = ultimatum2010, s2 = 3)

R> summary(ult1)

Call:
ultimatum(formulas = offer + accept ~ gender1 | gender2, data = ultimatum2010,
    maxOffer = 100, s2 = 3)

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
R1:(Intercept)   62.198     18.734    3.32   0.0009
R1:gender1      -25.409     39.290   -0.65   0.5178
R2:(Intercept)   37.684      1.616   23.31   <2e-16
R2:gender2       -0.897      1.150   -0.78   0.4355
log(s1)           3.934      0.430    9.14   <2e-16

Standard errors estimated from inverse Hessian
```

```
Fixed terms:
log(s2)
 1.0986

Log-likelihood: -292.14
AIC: 594.27
No. observations: 32
```

The results indicate that the female students have lower reservation values, meaning they are more likely to make high offers (as the proposer) or accept low ones (as the receiver). However, neither gender coefficient is statistically significant, so this may just be due to sampling error.

3.4. Convergence

The log-likelihood functions for strategic models are not globally concave, so convergence to a global maximum is not guaranteed. We provide two methods to avert convergence problems: well-chosen default starting values and a likelihood profiling method.

In all of the `egame` models, the default starting values come from statistical backward induction (SBI), an equation-by-equation method that uses ordinary probit or logistic regression models to obtain consistent estimates of the parameters (Bas *et al.* 2007).[4] For example, in an `egame12` model with logit link, the procedure is as follows. Let Y_i be an indicator for whether Player i chooses R.

1. Obtain the estimate $\hat{\beta}_{24}$ by running a logistic regression of Y_2 on X_{24} within the subset of observations for which $Y_1 = 1$ (i.e., Player 2's choice is observed).
2. Estimate $\hat{\beta}_{11}$, $\hat{\beta}_{13}$, and $\hat{\beta}_{14}$ as follows:
   (a) Use $\hat{\beta}_{24}$ to generate predicted probabilities $\hat{p}_4$ and $\hat{p}_3$ for Player 2's action.
   (b) Obtain the estimates by running a logistic regression of Y_1 on the expectation-transformed data matrix $\begin{bmatrix} -X_{11} & \hat{p}_3 X_{13} & \hat{p}_4 X_{14} \end{bmatrix}$. The estimated coefficient vector is $\begin{pmatrix} \hat{\beta}_{11} & \hat{\beta}_{13} & \hat{\beta}_{14} \end{pmatrix}^\top$.
3. Multiply the obtained estimates by $\sqrt{2}$ to obtain starting values for the full-information procedure.[5]

The applications of SBI to `egame122` and `egame123` are similar. It is less straightforward to generate starting values for the `ultimatum` model. We use a similar two-step procedure, but it has not been verified as consistent and sometimes yields non-finite likelihoods, in which case starting values of zero (except for the intercept) are used.

[4] SBI is based on the assumption of agent error, so the estimates technically are not consistent for private-information models. However, for strategic models of relatively low complexity like those available in the **games** package, the assumption of agent error or private information makes little difference to the parameter estimates (Signorino 2003b).

[5] This step is not necessary when SBI is used on its own, rather than to generate starting values. The correction is for the additional dispersion induced by the agent error model. Logistic regression uses a dispersion parameter of 1, but equations (2) and (3) along with the assumption $\sigma = 1$ imply a dispersion parameter of $\sqrt{2}$. The change makes no substantive difference for the results.

To assess convergence of an already-fitted model, we implement likelihood profiling via the `profile.game` method. As in the **MASS** package's (Venables and Ripley 2002) `profile.glm` method, this entails refitting the model numerous times, each time holding a single parameter at some value other than the original estimate. In the case of generalized linear models, this profiling procedure is typically used to estimate likelihood-ratio confidence regions (McCullagh and Nelder 1989, 254). However, it can also serve as a rough global convergence check: if the log-likelihood of any of the refit models is greater than that of the original fit, then by definition the original procedure did not converge to a global maximum. When this is the case, as in the following example, `profile.game` issues a warning.

```
R> data(student_offers)
R> stu1 <- ultimatum(offer + accept ~ gender1 | gender2, data = student_offers,
+     maxOffer = 100, s2 = 1)
R> profstu1 <- profile(stu1, which = 1:4)

Warning message:
In profile.game(stu1, which = 1:4) :
  some profiled fits have higher log-likelihood than original fit;
  refit the model using "profile" option
```

The returned object, inheriting from class `"profile.game"`, contains the estimates and log-likelihoods from each refitted model. For visualization of the profile log-likelihood, we provide the `plot.profile.game` method, which displays a spline approximation.

```
R> plot(profstu1)
```

See Figure 2 for the output from this example. Slightly lower values of both the intercept and the gender coefficient for Player 1 appear to yield better-fitting models.

When `profile.game` finds parameters that yield a higher log-likelihood than the original fit, these can be used as starting values in re-estimation of the model via the `profile` argument of the fitting function.

```
R> stu2 <- update(stu1, profile = profstu1)

R> logLik(stu1)

'log Lik.' -663.45 (df=5)

R> logLik(stu2)

'log Lik.' -663 (df=5)
```

3.5. Reporting results

A natural form to present the results of a fitted strategic model is in a table where each row is a covariate and each column is a utility equation. We provide the function `latexTable` to automatically generate LaTeX code for such tables. Table 1 was generated with the following code:

	u1(no attack)	u1(devaluation)	u1(defense)	u2(defense)
(Intercept)		−3.6648	−3.1385	0.4269
		(0.3855)	(0.4057)	(1.7442)
capcont	−0.4525			0.0656
	(0.3352)			(1.7098)
lreserves	0.2292			0.3099
	(0.0629)			(0.2046)
overval	−0.4413			
	(0.1400)			
creditgrow	−0.0648			
	(0.0380)			
USinterest	−0.0505			
	(0.0507)			
service	−0.0288			
	(0.0401)			
contagion	−0.1159			
	(0.0435)			
prioratt	−0.1218			
	(0.0457)			
unifgov				−0.3568
				(0.3862)
lexports				−0.1997
				(0.1891)
preelec				1.6632
				(1.9215)
postelec				1.0623
				(0.9031)
rightgov				−0.9358
				(0.5176)
realinterest				1.7955
				(1.0693)
Log-likelihood	−482.0155			
N	7240			

Table 1: Replication of Leblang's (2003) results.

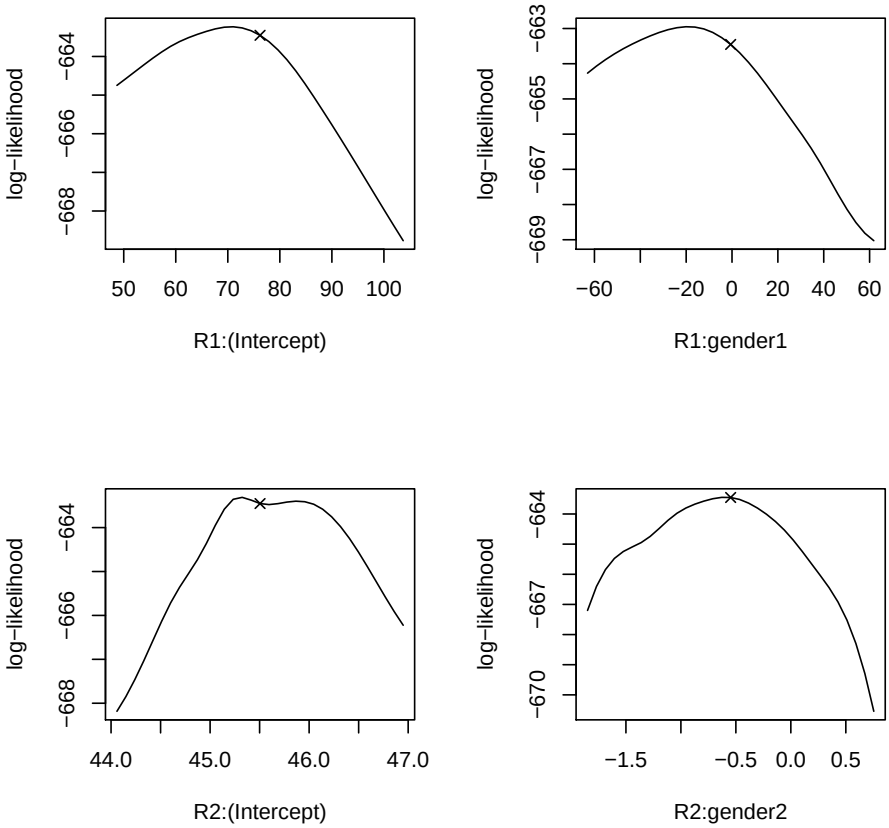


Figure 2: Output of `plot.profile`.

```
R> latexTable(leb1, caption = "Replication of \\citeauthor{Leblang2003}'s \\citeyearpar{Le
+     label = "tab:leb1", floatplace = "p")
```

Additional arguments include `digits` for the number of digits printed, `rowsep` for the point spacing between rows, and `useboot` for the use of bootstrap vs. normal-theory standard errors.

4. Analyzing fitted models

4.1. Predicted probabilities

The raw output from the model fitting functions in **games** describes the effect of each covariate on players' utilities for different outcomes. Some analysts may instead be interested in how each variable affects the probability of a particular outcome occurring. Such probabilities are nonlinear functions of the covariates; for example, they are given by equations (2) and (3) for `egame12` models with agent error. Following popular developments in the political science literature (King, Tomz, and Wittenberg 2000), we provide the function `predProbs` to analyze how the predicted probability of each outcome changes as a function of certain covariates.

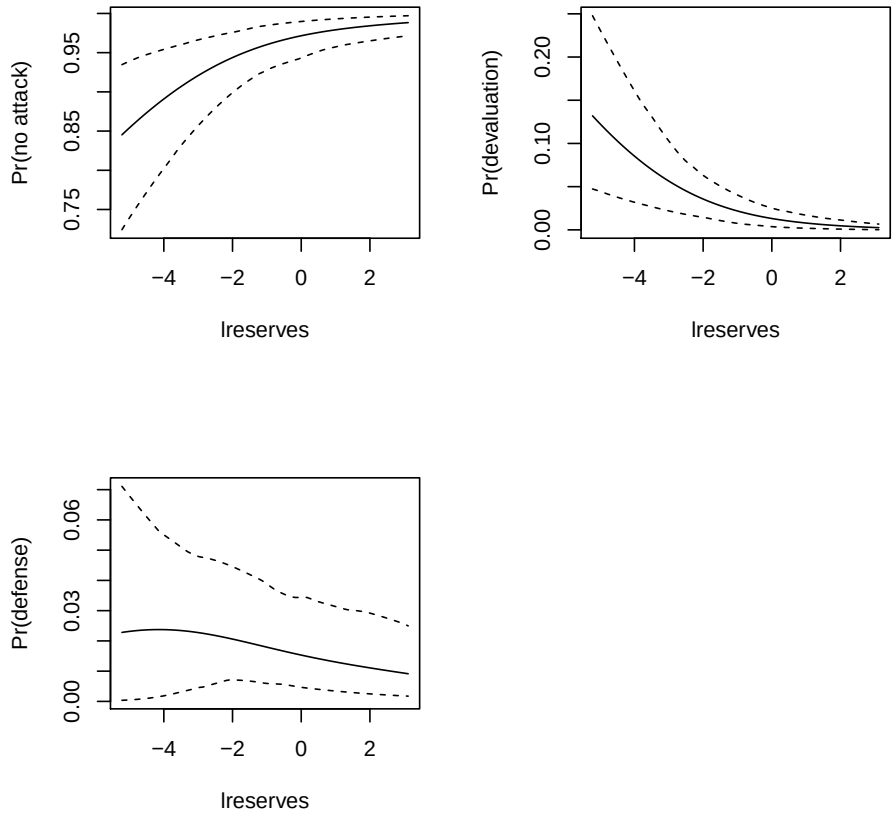


Figure 3: Output of `plot.predProbs`.

The general procedure is:

1. Select a "covariate of interest," X_j.
2. Hold all other variables at their central values — means for continuous variables, medians for binary or ordinal variables, modes for others — or some other pre-specified "profile" $X_{-j} = (X_{j'})_{j' \neq j}$.
3. Using the estimated model, find the predicted probability of each potential outcome over the observed range of X_j, while holding X_{-j} fixed.
4. Calculate confidence intervals for the predicted values using a parametric or nonparametric bootstrap.
5. Plot the results.

The only mandatory arguments for `predProbs` are `model`, for the fitted model object, and `x`, a character string containing the name of the variable of interest (partial matches are allowed). If `x` is numeric, then the default behavior is to evaluate predicted probabilities at

100 grid points along the range observed for `x` in the data used to fit the model (i.e., the data frame `model$model`). The number of grid points and range of values can be controlled via the arguments `n` and `xlim` respectively. If `x` is a factor variable, all available levels are used.

Additional named arguments can be used to change the default profile of values for the covariates other than `x`. These arguments should be specified as `varname = value`, where `varname` exactly matches the name of the variable in the data frame used to fit the model and `value` is an expression that is evaluated within the model frame, `model$model`. For example, to set a variable `y` to its observed 10th percentile, use the argument `y = quantile(y, probs = 0.1)`.

Confidence intervals for the predictions are calculated by resampling. If `model` has a `boot.matrix` element containing nonparametric bootstrap results, these are used. Otherwise, a matrix of parametric bootstrap results is constructed by taking 1,000 samples from a multivariate normal distribution whose mean is $\hat{\beta}$ and whose variance matrix is the inverse of the negative Hessian of the estimates. The default is to compute a 95% confidence interval; this can be controlled by the `ci` argument. It normally takes a few seconds to compute the fitted values for all of the bootstrapped coefficients, so a status bar is displayed. This can be suppressed by setting `report = FALSE`.

We illustrate with an example from Leblang's data. Suppose we are interested in the estimated effect of currency reserves on the outcome probabilities when contagion is high (currency attacks are occurring elsewhere) but all other variables are held at their central values. We would use `predProbs` as follows.

```
R> predleb1 <- predProbs(leb1, x = "lreserves", contagion = max(contagion))

Calculating confidence intervals...
==========================================================================
```

The return value is an object inheriting from classes `"predProbs"` and `"data.frame"`. This is a data frame with `n` (or `nlevels(x)`, if `x` is a factor) rows, each containing a profile, the predicted probabilities for each outcome, and the confidence bands on each predicted probability.

The method `plot.predProbs` can be used for visualization of the output. The number of plots that can be produced from `predProbs` output is equal to the number of outcomes in the corresponding fitted model (e.g., three for an `egame12` model). To deal with this, we have written `plot.predProbs` to behave similarly to `plot.gam` in the **gam** package (Hastie 2011), in which each fitted model corresponds to as many plots as there are covariates.

```
R> par(mfrow = c(2, 2))
R> plot(predleb1)
```

See Figure 3 for the output. If no additional arguments are specified to `plot.predProbs(x)`, all of the plots are printed in sequence. If `ask = TRUE` is specified, then an interactive menu is used for plot selection:

```
R> plot(predleb1, ask = TRUE)
Make a plot selection (or 0 to exit):
```

```
1: plot: Pr(no attack)
2: plot: Pr(devaluation)
3: plot: Pr(defense)
4: plot all terms
```

The argument `which` can be used to select one of these without bringing up the menu; e.g., `plot(predleb1, which = 2)` will produce only the plot for the devaluation outcome. In each case, all of the standard plotting arguments can be used to control the output. To change the line type used for the confidence bands, use the argument `lty.ci`.

4.2. Non-nested model comparisons

It is not possible to express traditional discrete-choice models like logistic regression as "restricted" strategic models, or vice versa. Therefore, standard likelihood ratio tests are inappropriate for comparing the fit of a strategic model to that of a generalized linear model; a non-nested model comparison is necessary (Clarke and Signorino 2010). The **games** package implements the tests of Vuong (1989) and Clarke (2006) via the `vuong` and `clarke` functions respectively. Each test compares two models, under the null hypothesis that the two have an equal Kullback-Leibler distance from the true model. Both use test statistics formed from the log-likelihood contributions of each individual observation. The main difference is that Clarke's test is unbiased in finite samples, whereas Vuong's depends on asymptotic properties. We implement both tests with the recommended BIC-based correction to penalize overparameterization.

The simplest use of the non-nested test functions is to compare two strategic models to each other. For example, we can use them to determine whether agent error or private information is more appropriate for Leblang's data.

```
R> lebagent <- update(leb1, type = "agent", boot = 0)

R> vuong(leb1, lebagent)

Vuong test for non-nested models

Model 1 log-likelihood: -482
Model 2 log-likelihood: -482
Observations: 7240
Test statistic: -0.15

Neither model is significantly preferred (p = 0.88)
```

Neither stochastic structure appears to be significantly preferred over the other.

It is somewhat less straightforward to compare strategic to non-strategic models. Vuong's and Clarke's tests can be applied only to pairs of models for which the dependent variable is exactly the same. In a strategic model lik `egame12`, the dependent variable for each observation is the outcome reached—i.e., the vector of all decisions made by each player. By contrast, in a standard (binary) logistic regression model, the dependent variable is an indicator for

whether one particular outcome was reached. To allow for comparisons in such cases, the `vuong` and `clarke` functions have `outcome` arguments. For example, we could compare the strategic model of Leblang's data to a logistic regression in terms of their ability to predict the occurrence of speculative attacks as follows.

```
R> leblang2003$noattack <- 1 - leblang2003$attack
R> logit1 <- glm(noattack ~ lreserves + overval + creditgrow + contagion +
+     prioratt + rightgov + realinterest, data = leblang2003, family = binomial)

R> vuong(model1 = leb1, outcome1 = 1, model2 = logit1)

Vuong test for non-nested models

Model 1 log-likelihood: -430
Model 2 log-likelihood: -431
Observations: 7240
Test statistic: -9.8

Model 2 is preferred (p < 2e-16)
```

The logistic regression is preferred despite having a lower log-likelihood, since it fits 8 parameters compared to the strategic model's 19. The argument `outcome1 = 1` is used to indicate that the strategic model should be evaluated in terms of its fit with the market's decision not to initiate a currency attack. We would have used `outcome1 = 2` to consider the outcome of a speculative attack followed by devaluation, and `outcome1 = 3` for an attack followed by defense. Of course, these could not have been compared to `logit1`, and `vuong` or `clarke` would stop with an error after detecting models with different dependent variables.

5. Conclusion

We have provided new software, the **games** package, for estimation of strategic statistical models in R. We argue that such models are appropriate for the analysis of data where agents' expectations of each other's actions determine their choices. The new software implements multiple strategic models, including a statistical bargaining game. The software is easy to use and includes post-estimation features such as non-nested comparison tests and plots of fitted values with measures of uncertainty. We show that the software can be used to easily replicate one well-known analysis of a strategic statistical model (Leblang 2003).

Acknowledgments

We are grateful to David Leblang for sharing his data. We thank the Wallis Institute for Political Economy at the University of Rochester for financial support.

References

Bas MA, Signorino CS, Walker RW (2007). "Statistical Backwards Induction: A Simple Method for Estimating Recursive Strategic Models." *Political Analysis*, **16**(1), 21–40.

Botelho A, Harrison GW, Hirsch MA, Rutström EE (2005). "Bargaining behavior, demographics and nationality: What can the experimental evidence show?" *Research in Experimental Economics*, **10**(4), 337–372.

Carson JL (2003). "Strategic Interaction and Candidate Competition in U.S. House Elections: Empirical Applications of Probit and Strategic Probit Models." *Political Analysis*, **11**(4), 368–380. ISSN 1047-1987. doi:10.1093/pan/mpg022.

Carson JL (2005). "Strategy, Selection, and Candidate Competition in U.S. House and Senate Elections." *The Journal of Politics*, **67**(1), 1–28. ISSN 0022-3816. doi:10.1111/j.1468-2508.2005.00305.x.

Carter DB (2010). "The Strategy of Territorial Conflict." *American Journal of Political Science*, **54**(4), 969–987. doi:10.1111/j.1540-5907.2010.00471.x.

Clarke KA (2006). "A Simple Distribution-Free Test for Nonnested Model Selection." *Political Analysis*, **15**(3), 347–363. ISSN 1047-1987. doi:10.1093/pan/mpm004. URL http://pan.oxfordjournals.org/cgi/doi/10.1093/pan/mpm004.

Clarke KA, Signorino CS (2010). "Discriminating Methods: Tests for Non-nested Discrete Choice Models." *Political Studies*, **58**(2), 368–388. ISSN 00323217. doi:10.1111/j.1467-9248.2009.00813.x. URL http://doi.wiley.com/10.1111/j.1467-9248.2009.00813.x.

Hastie T (2011). ***gam**: Generalized Additive Models*. R package version 1.04, URL http://CRAN.R-project.org/package=gam.

Helmke G (2010). "The Origins of Institutional Crises in Latin America." *American Journal of Political Science*, **54**(3), 737–750. ISSN 00925853. doi:10.1111/j.1540-5907.2010.00457.x.

King G, Tomz M, Wittenberg J (2000). "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science*, **44**(2), 347–361.

Leblang D (2003). "To Devalue or to Defend? The Political Economy of Exchange Rate Policy." *International Studies Quarterly*, **47**(4), 533–560. ISSN 0020-8833. doi:10.1046/j.0020-8833.2003.00278.x.

Lewis JB, Schultz KA (2003). "Revealing Preferences: Empirical Estimation of a Crisis Bargaining Game with Incomplete Information." *Political Analysis*, **11**(4), 345–367.

McCullagh P, Nelder J (1989). *Generalized Linear Models*. Second edition. Chapman and Hall, London.

McKelvey RD, Palfrey TR (1998). "Quantal response equilibria for extensive form games." *Experimental Economics*, **1**(1), 9–41. ISSN 1386-4157. doi:10.1007/BF01426213.

McLean EV, Whang T (2010). "Friends or Foes? Major Trading Partners and the Success of Economic Sanctions." *International Studies Quarterly*, **54**(2), 427–447. ISSN 00208833. doi:10.1111/j.1468-2478.2010.00594.x.

Ramsay KW, Signorino CS (2009). "A Statistical Model of the Ultimatum Game." *Working paper*.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Signorino CS (1999). "Strategic Interaction and the Statistical Analysis of International Conflict." *The American Political Science Review*, **93**(2), 279–297.

Signorino CS (2002). "Strategy and Selection in International Relations." *International Interactions*, **28**(1), 93–115.

Signorino CS (2003a). ***Strat**: A Program for Analyzing Statistical Strategic Models*. Gauss package version 1.4, URL http://www.rochester.edu/college/psc/signorino/research/stratman.pdf.

Signorino CS (2003b). "Structure and Uncertainty in Discrete Choice Models." *Political Analysis*, **11**(4), 316–344.

Signorino CS, Tarar A (2006). "A Unified Theory and Test of Extended Immediate Deterrence." *American Journal of Political Science*, **50**(3), 586–605.

Signorino CS, Yilmaz K (2003). "Strategic Misspecification in Regression Models." *American Journal of Political Science*, **47**(3), 551–566.

Toomet O, Henningsen A, with contributions from Spencer Graves, Croissant Y (2010). ***maxLik**: Maximum Likelihood Estimation*. R package version 1.0-0, URL http://CRAN.R-project.org/package=maxLik.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Vuong QH (1989). "Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses." *Econometrica*, **57**(2), 307–333. URL http://www.jstor.org/stable/1912557.

Wickham H (2010). ***stringr**: Make it easier to work with strings*. R package version 0.4, URL http://CRAN.R-project.org/package=stringr.

Zeileis A, Croissant Y (2010). "Extended Model Formulas in R: Multiple Parts and Multiple Responses." *Journal of Statistical Software*, **34**(1), 1–13. URL http://www.jstatsoft.org/v34/i01/.

Affiliation:

Brenton Kenkel
315A Harkness Hall
Department of Political Science
University of Rochester
Rochester, NY 14627
Email: brenton.kenkel@gmail.com

Curtis S. Signorino
303 Harkness Hall
Department of Political Science
University of Rochester
Rochester, NY 14627
Email: curt.signorino@rochester.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd