Discrete Choice 1.0

for GAUSS™ Mathematical and Statistical System

Aptech Systems, Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of Aptech Systems, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Aptech Systems, Inc.

©Copyright 1997-2010 by Aptech Systems, Inc., Black Diamond, WA.
All Rights Reserved.

GAUSS, **GAUSS Engine** and **GAUSS Light** are trademarks of Aptech Systems, Inc. Other trademarks are the property of their respective owners.

Part Number: `003651`
Version 1.0
Documentation Revision: 1861 May 24, 2010

Contents

Installation 1

1.1 UNIX/Linux/Mac

If you are unfamiliar with UNIX/Linux/Mac, see your system administrator or system documentation for information on the system commands referred to below.

1.1.1 Download

1. Copy the `.tar.gz` or `.zip` file to `/tmp`.
2. If the file has a `.tar.gz` extension, unzip it using `gunzip`. Otherwise skip to step 3.

   `gunzip` app_*appname*_*vernum*.*revnum*_`UNIX.tar.gz`
3. `cd` to your **GAUSS** or **GAUSS Engine** installation directory. We are assuming `/usr/local/gauss` in this case.

   ```
   cd /usr/local/gauss
   ```

4. Use `tar` or `unzip`, depending on the file name extension, to extract the file.

   `tar xvf /tmp/app_`*appname_vernum*`.`*revnum*`_UNIX.tar`

   – or –

   `unzip /tmp/app_`*appname_vernum*`.`*revnum*`_UNIX.zip`

1.1.2 CD

1. Insert the Apps CD into your machine's CD-ROM drive.
2. Open a terminal window.
3. `cd` to your current **GAUSS** or **GAUSS Engine** installation directory. We are assuming `/usr/local/gauss` in this case.

   ```
   cd /usr/local/gauss
   ```

4. Use `tar` or `unzip`, depending on the file name extensions, to extract the files found on the CD. For example:

   `tar xvf /cdrom/apps/app_`*appname_vernum*`.`*revnum*`_UNIX.tar`

   – or –

   `unzip /cdrom/apps/app_`*appname_vernum*`.`*revnum*`_UNIX.zip`

   However, note that the paths may be different on your machine.

1.2 Windows

1.2.1 Download

Unzip the `.zip` file into your **GAUSS** or **GAUSS Engine** installation directory.

1.2.2 CD

1. Insert the Apps CD into your machine's CD-ROM drive.

2. Unzip the `.zip` files found on the CD to your **GAUSS** or **GAUSS Engine** installation directory.

1.2.3 64-Bit Windows

If you have both the 64-bit version of **GAUSS** and the 32-bit Companion Edition installed on your machine, you need to install any **GAUSS** applications you own in both **GAUSS** installation directories.

1.3 Difference Between the UNIX and Windows Versions

- If the functions can be controlled during execution by entering keystrokes from the keyboard, it may be necessary to press ENTER after the keystroke in the UNIX version.

Getting Started 2

GAUSS 5.0.30+ is required to use the **Discrete Choice** routines. See **`_rtl_ver`** in `src/gauss.dec`.

The **Discrete Choice** version number is stored in a global variable:

`_dc_ver`	3×1 matrix, the first element contains the major version number, the second element the minor version number, and the third element the revision number.

If you call for technical support, you may be asked for the version of your copy of **Discrete Choice**.

2.0.1 README Files

If there is a **`README.dc`** file, it contains any last minute information on the **Discrete Choice** procedures. Please read it before using them.

2.0.2 Setup

In order to use the procedures in the **Discrete Choice** or **DC** Module, the **DC** library must be active. This is done by including **`dc`** in the **`library`** statement at the top of your program or command file along with the **`#include`** command for `dc.sdf`:

```
library dc;
#include dc.sdf
```

This enables **GAUSS** to find the **DC** procedures.

Discrete Choice 3

Discrete Choice estimation uses the **`sqpsolvemt`** procedure, a seqential quadratic programming method that solves general nonlinear programming problems. Data are passed in a **`dcDesc`** structure instance. The optimization process is controlled with a **`dcControl`** structure instance. Output arguments are in a **`dcOut`** structure instance and an array of **`ds`** structure instances.

3.1 Poisson Model

Given independent variables x_i for an observation with count y_i, the Poisson density function is

$$P(y_i|x_i) = \frac{\exp(-\mu_i)\mu_i^{y_i}}{y_i!}$$

where

$$\mu_i = E(y_i|x_i) = \exp(x_i\beta)$$

is the number of events expected to occur per unit time (or space).

The Poisson regression model log likelihood function is:

$$lnL = \sum_{i=1}^{n}[-\mu_i + y_i\beta' x_i - ln(y_i!)]$$

The Poisson distribution function is

$$F(c) = P(y_i \leq c) = \sum_{j=0}^{c} P(y_i = j|x_i)$$

3.1.1 Poisson Overdispersion

The **`psnprt`** function shows three tests for overdispersion when a Poisson model is estimated.

Following Cameron and Trivedi's (1998, p62) notation, let $\omega_i = V[y_i|\mathbf{x}_i]$ be the conditional variance of y_i. Two possible variance functions are the NB1 and NB2 functions:

$$\begin{aligned} NB1 &: \quad \omega_i = (1+\alpha)\mu_i \\ NB2 &: \quad \omega_i = \mu_i + \alpha\mu_i^2 \end{aligned}$$

Tests of $Ho : \alpha = 0$ in both cases are conducted using auxillary regressions. Overdispersion of the $NB1$ form is indicated by a significant t statistic for $\widehat{\alpha}$ in the regression $\frac{(y_i-\widehat{\mu}_i)^2-y_i}{\widehat{\mu}_i} = \alpha + u_i$. Overdispersion of the $NB2$ form is indicated by a significant t statistic for $\widehat{\alpha}$ in the regression $\frac{(y_i-\widehat{\mu}_i)^2-y_i}{\widehat{\mu}_i} = \alpha\widehat{\mu}_i + u_i$. In both cases u_i is an i.i.d. disturbance term. **`psnprt`** reports the t statistics for both cases and their probability values, against a two sided alternative hypothesis.

A Lagrange Multiplier test for overdispersion is presented by Greene (2000, pp. 885-886). The Poisson model is a restriction on the Negative Binomial model. The LM statistic has a $\chi^2(1)$ distribution under the null hypothesis that the mean equals the variance.

$$LM = \frac{(e'e - N\overline{y})^2}{2\widehat{\mu}'\widehat{\mu}}$$

where e is an $Nx1$ vector of residuals and $\widehat{\mu}$ the $Nx1$ vector of fitted values. ”psnprt” reports this statistic and its probability value.

3.2 Negative Binomial Model

The Poisson model assumes that the conditional variance always equal the conditional mean. Consistent but inefficient Poisson model estimates and downward biased standard errors result if this assumption is not true (Gourieroux et al., 1984, Cameron and Trivedi, 1986, p. 31).

The negative binomial regression model lets the conditional variance exceed the conditional mean. Let the conditional mean, μ_i be:

$$\mu_i = \exp(x_i\beta + \varepsilon_i)$$

where ε is random and uncorrelated with x. Rewrite (1) in terms of the Poisson mean to get

$$\mu_i = \exp(x_i\beta)\exp(\varepsilon_i) = \mu_i \exp(\varepsilon_i) = \mu_i\delta_i$$

Assume that δ_i has a gamma distribution with parameter υ_i (this sets $E(\delta_i) = 1$, identifying the model, and $Var(\delta_i) = 1/\upsilon_i$) and integrate $P(y_i|x_i;\delta_i)$ over the unknown δ_i to get the negative binomial density function:

$$P(y_i|x_i) = \frac{\Gamma(y_i+\upsilon_i)}{y_i!\Gamma(\upsilon_i)}(\frac{\upsilon_i}{\upsilon_i+\mu_i})^{\upsilon_i}(\frac{\mu_i}{\upsilon_i+\mu_i})^{y_i}$$

with distribution function

$$F(c) = P(y_i \le c) = \sum_{j=0}^{c} P(y_i = j|x_i)$$

The conditional variance is

$$Var(y_i|x_i) = \mu_i(1+\frac{\mu_i}{\upsilon_i}) = \exp(x_i\beta)(1+\frac{x_i\beta}{\upsilon_i})$$

which is greater than the conditional variance of the Poisson distribution.

3.3 Truncation and Censoring

This discussion of truncated and censored models closely follows Hayashi (2000) and Long (1997). It assumes that $\{y_t, \mathbf{x}_t\}$ is i.i.d.

y_t is truncated if observations above or below given levels are not in the sample. A double truncation rule is that y_t is observable if it is greater than c_l or less than c_u. The density function is

$$\begin{aligned} f(y|(y > c_l) and (y < c_u)) &= \frac{f(y)}{P((y>c_l) and (y<c_u))} \\ &= \frac{f(y)}{F(c_l)(1-F(c_u))} \end{aligned}$$

where F is the cumulative distribution function of y. The corresponding log conditional likelihood function is

$$\begin{aligned} L(y_t|x_t;\theta,c_l,c_u) &= \log(f(y_t|x_t;\theta,c_l,c_u)) - \log(F(c_l|x_t;\theta,c_l,c_u)) \\ &\quad - \log(1 - F(c_u|x_t;\theta,c_l,c_u)) \end{aligned}$$

where θ represents all parameters of the distribution.

A censored model is defined by

$$y_t^* = x_t\beta + \varepsilon_t,\ t = 1, 2, ..., n$$

with observed y_t values:

$$y_t = \left\{ \begin{array}{c} y_t^* \text{ if } y_t^* > c_l \text{ and } y_t^* < c_u \\ c_l \text{ if } y_t^* < c_l \\ c_u \text{ if } y_t^* > c_u \end{array} \right\}$$

where c_l and c_u are known. All observations are in the sample, though the observable values, y_t, for which $y_t^* > c_l$ and $y_t^* < c_u$ are set equal to c_l and c_u respectively.

The density of y_t is

$$[f(y_t|x_t, \theta, c_u, c_l)]^{1-(D_u+D_l)} \times [F(c_l)]^{D_l} \times [1 - F(c_u)]^{D_u}$$

where

$$\begin{aligned} D_l &= \begin{matrix} 0 \text{ if } y_t > c_l \text{ (i.e. } y_t^* > c_l) \\ 1 \text{ if } y_t = c_l \text{ (i.e. } y_t^* \leq c_l) \end{matrix} \\ D_u &= \begin{matrix} 0 \text{ if } y_t < c_u \text{ (i.e. } y_t^* \leq c_u) \\ 1 \text{ if } y_t = c_u \text{ (i.e. } y_t^* > c_u) \end{matrix} \end{aligned}$$

with the corresponding conditional log likelihood

$$\begin{aligned} \log\ f(y_t|x_t; \theta, c_u, c_l) &= (1 - (D_u + D_l)) \log f(y_t|xxx) \\ &+ D_l \log F(c_l) + D_u \log[1 - F(c_u)] \end{aligned}$$

3.4 Zero-Inflated Models

A zero-inflated (sometimes called zero-altered) model allows for the possibility that count outcomes equal to zero are generated by two regimes: a regime where the outcome is always zero and either a Poisson or Negative Binomial model with zero as one of the outcomes.

Suppose $z_i = 0$ when regime 1 generates outcome i (equalling zero) and $z_i = 1$ when regime two generates outcome i (possibly equalling zero).

$P[z_i = 1]$ is determined by a logit or probit model and $P[y_i = j|z_i = 1]$ is given by a Poisson probability density function.

Greene (2000, p. 890) summarizes these ideas, citing works by Mullahey (1986), Heilbron (1989), Lambert (1992), Johnson and Kotz (1970), and Greene (1994):

$$\begin{aligned} P[y_i = 0] &= P[y_i = 0|regime1]P[regime1] + P[y_i = 0|regime2]P[regime2] \\ &= P[regime1] + P[y_i = 0|regime2]P[regime2] \\ P[y_i = j] &= P[y_i = j|regime2]P[regime2] \quad j = 1, 2, ... \end{aligned}$$

3.4.1 Testing Zero-Inflated Regime Assumptions

Vuong (1989) proposes a method that can be used to test whether two regimes likely generate the data. The statistic compares the probabilities of counts occurring under two regimes. Following Greene's (2000, p. 891) notation, let $f_i(y_i|\mathbf{x}_i)$ be the predicted probability that y_i is observed assuming the data are sampled from distribution j, $j = 1, 2$. Compare these values with

$$m_i = \log\left(\frac{f_1(y_i|\mathbf{x}_i)}{f_1(y_i|\mathbf{x}_i)}\right)$$

Vuong's statistic is:

$$v = \frac{\sqrt{N}[\frac{1}{N}\sum_{i=1}^{N} m_i]}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(m_i - \overline{m})^2}}$$

which converges in distribution to a standard normal distribution. Large values of v suggest that model 1 more likely generates the data while small values of v suggest that model 2 more likely generates the data.

3.5 Multinomial Logit Model

The **`MultinomialLogit`** procedure estimates a multinomial logit model.

For the probability of observing $y_i = m$ we have

$$Pr(y_i = m|x_i) = \frac{exp(x_i\beta_m)}{\sum_{j=1}^{J} exp(x_i\beta_j)}$$

By default the set of coefficients for the first category, β_1, is set to a zero vector as a "reference" category. This can be modified by the user to any of the categories.

Estimates are found by minimizing

$$-lnL = -\sum_{i=1}^{N} Pr(y_i = m|x_i)$$

3.6 Adjacent Categories Multinomial Logit

The adjacent categories model is a special case of multinomial logit (Long, 1997, p. 146). It specifies that the log odds of one category versus the next higher category is linear in the cutpoints and explanatory variables, i.e.

$$\ln\left[\frac{P(y_i = j+1|\mathbf{x}_i)}{P(y_i = j|\mathbf{x}_i)}\right] = x_i\beta_j$$

This implies

$$\begin{aligned}
\beta_1^{mnl} &= \beta_1^{acl} \\
\beta_2^{mnl} &= \beta_2^{acl} + \beta_1^{acl} \\
\beta_3^{mnl} &= \beta_3^{acl} + \beta_2^{acl} + \beta_1^{acl} \\
&\vdots
\end{aligned}$$

`AdjacentCategories` first estimates the standard multinomial logit model, transforms the β_m^{mnl} parameters to the β_m^{acl} parameters, and computes the covariance matrix of the parameters by the delta method.

3.7 Stereotype Multinomial Logit

For the stereotype model, regression vectors across categories are constrained to a linear function of each other. For $Pr(y_i = m|x_i)$ we have

$$Pr(y_i = m|x_i) = \frac{exp(x_i\phi_m\beta_m)}{\sum_{j=1}^{J} exp(x_i\phi_j\beta_j)}$$

where ϕ_m is a distance coefficient. This model requires two reference categories, one with the distance set to zero, and another which is set to one. By default $\phi_0 = 0$ and $\phi_M = 1$. The remaining distances are constrained to be between zero and one.

3.8 Ordered Logit/Probit

Suppose $y_i^* = x_i\beta + \epsilon$ is an unobserved latent variable where x_i is $1xK$, β is $Kx1$, and ϵ is i.i.d. logistic with zero mean and variance $\frac{\pi^2}{3}$.There are J ordinal categories. The model is identified by excluding the constant term. (See Long, 1997, p. 124 for discussion of alternate parameterizations.)

The observed y for an individual depends on the intensity of y^* relative to cutpoint parameters $\tau_i \;\; i = 1, ..., J-1$, defined by

$$P(y_i = j|\mathbf{x}_i) = P(\tau_{j-1} \leq y_i^* < \tau_j|\mathbf{x}_i) = F(\tau_j - x_i\beta|\mathbf{x}_i) - F(\tau_{j-1} - x_i\beta|\mathbf{x}_i)$$

where $\tau_0 = -\infty, \;\; 0 < \tau_1 < ... < \tau_{J-1}$ and $F(j|\mathbf{x}_i) = P(y_i \leq j|\mathbf{x}_i) = \sum_{k=1}^{j} P(y_i = k|\mathbf{x}_i)$. F is a logit cumulative distribution function.

The cumulative log odds in the ordered logit model is linear in the cutpoints and explanatory variables, i.e.,

$$\ln\left[\frac{P(y_i \leq j|\mathbf{x}_i)}{P(y_i > j|\mathbf{x}_i)}\right] = \tau_j - x_i\beta$$

The ordered log likelihood is:

$$\ln L(\beta, \tau) = \sum_{j=1}^{J} \sum_{y_i=j} \ln[F(\tau_j - x_i\beta|\mathbf{x}_i) - F(\tau_{j-1} - x_i\beta|\mathbf{x}_i)]$$

For the ordered logit model, F is the cdf of the logistic distribution and for the ordered probit model, F is the Normal cdf.

3.9 Conditional Logit

In the conditional logit model, variables that measure the attributes of the categories are added to the model.

$$Pr(y_i = m|x_i, z_{ij}) = \frac{exp(x_i\beta_m + z_{im}\gamma)}{\sum_{j=1}^{J} exp(x_i\beta_j + z_{ij}\gamma)}$$

3.9.1 Example

We have 152 respondents reporting preferences for mode of transportation between Sydney and Melbourne by train, bus, and car (Hensher and Greene, 1995; air travel excluded for the purposes of this example). Several attributes of these categories were recorded, TTME - terminal waiting time (zero for car), INVT – in-vehicle time, INVC – in-vehicle cost, and GC – a generalized cost measure. The command file for this estimation is

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "Mode";
d1.catvarname = "choiceno";
d1.catnames = "train"$|"bus"$|"car";
d1.atnames = "ttme"$|"invc"$|"invt"$|"GC";

d1.noconstant = 1;

struct dcout dcout1;

dcout1 = dcConditionalLogit("powersxie",d1,dccontrolCreate);
call dcprt(dcout1);
```

The results are

```
                   CONDITIONAL LOGIT RESULTS
                      2003-08-17 14:50:27

Number of Observations:   152
Degrees of Freedom:       148

  1 - train
  2 - bus
  3 - car


DISTRIBUTION AMONG OUTCOME CATEGORIES for Mode


Dependent Variable       Proportion
train                     0.4145
bus                       0.1974
car                       0.3882





ATTRIBUTES

  ttme


Attribute Variable         Mean         Std Dev       Minimum       Maximum
train                    0.0000       0.0000        0.0000        0.0000
bus                      6.4934       17.1293      -46.0000       43.0000
car                    -34.6250       13.8744      -99.0000      -1.0000

  invc


Attribute Variable         Mean         Std Dev       Minimum       Maximum
train                    0.0000       0.0000        0.0000        0.0000
bus                    -15.4737       20.5984      -61.0000       25.0000
car                    -28.5263       23.8895      -87.0000       26.0000
```

```
  invt


Attribute Variable        Mean         Std Dev       Minimum       Maximum
train                   0.0000        0.0000        0.0000        0.0000
bus                     24.6842       94.7064      -317.0000      552.0000
car                    -30.5461       131.0412     -327.0000      522.0000

  GC


Attribute Variable        Mean         Std Dev       Minimum       Maximum
train                   0.0000        0.0000        0.0000        0.0000
bus                    -12.0461       27.4335      -86.0000       105.0000
car                    -32.4079       32.9388      -120.0000      78.0000

                        ATTRIBUTE COEFFICIENTS


       Variable      Coefficient     Std Err       t-stat       Prob
       ttme          -0.0022          0.0071       -0.3138       0.7537
       invc          -0.4351          0.1328       -3.2770       0.0010
       invt          -0.0772          0.0194       -3.9912       0.0001
       GC             0.4312          0.1332        3.2371       0.0012





                MARGINAL EFFECTS OF ATTRIBUTE VARIABLES
           partial probability with respect to mean attribute


train


train
```

```
        Variable    Coefficient  Std Err      t-stat       Prob
        ttme        -0.0003       0.0009      -0.3129       0.7544
        invc        -0.0564       0.0159      -3.5477       0.0004
        invt        -0.0100       0.0023      -4.4402       0.0000
        GC           0.0559       0.0161       3.4728       0.0005



bus


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme         0.0001       0.0003       0.3222       0.7473
        invc         0.0184       0.0053       3.5069       0.0005
        invt         0.0033       0.0008       4.2558       0.0000
        GC          -0.0183       0.0053      -3.4586       0.0005



car


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme         0.0002       0.0006       0.3085       0.7577
        invc         0.0379       0.0111       3.4094       0.0007
        invt         0.0067       0.0016       4.2251       0.0000
        GC          -0.0376       0.0113      -3.3322       0.0009



bus


train


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme         0.0001       0.0003       0.3222       0.7473
        invc         0.0184       0.0053       3.5069       0.0005
        invt         0.0033       0.0008       4.2558       0.0000
        GC          -0.0183       0.0053      -3.4586       0.0005


    bus


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme        -0.0002       0.0007      -0.3188       0.7499
        invc        -0.0423       0.0123      -3.4475       0.0006
        invt        -0.0075       0.0017      -4.3723       0.0000
        GC           0.0419       0.0122       3.4304       0.0006


    car


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme         0.0001       0.0004       0.3161       0.7520
        invc         0.0238       0.0074       3.2256       0.0013
        invt         0.0042       0.0010       4.0875       0.0000
        GC          -0.0236       0.0073      -3.2303       0.0012


    car


    train


        Variable    Coefficient  Std Err      t-stat       Prob
        ttme         0.0002       0.0006       0.3085       0.7577
        invc         0.0379       0.0111       3.4094       0.0007
        invt         0.0067       0.0016       4.2251       0.0000
        GC          -0.0376       0.0113      -3.3322       0.0009


    bus
```

```
       Variable    Coefficient  Std Err      t-stat       Prob
       ttme         0.0001       0.0004       0.3161       0.7520
       invc         0.0238       0.0074       3.2256       0.0013
       invt         0.0042       0.0010       4.0875       0.0000
       GC          -0.0236       0.0073      -3.2303       0.0012



car


       Variable    Coefficient  Std Err      t-stat       Prob
       ttme        -0.0003       0.0010      -0.3114       0.7555
       invc        -0.0618       0.0181      -3.4109       0.0006
       invt        -0.0110       0.0025      -4.3221       0.0000
       GC           0.0612       0.0182       3.3648       0.0008


********************SUMMARY STATISTICS********************

MEASURES OF FIT:

  -2 Ln(Lu):                                    192.6971
  -2 Ln(Lr): All coeffs equal zero              333.9781
  -2 Ln(Lr): J-1 intercepts                     320.0034
  LR Chi-Square (coeffs equal zero):            141.2811
       d.f.                                       4.0000
       p-value =                                  0.0000
  LR Chi-Square (J-1 intercepts):               127.3064
       d.f.                                       2.0000
       p-value =                                  0.0000
  Count R2, Percent Correctly Predicted:          0.8092
  Adjusted Percent Correctly Predicted:          -0.6988
  Madalla's pseudo R-square:                      0.5672
  McFadden's pseudo R-square:                     0.3978
  Ben-Akiva and Lerman's Adjusted R-square:       0.3978
  Cragg and Uhler's pseudo R-square:              0.1818
  Akaike Information Criterion:                   1.3204
  Bayesian Information Criterion1:                0.0796
  Hannan-Quinn Information Criterion:             1.3527
```

```
OBSERVED and PREDICTED OUTCOMES

          |           Predicted
 Observed |     Y01      Y02      Y03    Total
 --------------------------------------------------
      Y01 |      47        0       16       63
      Y02 |       0       23        7       30
      Y03 |       3        3       53       59

 --------------------------------------------------
    Total|      50       26       76      152
```

3.10 Nested Logit

NestedLogit is a generalization of the conditional logit model in which categories are grouped into subcategories. Define the probability of an observation being in the m-th category given being in the j-th subcategory:

$$P_{m|j} = \frac{exp(z_{m|j}\beta_1)}{\sum_k^J exp(z_{k|j}\beta_1)}$$

Now let

$$P_j = \frac{exp(z_j\beta_2 + \tau_j I_j)}{\sum_k^J exp(z_j\beta_2 + \tau_k I_k)}$$

where

$$I_j = ln\left(\sum_{k=1}^{K_j} exp(z_{m|j}\beta_1)\right)$$

$\rho_j = 1 - \tau_j$ can be interpreted as an approximate subcategory correlation (Maddala, 1983).

Then, the joint probability of category and subcategory is

$$P_{m,j} = P_{m|j} P_j$$

and maximum likelihood estimates are produced by minimizing

$$-lnL = -\sum_{i=1}^{N} P_{m,j}$$

This model can be generalized to any number of levels of subcategories (Maddala, 1983; Greene, 2000).

3.10.1 Example

This example is presented in Greene, 2000, p. 868, and the data are from Greene and Hensher, 1997. The dataset contains 210 observations on choices between air, train, bus, and car modes of transportation. Attributes of the first level categories are TTME – terminal time, and GC – a generalized cost of transportation. These categories are grouped into two subcategories, Air and Ground, and an attribute of these categories is AIRHINC – traveling by air times household income.

The command file for this problem is

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;
```

```
d1.level = reshape(d1.level,2,1);

d1.yname = "Mode";
d1.catnames = "Air"$|"Train"$|"Bus"$|"Car";
d1.refcatName = "Car";

d1.level[1].atNames = "TTME"$|"GC";
d1.level[1].nests = { 1, 2, 2, 2 };

d1.level[2].catnames = "Fly"$|"Ground";
d1.level[2].atNames = "airhinc";


struct dcout dcout1;
struct ds d0;

dcout1 = dcNestedLogit("hensher",d1,dccontrolCreate);
call dcprt(dcout1);
```

The results are

```
                      NESTED LOGIT RESULTS
                       2003-08-17 16:27:06

Number of Observations:   210
Degrees of Freedom:       202

  1 - Air
  2 - Train
  3 - Bus
  4 - Car


DISTRIBUTION AMONG OUTCOME CATEGORIES for Mode


Dependent Variable       Proportion
Air                       0.2762
Train                     0.3000
```

```
Bus                         0.1429
Car                         0.2810



                         CONSTANTS


Variable Comparison    Coefficient   Std Err     t-stat      Prob
  Air   1/4             6.0423        1.3313      4.5386      0.0000
Train   2/4             5.0646        0.6760      7.4919      0.0000
  Bus   3/4             4.0963        0.6289      6.5138      0.0000



                      ATTRIBUTE COEFFICIENTS


level 1


       Variable     Coefficient  Std Err     t-stat      Prob
       TTME         -0.1126       0.0118     -9.5232      0.0000
       GC           -0.0316       0.0074     -4.2490      0.0000



level 2


     Variable       Coefficient  Std Err     t-stat      Prob
     airhinc         0.0153       0.0111      1.3785      0.1681



1 - correlations


       Variable     Coefficient  Std Err     t-stat      Prob
       Fly           0.5860       0.1131      5.1833      0.0000
```

```
        Ground        0.3890        0.1579       2.4633       0.0138




                 MARGINAL EFFECTS OF ATTRIBUTE VARIABLES
               partial probability with respect to mean attributes


  Air


  Air


        Variable     Coefficient    Std Err       t-stat        Prob
        TTME         -0.0130         0.0031       -4.1503        0.0000
        GC           -0.0036         0.0010       -3.6320        0.0003



  Train


        Variable     Coefficient    Std Err       t-stat        Prob
        TTME          0.0055         0.0016        3.4465        0.0006
        GC            0.0015         0.0006        2.7311        0.0063



  Bus


        Variable     Coefficient    Std Err       t-stat        Prob
        TTME         -0.0017         0.0007       -2.5782        0.0099
        GC           -0.0005         0.0002       -2.1611        0.0307



  Car
```

```
       Variable    Coefficient   Std Err      t-stat       Prob
       TTME         0.0058        0.0015       3.8266       0.0001
       GC           0.0016        0.0008       2.0445       0.0409



Train


Air


       Variable    Coefficient   Std Err      t-stat       Prob
       TTME         0.0036        0.0018       2.0331       0.0420
       GC           0.0010        0.0003       3.2721       0.0011



Train


       Variable    Coefficient   Std Err      t-stat       Prob
       TTME        -0.0216        0.0034      -6.4167       0.0000
       GC          -0.0061        0.0017      -3.4736       0.0005



Bus


       Variable    Coefficient   Std Err      t-stat       Prob
       TTME         0.0022        0.0007       3.2351       0.0012
       GC           0.0006        0.0002       2.4789       0.0132



Car


       Variable    Coefficient   Std Err      t-stat       Prob
```

```
          TTME         0.0139       0.0032       4.3983       0.0000
          GC           0.0039       0.0012       3.1612       0.0016



   Bus


   Air


          Variable    Coefficient  Std Err      t-stat       Prob
          TTME         0.0011       0.0013       0.8568       0.3916
          GC           0.0003       0.0005       0.6127       0.5400



   Train


          Variable    Coefficient  Std Err      t-stat       Prob
          TTME         0.0041       0.0019       2.2059       0.0274
          GC           0.0011       0.0005       2.1280       0.0333



   Bus


          Variable    Coefficient  Std Err      t-stat       Prob
          TTME        -0.0033       0.0009      -3.4924       0.0005
          GC          -0.0009       0.0004      -2.5794       0.0099



   Car


          Variable    Coefficient  Std Err      t-stat       Prob
          TTME         0.0043       0.0019       2.2819       0.0225
          GC           0.0012       0.0009       1.3605       0.1737
```

```
Car


Air


      Variable    Coefficient   Std Err     t-stat      Prob
      TTME         0.0038        4.5025      0.0009      0.9993
      GC           0.0011        0.0006      1.7829      0.0746



Train


      Variable    Coefficient   Std Err     t-stat      Prob
      TTME         0.0139        5.7323      0.0024      0.9981
      GC           0.0039        0.0015      2.5925      0.0095



Bus


      Variable    Coefficient   Std Err     t-stat      Prob
      TTME         0.0023        1.1325      0.0020      0.9984
      GC           0.0006        0.0003      2.1380      0.0325



Car


      Variable    Coefficient   Std Err     t-stat      Prob
      TTME        -0.0220        9.2929     -0.0024      0.9981
      GC          -0.0062        0.0016     -3.7781      0.0002



Fly
```

```
Fly


     Variable      Coefficient   Std Err      t-stat       Prob
     airhinc        0.0030        0.0006       4.8805       0.0000



Ground


     Variable      Coefficient   Std Err      t-stat       Prob
     airhinc       -0.0013        0.0009      -1.3862       0.1657



Ground


Fly


     Variable      Coefficient   Std Err      t-stat       Prob
     airhinc       -0.0030        4.0953      -0.0007       0.9994



Ground


     Variable      Coefficient   Std Err      t-stat       Prob
     airhinc        0.0013        4.8130       0.0003       0.9998



********************SUMMARY STATISTICS********************

MEASURES OF FIT:
```

```
-2 Ln(Lu):                                    387.3123
-2 Ln(Lr): All coeffs equal zero              582.2436
-2 Ln(Lr): J-1 intercepts                     567.5175
LR Chi-Square (coeffs equal zero):            194.9313
     d.f.                                       8.0000
     p-value =                                  0.0000
LR Chi-Square (J-1 intercepts):               180.2052
     d.f.                                       5.0000
     p-value =                                  0.0000
Count R2, Percent Correctly Predicted:          0.7048
Adjusted Percent Correctly Predicted:          -0.4238
Madalla's pseudo R-square:                      0.5760
McFadden's pseudo R-square:                     0.3175
Ben-Akiva and Lerman's Adjusted R-square:       0.3175
Cragg and Uhler's pseudo R-square:              0.0976
Akaike Information Criterion:                   1.9205
Bayesian Information Criterion1:                0.1275
Hannan-Quinn Information Criterion:             1.9721


OBSERVED and PREDICTED OUTCOMES

           |                Predicted
  Observed |     Y01      Y02      Y03      Y04    Total
  -----------------------------------------------------------
       Y01 |      37        3        2       16       58
       Y02 |       2       49        1       11       63
       Y03 |       0        3       23        4       30
       Y04 |       5       14        1       39       59

  -----------------------------------------------------------
     Total |      44       69       27       70      210
```

3.11 Summary Statistics

Several goodness-of-fit measures are printed by **`mnlprt`**. Suppose the dependent variable is y; there are N observations and $K + 1$ explanatory variables (including a constant term); the fitted values are $\widehat{\mu}_i$; $L(r)$ is the restricted likelihood of the model with only an intercept and no other explanatory variables and $L(u)$ is the unrestricted likelihood, the model estimated with an intercept and all explanatory variables.

These include

1. The likelihood ratio statistic is:

$$LR = -2\ln[\frac{L(r)}{L(u)}]$$

   Under the null hypothesis that the $K-1$ explanatory variables have no information about the dependent variable, LR is distributed $\chi^2(K-1)$.

2. McFadden's (1973) pseudo R-square is:

$$R^2_{McF} = 1 - 2\ln[\frac{L(u)}{L(r)}]$$

3. Ben-Akiva and Lerman (1985) revise McFadden's measure to compensate for the effect of additional variables on a regression's explanatory power. Their measure, analogous to adjusted R^2, is

$$\overline{R}^2_{McF} = 1 - \frac{\ln L(u) - K}{\ln L(r)}$$

4. Greene (2000, p. 882) presents an R^2 measure based on standardized residuals.

$$R^2_p = 1 - \frac{\sum_{i=1}^{N}[\frac{y_i-\widehat{\mu}_i}{\sqrt{\widehat{\mu}_i}}]^2}{\sum_{i=1}^{N}[\frac{y_i-\overline{y}}{\sqrt{\overline{y}}}]^2}$$

5. As noted in Greene (2000, p. 883), Cameron and Windmeijer (1993) present an R^2 measure based on the deviances of individual observations, $d_i = 2[y_i\ln(\frac{y_i}{\widehat{\mu}_i}) - (y_i - \widehat{\mu}_i)]$:

$$R^2_d = 1 - \frac{\sum_{i=1}^{N}[y_i\log(\frac{y_i}{\widehat{\mu}_i}) - (y_i - \widehat{\mu}_i)]}{\sum_{i=1}^{N}[y_i\log(\frac{y_i}{\widehat{\mu}_i})]}$$

6. Cragg and Uhler (1970) propose a normed likelihood ratio, based on Maddala's (1983) showing that the maximum of R^2_{ML} is $1 - L(r)^{2/N}$

$$R^2_{C\&U} = \frac{R^2_{ML}}{\max R^2_{ML}} = \frac{1 - [L(r)/L(u)]^{2/N}}{1 - L(r)^{2/N}}$$

7. The count R^2 is the proportion of correct predictions, i.e.

$$R^2_{Count} = \frac{1}{N} \sum_j n_{jj}$$

   where n_{jj} is the number of correct predictions for outcome j.

8. The adjusted count R^2 uses the highest marginal frequency to adjust for the "spurious" successes that result by predicting that an outcome will fall in the category with the greatest percentage of observed successes. It is the proportion of successful categorizations occurring above what would occur by simply choosing the category with the greatest prior chance of success.

$$R^2_{AdjCount} = \frac{\sum_j n_{jj} - \max_r(n_{r+})}{N - \max_r(n_{r+})}$$

   where $\max(n_{r+})$ is the maximum of the contingency table row marginals, the "number of cases in the outcome with the most observations" (Long, 1997, p. 108).

9. The average Akaike information criterion (AIC) is

$$AIC = \frac{-2[\ln L(u) - K]}{N}$$

10. The average Bayesian (Schwarz) information criterion (BIC) is

$$BIC = \frac{-2 \ln L(u) + K \ln(N)}{N}$$

11. The average Hannan-Quinn criterion is

$$HQIC = \frac{-2[\ln L(u) - K \ln(\ln(N))]}{N}$$

Estimation and Optimization 4

A general constrained maximum likelihood estimation problem is:

$$\max_{\theta} L = \sum_{i=1}^{N} \log P(Y_i|x_i;\theta)$$

where N is the number of observations, $P(Y_i|x_i;\theta)$ is the probability of Y_i given x_i, and θ, a vector of parameters subject to linear constraints, nonlinear constraints, and bounds constraints.

The linear constraints are:

$$A\theta = B$$
$$C\theta \geq D$$

The nonlinear constraints are:

$$G(\theta) = 0$$
$$H(\theta) \geq 0$$

The bounds constraints are:

$$\theta_l \leq \theta \leq \theta_u$$

$G(\theta)$ and $H(\theta)$ are functions provided by the user and must be differentiable at least once with respect to θ.

Under **sqpSolvemt**, parameters are updated in a series of iterations beginning with starting values provided by the user. Let θ_t be the current parameter values. Successive values are

$$\theta_{t+1} = \theta_t + \rho\delta$$

where δ is a $K \times 1$ *direction* vector, and ρ a scalar *step length*.

sqpSolvemt finds values for the parameters in θ such that L is maximized (the actual procedure is to minimize $-L$).

Numerous user controllable variables affect the **sqpSolvemt** optimization. These are put into a **dcControl** structure instance. Suppose this instance has the name **dc1**, i.e.

```
struct dcControl cont;
cont = dcControlCreate;
```

The following are the members of the **`dcControl`** structure relevant to the management of the optimization:

cont.A $M \times K$ matrix, linear equality constraint coefficients: `cont.A * p = cont.B` where p is a vector of the parameters.

cont.B $M \times 1$ vector, linear equality constraint constants: `cont.A * p = cont.B` where p is a vector of the parameters.

cont.C $M \times K$ matrix, linear inequality constraint coefficients: `cont.C * p >= cont.D` where p is a vector of the parameters.

cont.D $M \times 1$ vector, linear inequality constraint constants: `cont.C * p >= cont.D` where p is a vector of the parameters.

cont.eqProc scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure.

cont.IneqProc scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure.

cont.Bounds 1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }.

cont.GradProc scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no gradient procedure has been provided.

cont.HessProc scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = ., i.e., no Hessian procedure has been provided.

cont.MaxIters scalar, maximum number of iterations. Default = $1e + 5$.

cont.MaxTries scalar, maximum number of attempts in random search. Default = 100.

cont.DirTol scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed sqpSolvemt exits the iterations.

cont.FeasibleTest scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1;

cont.randRadius scalar, If zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.

cont.trustRadius scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.

cont.output scalar, if nonzero, optimization results are printed. Default = 0.

cont.PrintIters scalar, if nonzero, prints iteration information. Default = 0.

4.0.1 Constraints

The **`c1.A, dc1.B, dc1.C, dc1.D, dc1.EqProc, dc1.IneqProc`**, and **`dc1.Bounds`** matrix structure members control constraints in the **Discrete Choice** procedures. Each row in one of these matrices is associated with a single constraint.

For computational convenience, nonlinear equality constraints and nonlinear inequality constraints are divided into five types: linear equality, linear inequality, nonlinear equality, nonlinear inequality, and bounds constraints.

4.0.2 Linear Equality Constraints

Linear constraints are of the form:

$$A\theta = B$$

where A is an $m_1 \times k$ matrix of known constants, B an $m_1 \times 1$ vector of known constants, and θ the vector of parameters.

To specify linear equality constraints, assign the *A* and *B* matrices to the **dc1.A** and **dc1.B** structure members. To constrain the first of four parameters to equal the third,

```
dc1.A = { 1 0 -1 0 };
dc1.B = { 0 };
```

4.0.3 Linear Inequality Constraints

Linear constraints are of the form:

$$C\theta \geq D$$

where C is an $m_2 \times k$ matrix of known constants, D an $m_2 \times 1$ vector of known constants, and θ the vector of parameters.

To specify linear inequality constraints, assign the *C* and *D* matrices to the structure members **dc1.C** and **dc1.D**. To constrain the first of four parameters to be greater than the third, and the second plus the fourth to be greater than 10:

```
dc1.C = { 1 0 -1 0,
          0 1  0 1 };

dc1.D = {  0,
          10 };
```

4.0.4 Nonlinear Equality

Nonlinear equality constraints are of the form:

$$G(\theta) = 0$$

where θ is the vector of parameters and $G(\theta)$ is an arbitrary, user-supplied function.

To specify nonlinear equality constraints, assign the pointer to the user-supplied constraint function to the **dc1.EqProc** member. To constrain the norm of the parameters to equal 1:

```
proc eqp(b);
    retp(b'b - 1);
endp;
dc1.EqProc = &eqp;
```

4.0.5 Nonlinear Inequality

Nonlinear inequality constraints are of the form:

$$H(\theta) \geq 0$$

where θ is the vector of parameters, and $H(\theta)$ is an arbitrary, user-supplied function.

To specify nonlinear inequality constraints, assign the pointer to the user-supplied constraint function to the structure member **dc1.IneqProc**. To constrain a covariance matrix to be positive definite, the lower left nonredundant portion of which is stored in elements $r : r + s$ of the parameter vector:

```
proc ineqp(b);
    local v;
    v = xpnd(b[r:r+s]); /* r and s defined elsewhere */
    retp(minc(eigh(v)) - 1e-5);
endp;
dc1.IneqProc = &ineqp;
```

This constrains the minimum eigenvalue of the covariance matrix to be greater than a small number (1e-5), guaranteeing that the covariance matrix is positive definite.

4.0.6 Bounds

Bounds are a type of linear inequality constraint. For computational convenience they are specified separately from the other inequality constraints.

To specify bounds constraints, enter the lower and upper bounds respectively in the first and second columns of a matrix that has the same number of rows as the parameter vector. Assign this matrix to the structure member **dc1.Bounds**. Only the first row is necessary if the bounds are the same for all of the parameters. To bound four parameters:

```
dc1.Bounds = { -10 10,
               -10  0,
                 1 10,
                 0  1 };
```

To bound all the parameters between -50 and +50:

```
dc1.Bounds = { -50 50 };
```

4.0.7 Imposing Constraints in Discrete Choice Models

To impose constraints in **Discrete Choice** models, you will need to know the order of parameters in the parameter vector. The simplest way to do this is to first run the model unconstrained and inspect the parameter vector upon output. For example, run your command file adding a call to **`pvGetParNames`**:

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "occatt";
d1.xnames = "exper" $| "educ" $| "white";
d1.catnames = "Menial" $| "BC" $| "Craft" $| "WC" $| "Pro";

struct dcControl c0;
c0 = dcControlCreate;

struct dcout dcout1;
dcout1 = mnl("gssocc",d1,c0);

print (ftostrC(seqa(1,1,pvLength(dcout1.par)),"%1.0lf")
    $~ pvGetParNames(dcout1.par));

    1          b0[1,2]
    2          b0[1,3]
    3          b0[1,4]
    4          b0[1,5]
    5           b[1,2]
```

6	b[1,3]
7	b[1,4]
8	b[1,5]
9	b[2,2]
10	b[2,3]
11	b[2,4]
12	b[2,5]
13	b[3,2]
14	b[3,3]
15	b[3,4]
16	b[3,5]

Now suppose you want to constrain columns two and three of *b* to be equal to each other (the first column is the reference column fixed to zeros), the last two columns to be equal to each other (a type of adjacent categories model), i.e., b[1,3] = b[1,2], b[2,3] = b[2,2], etc., and b[1,5] = b[1,4], b[2,5] = b[2,4], etc., and as well, b[1,4] ¿= b[1,2], b[2,4] ¿= b[2,2], etc.

To accomplish this we set up the following constraint matrices:

```
/*                 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16  */
         c0.A = { 0  0  0  0  1 -1  0  0  0  0  0  0  0  0  0  0,
                  0  0  0  0  0  0  0  0  1 -1  0  0  0  0  0  0,
                  0  0  0  0  0  0  0  0  0  0  0  0  1 -1  0  0,
                  0  0  0  0  0  0  1 -1  0  0  0  0  0  0  0  0,
                  0  0  0  0  0  0  0  0  0  0  1 -1  0  0  0  0,
                  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 -1 };

         c0.B = { 0,
                  0,
                  0,
                  0,
                  0,
                  0 };
```

Now suppose we wish to constrain the second column to be equal to the square of the third column, i.e., *b[1,2] = b[1,3]2̂*, *b[2,2] = b[2,3]2̂*, etc. For nonlinear constraints we must provide a procedure for computing the constraint. Our command file now looks like this:

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "occatt";
d1.xnames = "exper" $| "educ" $| "white";
d1.catnames = "Menial" $| "BC" $| "Craft" $| "WC" $| "Pro";

struct dcControl c0;
c0 = dcControlCreate;


proc eqp(struct PV par, struct DS d);
     local p,r;
     p = pvGetParVector(par);

     r = zeros(3,1);
     r[1] = p[5] - p[6]^2;
     r[2] = p[9] - p[10]^2;
     r[3] = p[13] - p[14]^2;

     retp(r);
endp;

c0.eqProc = &eqp;

struct dcout dcout1;
dcout1 = mnl("gssocc",d1,c0);
```

Equality constraints are not required to be feasible. Inequality constraints however must be feasible. If you are imposing inequality constraints, start values computed by the procedures may not be feasible and the optimization will fail. In that case you will have to supply feasible start values.

4.1 Direction

Define the likelihood function's gradient and Hessian:

$$\begin{aligned}\Psi(\theta) &= \frac{\partial L}{\partial \theta} \\ \Sigma(\theta) &= \frac{\partial^2 L}{\partial\theta\partial\theta'}\end{aligned}$$

and the Jacobians

$$\begin{aligned}\dot{G}(\theta) &= \frac{\partial G(\theta)}{\partial \theta} \\ \dot{H}(\theta) &= \frac{\partial H(\theta)}{\partial \theta}\end{aligned}$$

For the purposes of this exposition and without loss of generality, assume that the linear constraints and bounds have been incorporated into G and H.

In practice, linear constraints are specified separately from the G and H because their Jacobians are known and easy to compute. The bounds are more easily handled separately from the linear inequality constraints.

The direction, δ, solves the quadratic program

$$\textit{minimize}\ \frac{1}{2}\delta'\Sigma(\theta_t)\delta + \Psi(\theta_t)\delta$$

$$\begin{aligned} \textit{subject to}\quad & \dot{G}(\theta_t)\delta + G(\theta_t) = 0 \\ & \dot{H}(\theta_t)\delta + H(\theta_t) \geq 0 \end{aligned}$$

This solution requires that Σ be positive semi-definite.

4.2 Line Search

Define the merit function

$$m(\theta) = L + \max \mid \kappa \mid \sum_j \mid g_j(\theta) \mid - \max \mid \lambda \mid \sum_\ell \min(0, h_\ell(\theta))$$

where g_j is the j-th row of G, h_ℓ is the ℓ-th row of H, κ is the vector of Lagrangean coefficients of the equality constraints, and λ the Lagrangean coefficients of the inequality constraints.

The line search finds a value of ρ that minimizes or decreases $m(\theta_t + \rho\delta)$.

4.2.1 Line Search Methods

Given a direction vector d, the updated estimate of the parameters is computed

$$\theta_{t+1} = \theta_t + \rho\delta$$

where ρ is a constant, usually called the *step length*, that increases the descent of the function given the direction. The value of the function to be minimized as a function of ρ is

$$m(\theta_t + \rho\delta)$$

Given θ and d, this is a function of a single variable ρ. The STEPBT polynomial line fitting/line search method attempts to find a value for ρ that decreases m.

STEPBT is an implementation of a similarly named algorithm described in Dennis and Schnabel (1983). It first attempts to fit a quadratic function to $m(\theta_t + \rho\delta)$, computing a ρ that minimizes the quadratic. If that fails it attempts to fit a cubic function. The cubic function is more costly to compute.

If **`dc1.RandRadius`** is greater than zero, a random search is tried if STEPBT fails. The random search uses the radius specified by **`dc1.RandRadius`**.

4.3 Managing Optimization

The critical elements in optimization are scaling, the starting point, and the condition of the model. When the data are scaled, the starting point is reasonably close to the solution, and the data and model go together well, the iterations converge quickly and without difficulty.

When the optimization is not proceeding well, it is sometimes useful to examine the function, the gradient Ψ , the direction δ, the Hessian Σ, the parameters θ_t, or the step length ρ, during the iterations.

Unless user-supplied functions are provided, **`sqpSolvemt`** calculates the gradient and Hessian numerically, using **`gradmt`** and **`hessmt`**. They have the same input arguments as **`sqpSolvemt`**, a **`PV`** instance containing the parameters and a **`DS`** instance containing the data.

Pointers to explicit gradient and Hessian functions are assigned to **`dc1.gradproc`** and **`dc1.hessproc`** respectively, i.e.

```
dc1.gradproc = &mygradproc;
dc1.hessproc = &myhessproc;
```

4.3.1 Scaling

For best performance, the diagonal elements of the Hessian matrix should be roughly equal. If some diagonal elements contain numbers that are very large and/or very small with respect to the others, **`sqpSolvemt`** has difficulty converging. It is not always obvious how to scale the diagonal elements of the Hessian. One rule-of-thumb is that the data be of roughly the same magnitude.

4.3.2 Condition

The specification of the model may be measured by the condition of the Hessian, the ratio of the Hessian's largest to smallest eigenvalues.

The optimization solution is found by searching for parameter values for which the gradient is zero. It is difficult to determine a parameter's optimal value when the gradient of the function with respect to a parameter is nearly flat. When this occurs, elements of the

Hessian associated with the parameter are very small and the inverse of the Hessian contains very large numbers. The search direction gets buried in the large numbers. In this case it is necessary to respecify the model to exclude the parameter.

Poor condition can be caused by bad scaling. It can also be caused by a poor specification of the model or by bad data. A poorly specified model and bad data are two sides of the same coin.

If the problem is highly nonlinear, it is important that data be available to describe the features of the curve described by each of the parameters. For example, one of the parameters of the Weibull function describes the shape of the curve as it approaches the upper asymptote. This parameter is poorly estimated if data are not available for that portion of the curve.

4.3.3 Starting Point

When the model is not particularly well-defined, the starting point can be critical. Try different starting points when the optimization does not seem to be working. A closed form solution may exist for a simpler problem with the same parameters. For example, ordinary least squares estimates may be used for nonlinear least squares problems or nonlinear regressions like probit or logit. There are no general methods for computing starting values. It may also be necessary to attempt the estimation from a variety of starting points.

References 5

1. Ben-Akiva, M. and Lerman. S.R. 1985. *Discrete Choice Analysis: Theory and Application to Travel Demand*, Cambridge, MA: MIT Press.

2. Cameron, A. Colin and Trivedi, P.K. *Regression Analysis of Count Data*, Cambridge, UK: Cambridge University Press.

3. Cragg, J.G. and Uhler, R., 1970. *The Demand for Automobiles*, **Canadian Journal of Economics**, 3:386-406.

4. Dennis, Jr., J.E., and Schnabel, R.B., 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall.

5. Greene, W.H., 2000. *Econometric Analysis*, 4th ed, Prentice Hall, NJ.

6. Greene, W.H., 1994. “Accounting for Excess Zeros and Sample Selection in Poisson and Negative Binomial Regression Models,” Working Paper No. 94-10, New York: Stern School of Business, New York University, Department of Economics.

7. Hayashi, F. 2000. *Econometrics*, Princeton University Press, NJ.

8. Heilbron, D. 1989. “Generalized Linear Models for Altered Zero Probabilities and Overdispersion in Count Data,” Technical Report, Department of Epidemiology and Biostatistics, University of California, San Francisco.

9. Greene, W. and D. Hensher, 1997, “Multinomial Logit and Discrete Choice Models,” in Greene, W., *LIMDEP, Version 7.0 User’s Manual, Revised*, Plainview NY: Econometric Software, Inc.

10. Johnson, N.L. and Kotz, S., and Balakrishnan, N. 1994., *Continuous Univariate Distributions*, vol 1 (2nd ed.) New York: John Wiley.

11. Lambert, D. 1992. Zero-inflated Poisson Regression with an Application to Manufacturing. *Technometrics*, 34:1-14.

12. Long, J.S. 1997. *Regression Models for Categorical and Limited Dependent Variables*, Sage Publications.

13. Maddala, G. 1983. *Limited Dependent and Qualitative Variables in Econometrics*, New York: Cambridge University Press.

14. McFadden, D. 1974. ‘Conditional Logit Analysis of Qualitative Choice Behavior,’ in P. Zarembka (ed.), *Frontiers of Econometrics*, New York: Academic Press.

15. McKelvey, R.D. and Zavoina, W. 1975. “A Statistical Model for the Analysis of Ordinal Level Dependent Variables,” *Journal of Mathematical Sociology*, 4:103-120.

16. Mullahey, J. 1986. “Specification and Testing of Some Modified Count Models,” *Journal of Econometrics*, 33:341-365.

Discrete Choice Reference 6

dcAdjacentCategories

PURPOSE	Estimates the Adjacent Categories Multinomial Logit model.
LIBRARY	**dc**
FORMAT	{ *out* } = **dcAdjacentCategories**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data	
desc1	an instance of a **dcDesc** structure	
	desc.*yname*	name of dependent variable

`desc.`*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
`desc.`*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
`desc.`*xnames*	$K \times 1$ string vector, names of the independent variable(s)
`desc.`*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories
`desc.`*refcat*	reference category. If **desc**.*refcatName* is specified, **desc**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **desc**.*refcat* has been specified, **desc**.*refcatName* is optional. Default = **desc**.*catnames[1]*.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.
`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the

		specification of **`desc`**.*wgtname* is optional. Default = “”.
	`desc.`*wgtvar*	scalar, index of weight variable. If **`desc`**.*wgtname* is specified, the specification of **`desc`**.*wgtvar* is optional. Default = 0.
cont	an instance of a **`dcControl`** structure	
	`cont.`*startValues*	instance of **`PV`** structure containing starting values; if not provided, **`AdjacentCategories`** computes start values

b0	1	$1 \times L$ matrix, constants in regression
b	2	$K \times L$ matrix, regression coefficients (if any). Coefficients associated with reference category are fixed to zeros.

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = { 0  1  1 };

b = { 0 .1 .1
      0 .1 .1 };

mask = { 0  1  1,
         0  1  1,
         0  1  1 };

cont.startValues =
  pvPackmi(cont.startValues,
  b0,"b0",mask[1,.],1);
cont.startValues =
  pvPackmi(cont.startValues,
  b,"b",mask,2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality

	constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one

	output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par*	instance of PV structure containing estimates
	b0 1 $L \times 1$ matrix, constants in regression
	b 2 $L \times K$ matrix, regression coefficients (if any). Coefficients associated with reference category are fixed to zeros.
	To retrieve, e.g., regression coefficients:
	`b = pvUnpack(out.par,"b");`
	or
	`b = pvUnpack(out.par,2);`
	The coefficients may also be retrieved as a single parameter vector:
	`b = pvGetParVector(out.par);`
	The location of the coefficients in *out.par* can be described by
	`b = pvgetParNames(out.par);`
	if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.

`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables

`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)

13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "occatt";
d1.xnames =  "exper" $| "educ" $| "white";
d1.catnames = "Menial" $| "BC" $| "Craft" $| "WC" $| "Pro";

struct dcout dcout1;

dcout1 = dcAdjacentCategories("gssocc",d1,dccontrolCreate);

call dcprt(dcout1);
```

REMARKS The adjacent category model is a special case of the multinomial logit model where the coefficients of succeeding categories are constrained to be greater than their preceding counterparts.

SOURCE `dcaclogit.src`

dcBinaryLogit

PURPOSE Estimates a logit regression model.

LIBRARY **dc**

FORMAT { *out* } = **dcBinaryLogit**(*data*, *desc*, *cont*)

INPUT *data* string or $N \times K$ matrix, if string, the name of a **GAUSS** data

	set or if matrix, matrix of data	
desc1	an instance of a **dcDesc** structure	
	desc.*yname*	name of dependent variable
	desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
	desc.*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
	desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s).
	desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
	desc.*catnames*	$L \times 1$ string vector, names of categories.
	desc.*refcat*	reference category. If **desc**.*refcatName* is specified, **desc**.*refcat* is optional. Default = 1.
	desc.*refcatName*	string, reference category name. If **desc**.*refcat* has been specified, **desc**.*refcatName* is optional. Default = **desc**.*catnames[1]*.
	desc.*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.

`desc.`*wgtvar*	scalar, index of weight variable. If **`desc`**.*wgtname* is specified, the specification of **`desc`**.*wgtvar* is optional. Default = 0.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`dcBinaryLogit`** computes start values

b0	1 constant in regression
b	2 regression coefficients (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = 1;
b = { .1, .2 };
cont.startValues =
  pvPacki(cont.startValues,
  b0,"b0",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
```

`cont.`*A* $M \times K$ matrix, linear equality constraint coefficients: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.

`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of

	computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.

`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e+5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e-5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par*	instance of PV structure containing estimates
	b0 1 constant in regression

	b 2 regression coefficients (if any)

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
  b = pvGetParVector(out.par);
```

The location of the coefficients in **`out`**.*par* can be described by

```
  b = pvgetParNames(out.par);
```

if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.

`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts

`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf


struct dcDesc d1;
d1 = dcDescCreate;
```

```
d1.yname = "A";
d1.xnames = "GPA" $| "TUCE" $| "PSI";

struct dcout dcout1;

dcout1 = dcBinaryLogit("aldnel",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcbin.src`

dcBinaryProbit

PURPOSE Estimates a probit regression model.

LIBRARY **dc**

FORMAT { *out* } = **dcBinaryProbit**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set, or if matrix, matrix of data	
desc1	an instance of a **dcDesc** structure	
	desc.*yname*	name of dependent variable
	desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
	desc.*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.

DC Reference

`desc.`*xnames*	$K \times 1$ string vector, names of the independent variable(s)
`desc.`*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **`desc`**.*xnames* or **`desc`**.*xvars* may be specified. If data is matrix of data, **`desc`**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories
`desc.`*refcat*	reference category. If **`desc`**.*refcatName* is specified, **`desc`**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **`desc`**.*refcat* has been specified, **`desc`**.*refcatName* is optional. Default = **`desc`**.*catnames[1]*.
`desc.`*wgtname*	string, name of weight variable. If **`desc`**.*wgtvar* is specified, the specification of **`desc`**.*wgtname* is optional. Default =
`desc.`*wgtvar*	scalar, index of weight variable. If **`desc`**.*wgtname* is specified, the specification of **`desc`**.*wgtvar* is optional. Default = 0.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues*	instance of **`PV`** structure containing

starting values; if not provided, **dcBinaryProbit** computes start values

b0	1 constant in regression
b	2 regression coefficients (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = 1;
b = { .1, .2 };
cont.startValues =
  pvPacki(cont.startValues,
  b0,"b0",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* $* p =$ *cont.B* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* $* p =$ *cont.B* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* $* p$ ¿= *cont.D* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* $* p$ ¿= *cont.D* where p is a vector of the

	parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When

	such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality

	boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par* instance of PV structure containing estimates

b0	1 constant in regression
b	2 regression coefficients (if any)

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
  b = pvGetParVector(out.par);
```

The location of the coefficients in **`out`**.*par* can be described by

```
  b = pvgetParNames(out.par);
```

	if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)

6	Probability values for the likelihood ratio statistics
7	McFadden’s Pseudo R-Squared
8	McKelvey and Zovcina’s Pseudo R-Squared
9	Cragg and Uhler’s normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf


struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "A";
d1.xnames = "GPA" $| "TUCE" $| "PSI";

struct dcout dcout1;

dcout1 = dcBinaryProbit("aldnel",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcbin.src`

dcConditionalLogit

PURPOSE	Estimates the Conditional Logit model.
LIBRARY	**dc**
FORMAT	{ *out* } = **dcConditionalLogit**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data
desc1	an instance of a **dcDesc** structure

`desc.`*dataType*	scalar, if 1, the dataset contains a single row for each observation and attribute variables are stored in separate columns in that row. If 0, category data are stored by row within observation and attribute data are stored in single columns.
`desc.`*yname*	name of dependent variable
`desc.`*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
`desc.`*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
`desc.`*xnames*	$K \times 1$ string vector, names of the independent variable(s).
`desc.`*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is

	name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories.
`desc.`*refcat*	reference category. If **desc**.*refcatName* is specified, **desc**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **desc**.*refcat* has been specified, **desc**.*refcatName* is optional. Default = **desc**.*catnames[1]*.
`desc.`*atNames*	$P \times 1$ string vector, names of the attribute variable(s)
`desc.`*atVars*	$P \times 1$ numeric vector, indices of the attribute variable(s)
`desc.`*atCatNames*	$P \times L$ string array, names of the categories of attribute variable(s). Required if **desc**.*datatype* = 1 and **desc**.*atCatVars* not specified.
`desc.`*atCatVars*	$P \times L$ numeric vector, indices of the categories of attribute variable(s). Required if **desc**.*datatype* = 1 and **desc**.*atCatNames* not specified.
`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.

`desc.`*noconstant* scalar, 1 if no constants in model. Default = 0.

`desc.`*marginType* scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`AdjacentCategories`** computes start values

b0	1 $1 \times L$ vector, constants in regression
b	2 $K \times L$ Matrix, regression coefficients of independent variables if any. Coefficients associated with reference category are fixed to zeros.
gm	3 $M \times 1$ vector, coefficients of attribute variables

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = { 0  1  1 };

b = { 0 .1 .1
      0 .1 .1 };

mask = { 0  1  1,
         0  1  1,
         0  1  1 };

gm = { .1, .1 };

cont.startValues =
  pvPackmi(cont.startValues,
  b0,"b0",mask[1,.],1);
cont.startValues =
  pvPackmi(cont.startValues,
  b,"b",mask,2);
cont.startValues =
  pvPackmi(cont.startValues,
  gm,"gm",mask,3);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of

	computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that

	computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each

			iteration. Default = .001.
		`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
		`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.
OUTPUT	*out*	an instance of a **`dcOut`** structure	

`out.`*par* instance of PV structure containing estimates

b0	1	$L \times 1$ matrix, constants in regression
b	2	$L \times K$ matrix, regression coefficients (if any). Coefficients associated with reference category are fixed to zeros.
gm	3	$M \times 1$ vector, coefficients of attribute variables.

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
  b = pvGetParVector(out.par);
```

The location of the coefficients in *out.par* can be described by

```
  b = pvgetParNames(out.par);
```

if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.

`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*atmargineffects*	$L \times L \times 1 \times R$ array, marginal effects by category of attribute variables by category of dependent variable
`out.`*atmarginvc*	$L \times L \times R \times R$ array, covariance matrices of marginal effects by category of attribute variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic

4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
new;
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "Mode";
d1.catvarname = "choiceno";
d1.catnames = "train"$|"bus"$|"car";
d1.atnames = "ttme"$|"invc"$|"invt"$|"GC";

d1.noconstant = 1;

struct dcout dcout1;
dcout1 = dcConditionalLogit("powersxie",d1,dccontrolCreate);
call dcprt(dcout1);
```

dcMultinomialLogit

PURPOSE	Estimates the Multinomial Logit model.
LIBRARY	**dc**
FORMAT	{ *out* } = **`dcMultinomialLogit`**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data
desc1	an instance of a **`dcDesc`** structure

`desc.`*yname*	name of dependent variable
`desc.`*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **`desc`**.*yname* or **`desc`**.*yvar* may be specified. If data is matrix of data, **`desc`**.*yvar* must be specified.
`desc.`*ytype*	scalar, 0 if **`desc`**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
`desc.`*xnames*	$K \times 1$ string vector, names of the independent variable(s)
`desc.`*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **`desc`**.*xnames* or **`desc`**.*xvars* may be specified. If data is matrix of data, **`desc`**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories

`desc.`*refcat*	reference category. If **`desc`**.*refcatName* is specified, **`desc`**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **`desc`**.*refcat* has been specified, **`desc`**.*refcatName* is optional. Default = **`desc`**.*catnames[1]*.
`desc.`*wgtname*	string, name of weight variable. If **`desc`**.*wgtvar* is specified, the specification of **`desc`**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **`desc`**.*wgtname* is specified, the specification of **`desc`**.*wgtvar* is optional. Default = 0.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`AdjacentCategories`** computes start values

b0	1 $1 \times L$ matrix, constants in regression
b	2 $K \times L$ matrix, regression coefficients (if any). Coefficients associated with reference category are fixed to zeros.

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = { 0  1  1 };

b = { 0 .1 .1
      0 .1 .1 };

mask = { 0  1  1,
         0  1  1,
         0  1  1 };

cont.startValues =
  pvPackmi(cont.startValues,
  b0,"b0",mask[1,.],1);
cont.startValues =
  pvPackmi(cont.startValues,
  b,"b",mask,2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C * p ¿= cont.D* where *p* is a vector of the

	parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When

	such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality

			boundaries, then this test can be turned off. Default = 1.
		`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
		`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
		`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
		`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.
OUTPUT	*out*	an instance of a **`dcOut`** structure	
		`out.`*par*	instance of PV structure containing estimates

b0	1 $L \times 1$ matrix, constants in regression
b	2 $L \times K$ matrix, regression coefficients (if any). Coefficients associated with reference category are fixed to zeros.

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

	`b = pvGetParVector(out.par);` The location of the coefficients in *out.par* can be described by `b = pvgetParNames(out.par);` if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic

5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "occatt";
d1.xnames =  "exper" $| "educ" $| "white";
d1.catnames = "Menial" $| "BC" $| "Craft" $| "WC" $| "Pro";

struct dcout dcout1;

dcout1 = dcMultinomialLogit("gssocc",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcmnlogit.src`

dcNegativeBinomial

PURPOSE	Estimates a negative binomial regression model.
LIBRARY	**dc**
FORMAT	{ *out* } = **dcNegativeBinomial**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data
desc1	an instance of a **dcDesc** structure

desc.*yname*	name of dependent variable
desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s)
desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
desc.*znames*	$L \times 1$ string vector, names of the exogenous variable(s), if any, for zero-inflated model
desc.*zvars*	$K \times 1$ vector, indices of the exogenous variable(s), if any, for

	zero-inflated model. If data is name of **GAUSS** dataset, either **desc**.*znames* or **desc**.*zvars* may be specified. If data is matrix of data, **desc**.*zvars* must be specified.
`desc.`*timeName*	string, name of variable for inclusion as a fixed exogenous log-variable. If **desc**.*timeVar* is specified, **desc**.*timeName* is optional.
`desc.`*timeVar*	string, index of variable for inclusion as a fixed exogenous log-variable. If **desc**.*timeName* is specified, **desc**.*timeVar* is optional.
`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.
`desc.`*limited*	scalar, 0 - no censoring or truncation, 1 - truncated model, 2 - censored model
`desc.`*lh*	scalar, value of left side truncation or censoring If the data are truncated on the left, all values must be greater than or equal to **desc**.*lh* (i.e. specify *desc.lh = 1* if there are no zeros in the dependent variable). If the data are censored on the left, all values must be greater than or equal to **desc**.*lh*.

`desc.`*rh*	scalar value of right side truncation or censoring If the data are truncated on the right, all values must be less than or equal to **`desc`**.*rh*. If the data are censored on the left, all values must be less than or equal to **`desc`**.*rh*.
`desc.`*zeroInflated*	scalar, if nonzero a zero-inflated model is estimated. Mixture probability can be a function of exogenous variables as specifed in **`desc`**.*zvars*.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`dcNegativeBinomial`** computes start values

b0	1 constant in regression
b	2 regression coefficients (if any)
alpha	3 dispersion parameter
p0	4 constant in zero-inflated model
p	5 coefficients in zero-inflated model (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;
```

```
b0 = .5;
b = { .1, .1, .1 };
a = .01;
cont.startValues =
  pvPacki(cont.startValues,
  b0,"b0",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
cont.startValues =
  pvPacki(cont.startValues,
  a,"alpha",3);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input

	arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see

	Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If

			scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
		`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
		`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par* instance of PV structure containing estimates

b0	1	constant in regression
b	2	regression coefficients (if any)
alpha	3	dispersion parameter
p0	4	constant in zero-inflated model
p	5	coefficients in zero-inflated model (if any)

To retrieve, e.g., regression coefficients:

```
b = pvUnpack(out.par,"b");
```

or

```
b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
b = pvGetParVector(out.par);
```

The location of the coefficients in *out.par* can be described by

```
b = pvgetParNames(out.par);
```

	if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)

6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "ACC";
d1.xnames = "TB" $| "TC" $| "TD" $| "TE" $| "T6569" $|
            "T7074" $| "T7579" $| "O7579";
d1.timeName = "months";

struct dcOut dcOut1;

dcout1 = dcNegativeBinomial("greenedata",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcbin.src`

dcNestedLogit

PURPOSE	Estimates the Conditional Logit model.
LIBRARY	**dc**
FORMAT	{ *out* } = **dcNestedLogit**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data		
desc1	an instance of a **dcDesc** structure		
		desc.*dataType*	scalar, if 1, the dataset contains a single row for each observation and attribute variables are stored in separate columns in that row. If 0, category data are stored by row within observation and attribute data are stored in single columns.
		desc.*yname*	name of dependent variable
		desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
		desc.*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
		desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s)
		desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is

	name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories
`desc.`*refcat*	reference category. If **desc**.*refcatName* is specified, **desc**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **desc**.*refcat* has been specified, **desc**.*refcatName* is optional. Default = **desc**.*catnames[1]*.
`desc.`*level*	$M \times 1$ vector of instances of a dcLevel structure, one for each level of the model

desc.*level[m].catnames* $L_m \times 1$ string array, names of categories

desc.*level[m].atNames* if **desc**.*datatype = 0*, $P_m \times 1$ string vector, names of the attribute variable(s). If **desc**.*level[m].atVars* is specified, the specification of **desc**.*level[m].atNames* is optional.

desc.*level[m].atVars* if **desc**.*datatype = 0*, $P_m \times 1$ numeric vector, indices of the attribute variable(s). If **desc**.*level[m].atNames* is specified, the specification of **desc**.*level[m].atVars* is optional.

desc.*level[m].nests* $L_m \times 1$ vector, category number in the next higher level of each category at this level. The highest category does not contain one.

desc.*level[m].atCatnames* $R_m \times L_m$
string array, L_m names of categories in **GAUSS** dataset of R_m attribute variables in level m. Required only if **desc**..*dataType = 1*. If **desc**.*level[m].atCatvars* is specified, the specification of **desc**.*level[m].atCatnames* is optional.

desc.*level[m].atCatvars* $R_m \times L_m$
matrix, L_m indices of categories in data matrix or **GAUSS** dataset of R_m attribute variables in level m. Required only if tructmemdesc.*dataType = 1*. If **desc**.*level[m].atCatnames* is specified, the specification of **desc**.*level[m].atCatvars* is optional.

`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability

with respect to independent variables; 0 - partial probability with respect to mean x. Default = 0.

cont an instance of a `dcControl` structure

`cont.`*startValues* instance of `PV` structure containing starting values; if not provided, `AdjacentCategories` computes start values

b0	1	$1 \times L$ vector, constants in regression
b	2	$K \times L$ Matrix, regression coefficients of independent variables if any. Coefficients associated with reference category are fixed to zeros.
g1	3	$R_1 \times 1$ vector, coefficients of attribute variables for first level
g2	4	$R_2 \times 1$ vector, coefficients of attribute variables for second level
	. . .	
gM	2+M	$R_M \times 1$ vector, coefficients of attribute variables for M-th level
t2	3+M	$L_2 \times 1$ vector, proportionality coefficients for second level (first level does not have these coefficients)
t3	4+M	$L_3 \times 1$ vector, proportionality coefficients for third level (first level does not have these coefficients)

. . .

tM	2M+1 $L_M \times 1$ vector, proportionality coefficients for M-th level (first level does not have these coefficients)

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = { 0  1  1 };
/* three categories at first level */

b = { 0 .1 .1
      0 .1 .1 };
/* two independent variables */

mask = { 0  1  1,
         0  1  1,
         0  1  1 };

g1 = { .1, .1 };
/* two attribute variables at first level */
g2 = { .1 };
/* one attribute variable at second level */
t2 = { .1, .1 };
/* two categories at second level */


cont.startValues =
  pvPackmi(cont.startValues,
  b0,"b0",mask[1,.],1);
cont.startValues =
  pvPackmi(cont.startValues,
  b,"b",mask,2);
cont.startValues =
  pvPackmi(cont.startValues,
  g1,"g1",3);
cont.startValues =
  pvPacki(cont.startValues,
  g2,"g2",4);
cont.startValues =
  pvPacki(cont.startValues,
  t2,"t2",5);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A * p = cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.

`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input

	arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **sqpSolvemt** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out`.*par* instance of PV structure containing estimates

b0	1 $1 \times L$ vector, constants in regression
b	2 $K \times L$ Matrix, regression coefficients of independent variables if any. Coefficients associated with reference category are fixed to zeros.
g1	3 $R_1 \times 1$ vector, coefficients of attribute variables for first level
g2	4 $R_2 \times 1$ vector, coefficients of attribute variables for second level
	. . .
gM	2+M $R_M \times 1$ vector, coefficients of attribute variables for M-th level
t2	3+M $L_2 \times 1$ vector, proportionality coefficients for second level (first level does not have these coefficients)
t3	4+M $L_3 \times 1$ vector, proportionality coefficients for third level (first level does not have these coefficients)
	. . .
tM	2M+1 $L_M \times 1$ vector, proportionality coefficients for M-th level (first level does not have these coefficients)

To retrieve, e.g., regression coefficients:

```
b = pvUnpack(out.par,"b");
```

or

```
b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
b = pvGetParVector(out.par);
```

The location of the coefficients in *out.par* can be described by

```
b = pvgetParNames(out.par);
```

if model does not contain a parameter, **pvUnpack** returns a scalar missing value with error code = 99.

`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*atmargineffects*	$M \times 1$ DS structure containing M $L_m \times L_m \times 1 \times R_m$ arrays, marginal effects by category of attribute

	variables by categories at the *m*-th level
`out.`*atmarginvc*	$M \times 1$ DS structure containing M $L_m \times L_m \times R_m \times R_m$ arrays, covariance matrices of marginal effects by category of attribute variables by categories at the *m*-th level
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)

13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf


struct dcDesc d1;
d1 = dcDescCreate;

d1.level = reshape(d1.level,2,1);

d1.yname = "Mode";
d1.catnames = "Air"$|"Train"$|"Bus"$|"Car";
d1.refcatName = "Car";

d1.level[1].atNames = "TTME"$|"GC";
d1.level[1].nests = { 1, 2, 2, 2 };

d1.level[2].catnames = "Fly"$|"Ground";
d1.level[2].atNames = "airhinc";

struct dcout dcout1;

dcout1 = dcNestedLogit("hensher",d1,dccontrolCreate);
call dcprt(dcout1);
```

SOURCE `dcnlogit.src`

dcOrderedLogit

PURPOSE Estimates an ordered logit regression model.

LIBRARY **dc**

FORMAT { *out* } = **dcOrderedLogit**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data
desc1	an instance of a **dcDesc** structure

desc.*yname*	name of dependent variable.
desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
desc.*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s)
desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
desc.*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
desc.*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.
desc.*catnames*	$L \times 1$ string vector, names of categories

`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`dcOrderedLogit`** computes start values

tau	1 thresholds
b	2 regression coefficients (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;

tau = { -5, -2 };
b = { .1, .1, .1 };
cont.startValues =
  pvPacki(cont.startValues,
  tau,"tau",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * *p* ¿=

	cont.D where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters

	have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion

has been satisifed, **`sqpSolvemt`** exits the iterations.

`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par* instance of PV structure containing estimates

tau	1 thresholds
b	2 regression coefficients (if any)

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

	The coefficients may also be retrieved as a single parameter vector: `b = pvGetParVector(out.par);` The location of the coefficients in *out.par* can be described by `b = pvgetParNames(out.par);` if model does not contain a parameter, **pvUnpack** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)

3	Chi-square statistic
4	Agresti’s G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden’s Pseudo R-Squared
8	McKelvey and Zovcina’s Pseudo R-Squared
9	Cragg and Uhler’s normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf


struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "ABC";
d1.xnames = "GPA" $| "TUCE" $| "PSI";

struct dcout dcout1;

dcout1 = dcOrderedLogit("aldnel",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcord.src`

dcOrderedProbit

PURPOSE Estimates an ordered probit regression model.

LIBRARY **`dc`**

FORMAT { *out* } = **`dcOrderedProbit`**(*data*, *desc*, substitutecont)

INPUT

data string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data

desc1 an instance of a **`dcDesc`** structure

`desc.`*yname*	name of dependent variable
`desc.`*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **`desc`**.*yname* or **`desc`**.*yvar* may be specified. If data is matrix of data, **`desc`**.*yvar* must be specified.
`desc.`*ytype*	scalar, 0 if **`desc`**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
`desc.`*xnames*	$K \times 1$ string vector, names of the independent variable(s)
`desc.`*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **`desc`**.*xnames* or **`desc`**.*xvars* may be specified. If data is matrix of data, **`desc`**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories

`desc.`*wgtname*	string, name of weight variable. If **`desc`**.*wgtvar* is specified, the specification of **`desc`**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **`desc`**.*wgtname* is specified, the specification of **`desc`**.*wgtvar* is optional. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`dcOrderedProbit`** computes start values

tau	1 thresholds
b	2 regression coefficients (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;

tau = { -5, -2 };
b = { .1, .1, .1 };
cont.startValues =
  pvPacki(cont.startValues,
  tau,"tau",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* $* p =$ *cont.B* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* $* p =$ *cont.B* where p is

	a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have

	same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested

	for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

“ttfamily “bfseries “upshape LaTeX Error: “beginitList on input line 2731 ended by “endargumentlist.ΩΩSee the LaTeX manual or LaTeX Companion for explanation.ΩType˝H˝¡return¿˝˝for˝immediate˝help

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par* instance of PV structure containing estimates

tau	1 thresholds
b	2 regression coefficients (if any)

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

	`b = pvGetParVector(out.par);` The location of the coefficients in *out.par* can be described by `b = pvgetParNames(out.par);` if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic

5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden’s Pseudo R-Squared
8	McKelvey and Zovcina’s Pseudo R-Squared
9	Cragg and Uhler’s normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf


struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "A";
d1.xnames = "GPA" $| "TUCE" $| "PSI";

struct dcout dcout1;

dcout1 = dcOrderedProbit("aldnel",d1,dccontrolCreate);

call dcprt(dcout1);
```

SOURCE `dcord.src`

dcPoisson

PURPOSE	Estimates a Poisson regression model.
LIBRARY	**dc**
FORMAT	{ *out* } = **dcPoisson**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data
desc1	an instance of a **dcDesc** structure

desc.*yname*	name of dependent variable
desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s)
desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
desc.*znames*	$L \times 1$ string vector, names of the exogenous variable(s), if any, for zero-inflated model
desc.*zvars*	$K \times 1$ vector, indices of the exogenous variable(s), if any, for

	zero-inflated model. If data is name of **GAUSS** dataset, either **desc**.*znames* or **desc**.*zvars* may be specified. If data is matrix of data, **desc**.*zvars* must be specified.
`desc.`*timeName*	string, name of variable for inclusion as a fixed exogenous log-variable. If **desc**.*timeVar* is specified, **desc**.*timeName* is optional.
`desc.`*timeVar*	string, index of variable for inclusion as a fixed exogenous log-variable. If **desc**.*timeName* is specified, **desc**.*timeVar* is optional.
`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.
`desc.`*limited*	scalar, 0 - no censoring or truncation, 1 - truncated model, 2 - censored model
`desc.`*lh*	scalar, value of left side truncation or censoring If the data are truncated on the left, all values must be greater than or equal to **desc**.*lh* (i.e. specify desc.lh = 1 if there are no zeros in the dependent variable). If the data are censored on the left, all values must be greater than or equal to **desc**.*lh*.

`desc.`*rh*	scalar value of right side truncation or censoring If the data are truncated on the right, all values must be less than or equal to **`desc`**.*rh*. If the data are censored on the left, all values must be less than or equal to **`desc`**.*rh*.
`desc.`*zeroInflated*	scalar, if nonzero a zero-inflated model is estimated. Mixture probability can be a function of exogenous variables as specifed in **`desc`**.*zvars*.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **`dcControl`** structure

`cont.`*startValues* instance of **`PV`** structure containing starting values; if not provided, **`dcPoisson`** computes start values

b0	1 constant in regression
b	2 regression coefficients (if any)
p0	3 constant in zero-inflated model
p	4 coefficients in zero-inflated model (if any)

For example:

```
struct dcControl cont;
cont = dcControlCreate;
b0 = { .5 };
b = { .1, .1, .1 };
cont.startValues =
```

DC Reference

```
  pvPacki(cont.startValues,
  b0,"b0",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* * p = *cont.B* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* * p = *cont.B* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * p ¿= *cont.D* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality constraint constants: *cont.C* * p ¿= *cont.D* where p is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality

	procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the

	parameters. When such a procedure has been provided, it has two input arguments, a `PV` parameter structure and a `DS` data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **`sqpSolvemt`** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.

		`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.
OUTPUT	*out*	an instance of a **`dcOut`** structure	
		`out.`*par*	instance of PV structure containing estimates

b0	1 constant in regression
b	2 regression coefficients (if any)
p0	3 constant in zero-inflated model
p	4 coefficients in zero-inflated model (if any)

To retrieve, e.g., regression coefficients:

```
b = pvUnpack(out.par,"b");
```

or

```
b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
b = pvGetParVector(out.par);
```

The location of the coefficients in *out.par* can be described by

```
b = pvgetParNames(out.par);
```

if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.

`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category

`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)
6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared

11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yname = "ACC";
d1.xnames = "TB" $| "TC" $| "TD" $| "TE" $| "T6569" $|
            "T7074" $| "T7579" $| "O7579";
d1.timeName = "months";

struct dcOut dcOut1;

dcout1 = dcPoisson("greenedata",d1,dccontrolCreate);

call psnprt(dcout1);
```

SOURCE `dcpsn.src`

dcprt

PURPOSE Prints output from **Discrete Choice** procedures.

LIBRARY **dc**

FORMAT { *out* } = **dcprt**(*out*)

INPUT *out* an instance of a **dcOut** structure

OUTPUT	*out*	an instance of a **dcOut** structure

REMARKS The input argument is returned unchanged.

SOURCE `dc.src`

dcStereo

PURPOSE Estimates the Stereotype Multinomial Logit model.

LIBRARY **dc**

FORMAT { *out* } = **dcStereo**(*data*, *desc*, substitutecont)

INPUT

data	string or $N \times K$ matrix, if string, the name of a **GAUSS** data set or if matrix, matrix of data		
desc1	an instance of a **dcDesc** structure		
		desc.*yname*	name of dependent variable
		desc.*yvar*	scalar, index of dependent variable. If data is name of **GAUSS** dataset, either **desc**.*yname* or **desc**.*yvar* may be specified. If data is matrix of data, **desc**.*yvar* must be specified.
		desc.*ytype*	scalar, 0 if **desc**.*yvar* character variable, otherwise 1 if numeric. Default = 1.
		desc.*xnames*	$K \times 1$ string vector, names of the independent variable(s)
		desc.*xvars*	$K \times 1$ vector, indices of the independent variable(s). If data is

	name of **GAUSS** dataset, either **desc**.*xnames* or **desc**.*xvars* may be specified. If data is matrix of data, **desc**.*xvars* must be specified.
`desc.`*catnames*	$L \times 1$ string vector, names of categories
`desc.`*refcat*	reference category. If **desc**.*refcatName* is specified, **desc**.*refcat* is optional. Default = 1.
`desc.`*refcatName*	string, reference category name. If **desc**.*refcat* has been specified, **desc**.*refcatName* is optional. Default = **desc**.*catnames[1]*.
`desc.`*wgtname*	string, name of weight variable. If **desc**.*wgtvar* is specified, the specification of **desc**.*wgtname* is optional. Default = “”.
`desc.`*wgtvar*	scalar, index of weight variable. If **desc**.*wgtname* is specified, the specification of **desc**.*wgtvar* is optional. Default = 0.
`desc.`*noconstant*	scalar, 1 if no constants in model. Default = 0.
`desc.`*marginType*	scalar, 1 - average partial probability with respect to independent variables; 0 - partial probability with respect to mean *x*. Default = 0.

cont an instance of a **dcControl** structure

`cont.`*startValues*	instance of **PV** structure containing starting values; if not provided, **stereo** computes start values

b0	1 $1 \times L$ vector, constants in regression

b	2 $K \times 1$ vector, regression coefficients
distance	3 $(L - 1) \times 1$ vector, distance coefficients

For example:

```
struct dcControl cont;
cont = dcControlCreate;

b0 = 1;
b = { .1, .2 };
d = .01;
cont.startValues =
  pvPacki(cont.startValues,
  b0,"b0",1);
cont.startValues =
  pvPacki(cont.startValues,
  b,"b",2);
cont.startValues =
  pvPacki(cont.startValues,
  d,"distance",2);
```

`cont.`*A*	$M \times K$ matrix, linear equality constraint coefficients: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*B*	$M \times 1$ vector, linear equality constraint constants: *cont.A* * *p* = *cont.B* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*C*	$M \times K$ matrix, linear inequality constraint coefficients: *cont.C* * *p* ¿= *cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*D*	$M \times 1$ vector, linear inequality

	constraint constants: *cont.C * p ¿= cont.D* where *p* is a vector of the parameters. For more details see Section 4.0.7.
`cont.`*eqProc*	scalar, pointer to a procedure that computes the nonlinear equality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed equality constraints. For more details see Remarks below. Default = {.}, i.e., no equality procedure. For more details see Section 4.0.7.
`cont.`*IneqProc*	scalar, pointer to a procedure that computes the nonlinear inequality constraints. When such a procedure has been provided, it has two input arguments, a **`PV`** parameter structure and a **`DS`** data structure, and one output argument, a vector of computed inequality constraints. For more details see Remarks below. Default = {.}, i.e., no inequality procedure. For more details see Section 4.0.7.
`cont.`*Bounds*	1×2 or $K \times 2$ matrix, bounds on parameters. If 1×2 all parameters have same bounds. Default = { -1e256 1e256 }. For more details see Section 4.0.7.
`cont.`*GradProc*	scalar, pointer to a procedure that

	computes the gradient of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = {.}, i.e., no gradient procedure has been provided.
`cont.`*HessProc*	scalar, pointer to a procedure that computes the Hessian, i.e., the matrix of second order partial derivatives of the function with respect to the parameters. When such a procedure has been provided, it has two input arguments, a **PV** parameter structure and a **DS** data structure, and one output argument, a vector of computed inequality constraints. For more details see Section 4.0.7. Default = ., i.e., no Hessian procedure has been provided.
`cont.`*MaxIters*	scalar, maximum number of iterations. Default = $1e + 5$.
`cont.`*MaxTries*	scalar, maximum number of attempts in random search. Default = 100.
`cont.`*DirTol*	scalar, convergence tolerance for gradient of estimated coefficients. Default = $1e - 5$. When this criterion has been satisifed, **sqpSolvemt** exits the iterations.
`cont.`*FeasibleTest*	scalar, if nonzero, parameters are tested for feasibility before

	computing function in line search. If function is defined outside inequality boundaries, then this test can be turned off. Default = 1.
`cont.`*randRadius*	scalar, if zero, no random search is attempted. If nonzero, it is the radius of the random search. Default = .001.
`cont.`*trustRadius*	scalar, radius of the trust region. If scalar missing, trust region not applied. The trust sets a maximum amount of the direction at each iteration. Default = .001.
`cont.`*output*	scalar, if nonzero, optimization results are printed. Default = 0.
`cont.`*PrintIters*	scalar, if nonzero, prints iteration information. Default = 0.

OUTPUT *out* an instance of a **`dcOut`** structure

`out.`*par* instance of PV structure containing estimates

b0	1	constant in regression
b	2	regression coefficients
distance	3	distance coefficients

To retrieve, e.g., regression coefficients:

```
  b = pvUnpack(out.par,"b");
```

or

```
  b = pvUnpack(out.par,2);
```

The coefficients may also be retrieved as a single parameter vector:

```
  b = pvGetParVector(out.par);
```

The location of the coefficients in *out.par* can be described by

	`b = pvgetParNames(out.par);` if model does not contain a parameter, **`pvUnpack`** returns a scalar missing value with error code = 99.
`out.`*vc*	$NPARM \times NPARM$ variance-covariance matrix of coefficient estimates
`out.`*ydist*	$L \times 1$ vector, percentages of dependent variable by category
`out.`*xdata*	$K \times 4$ matrix, the means, standard deviations, minimums, and maximums of independent variables
`out.`*margineffects*	$L \times 1 \times K$ array, marginal effects of independent variables by category of dependent variable
`out.`*marginvc*	$L \times K \times K$ array, covariance matrices of marginal effects of independent variables by category of dependent variable
`out.`*fittedvals*	$N \times 1$ matrix of predicted (fitted) counts
`out.`*resids*	$N \times 1$ matrix of residuals
`out.`*gf*	12×1 matrix of goodness-of-fit measures

1	Log-Likelihood, full model
2	Log-Likelihood, restricted model (all slope coefficients equal zero)
3	Chi-square statistic
4	Agresti's G-Squared statistic
5	Likelihood Ratio statistics (from the full and restricted models above)

6	Probability values for the likelihood ratio statistics
7	McFadden's Pseudo R-Squared
8	McKelvey and Zovcina's Pseudo R-Squared
9	Cragg and Uhler's normed likelihood ratios
10	Count R-Squared
11	Adjusted Count R-Squared
12	Akaike information criterion (AIC)
13	Bayesian information criterion (BIC)

EXAMPLE

```
library dc;
#include dc.sdf

struct dcDesc d1;
d1 = dcDescCreate;

d1.yvar = 1;
let d1.xvars = { 2,3,4 };

struct dcout dcout1;

dcout1 = dcStereo("aldnel",d1,dcControlCreate);
call dcprt(dcout1);
```

REMARKS The stereotype model is a special case of the multinomial logit model where the coefficients of succeeding categories are a linear function of a single vector of coefficients.

SOURCE `dcstereo.src`

Index