# Package ‘DMCfun’

October 25, 2021

**Type** Package

**Title** Diffusion Model of Conflict (DMC) in Reaction Time Tasks

**Version** 2.0.2

**Date** 2021-10-12

**Description** DMC model simulation detailed in Ulrich, R., Schroeter, H., Leuthold, H., & Birngruber, T. (2015).
Automatic and controlled stimulus processing in conflict tasks: Superimposed diffusion processes and delta functions.
Cognitive Psychology, 78, 148-174. Ulrich et al. (2015) <doi:10.1016/j.cogpsych.2015.02.005>.
Decision processes within choice reaction-
time (CRT) tasks are often modelled using evidence accumulation models (EAMs),
a variation of which is the Diffusion Decision Model (DDM, for a review, see Ratcliff & McKoon, 2008).
Ulrich et al. (2015) introduced a Diffusion Model for Con-
flict tasks (DMC). The DMC model combines common
features from within standard diffusion models with the addition of superimposed controlled and automatic activation.
The DMC model is used to explain distributional reaction time (and error rate) patterns in common behavioural
conflict-like tasks (e.g., Flanker task, Simon task). This R-
package implements the DMC model and provides functionality
to fit the model to observed data. Further details are provided in the following paper:
Mackenzie, I. G., & Dudschig, C. (2021). DMCfun: An R package for fitting Diffusion Model of Conflict (DMC) to reaction
time and error rate data. Methods in Psychology, 100074. <doi:10.1016/j.metip.2021.100074>.

**URL** https://github.com/igmmgi/DMCfun,
https://CRAN.R-project.org/package=DMCfun,
https://www.sciencedirect.com/science/article/pii/S259026012100031X

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 3.5.0)

**Imports** DEoptim, Rcpp (>= 0.12.16), dplyr (>= 1.0.0), parallel,
pbapply, shiny, tidyr

**Suggests** testthat

**LinkingTo** Rcpp, BH

**RoxygenNote** 7.1.2

**LazyData** true

**SystemRequirements** C++11

**NeedsCompilation** yes

**Author** Ian G. Mackenzie [cre, aut],
Carolin Dudschig [aut]

**Maintainer** Ian G. Mackenzie <ian.mackenzie@uni-tuebingen.de>

**Repository** CRAN

**Date/Publication** 2021-10-25 07:30:14 UTC

## R topics documented:

## addDataDF *addDataDF*

### Description

Add simulated ex-gaussian reaction-time (RT) data and binary error (Error = 1, Correct = 0) data to an R DataFrame. This function can be used to create simulated data sets.

### Usage

```
addDataDF(dat, RT = NULL, Error = NULL)
```

### Arguments

| | |
|---|---|
| dat | DataFrame (see createDF) |
| RT | RT parameters (see rtDist) |
| Error | Error parameters (see errDist) |

### Value

DataFrame with RT (ms) and Error (bool) columns

### Examples

```
# Example 1: default dataframe
dat <- createDF()
dat <- addDataDF(dat)
head(dat)
hist(dat$RT, 100)
table(dat$Error)

# Example 2: defined overall RT parameters
dat <- createDF(nSubjects = 50, nTrl = 50, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat, RT = c(500, 150, 100))
boxplot(dat$RT ~ dat$Comp)
table(dat$Comp, dat$Error)

# Example 3: defined RT + Error parameters across conditions
dat <- createDF(nSubjects = 50, nTrl = 50, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"   = c(500, 80, 100),
                           "Comp_incomp" = c(600, 80, 140)),
                 Error = list("Comp_comp"   = 5,
                              "Comp_incomp" = 15))
boxplot(dat$RT ~ dat$Comp)
table(dat$Comp, dat$Error)
```

```
# Example 4:
# create dataframe with defined RT + Error parameters across different conditions
dat <- createDF(nSubjects = 50, nTrl = 50, design = list("Comp" = c("comp", "incomp", "neutral")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"      = c(500, 150, 100),
                           "Comp_neutral"   = c(550, 150, 100),
                           "Comp_incomp"    = c(600, 150, 100)),
                 Error = list("Comp_comp"    =  5,
                              "Comp_neutral" = 10,
                              "Comp_incomp"  = 15))
boxplot(dat$RT ~ dat$Comp)
table(dat$Comp, dat$Error)

# Example 5:
# create dataframe with defined RT + Error parameters across different conditions
dat <- createDF(nSubjects = 50, nTrl = 50,
                design = list("Hand" = c("left", "right"),
                              "Side" = c("left", "right")))
dat <- addDataDF(dat,
                 RT = list("Hand:Side_left:left"   = c(400, 150, 100),
                           "Hand:Side_left:right"  = c(500, 150, 100),
                           "Hand:Side_right:left"  = c(500, 150, 100),
                           "Hand:Side_right:right" = c(400, 150, 100)),
                 Error = list("Hand:Side_left:left"   = c(5,4,2,2,1),
                              "Hand:Side_left:right"  = c(15,4,2,2,1),
                              "Hand:Side_right:left"  = c(15,7,4,2,1),
                              "Hand:Side_right:right" = c(5,8,5,3,1)))

boxplot(dat$RT ~ dat$Hand + dat$Side)
table(dat$Error, dat$Hand, dat$Side)
```

## `addErrorBars` *addErrorBars: Add errorbars to plot.*

### Description

Add error bars to current plot (uses base arrows function).

### Usage

```
addErrorBars(xpos, ypos, errorSize, arrowSize = 0.1)
```

### Arguments

| | |
|---|---|
| `xpos` | x-position of data-points |
| `ypos` | y-position of data-points |
| `errorSize` | +- size of error bars |
| `arrowSize` | Width of the errorbar arrow |

### Value

Plot (no return value)

### Examples

```
# Example 1
plot(c(1, 2), c(450, 500), xlim = c(0.5, 2.5), ylim = c(400, 600), type = "o")
addErrorBars(c(1, 2), c(450, 500), errorSize = c(20, 20))

# Example 2
plot(c(1, 2), c(450, 500), xlim = c(0.5, 2.5), ylim = c(400, 600), type = "o")
addErrorBars(c(1, 2), c(450, 500), errorSize = c(20, 40), arrowSize = 0.1)
```

## `calculateBinProbabilities` *calculateBinProbabilities*

### Description

Calculate bin probabilities in observed data

### Usage

```
calculateBinProbabilities(resOb)
```

### Arguments

| | |
|---|---|
| `resOb` | Observed data (see dmcObservedData) |

### Value

resOb Observed data with additional $probSubject/$prob table

### Examples

```
# Example 1:
resOb <- flankerData
resOb <- calculateBinProbabilities(resOb)
resOb$prob
```

## calculateCAF *calculateCAF*

### Description

Calculate conditional accuracy function (CAF). The DataFrame should contain columns defining the participant, compatibility condition, RT and error (Default column names: "Subject", "Comp", "RT", "Error"). The "Comp" column should define compatibility condition (Default: c("comp", "incomp")) and the "Error" column should define if the trial was an error or not (Default: c(0, 1) ).

### Usage

```
calculateCAF(
  dat,
  nCAF = 5,
  columns = c("Subject", "Comp", "RT", "Error"),
  compCoding = c("comp", "incomp"),
  errorCoding = c(0, 1)
)
```

### Arguments

| | |
|---|---|
| `dat` | DataFrame with columns containing the participant number, condition compatibility, RT data (in ms) and an Error column. |
| `nCAF` | Number of CAF bins. |
| `columns` | Name of required columns Default: c("Subject", "Comp", "RT", "Error") |
| `compCoding` | Coding for compatibility Default: c("comp", "incomp") |
| `errorCoding` | Coding for errors Default: c(0, 1)) |

### Value

calculateCAF returns a DataFrame with conditional accuracy function (CAF) data (Bin, comp, incomp, effect)

### Examples

```
# Example 1
dat <- createDF(nSubjects = 1, nTrl = 10000, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"   = c(500, 80, 100),
                           "Comp_incomp" = c(600, 80, 140)),
                 Error = list("Comp_comp"   = c( 5, 4, 3, 2, 1),
                              "Comp_incomp" = c(20, 8, 6, 4, 2)))
caf <- calculateCAF(dat)

# Example 2
dat <- createDF(nSubjects = 1, nTrl = 10000, design = list("Congruency" = c("cong", "incong")))
dat <- addDataDF(dat,
                 RT = list("Congruency_cong"   = c(500, 80, 100),
                           "Congruency_incong" = c(600, 80, 140)),
                 Error = list("Congruency_cong"   = c( 5, 4, 3, 2, 1),
                              "Congruency_incong" = c(20, 8, 6, 4, 2)))
head(dat)
caf <- calculateCAF(dat, columns = c("Subject", "Congruency", "RT", "Error"),
                    compCoding = c("cong", "incong"))
```

---

## `calculateCostValueCS` *calculateCostValueCS*

---

### Description

Calculate cost value (fit) using chi-square (CS) from correct and incorrect RT data.

### Usage

```
calculateCostValueCS(resTh, resOb)
```

### Arguments

| | |
|---|---|
| `resTh` | list containing simulation $sim values (output from dmcSim) for rts_comp, rts_incomp, errs_comp, errs_incomp |
| `resOb` | list containing raw observed data (see dmcObservedData with keepRaw = TRUE) |

### Value

cost value (CS)

### Examples

```
# Example 1:
resTh <- dmcSim()
resOb <- flankerData
resOb <- calculateBinProbabilities(resOb)
cost <- calculateCostValueCS(resTh, resOb)
```

## `calculateCostValueGS` *calculateCostValueGS*

### Description

Calculate cost value (fit) using likelihood-ratio chi-square statistic (GS) from correct and incorrect RT data.

### Usage

```
calculateCostValueGS(resTh, resOb)
```

### Arguments

| | |
|---|---|
| `resTh` | list containing simulation $sim values (output from dmcSim) for rts_comp, rts_incomp, errs_comp, errs_incomp |
| `resOb` | list containing raw observed data (see dmcObservedData with keepRaw = TRUE) |

### Value

cost value (GS)

### Examples

```
# Example 1:
resTh <- dmcSim()
resOb <- flankerData
resOb <- calculateBinProbabilities(resOb)
cost  <- calculateCostValueGS(resTh, resOb)
```

## `calculateCostValueRMSE` *calculateCostValueRMSE*

### Description

Calculate cost value (fit) using root-mean-square error (RMSE) from a combination of RT and error rate.

### Usage

```
calculateCostValueRMSE(resTh, resOb)
```

### Arguments

| | |
|---|---|
| `resTh` | list containing caf values for comp/incomp conditions (nbins * 4 columns) and delta values for comp/incomp conditions (nbins * 5 columns). See output from dmcSim (.$caf). |
| `resOb` | list containing caf values for comp/incomp conditions (n * 4 columns) and delta values for comp/incomp conditions (nbins * 5 columns). See output from dmc-Sim (.$delta). |

### Value

cost value (RMSE)

### Examples

```
# Example 1:
resTh <- dmcSim()
resOb <- dmcSim()
cost <- calculateCostValueRMSE(resTh, resOb)

# Example 2:
resTh <- dmcSim()
resOb <- dmcSim(tau = 150)
cost <- calculateCostValueRMSE(resTh, resOb)
```

---

## `calculateCostValueSPE` *calculateCostValueSPE*

---

### Description

Calculate cost value (fit) using squared percentage errror (SPE) from combination of RT and error rate.

### Usage

```
calculateCostValueSPE(resTh, resOb)
```

### Arguments

| | |
|---|---|
| `resTh` | list containing caf values for comp/incomp conditions (nbins * 4 columns) and delta values for comp/incomp conditions (nbins * 5 columns). See output from dmcSim (.$caf). |
| `resOb` | list containing caf values for comp/incomp conditions (n * 4 columns) and delta values for comp/incomp conditions (nbins * 5 columns). See output from dmc-Sim (.$delta). |

### Value

cost value (SPE)

## Examples

```
# Example 1:
resTh <- dmcSim()
resOb <- dmcSim()
cost <- calculateCostValueSPE(resTh, resOb)

# Example 2:
resTh <- dmcSim()
resOb <- dmcSim(tau = 150)
cost <- calculateCostValueSPE(resTh, resOb)
```

---

## `calculateDelta` *calculateDelta*

---

### Description

Calculate delta plot. Here RTs are split into n bins (Default: 5) for compatible and incompatible trials separately. Mean RT is calculated for each condition in each bin then subtracted (incompatible - compatible) to give a compatibility effect (delta) at each bin.

### Usage

```
calculateDelta(
  dat,
  nDelta = 19,
  tDelta = 1,
  columns = c("Subject", "Comp", "RT"),
  compCoding = c("comp", "incomp"),
  quantileType = 5
)
```

### Arguments

| | |
|---|---|
| `dat` | DataFrame with columns containing the participant number, condition compatibility, and RT data (in ms). |
| `nDelta` | The number of delta bins. |
| `tDelta` | type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `columns` | Name of required columns Default: c("Subject", "Comp", "RT") |
| `compCoding` | Coding for compatibility Default: c("comp", "incomp") |
| `quantileType` | Argument (1-9) from R function quantile specifying the algorithm (?quantile) |

### Value

calculateDelta returns a DataFrame with distributional delta analysis data (Bin, comp, incomp, meanBin, Effect)

## Examples

```
# Example 1
dat <- createDF(nSubjects = 1, nTrl = 10000, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"   = c(500, 80, 100),
                           "Comp_incomp" = c(600, 80, 140)))
delta <- calculateDelta(dat)

# Example 2
dat <- createDF(nSubject = 1, nTrl = 10000, design = list("Congruency" = c("cong", "incong")))
dat <- addDataDF(dat,
                 RT = list("Congruency_cong"   = c(500, 80, 100),
                           "Congruency_incong" = c(600, 80, 140)))
head(dat)
delta <- calculateDelta(dat, nDelta = 9, columns = c("Subject", "Congruency", "RT"),
                        compCoding = c("cong", "incong"))
```

---

`createDF` *createDF*

---

## Description

Create dataframe (see also addDataDF)

## Usage

```
createDF(
  nSubjects = 20,
  nTrl = 50,
  design = list(A = c("A1", "A2"), B = c("B1", "B2"))
)
```

## Arguments

| | |
|---|---|
| `nSubjects` | Number of subjects |
| `nTrl` | Number of trials per factor/level for each participant |
| `design` | Factors and levels |

## Value

DataFrame with Subject, Factor(s) columns

## Examples

```
# Example 1
dat <- createDF()

# Example 2
dat <- createDF(nSubjects = 50, nTrl = 50, design = list("Comp" = c("comp", "incomp")))

# Example 3
dat <- createDF(nSubjects = 50, nTrl = 50, design = list("Comp" = c("comp", "incomp"),
                                                  "Side" = c("left", "right", "middle")))
```

---

`dmcCombineObservedData` *dmcCombineObservedData*

---

## Description

Combine observed datasets

## Usage

```
dmcCombineObservedData(...)
```

## Arguments

| | |
|---|---|
| `...` | Any number of outputs from dmcObservedData |

## Value

dmcCombineObservedData returns a list of objects of class "dmcob"

## Examples

```
# Example 1
dat <- dmcCombineObservedData(flankerData, simonData)  # combine flanker/simon data
plot(dat, figType = "delta", xlimDelta = c(200, 700), ylimDelta = c(-20, 80),
     cols = c("black", "darkgrey"), legend = FALSE, resetPar = FALSE)
legend(200, 80, legend = c("Flanker Task", "Simon Task"),
       col = c("black", "darkgrey"), lty = c(1, 1))
```

---

`dmcCppR` *dmcCppR*

---

## Description

dmcCppR

## dmcFit *dmcFit*

### Description

Fit theoretical data generated from dmcSim to observed data by minimizing the root-mean-square error ("RMSE") between a weighted combination of the CAF and CDF functions using optim (Nelder-Mead). Alternative cost functions include squared percentage error ("SPE"), and g-squared statistic ("GS").

### Usage

```
dmcFit(
  resOb,
  nTrl = 1e+05,
  startVals = list(),
  minVals = list(),
  maxVals = list(),
  fixedFit = list(),
  fitInitialGrid = TRUE,
  fitInitialGridN = 10,
  fixedGrid = list(),
  nCAF = 5,
  nDelta = 19,
  pDelta = vector(),
  tDelta = 1,
  spDist = 1,
  drDist = 0,
  drShape = 3,
  drLim = c(0.1, 0.7),
  rtMax = 5000,
  costFunction = "RMSE",
  printInputArgs = TRUE,
  printResults = FALSE,
  optimControl = list(),
  numCores = 2
)
```

### Arguments

| | |
|---|---|
| `resOb` | Observed data (see flankerData and simonTask for data format) and the function dmcObservedData to create the required input from either an R data frame or external *.txt/*.csv files |
| `nTrl` | Number of trials to use within dmcSim. |
| `startVals` | Starting values for to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, |

| | |
|---|---|
| | sigm (e.g., startVals = list(amp = 20, tau = 200, drc = 0.5, bnds = 75, resMean = 300, resSD = 30, aaShape = 2, spShape = 3, spBias = 0, sigm = 4)). |
| `minVals` | Minimum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., minVals = list(amp = 10, tau = 5, drc = 0.1, bnds = 20, resMean = 200, resSD = 5, aaShape = 1, spShape = 2, spBias = -20, sigm = 1)). |
| `maxVals` | Maximum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., maxVals = list(amp = 40, tau = 300, drc = 1.0, bnds = 150, resMean = 800, resSD = 100, aaShape = 3, spShape = 4, spBias = 20, sigm = 10)) |
| `fixedFit` | Fix parameter to starting value. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., fixedFit = list(amp = F, tau = F, drc = F, bnds = F, resMean = F, resSD = F, aaShape = F, spShape = F, spBias = T, sigm = T)) |
| `fitInitialGrid` | TRUE/FALSE |
| `fitInitialGridN` | 10 linear steps between parameters min/max values (reduce if searching more than ~2/3 initial parameters) |
| `fixedGrid` | Fix parameter for initial grid search. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., fixedGrid = list(amp = T, tau = F, drc = T, bnds = T, resMean = T, resSD = T, aaShape = T, spShape = T, spBias = T, sigm = T)). As a default, the initial gridsearch only searches the tau space. |
| `nCAF` | The number of CAF bins. |
| `nDelta` | The number of delta bins. |
| `pDelta` | An alternative option to nDelta by directly specifying required percentile values (vector of values 0-100) |
| `tDelta` | The type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `spDist` | The starting point (sp) distribution (0 = constant, 1 = beta, 2 = uniform) |
| `drDist` | The drift rate (dr) distribution type (0 = constant, 1 = beta, 2 = uniform) |
| `drShape` | The drift rate (dr) shape parameter |
| `drLim` | The drift rate (dr) range |
| `rtMax` | The limit on simulated RT (decision + non-decisional components) |
| `costFunction` | The cost function to minimise: root mean square error ("RMSE": default), squared percentage error ("SPE"), or likelihood-ratio chi-square statistic ("GS") |
| `printInputArgs` | TRUE (default) /FALSE |
| `printResults` | TRUE/FALSE (default) |
| `optimControl` | Additional control parameters passed to optim (see optim details section) |
| `numCores` | Number of cores to use |

## Value

dmcfit returns an object of class "dmcfit" with the following components:

| | |
|---|---|
| `sim` | Individual trial data points (RTs for all trial types e.g., correct/error trials) and activation vectors from the simulation |
| `summary` | Condition means for reaction time and error rate |
| `caf` | Conditional Accuracy Function (CAF) data per bin |
| `delta` | DataFrame with distributional delta analysis data correct trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `delta_errs` | DataFrame with distributional delta analysis data incorrect trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `par` | The fitted model parameters + final cost value of the fit |

## Examples

```
# Code below can exceed CRAN check time limit, hence donttest
# Example 1: Flanker data from Ulrich et al. (2015)
fit <- dmcFit(flankerData) # only initial search tau
plot(fit, flankerData)
summary(fit)

# Example 2: Simon data from Ulrich et al. (2015)
fit <- dmcFit(simonData) # only initial search tau
plot(fit, simonData)
summary(fit)

# Example 3: Flanker data from Ulrich et al. (2015) with non-default
# start vals and some fixed values
fit <- dmcFit(flankerData,
  startVals = list(drc = 0.6, aaShape = 2.5),
  fixedFit = list(drc = TRUE, aaShape = TRUE)
)

# Example 4: Simulated Data (+ve going delta function)
dat <- createDF(nSubjects = 20, nTrl = 500, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
  RT = list(
    "Comp_comp" = c(510, 100, 100),
    "Comp_incomp" = c(540, 130, 85)
  ),
  Error = list(
    "Comp_comp" = c(4, 3, 2, 1, 1),
    "Comp_incomp" = c(20, 4, 3, 1, 1)
  )
)
datOb <- dmcObservedData(dat, columns = c("Subject", "Comp", "RT", "Error"))
plot(datOb)
fit <- dmcFit(datOb, nTrl = 5000)
plot(fit, datOb)
```

```
summary(fit)
```

---

dmcFitDE *dmcFitDE*

---

## Description

Fit theoretical data generated from dmcSim to observed data by minimizing the root-mean-square error (RMSE) between a weighted combination of the CAF and CDF functions using the R-package DEoptim. Alternative cost functions include squared percentage error ("SPE"), and g-squared statistic ("GS").

## Usage

```
dmcFitDE(
  resOb,
  nTrl = 1e+05,
  minVals = list(),
  maxVals = list(),
  fixedFit = list(),
  nCAF = 5,
  nDelta = 19,
  pDelta = vector(),
  tDelta = 1,
  costFunction = "RMSE",
  spDist = 1,
  drDist = 0,
  drShape = 3,
  drLim = c(0.1, 0.7),
  rtMax = 5000,
  deControl = list(),
  numCores = 2
)
```

## Arguments

| | |
|---|---|
| resOb | Observed data (see flankerData and simonTask for data format) |
| nTrl | The number of trials to use within dmcSim. |
| minVals | Minimum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., minVals = list(amp = 10, tau = 5, drc = 0.1, bnds = 20, resMean = 200, resSD = 5, aaShape = 1, spShape = 2, spBias = -20, sigm = 1)). |

| | |
|---|---|
| `maxVals` | Maximum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., maxVals = list(amp = 40, tau = 300, drc = 1.0, bnds = 150, resMean = 800, resSD = 100, aaShape = 3, spShape = 4, spBias = 20, sigm = 10)) |
| `fixedFit` | Fix parameter to starting value. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., fixedFit = list(amp = F, tau = F, drc = F, bnds = F, resMean = F, resSD = F, aaShape = F, spShape = F, spBias = T, sigm = T)) |
| `nCAF` | The number of CAF bins. |
| `nDelta` | The number of delta bins. |
| `pDelta` | An alternative option to nDelta by directly specifying required percentile values (vector of values 0-100) |
| `tDelta` | The type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `costFunction` | The cost function to minimise: root mean square error ("RMSE": default), squared percentage error ("SPE"), or likelihood-ratio chi-square statistic ("GS") |
| `spDist` | The starting point distribution (0 = constant, 1 = beta, 2 = uniform) |
| `drDist` | The drift rate (dr) distribution type (0 = constant, 1 = beta, 2 = uniform) |
| `drShape` | The drift rate (dr) shape parameter |
| `drLim` | The drift rate (dr) range |
| `rtMax` | The limit on simulated RT (decision + non-decisional components) |
| `deControl` | Additional control parameters passed to DEoptim (see DEoptim.control) |
| `numCores` | Number of cores to use |

## Value

dmcfit returns an object of class "dmcfit" with the following components:

| | |
|---|---|
| `sim` | Individual trial data points (RTs for all trial types e.g., correct/error trials) and activation vectors from the simulation |
| `summary` | Condition means for reaction time and error rate |
| `caf` | Conditional Accuracy Function (CAF) data per bin |
| `delta` | DataFrame with distributional delta analysis data correct trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `delta_errs` | DataFrame with distributional delta analysis data incorrect trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `par` | The fitted model parameters + final cost value of the fit |

## Examples

```
# The code below can exceed CRAN check time limit, hence donttest
# NB. The following code when using numCores = 2 (default) takes approx 20 minutes on
# a standard desktop, whilst when increasing the number of cores used, (numCores = 12),
# the code takes approx 5 minutes.

# Example 1: Flanker data from Ulrich et al. (2015)
fit <- dmcFitDE(flankerData);
plot(fit, flankerData)
summary(fit)

# Example 2: Simon data from Ulrich et al. (2015)
fit <- dmcFitDE(simonData, nTrl = 20000)
plot(fit, simonData)
summary(fit)
```

---

## dmcFitSubject *dmcFitSubject*

---

### Description

Fit theoretical data generated from dmcSim to observed data by minimizing the root-mean-square error ("RMSE") between a weighted combination of the CAF and CDF functions using optim (Nelder-Mead). Alternative cost functions include squared percentage error ("SPE"), and g-squared statistic ("GS").

### Usage

```
dmcFitSubject(
  resOb,
  nTrl = 1e+05,
  startVals = list(),
  minVals = list(),
  maxVals = list(),
  fixedFit = list(),
  fitInitialGrid = TRUE,
  fitInitialGridN = 10,
  fixedGrid = list(),
  nCAF = 5,
  nDelta = 19,
  pDelta = vector(),
  tDelta = 1,
  costFunction = "RMSE",
  spDist = 1,
  drDist = 0,
  drShape = 3,
  drLim = c(0.1, 0.7),
  rtMax = 5000,
  subjects = c(),
  printInputArgs = TRUE,
  printResults = FALSE,
  optimControl = list(),
  numCores = 2
)
```

## Arguments

| | |
|---|---|
| `resOb` | Observed data (see flankerData and simonTask for data format) and the function dmcObservedData to create the required input from either an R data frame or external *.txt/*.csv files |
| `nTrl` | Number of trials to use within dmcSim. |
| `startVals` | Starting values for to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., startVals = list(amp = 20, tau = 200, drc = 0.5, bnds = 75, resMean = 300, resSD = 30, aaShape = 2, spShape = 3, spBias = 0, sigm = 4)). |
| `minVals` | Minimum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., minVals = list(amp = 10, tau = 5, drc = 0.1, bnds = 20, resMean = 200, resSD = 5, aaShape = 1, spShape = 2, spBias = -20, sigm = 1)). |
| `maxVals` | Maximum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., maxVals = list(amp = 40, tau = 300, drc = 1.0, bnds = 150, resMean = 800, resSD = 100, aaShape = 3, spShape = 4, spBias = 20, sigm = 10)) |
| `fixedFit` | Fix parameter to starting value. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., fixedFit = list(amp = F, tau = F, drc = F, bnds = F, resMean = F, resSD = F, aaShape = F, spShape = F, spBias = T, sigm = T)) |
| `fitInitialGrid` | TRUE/FALSE |
| `fitInitialGridN` | 10 linear steps between parameters min/max values (reduce if searching more than ~2/3 initial parameters) |
| `fixedGrid` | Fix parameter for initial grid search. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, spBias, sigm (e.g., fixedGrid = list(amp = T, tau = F, drc = T, bnds = T, resMean = T, resSD = T, aaShape = T, spShape = T, spBias = T, sigm = T)). As a default, the initial gridsearch only searches the tau space. |
| `nCAF` | Number of CAF bins. |
| `nDelta` | Number of delta bins. |
| `pDelta` | An alternative option to nDelta by directly specifying required percentile values (vector of values 0-100) |

| | |
|---|---|
| `tDelta` | The type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `costFunction` | The cost function to minimise: root mean square error ("RMSE": default), squared percentage error ("SPE"), or likelihood-ratio chi-square statistic ("GS") |
| `spDist` | The starting point (sp) distribution (0 = constant, 1 = beta, 2 = uniform) |
| `drDist` | The drift rate (dr) distribution type (0 = constant, 1 = beta, 2 = uniform) |
| `drShape` | The drift rate (dr) shape parameter |
| `drLim` | The drift rate (dr) range |
| `rtMax` | The limit on simulated RT (decision + non-decisional components) |
| `subjects` | NULL (aggregated data across all subjects) or integer for subject number |
| `printInputArgs` | TRUE (default) /FALSE |
| `printResults` | TRUE/FALSE (default) |
| `optimControl` | Additional control parameters passed to optim (see optim details section) |
| `numCores` | Number of cores to use |

## Value

dmcFitSubject returns a list of objects of class "dmcfit"

## Examples

```
# Code below can exceed CRAN check time limit, hence donttest
# Example 1: Flanker data from Ulrich et al. (2015)
fit <- dmcFitSubject(flankerData, nTrl = 1000, subjects = c(1, 2));
plot(fit, flankerData, subject = 1)
plot(fit, flankerData, subject = 2)
summary(fit)

# Example 2: Simon data from Ulrich et al. (2015)
fit <- dmcFitSubject(simonData, nTrl = 1000, subject = c(1, 2))
plot(fit, simonData, subject = 1)
plot(fit, simonData, subject = 2)
summary(fit)
```

---

# `dmcFitSubjectDE` *dmcFitSubjectDE*

---

## Description

Fit theoretical data generated from dmcSim to observed data by minimizing the root-mean-square error (RMSE) between a weighted combination of the CAF and CDF functions using the R-package DEoptim. Alternative cost functions include squared percentage error ("SPE"), and g-squared statistic ("GS").

## Usage

```
dmcFitSubjectDE(
  resOb,
  nTrl = 1e+05,
  minVals = list(),
  maxVals = list(),
  fixedFit = list(),
  nCAF = 5,
  nDelta = 19,
  pDelta = vector(),
  tDelta = 1,
  costFunction = "RMSE",
  spDist = 1,
  drDist = 0,
  drShape = 3,
  drLim = c(0.1, 0.7),
  rtMax = 5000,
  subjects = c(),
  deControl = list(),
  numCores = 2
)
```

## Arguments

| | |
|---|---|
| `resOb` | Observed data (see flankerData and simonTask for data format) |
| `nTrl` | The number of trials to use within dmcSim. |
| `minVals` | Minimum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., minVals = list(amp = 10, tau = 5, drc = 0.1, bnds = 20, resMean = 200, resSD = 5, aaShape = 1, spShape = 2, spBias = -20, sigm = 1)). |
| `maxVals` | Maximum values for the to-be estimated parameters. This is a list with values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., maxVals = list(amp = 40, tau = 300, drc = 1.0, bnds = 150, resMean = 800, resSD = 100, aaShape = 3, spShape = 4, spBias = 20, sigm = 10)) |
| `fixedFit` | Fix parameter to starting value. This is a list with bool values specified individually for amp, tau, drc, bnds, resMean, resSD, aaShape, spShape, sigm (e.g., fixedFit = list(amp = F, tau = F, drc = F, bnds = F, resMean = F, resSD = F, aaShape = F, spShape = F, spBias = T, sigm = T)) |
| `nCAF` | The number of CAF bins. |
| `nDelta` | The number of delta bins. |
| `pDelta` | An alternative option to nDelta by directly specifying required percentile values (vector of values 0-100) |
| `tDelta` | The type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |

| | |
|---|---|
| `costFunction` | The cost function to minimise: root mean square error ("RMSE": default), squared percentage error ("SPE"), or likelihood-ratio chi-square statistic ("GS") |
| `spDist` | The starting point distribution (0 = constant, 1 = beta, 2 = uniform) |
| `drDist` | The drift rate (dr) distribution type (0 = constant, 1 = beta, 2 = uniform) |
| `drShape` | The drift rate (dr) shape parameter |
| `drLim` | The drift rate (dr) range |
| `rtMax` | The limit on simulated RT (decision + non-decisional components) |
| `subjects` | NULL (aggregated data across all subjects) or integer for subject number |
| `deControl` | Additional control parameters passed to DEoptim (see DEoptim.control) |
| `numCores` | Number of cores to use |

### Value

dmcFitSubjectDE returns a list of objects of class "dmcfit"

### Examples

```
# Code below can exceed CRAN check time limit, hence donttest
# Example 1: Flanker data from Ulrich et al. (2015)
fit <- dmcFitSubjectDE(flankerData, nTrl = 1000, subjects = c(1, 2))
plot(fit, flankerData, subject = 1)
plot(fit, flankerData, subject = 2)
summary(fit)

# Example 2: Simon data from Ulrich et al. (2015)
fit <- dmcFitSubjectDE(simonData, nTrl = 1000, subject = c(1, 2))
plot(fit, simonData, subject = 1)
plot(fit, simonData, subject = 2)
summary(fit)
```

## `dmcObservedData` *dmcObservedData*

### Description

Basic analysis to create data object required for observed data. Example raw *.txt files are flankerData.txt and simonData.txt. There are four critical columns:

1. column containing subject number
2. column coding for compatible or incompatible
3. column with RT (in ms)
4. column indicating of the response was correct

## Usage

```
dmcObservedData(
  dat,
  nCAF = 5,
  nDelta = 19,
  pDelta = vector(),
  tDelta = 1,
  outlier = c(200, 1200),
  columns = c("Subject", "Comp", "RT", "Error"),
  compCoding = c("comp", "incomp"),
  errorCoding = c(0, 1),
  quantileType = 5,
  keepRaw = FALSE,
  delim = "\t",
  skip = 0
)
```

## Arguments

| | |
|---|---|
| `dat` | A text file(s) containing the observed data or an R DataFrame (see createDF/addDataDF) |
| `nCAF` | The number of CAF bins. |
| `nDelta` | The number of delta bins. |
| `pDelta` | An alternative option to nDelta by directly specifying required percentile values (vector of values 0-100) |
| `tDelta` | The type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `outlier` | Outlier limits in ms (e.g., c(200, 1200)) |
| `columns` | Name of required columns DEFAULT = c("Subject", "Comp", "RT", "Error") |
| `compCoding` | Coding for compatibility DEFAULT = c("comp", "incomp") |
| `errorCoding` | Coding for errors DEFAULT = c(0, 1)) |
| `quantileType` | Argument (1-9) from R function quantile specifying the algorithm (?quantile) |
| `keepRaw` | TRUE/FALSE |
| `delim` | Single character used to separate fields within a record if reading from external text file. |
| `skip` | The number of lines to skip before reading data if reading from external text file. |

## Value

dmcObservedData returns an object of class "dmcob" with the following components:

| | |
|---|---|
| `summarySubject` | DataFrame within individual subject data (rtCor, perErr, rtErr) for compatibility condition |
| `summary` | DataFrame within aggregated subject data (rtCor, sdRtCor, seRtCor, perErr, sdPerErr, sePerErr, rtErr, sdRtErr, seRtErr) for compatibility condition |

| | |
|---|---|
| `cafSubject` | DataFrame within individual subject conditional accuracy function (CAF) data (Bin, accPerComp, accPerIncomp, meanEffect) |
| `caf` | DataFrame within aggregated subject conditional accuracy function (CAF) data (Bin, accPerComp, accPerIncomp, meanEffect, sdEffect, seEffect) |
| `deltaSubject` | DataFrame within individual subject distributional delta analysis data correct trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `delta` | DataFrame within aggregated subject distributional delta analysis data correct trials (Bin, meanComp, meanIncomp, meanBin, meanEffect, sdEffect, seEffect) |
| `deltaErrorsSubject` | DataFrame within individual subject distributional delta analysis data incorrect trials (Bin, meanComp, meanIncomp, meanBin, meanEffect) |
| `deltaErrors` | DataFrame within aggregated subject distributional delta analysis data incorrect trials (Bin, meanComp, meanIncomp, meanBin, meanEffect, sdEffect, seEffect) |

## Examples

```
# Example 1
plot(flankerData)  # flanker data from Ulrich et al. (2015)
plot(simonData)    # simon data from Ulrich et al. (2015)

# Example 2 (Basic behavioural analysis from Ulrich et al. 2015)
flankerDat <- cbind(Task = "flanker", flankerData$summarySubject)
simonDat   <- cbind(Task = "simon",   simonData$summarySubject)
datAgg     <- rbind(flankerDat, simonDat)

datAgg$Subject <- factor(datAgg$Subject)
datAgg$Task    <- factor(datAgg$Task)
datAgg$Comp    <- factor(datAgg$Comp)

aovErr <- aov(perErr ~ Comp*Task + Error(Subject/(Comp*Task)), datAgg)
summary(aovErr)
model.tables(aovErr, type = "mean")

aovRt <- aov(rtCor ~ Comp*Task + Error(Subject/(Comp*Task)), datAgg)
summary(aovRt)
model.tables(aovRt, type = "mean")

# Example 3
dat <- createDF(nSubjects = 50, nTrl = 500, design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"    = c(500, 75, 120),
                           "Comp_incomp"  = c(530, 75, 100)),
                 Error = list("Comp_comp" = c(3, 2, 2, 1, 1),
                            "Comp_incomp" = c(21, 3, 2, 1, 1)))
datOb <- dmcObservedData(dat)
plot(datOb)
plot(datOb, subject = 1)

# Example 4
dat <- createDF(nSubjects = 50, nTrl = 500, design = list("Congruency" = c("cong", "incong")))
dat <- addDataDF(dat,
                 RT = list("Congruency_cong"   = c(500, 75, 100),
                           "Congruency_incong" = c(530, 100, 110)),
                 Error = list("Congruency_cong"   = c(3, 2, 2, 1, 1),
                              "Congruency_incong" = c(21, 3, 2, 1, 1)))
datOb <- dmcObservedData(dat, nCAF = 5, nDelta = 9,
                         columns = c("Subject", "Congruency", "RT", "Error"),
                         compCoding = c("cong", "incong"))
plot(datOb, labels = c("Congruent", "Incongruent"))
plot(datOb, subject = 1)
```

## `dmcSim` *dmcSim*

### Description

DMC model simulation detailed in Ulrich, R., Schroeter, H., Leuthold, H., & Birngruber, T. (2015). Automatic and controlled stimulus processing in conflict tasks: Superimposed diffusion processes and delta functions. Cognitive Psychology, 78, 148-174. This function is essentially a wrapper around the c++ function runDMC

### Usage

```
dmcSim(
  amp = 20,
  tau = 30,
  drc = 0.5,
  bnds = 75,
  resDist = 1,
  resMean = 300,
  resSD = 30,
  aaShape = 2,
  spShape = 3,
  sigm = 4,
  nTrl = 1e+05,
  tmax = 1000,
  spDist = 0,
  spLim = c(-75, 75),
  spBias = 0,
  drDist = 0,
  drShape = 3,
  drLim = c(0.1, 0.7),
  rtMax = 5000,
  fullData = FALSE,
  nTrlData = 5,
  nDelta = 9,
  pDelta = vector(),
  tDelta = 1,
  nCAF = 5,
  printInputArgs = TRUE,
  printResults = TRUE,
  setSeed = FALSE,
  seedValue = 1
)
```

## Arguments

| | |
|---|---|
| `amp` | amplitude of automatic activation |
| `tau` | time to peak automatic activation |
| `drc` | drift rate of controlled processes |
| `bnds` | +- response criterion |
| `resDist` | residual distribution type (1=normal, 2=uniform) |
| `resMean` | residual distribution mean |
| `resSD` | residual distribution standard deviation |
| `aaShape` | shape parameter of automatic activation |
| `spShape` | starting point (sp) shape parameter |
| `sigm` | diffusion constant |
| `nTrl` | number of trials |
| `tmax` | number of time points per trial |
| `spDist` | starting point (sp) distribution (0 = constant, 1 = beta, 2 = uniform) |
| `spLim` | starting point (sp) range |
| `spBias` | starting point (sp) bias |
| `drDist` | drift rate (dr) distribution type (0 = constant, 1 = beta, 2 = uniform) |
| `drShape` | drift rate (dr) shape parameter |
| `drLim` | drift rate (dr) range |
| `rtMax` | limit on simulated RT (decision + non-decisional component) |
| `fullData` | TRUE/FALSE (Default: FALSE) NB. only required when plotting activation function and/or individual trials |
| `nTrlData` | Number of trials to plot |
| `nDelta` | number of delta bins |
| `pDelta` | alternative to nDelta by directly specifying required percentile values (0-100) |
| `tDelta` | type of delta calculation (1=direct percentiles points, 2=percentile bounds (tile) averaging) |
| `nCAF` | Number of CAF bins |
| `printInputArgs` | TRUE/FALSE |
| `printResults` | TRUE/FALSE |
| `setSeed` | TRUE/FALSE If true, set seed to seed value |
| `seedValue` | 1 |

## Value

dmcSim returns an object of class "dmcsim" with the following components:

| | |
|---|---|
| `sim` | Individual trial data points (reaction times/error) and activation vectors from simulation |
| `summary` | Condition means for reaction time and error rate |
| `caf` | Accuracy per bin for compatible and incompatible trials |
| `delta` | Mean RT and compatibility effect per bin |
| `delta_errs` | Mean RT and compatibility effect per bin |
| `prms` | The input parameters used in the simulation |

## Examples

```
# Example 1
dmc <- dmcSim(fullData = TRUE)  # fullData only required for activation/trials (top/bottom left)
plot(dmc)
dmc <- dmcSim() # faster!
plot(dmc)

# Example 2
dmc <- dmcSim(tau = 130)
plot(dmc)

# Example 3
dmc <- dmcSim(tau = 90)
plot(dmc)

# Example 4
dmc <- dmcSim(spDist = 1)
plot(dmc, "delta")

# Example 5
dmc <- dmcSim(tau = 130, drDist = 1)
plot(dmc, "caf")

# Example 6
dmc <- dmcSim(nDelta = 10, nCAF = 10)
plot(dmc)
```

---

`dmcSimApp` *dmcSimApp*

---

## Description

A shiny app allowing interactive exploration of DMC parameters

## Usage

```
dmcSimApp()
```

## Value

Shiny App

---

`dmcSims` *dmcSims: Run multiple dmc simulations*

---

## Description

Run dmcSim with range of input parameters.

## Usage

```
dmcSims(params, printInputArgs = FALSE, printResults = FALSE)
```

## Arguments

| | |
|---|---|
| `params` | (list of parameters to dmcSim) |
| `printInputArgs` | Print DMC input arguments to console |
| `printResults` | Print DMC output to console |

## Value

dmcSims returns a list of objects of class "dmcsim"

## Examples

```
# Example 1
params <- list(amp = seq(10, 20, 5), tau = c(50, 100, 150), nTrl = 50000)
dmc <- dmcSims(params)
plot(dmc[[1]])    # full combination 1
plot(dmc)         # delta plots for all combinations
plot(dmc[c(1:3)]) # delta plots for specific combinations

# Example 2
params <- list(amp = seq(10, 20, 5), tau = seq(20, 40, 20), bnds = seq(50, 100, 25))
dmc <- dmcSims(params)
plot(dmc[[1]])  # combination 1
plot(dmc, ncol = 2)       # delta plots for all combinations
plot(dmc[c(1:3)]) # delta plots for specific combinations
```

## `errDist` *errDist*

### Description

Returns a random vector of 0's (correct) and 1's (incorrect) with defined proportions (default = 10% errors).

### Usage

```
errDist(n = 10000, proportion = 10)
```

### Arguments

| | |
|---|---|
| `n` | Number |
| `proportion` | Approximate proportion of errors in percentage |

### Value

double

### Examples

```
# Example 1
x <- errDist(1000, 10)
table(x)
```

## `flankerData` *A summarised dataset: This is the flanker task data from Ulrich et al. (2015)*

### Description

- $summary –> Reaction time correct, standard deviation correct, standard error correct, percentage error, standard deviation error, standard error error, reaction time incorrect, standard deviation incorrect, and standard error incorrect trials for both compatible and incompatible trials
- $caf –> Proportion correct for compatible and incompatible trials across 5 bins
- $delta –> Compatible reactions times, incompatible mean reaction times, mean reaction times, incompatible - compatible reaction times (effect), and standard deviation + standard error of this effect across 19 bins
- $data –> Raw data from flankerData.txt + additional outlier column

### Usage

```
flankerData
```

### Format

dmcob

---

## mean.dmcfit *mean.dmcfit*

---

### Description

Aggregate simulation results from dmcFitSubject/dmcFitSubjectDE.

### Usage

```
## S3 method for class 'dmcfit'
mean(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | Output from dmcFitSubject/dmcFitSubjectDE |
| `...` | pars |

### Value

mean.dmcfit return an object of class "dmcfit" with the following components:

| | |
|---|---|
| `summary` | DataFrame within aggregated subject data (rtCor, sdRtCor, seRtCor, perErr, sd-PerErr, sePerErr, rtErr, sdRtErr, seRtErr) for compatibility condition |
| `delta` | DataFrame within aggregated subject distributional delta analysis data correct trials (Bin, meanComp, meanIncomp, meanBin, meanEffect, sdEffect, seEffect) |
| `caf` | DataFrame within aggregated subject conditional accuracy function (CAF) data (Bin, accPerComp, accPerIncomp, meanEffect, sdEffect, seEffect) |
| `par` | The fitted model parameters + final cost value of the fit |

### Examples

```
# Code below can exceed CRAN check time limit, hence donttest
# Example 1: Fit individual data then aggregate
fitSubjects <- dmcFitSubject(flankerData, nTrl = 1000, subjects = c(1, 2))
fitAgg <- mean(fitSubjects)
plot(fitAgg, flankerData)
```

`plot.dmcfit` *plot.dmcfit: Plot observed + fitted data*

## Description

Plot the simulation results from the output of dmcFit. The plot can be an overall summary, or individual plots (activation, trials, pdf, cdf, caf, delta, all). Plot type summary1 contains an activation plot, example individual trials, the probability distribution function (PDF), the cumulative distribution function (CDF), the conditional accuracy function (CAF) and delta plots. This required that dmcSim is run with fullData = TRUE. Plot type summary2 contains only the PDF, CDF, CAF and delta plots and does not require that dmcSim is run with fullData = TRUE.

## Usage

```
## S3 method for class 'dmcfit'
plot(
  x,
  y,
  subject = NULL,
  figType = "summary",
  legend = TRUE,
  labels = c("Compatible", "Incompatible", "Observed", "Predicted"),
  cols = c("black", "green", "red"),
  ylimRt = NULL,
  ylimErr = NULL,
  xlimCDF = NULL,
  ylimCAF = NULL,
  cafBinLabels = FALSE,
  ylimDelta = NULL,
  xlimDelta = NULL,
  xlabs = TRUE,
  ylabs = TRUE,
  xaxts = TRUE,
  yaxts = TRUE,
  resetPar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Output from dmcFit |
| `y` | Observed data |
| `subject` | NULL (aggregated data across all subjects) or integer for subject number |
| `figType` | summary, rtCorrect, errorRate, rtErrors, cdf, caf, delta, all |
| `legend` | TRUE/FALSE (or FUNCTION) plot legend on each plot |

| | |
|---|---|
| `labels` | Condition labels c("Compatible", "Incompatible", "Observed", "Predicted") default |
| `cols` | Condition colours c("green", "red") default |
| `ylimRt` | ylimit for Rt plots |
| `ylimErr` | ylimit for error rate plots |
| `xlimCDF` | ylimit for CDF plot |
| `ylimCAF` | ylimit for CAF plot |
| `cafBinLabels` | TRUE/FALSE |
| `ylimDelta` | ylimit for delta plot |
| `xlimDelta` | xlimit for delta plot |
| `xlabs` | TRUE/FALSE |
| `ylabs` | TRUE/FALSE |
| `xaxts` | TRUE/FALSE |
| `yaxts` | TRUE/FALSE |
| `resetPar` | TRUE/FALSE Reset graphical parameters |
| `...` | additional plot pars |

## Value

Plot (no return value)

## Examples

```
# Example 1
resTh <- dmcFit(flankerData, nTrl = 5000)
plot(resTh, flankerData, figType = "rtcorrect")

# Example 2
resTh <- dmcFit(flankerData, nTrl = 5000)
plot(resTh, flankerData)
plot(resTh, flankerData, figType = "all")

# Example 3
resTh <- dmcFit(simonData, nTrl = 5000)
plot(resTh, simonData)
```

`plot.dmclist` *plot.dmclist: Plot delta plots from multiple dmc simulations.*

## Description

Plot delta function from multiple dmc simulations (i.e., dmcSims).

## Usage

```
## S3 method for class 'dmclist'
plot(
  x,
  ylim = NULL,
  xlim = NULL,
  figType = "delta",
  xlab = "Time [ms]",
  col = c("black", "lightgrey"),
  lineType = "l",
  legend = TRUE,
  legendPos = "topleft",
  legendLabels = NULL,
  ncol = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Output from dmcSims |
| `ylim` | ylimit for delta plot |
| `xlim` | xlimit for delta plot |
| `figType` | delta (default), deltaErrors |
| `xlab` | x-label |
| `col` | color range start/end color |
| `lineType` | line type ("l", "b", "o") for delta plot |
| `legend` | TRUE/FALSE Show legend |
| `legendPos` | legend position |
| `legendLabels` | Custom legend labels |
| `ncol` | number of legend columns |
| `...` | pars for legend |

## Value

Plot (no return value)

## Examples

```
# Example 1
params <- list(amp = seq(20, 30, 2))
dmc <- dmcSims(params)
plot(dmc, ncol = 2, col = c("red", "green"), legendPos = "topright")

# Example 2
params <- list(amp=c(10, 20), tau = seq(20, 80, 40), drc = seq(0.2, 0.6, 0.2), nTrl = 50000)
dmc <- dmcSims(params)
plot(dmc, ncol = 2, col=c("green", "blue"), ylim = c(-10, 120))
```

---

## plot.dmcob *plot.dmcob: Plot observed data*

---

## Description

Plot results from the output of dmcObservedData. The plot can be an overall summary, or individual plots (rtCorrect, errorRate, rtErrors, cdf, caf, delta, deltaErrors, all).

## Usage

```
## S3 method for class 'dmcob'
plot(
  x,
  figType = "summary",
  subject = NULL,
  legend = TRUE,
  labels = c("Compatible", "Incompatible"),
  cols = c("black", "green", "red"),
  errorBars = FALSE,
  errorBarType = "sd",
  ylimRt = NULL,
  ylimErr = NULL,
  xlimCDF = NULL,
  ylimCAF = NULL,
  cafBinLabels = FALSE,
  ylimDelta = NULL,
  xlimDelta = NULL,
  ylimDeltaErrors = NULL,
  xlabs = TRUE,
  ylabs = TRUE,
  xaxts = TRUE,
  yaxts = TRUE,
  resetPar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Output from dmcObservedData |
| `figType` | summary, rtCorrect, errorRate, rtErrors, cdf, caf, delta, deltaErrors, all |
| `subject` | NULL (aggregated data across all subjects) or integer for subject number |
| `legend` | TRUE/FALSE (or FUNCTION) plot legend on each plot |
| `labels` | Condition labels c("Compatible", "Incompatible") default |
| `cols` | Condition colours c("green", "red") default |
| `errorBars` | TRUE(default)/FALSE Plot errorbars |
| `errorBarType` | sd(default), or se |
| `ylimRt` | ylimit for Rt plots |
| `ylimErr` | ylimit for error rate plots |
| `xlimCDF` | xlimit for CDF plot |
| `ylimCAF` | ylimit for CAF plot |
| `cafBinLabels` | TRUE/FALSE |
| `ylimDelta` | ylimit for delta plot |
| `xlimDelta` | xlimit for delta plot |
| `ylimDeltaErrors` | ylimit for delta plot errors |
| `xlabs` | TRUE/FALSE |
| `ylabs` | TRUE/FALSE |
| `xaxts` | TRUE/FALSE |
| `yaxts` | TRUE/FALSE |
| `resetPar` | TRUE/FALSE Reset graphical parameters |
| `...` | additional plot pars |

## Value

Plot (no return value)

## Examples

```
# Example 1 (real dataset)
plot(flankerData)
plot(flankerData, errorBars = TRUE, errorBarType = "se")
plot(flankerData, figType = "delta")
plot(flankerData, figType = "caf")
```

```
# Example 2 (real dataset)
plot(simonData)
plot(simonData, errorBars = TRUE, errorBarType = "se")
plot(simonData, figType = "delta", errorBars = TRUE, errorBarType = "sd")

# Example 3 (simulated dataset)
dat <- createDF(nSubjects = 50, nTrl = 50,
                design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"   = c(420, 100, 80),
                           "Comp_incomp" = c(470, 100, 95)),
                 Error = list("Comp_comp"   = c(5, 3, 2, 1, 2),
                              "Comp_incomp" = c(15, 8, 4, 2, 2)))
datOb <- dmcObservedData(dat)
plot(datOb, errorBars = TRUE, errorBarType = "sd")

# Example 4 (simulated dataset)
dat <- createDF(nSubjects = 50, nTrl = 50,
                design = list("Comp" = c("comp", "incomp")))
dat <- addDataDF(dat,
                 RT = list("Comp_comp"   = c(420, 100, 150),
                           "Comp_incomp" = c(470, 100, 120)),
                 Error = list("Comp_comp"   = c(5, 3, 2, 1),
                              "Comp_incomp" = c(15, 8, 4, 2)))
datOb <- dmcObservedData(dat, nCAF = 4)
plot(datOb)
```

## `plot.dmcobs` *plot.dmcobs: Plot combined observed data*

### Description

Plot delta results from the output of dmcObservedData. The plot can be an overall rtCorrect, errorRate, rtErrors, cdf, caf, delta, or all of the previous plots.

### Usage

```
## S3 method for class 'dmcobs'
plot(
  x,
  figType = "all",
  subject = NULL,
  legend = TRUE,
  legendLabels = c(),
  labels = c("Compatible", "Incompatible"),
  cols = c("black", "gray"),
  ltys = c(1, 1),
  pchs = c(1, 1),
  errorBars = FALSE,
  errorBarType = "sd",
  ylimRt = NULL,
  ylimErr = NULL,
  xlimCDF = NULL,
  ylimCAF = NULL,
  cafBinLabels = FALSE,
  ylimDelta = NULL,
  xlimDelta = NULL,
  xlabs = TRUE,
  ylabs = TRUE,
  xaxts = TRUE,
  yaxts = TRUE,
  resetPar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Output from dmcObservedData |
| `figType` | rtCorrect, errorRate, rtErrors, cdf, caf, delta, all |
| `subject` | NULL (aggregated data across all subjects) or integer for subject number |
| `legend` | TRUE/FALSE (or FUNCTION) plot legend on each plot |
| `legendLabels` | legend labels |
| `labels` | Condition labels c("Compatible", "Incompatible") default |
| `cols` | Condition colours c("green", "red") default |
| `ltys` | Linetype see par |
| `pchs` | Symbols see par |
| `errorBars` | TRUE(default)/FALSE Plot errorbars |
| `errorBarType` | sd(default), or se |
| `ylimRt` | ylimit for Rt plots |
| `ylimErr` | ylimit for error rate plots |
| `xlimCDF` | xlimit for CDF plot |
| `ylimCAF` | ylimit for CAF plot |
| `cafBinLabels` | TRUE/FALSE |
| `ylimDelta` | ylimit for delta plot |
| `xlimDelta` | xlimit for delta plot |
| `xlabs` | TRUE/FALSE |
| `ylabs` | TRUE/FALSE |
| `xaxts` | TRUE/FALSE |
| `yaxts` | TRUE/FALSE |
| `resetPar` | TRUE/FALSE Reset graphical parameters |
| `...` | additional plot pars |

## Value

Plot (no return value)

## Examples

```
# Example 1
dat <- dmcCombineObservedData(flankerData, simonData)  # combine flanker/simon data
plot(dat, figType = "delta", xlimDelta = c(200, 700), ylimDelta = c(-20, 80),
     cols = c("black", "darkgrey"), pchs = c(1, 2), legend = FALSE, resetPar = FALSE)
legend(200, 80, legend = c("Flanker Task", "Simon Task"),
       col = c("black", "darkgrey"), lty = c(1, 1))
```

---

## plot.dmcsim *plot.dmcsim: Plot dmc simulation*

---

### Description

Plot the simulation results from the output of dmcSim. The plot can be an overall summary, or individual plots (activation, trials, pdf, cdf, caf, delta, all). Plot type summary1 contains an activation plot, example individual trials, the probability distribution function (PDF), the cumulative distribution function (CDF), the conditional accuracy function (CAF) and delta plot. This requires that dmcSim is run with fullData = TRUE. Plot type summary2 contains only the PDF, CDF, CAF and delta plots and does not require that dmcSim is run with fullData = TRUE.

### Usage

```
## S3 method for class 'dmcsim'
plot(
  x,
  figType = "summary1",
  xlimActivation = NULL,
  xlimTrials = NULL,
  xlimPDF = NULL,
  ylimPDF = NULL,
  xlimCDF = NULL,
  ylimCAF = NULL,
  cafBinLabels = FALSE,
  ylimDelta = NULL,
  xlimDelta = NULL,
  ylimDeltaErrors = NULL,
  ylimRt = NULL,
  ylimErr = NULL,
  legend = TRUE,
  labels = c("Compatible", "Incompatible"),
  cols = c("black", "green", "red"),
  errorBars = FALSE,
  xlabs = TRUE,
  ylabs = TRUE,
  xaxts = TRUE,
  yaxts = TRUE,
  resetPar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Output from dmcSim |
| `figType` | summary1, summary2, summary3, activation, trials, pdf, cdf, caf, delta, rtCorrect, rtErrors, errorRate, all |
| `xlimActivation` | xlimit for activation plot |
| `xlimTrials` | xlimit for trials plot |
| `xlimPDF` | xlimit for PDF plot |
| `ylimPDF` | ylimit for PDF plot |
| `xlimCDF` | xlimit for CDF plot |
| `ylimCAF` | ylimit for CAF plot |
| `cafBinLabels` | TRUE/FALSE |
| `ylimDelta` | ylimit for delta plot |
| `xlimDelta` | xlimit for delta plot (Default is 0 to tmax) |
| `ylimDeltaErrors` | ylimit for delta plot of errors |
| `ylimRt` | ylimit for rt plot |
| `ylimErr` | ylimit for er plot |
| `legend` | TRUE/FALSE (or FUNCTION) plot legend on each plot |
| `labels` | Condition labels c("Compatible", "Incompatible") default |
| `cols` | Condition colours c("green", "red") default |
| `errorBars` | TRUE/FALSE |
| `xlabs` | TRUE/FALSE |
| `ylabs` | TRUE/FALSE |
| `xaxts` | TRUE/FALSE |
| `yaxts` | TRUE/FALSE |
| `resetPar` | TRUE/FALSE Reset graphical parameters |
| `...` | additional plot pars |

## Value

Plot (no return value)

## Examples

```
# Example 1
dmc = dmcSim(fullData = TRUE)
plot(dmc)

# Example 2
dmc = dmcSim()
plot(dmc)

# Example 3
dmc = dmcSim()
plot(dmc, figType = "all")

# Example 4
dmc = dmcSim()
plot(dmc, figType = "summary3")
```

---

## `rtDist` *rtDist*

---

### Description

Returns value(s) from a distribution appropriate to simulate reaction times. The distribution is a combined exponential and gaussian distribution called an exponentially modified Gaussian (EMG) distribution or ex-gaussian distribution.

### Usage

```
rtDist(n = 10000, gaussMean = 600, gaussSD = 50, expRate = 200)
```

### Arguments

| | |
|---|---|
| `n` | Number of observations |
| `gaussMean` | Mean of the gaussian distribution |
| `gaussSD` | SD of the gaussian distribution |
| `expRate` | Rate of the exponential function |

### Value

double

### Examples

```
# Example 1
x <- rtDist()
hist(x, 100, xlab = "RT [ms]")

# Example 2
x <- rtDist(n=20000, gaussMean=800, gaussSD=50, expRate=100)
hist(x, 100, xlab = "RT [ms]")
```

---

## `simonData` *A summarised dataset: This is the simon task data from Ulrich et al. (2015)*

---

### Description

- $summary –> Reaction time correct, standard deviation correct, standard error correct, percentage error, standard deviation error, standard error error, reaction time incorrect, standard deviation incorrect, and standard error incorrect trials for both compatible and incompatible trials
- $caf –> Proportion correct for compatible and incompatible trials across 5 bins
- $delta –> Compatible reactions times, incompatible mean reaction times, mean reaction times, incompatible - compatible reaction times (effect), and standard deviation + standard error of this effect across 19 bins
- $data –> Raw data from simonData.txt + additional outlier column

### Usage

```
simonData
```

### Format

dmcob

---

## `summary.dmcfit` *summary.dmcfit: dmc fit aggregate summary*

---

### Description

Summary of the simulation results from dmcFitAgg

### Usage

```
## S3 method for class 'dmcfit'
summary(object, digits = 2, ...)
```

## Arguments

| | |
|---|---|
| `object` | Output from dmcFitAgg |
| `digits` | Number of digits in the output |
| `...` | pars |

## Value

DataFrame

## Examples

```
# Example 1
fitAgg <- dmcFit(flankerData, nTrl = 1000)
summary(fitAgg)
```

---

`summary.dmcsim` *summary.dmcsim: dmc simulation summary*

---

## Description

Summary of the overall results from dmcSim

## Usage

```
## S3 method for class 'dmcsim'
summary(object, digits = 1, ...)
```

## Arguments

| | |
|---|---|
| `object` | Output from dmcSim |
| `digits` | Number of digits in the output |
| `...` | pars |

## Value

DataFrame

## Examples

```
# Example 1
dmc <- dmcSim()
summary(dmc)

# Example 2
dmc <- dmcSim(tau = 90)
summary(dmc)
```

# Index