Package 'cas.simples'

June 8, 2016

Type Package

Title Implementation des cas simples de calcul de puissance (t-test de Student).

Version 1.0

Date 2016-06-08

Author Bonjean Gregoire, Crepin Baptiste et Lair Thomas

Maintainer Lair Thomas <thomas.lair@ensimag.grenoble-inp.fr>

Description Ce package permet le calcul de la puissance d'un test d'hypothese (presence d'un effet sous la forme d'un ecart a la moyenne) dans le cas d'un echantillon, de deux echantillons independants et de deux echantillons apparies. Il comprends des fonctions qui permettent de generer des pilotes tests, ainsi que des fonctions pour le Bootstrap et la methode de Monte-Carlo.

License x

R topics documented:

cas.simples-package *Calcul de puissance de cas usuels de tests d'hypothese.*

Description

Ce package permet le calcul de la puissance d'un test d'hypothese (presence d'un effet sous la forme d'un ecart à la moyenne) dans le cas d'un echantillon, de deux echantillons independants et de deux echantillons apparies. Il comprends des fonctions qui permettent de generer des pilotes tests, ainsi que des fonctions pour le Bootstrap et la methode de Monte-Carlo.

Details

Package:	cas.simples
Type:	Package
Version:	1.0
Date:	2016-06-08
License:	None

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Maintainer: Lair Thomas <thomas.lair@ensimag.grenoble-inp.fr>

References

Tests statistiques parametriques : Puissance, taille d'effet et taille d'echantillon (sous R). – Stéphane CHAMPELY, Universite Lyon 1, France.

calcul_n *Calcul de taille d'echantillon.*

Description

Cette fonction permet le calcul par approximation affine de la taille d'echantillon necessaire à obtenir la puissance visee.

Usage

```
calcul_n(npoints, puissance, puissances, tailles)
```

Arguments

npoints	Nombre de points du graphe puissance = f(taille d'echantillon).
puissance	Puissance visee pour le test d'hypothese.
puissances	Vecteur des puissances calcules en les npoints points du graphe.
tailles	Vecteur des tailles d'echantillons (simulé par Monte-Carlo).

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (npoints, puissance, puissances, tailles)
{
    if (is.null(puissance)) {
        return(-1)
    }
    else {
        i = 0
        p_cour = 0
        while ((i < npoints + 1) && (p_cour < puissance)) {
            i = i + 1
            p_cour = puissances[i]
        }
        if (i == npoints + 1) {
            return(0)
        }
        else {
            n2 = tailles[i]
            p2 = puissances[i]
            if (i == 1) {
                n1 = 0
                p1 = 0
            }
            else {
                n1 = tailles[i - 1]
                p1 = puissances[i - 1]
            }
            pente = (p2 - p1)/(n2 - n1)
            if (pente == 0) {
                n_calc = (n2 - n1)/2
            }
            else {
                n_calc = (puissance - p1)/pente + n1
            }
            n_calc = ceiling(n_calc)
            return(n_calc)
        }
    }
}
```

MC *Methode de Monte-Carlo, pour le cas "Un Echantillon".*

Description

Cette fonction implemente la methode de duplication d'observations de Monte-Carlo dans le cas "Un Echantillon".

Usage

```
MC(n, runs, meand, sd)
```

Arguments

n	Nombre d'observations de l'echantillon.
runs	Nombre d'iteration de la methode de Monte-Carlo.
meand	Ecart à la moyenne (seuil).
sd	Ecart-type de l'echantillon.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (n, runs, meand, sd)
{
    alpha = 0.05
    x = c(rep(1, n))
    x = factor(x)
    tval = numeric(runs)
    tval_hand = numeric(runs)
    pval = numeric(runs)
    for (r in 1:runs) {
        y = c(rnorm(n, mean = meand, sd = sd))
        model = t.test(y)
        tval[r] = model$statistic
        pval[r] = model$p.value
        y1 = y[x == 1]
        s12 = sd(y1)
        tval_hand[r] = mean(y1)/(s12/sqrt(n))
    }
    nb = sum(pval < alpha)
    p5_model = nb/runs
    thr_low = qt(alpha/2, n - 1, lower.tail = T)
    thr_upp = qt(alpha/2, n - 1, lower.tail = F)
    nb = sum(tval_hand > thr_upp | tval_hand < thr_low)
    p5_hand = nb/runs
```

```
    p5_package = power.t.test(n = n, d = meand/sd, sig.level = alpha,
        type = "one.sample", alternative = "two.sided")$power
    return(list(meand = meand, sd = sd, p5_model = p5_model,
        p5_hand = p5_hand, p5_package = p5_package))
}
```

MC_ap *Methode de Monte-Carlo, pour le cas "Deux Echantillons Apparies".*

Description

Cette fonction implemente la methode de duplication d'observations de Monte-Carlo dans le cas "Deux Echantillons Apparies".

Usage

```
MC_ap(n, runs, meand, s1, s2, cf)
```

Arguments

n	Nombre d'observations des echantillons.
runs	Nombre d'iteration de la methode de Monte-Carlo.
meand	Difference de moyennes des echantillons.
s1	Ecart-type du premier echantillon.
s2	Ecart-type du second echantillon.
cf	Facteur de correlation des echantillons.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (n, runs, meand, s1, s2, cf)
{
    alpha = 0.05
    x = c(rep(1, n))
    x = factor(x)
    tval = numeric(runs)
    tval_hand = numeric(runs)
    pval = numeric(runs)
    sz = sqrt(s1^2 + s2^2 - (2 * cf * s1 * s2))
    for (r in 1:runs) {
        y = c(rnorm(n, mean = meand, sd = sz))
        model = t.test(y)
        tval[r] = model$statistic
        pval[r] = model$p.value
        y1 = y[x == 1]
```

```
        s12 = sd(y1)
        tval_hand[r] = mean(y1)/(s12/sqrt(n))
    }
    res = data.frame(t_statistic = tval, t_statistic_hand = tval_hand,
        p_value = pval)
    head(res, 10)
    nb = sum(pval < alpha)
    p5_model = nb/runs
    thr_low = qt(alpha/2, n + n - 2, lower.tail = T)
    thr_upp = qt(alpha/2, n + n - 2, lower.tail = F)
    nb = sum(tval < thr_low | tval > thr_upp)
    p5_hand = nb/runs
    p5_package = power.t.test(n = n, d = (meand)/sz, sig.level = alpha,
        type = "paired", alternative = "two.sided")$power
    return(list(meand = meand, sd = sd, p5_model = p5_model,
        p5_hand = p5_hand, p5_package = p5_package))
}
```

`MC_ind` *Methode de Monte-Carlo, pour le cas "Deux Echantillons Independants".*

Description

Cette fonction implemente la methode de duplication d'observations de Monte-Carlo dans le cas "Deux Echantillons Independants".

Usage

```
MC_ind(n1, n2, runs, meand, sd)
```

Arguments

`n1`	Nombre d'observations du premier echantillon.
`n2`	Nombre d'observations du deuxieme echantillon.
`runs`	Nombre d'iteration de la methode de Monte-Carlo.
`meand`	Difference de moyennes des echantillons.
`sd`	Ecart-type des echantillons.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (n1, n2, runs, meand, sd)
{
    alpha = 0.05
    x = c(rep(1, n1), rep(2, n2))
    x = factor(x)
    tval = numeric(runs)
    tval_hand = numeric(runs)
    pval = numeric(runs)
    for (r in 1:runs) {
        y = c(rnorm(n1, mean = 0, sd = sd), rnorm(n2, mean = meand,
            sd = sd))
        y1 = y[x == 1]
        y2 = y[x == 2]
        s12 = sqrt((sd(y1)^2 + sd(y2)^2)/(n1 + n2 - 2))
        tval_hand[r] = (mean(y1) - mean(y2))/(s12 * sqrt((1/n1) +
            (1/n2)))
        model = t.test(y2, y1, var.equal = TRUE)
        tval[r] = model$statistic
        pval[r] = model$p.value
    }
    res = data.frame(t_statistic = tval, t_statistic_hand = tval_hand,
        p_value = pval)
    head(res, 10)
    nb = sum(pval < alpha)
    p5_model = nb/runs
    thr_low = qt(alpha/2, n1 + n2 - 2, lower.tail = T)
    thr_upp = qt(alpha/2, n1 + n2 - 2, lower.tail = F)
    nb = sum(tval < thr_low | tval > thr_upp)
    p5_hand = nb/runs
    p5_package = power.t.test(n = (n1 + n2)/2, d = meand/sd,
        sig.level = alpha, type = "two.sample", alternative = "two.sided")$power
    return(list(meand = meand, sd = sd, p5_model = p5_model,
        p5_hand = p5_hand, p5_package = p5_package))
  }
```

`pilote_ttest_independants` *Etude Pilote, dans le cas "Deux Echantillons Independants".*

Description

Cette fonction genere et affiche l'histogramme d'un pilote, pour le cas "Deux Echantillons Independants". Elle utilise du Bootstrap (qui sélectionne 80

Usage

```
pilote_ttest_independants(npilote_congruent, npilote_incongruent, meand, s, runs_bs_pilote, dest_pil
```

Arguments

npilote_congruent	Nombre d'observation du premier echantillon de l'etude pilote.
npilote_incongruent	Nombre d'observation du second echantillon de l'etude pilote.
meand	Difference de moyenne entre les deux normales permettant de generer les deux echantillons.
s	Ecart-type des deux normales permettant de generer les deux echantillons.
runs_bs_pilote	Nombre d'iteration de Bootstrap applique au pilote pour le calcul des intervalles de confiance.
dest_pilote	Destination de l'image contenant l'histogramme du pilote.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (npilote_congruent, npilote_incongruent, meand, s, runs_bs_pilote,
    dest_pilote)
{
    alpha = 0.05
    congruent = rnorm(npilote_congruent, mean = 0, sd = s)
    incongruent = rnorm(npilote_incongruent, mean = meand, sd = s)
    mean1 = mean(congruent)
    mean2 = mean(incongruent)
    meand2 = mean2 - mean1
    ecart_type = (sd(congruent) + sd(incongruent))/2
    conf_mean = t.test(incongruent, congruent, conf.level = alpha)$conf.int
    table_sd = numeric(runs_bs_pilote)
    for (i in 1:runs_bs_pilote) {
        n_bs_congruent = ceiling(0.8 * npilote_congruent)
        n_bs_incongruent = ceiling(0.8 * npilote_incongruent)
        table_sd[i] = (sd((sample(congruent, n_bs_congruent,
            replace = T))) + sd((sample(incongruent, n_bs_incongruent,
            replace = T))))/2
    }
    table_sd_sorted = sort(table_sd)
    conf_sd = c(table_sd_sorted[floor(runs_bs_pilote * alpha)],
        table_sd_sorted[floor(runs_bs_pilote * (1 - alpha))])
    jpeg(dest_pilote)
    densCongruent <- density(congruent)
    densIncongruent <- density(incongruent)
    histCongruent <- hist(congruent, breaks = 10, plot = FALSE)
    histIncongruent <- hist(incongruent, breaks = 10, plot = FALSE)
    xlim <- range(histIncongruent$breaks, histCongruent$breaks)
    ylim <- range(0, histIncongruent$density, histCongruent$density)
    plot(histCongruent, xlim = xlim, ylim = ylim, col = rgb(1,
        0, 0, 0.4), xlab = "congruent", freq = FALSE, main = "Distribution")
    opar <- par(new = FALSE)
```

```
  plot(histIncongruent, xlim = xlim, ylim = ylim, xaxt = "n",
      yaxt = "n", col = rgb(0, 0, 1, 0.4), add = TRUE, freq = FALSE)
  legend("topleft", c("Congruent", "Incongruent"), fill = rgb(1:0,
      0, 0:1, 0.4), bty = "n", border = NA)
  par(opar)
  xfit1 <- seq(min(congruent), max(congruent), length = 40)
  yfit1 <- dnorm(xfit1, mean = mean1, sd = ecart_type)
  yfit1 <- yfit1 * diff(histCongruent$mids[1:2]) * length(congruent)
  lines(xfit1, yfit1, col = rgb(1, 0, 0, 0.4), lwd = 2)
  xfit2 <- seq(min(incongruent), max(incongruent), length = 40)
  yfit2 <- dnorm(xfit2, mean = mean2, sd = ecart_type)
  yfit2 <- yfit2 * diff(histIncongruent$mids[1:2]) * length(incongruent)
  lines(xfit2, yfit2, col = rgb(0, 0, 1, 0.4), lwd = 2)
  dev.off()
  return(list(ecart_type = ecart_type, mean = meand2, conf_mean = conf_mean,
      conf_sd = conf_sd))
}
```

`pilote_ttest_normal` *Etude Pilote, dans le cas "Un Echantillon".*

Description

Cette fonction genere et affiche l'histogramme d'un pilote, pour le cas "Un Echantillon". Elle utilise du Bootstrap (qui sélectionne 80

Usage

```
pilote_ttest_normal(npilote, meand, sd, runs_bs_pilote, dest_pilote)
```

Arguments

`npilote`	Nombre d'observation de l'etude pilote.
`meand`	Moyenne de la loi normale permettant de generer l'etude pilote. C'est aussi l'ecart a la moyenne d'une normale centree.
`sd`	Ecart-type de la loi normale permettant de generer l'etude pilote.
`runs_bs_pilote`	Nombre d'iteration de Bootstrap applique au pilote pour le calcul des intervalles de confiance.
`dest_pilote`	Destination de l'image contenant l'histogramme du pilote.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (npilote, meand, sd, runs_bs_pilote, dest_pilote)
{
    alpha = 0.05
    echantillon = rnorm(npilote, mean = meand, sd = sd)
    meand2 = mean(echantillon)
    ecart_type = sd(echantillon)
    conf_mean = t.test(echantillon, conf.level = alpha)$conf.int
    table_sd = numeric(runs_bs_pilote)
    for (i in 1:runs_bs_pilote) {
        n_bs = ceiling(0.8 * npilote)
        table_sd[i] = sd((sample(echantillon, n_bs, replace = T)))
    }
    table_sd_sorted = sort(table_sd)
    conf_sd = c(table_sd_sorted[floor(runs_bs_pilote * alpha)],
        table_sd_sorted[floor(runs_bs_pilote * (1 - alpha))])
    jpeg(dest_pilote)
    redleger = rgb(1, 0, 0)
    redfort = rgb(0.3, 0, 0)
   h <- hist(echantillon, breaks = 10, col = redleger, main = "Histogram with Normal Curve")
    xfit <- seq(min(echantillon), max(echantillon), length = 40)
    yfit1 <- dnorm(xfit, mean = meand2, sd = ecart_type)
    yfit1 <- yfit1 * diff(h$mids[1:2]) * length(echantillon)
    lines(xfit, yfit1, col = redfort, lwd = 2)
    yfit2 <- dnorm(xfit, sd = ecart_type)
    yfit2 <- yfit2 * diff(h$mids[1:2]) * length(echantillon)
    lines(xfit, yfit2, col = "blue", lwd = 2)
    legend("topleft", c("R<U+00E9>f<U+00E9>rence nulle", "Echantillon"),
        fill = c("blue", redfort), bty = "n", border = NA)
    dev.off()
    return(list(ecart_type = ecart_type, mean = meand2, conf_mean = conf_mean,
        conf_sd = conf_sd))
  }
```

`pilote_ttest_paired` *Etude Pilote, dans le cas "Deux Echantillons Apparies".*

Description

Cette fonction genere et affiche l'histogramme d'un pilote, pour le cas "Deux Echantillons Apparies". Elle utilise du Bootstrap (qui sélectionne 80

Usage

```
pilote_ttest_paired(npilote, meand, s1, s2, cf, runs_bs_pilote, dest_pilote)
```

Arguments

npilote	Nombre d'observation des deux echantillons de l'etude pilote.
meand	Difference de moyenne entre les deux normales permettant de generer les deux echantillons.
s1	Ecart-type de la normale permettant de generer le premier echantillon.
s2	Ecart-type de la normale permettant de generer le second echantillon.
cf	Facteur de correlation entre les deux lois normales permettant de generer les deux echantillons.
runs_bs_pilote	Nombre d'iteration de Bootstrap applique au pilote pour le calcul des intervalles de confiance.
dest_pilote	Destination de l'image contenant l'histogramme du pilote.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas.

Examples

```
function (npilote, meand, s1, s2, cf, runs_bs_pilote, dest_pilote)
{
    alpha = 0.05
    congruent = rnorm(npilote, mean = 0, sd = s1)
    a = meand
    b = cf * (s1 * s2)
    sd_eps = sqrt(s2^2 - b^2 * s1^2)
    eps = rnorm(npilote, mean = 0, sd = sd_eps)
    incongruent = a + b * congruent + eps
    mean1 = mean(congruent)
    mean2 = mean(incongruent)
    meand2 = mean2 - mean1
    ecart_type_congruent = sd(congruent)
    ecart_type_incongruent = sd(incongruent)
    cf_estime = cor(congruent, incongruent)
    conf_mean = t.test(incongruent, congruent, paired = TRUE)$conf.int
    table_sd1 = numeric(runs_bs_pilote)
    table_sd2 = numeric(runs_bs_pilote)
    for (i in 1:runs_bs_pilote) {
        n_bs = ceiling(0.8 * npilote)
        table_sd1[i] = sd((sample(congruent, n_bs, replace = T)))
        table_sd2[i] = sd((sample(incongruent, n_bs, replace = T)))
    }
    table_sd1_sorted = sort(table_sd1)
    table_sd2_sorted = sort(table_sd2)
    conf_sd1 = c(table_sd1_sorted[floor(runs_bs_pilote * alpha)],
        table_sd1_sorted[floor(runs_bs_pilote * (1 - alpha))])
    conf_sd2 = c(table_sd2_sorted[floor(runs_bs_pilote * alpha)],
        table_sd2_sorted[floor(runs_bs_pilote * (1 - alpha))])
    jpeg(dest_pilote)
    densCongruent <- density(congruent)
```

```
    densIncongruent <- density(incongruent)
    histCongruent <- hist(congruent, breaks = 10, plot = FALSE)
    histIncongruent <- hist(incongruent, breaks = 10, plot = FALSE)
    xlim <- range(histIncongruent$breaks, histCongruent$breaks)
    ylim <- range(0, histIncongruent$density, histCongruent$density)
    plot(histCongruent, xlim = xlim, ylim = ylim, col = rgb(1,
        0, 0, 0.4), xlab = "congruent", freq = FALSE, main = "Distribution")
    opar <- par(new = FALSE)
    plot(histIncongruent, xlim = xlim, ylim = ylim, xaxt = "n",
        yaxt = "n", col = rgb(0, 0, 1, 0.4), add = TRUE, freq = FALSE)
    legend("topleft", c("Congruent", "Incongruent"), fill = rgb(1:0,
        0, 0:1, 0.4), bty = "n", border = NA)
    par(opar)
    xfit1 <- seq(min(congruent), max(congruent), length = 40)
    yfit1 <- dnorm(xfit1, mean = mean1, sd = ecart_type_congruent)
    yfit1 <- yfit1 * diff(histCongruent$mids[1:2]) * length(congruent)
    lines(xfit1, yfit1, col = rgb(1, 0, 0, 0.4), lwd = 2)
    xfit2 <- seq(min(incongruent), max(incongruent), length = 40)
    yfit2 <- dnorm(xfit2, mean = mean2, sd = ecart_type_incongruent)
    yfit2 <- yfit2 * diff(histIncongruent$mids[1:2]) * length(incongruent)
    lines(xfit2, yfit2, col = rgb(0, 0, 1, 0.4), lwd = 2)
    dev.off()
    return(list(ecart_type_congruent = ecart_type_congruent,
        ecart_type_incongruent = ecart_type_incongruent, mean = meand2,
        conf_mean = conf_mean, conf_sd1 = conf_sd1, conf_sd2 = conf_sd2))
}
```

`Puissance_ap` *Fonction principale du package, cas "Deux Echantillons Apparies".*

Description

Cette fonction appelle toutes les sous-fonctions relatives au cas "Deux Echantillons Apparies". Elle genere donc un pilote et en affiche un histogramme. Ensuite, elle effectue des calculs (grace au Bootstrap et a Monte-Carlo) pour la puissance et les intervalles de confiance, et elle termine par l'affichage de la courbe puissance = f(taille d'echantillon). Des valeurs "typiques" des variables sont renseignees par default.

Usage

```
Puissance_ap(npoints = 15, npilote = 20, meand = 0.1, s1 = 0.3, s2 = 0.3, cf = 0.6, runs_bs_pilote = 10
```

Arguments

`npoints`	Nombre de points à afficher sur la graphe puissance = f(taille d'echantillon).
`npilote`	Nombre d'observations du pilote.
`meand`	Difference de moyennes entre les deux normales qui permettent de generer les echantillons.

s1	Ecart-type de la normale qui permet de generer le premier echantillon.
s2	Ecart-type de la normale qui permet de generer le second echantillon.
cf	Facteur de corrélation des normales qui permettent de generer les deux echantillons.
runs_bs_pilote	Nombre d'iterations du Bootstrap pour le calcul des intervalles de confiance.
runs_MC	Nombre d'iterations de la methode de Monte-Carlo pour les calculs de puissance.
taille_max	Taille maximum envisageable pour l'etude reelle qui se basera sur les resultats de l'etude pilote.
dest_puissance	Emplacement où enregistrer le graphe des puissances.
dest_pilote	Emplacement où enregistrer l'histogramme du pilote.
puissance	Puissance désirée pour le test d'hypothèse.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (npoints = 15, npilote = 20, meand = 0.1, s1 = 0.3,
    s2 = 0.3, cf = 0.6, runs_bs_pilote = 1000, runs_MC = 1000,
    taille_max = 100, dest_puissance, dest_pilote, puissance = NULL)
{
    alpha = 0.05
    library(gplots)
    pilote = pilote_ttest_paired(npilote, meand, s1, s2, cf,
        runs_bs_pilote, dest_pilote)
    tailles = seq(from = 20, to = taille_max, length.out = npoints)
    longueur = length(tailles)
    puissances = numeric(longueur)
    IC_low_width = numeric(longueur)
    IC_up_width = numeric(longueur)
    for (i in 1:longueur) {
        results = ttest_paired(tailles[i], runs_MC, pilote, cf)
        puissances[i] = results$Puissance_moy_hand
        IC_low_width[i] = puissances[i] - results$IC_Puissance_hand_inf
        IC_up_width[i] = results$IC_Puissance_hand_sup - puissances[i]
    }
    jpeg(dest_puissance)
    plotCI(tailles, puissances, uiw = IC_up_width, liw = IC_low_width,
        type = "o", barcol = "red")
    dev.off()
    results
    return(calcul_n(npoints, puissance, puissances, tailles))
  }
```

`Puissance_ind` *Fonction principale du package, cas "Deux Echantillons Independants".*

Description

Cette fonction appelle toutes les sous-fonctions relatives au cas "Deux Echantillons Independants". Elle genere donc un pilote et en affiche un histogramme. Ensuite, elle effectue des calculs (grace au Bootstrap et a Monte-Carlo) pour la puissance et les intervalles de confiance, et elle termine par l'affichage de la courbe puissance = f(taille d'echantillon). Des valeurs "typiques" des variables sont renseignees par default.

Usage

```
Puissance_ind(npoints = 15, npilote_congruent = 20, npilote_incongruent = 20, meand = 0.4, sd = 0.3, r
```

Arguments

`npoints`	Nombre de points à afficher sur la graphe puissance = f(taille d'echantillon).
`npilote_congruent`	Nombre d'observations du premier echantillon du pilote.
`npilote_incongruent`	Nombre d'observations du second echantillon du pilote.
`meand`	Difference de moyennes entre les deux normales qui permettent de generer les echantillons.
`sd`	Ecart-type des normales qui permettent de generer les echantillons.
`runs_bs_pilote`	Nombre d'iterations du Bootstrap pour le calcul des intervalles de confiance.
`runs_MC`	Nombre d'iterations de la methode de Monte-Carlo pour les calculs de puissance.
`taille_max`	Taille maximum envisageable pour l'etude reelle qui se basera sur les resultats de l'etude pilote.
`dest_puissance`	Emplacement où enregistrer le graphe des puissances.
`dest_pilote`	Emplacement où enregistrer l'histogramme du pilote.
`puissance`	Puissance désirée pour le test d'hypothèse.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (npoints = 15, npilote_congruent = 20, npilote_incongruent = 20,
    meand = 0.4, sd = 0.3, runs_bs_pilote = 1000, runs_MC = 1000,
    taille_max = 100, dest_puissance, dest_pilote, puissance = NULL)
{
    alpha = 0.05
    library(gplots)
    pilote = pilote_ttest_independants(npilote_congruent, npilote_incongruent,
        meand, sd, runs_bs_pilote, dest_pilote)
    tailles = seq(from = 20, to = taille_max, length.out = npoints)
    longueur = length(tailles)
    puissances = numeric(longueur)
    IC_low_width = numeric(longueur)
    IC_up_width = numeric(longueur)
    for (i in 1:longueur) {
        results = ttest_independants(tailles[i], tailles[i],
            runs_MC, pilote)
        puissances[i] = results$Puissance_moy_hand
        IC_low_width[i] = puissances[i] - results$IC_Puissance_hand_inf
        IC_up_width[i] = results$IC_Puissance_hand_sup - puissances[i]
    }
    jpeg(dest_puissance)
    plotCI(tailles, puissances, uiw = IC_up_width, liw = IC_low_width,
        type = "o", barcol = "red")
    dev.off()
    results
    return(calcul_n(npoints, puissance, puissances, tailles))
  }
```

`Puissance_un` *Fonction principale du package, cas "Un Echantillon".*

Description

Cette fonction appelle toutes les sous-fonctions relatives au cas "Un Echantillon". Elle genere donc un pilote et en affiche un histogramme. Ensuite, elle effectue des calculs (grace au Bootstrap et a Monte-Carlo) pour la puissance et les intervalles de confiance, et elle termine par l'affichage de la courbe puissance = f(taille d'echantillon). Des valeurs "typiques" des variables sont renseignees par default.

Usage

```
Puissance_un(npoints = 15, npilote = 20, meand = 0.1, sd = 0.3, runs_bs_pilote = 1000, runs_MC = 1000,
```

Arguments

`npoints`	Nombre de points à afficher sur la graphe puissance = f(taille d'echantillon).
`npilote`	Nombre d'observations du pilote.

meand	Moyenne de la normale qui permet de generer le pilote. C'est aussi l'ecart entre la moyenne de cette normale et celle d'une normale centree.
sd	Ecart-type des normales qui permettent de generer les echantillons.
runs_bs_pilote	Nombre d'iterations du Bootstrap pour le calcul des intervalles de confiance.
runs_MC	Nombre d'iterations de la methode de Monte-Carlo pour les calculs de puissance.
taille_max	Taille maximum envisageable pour l'etude reelle qui se basera sur les resultats de l'etude pilote.
dest_puissance	Emplacement où enregistrer le graphe des puissances.
dest_pilote	Emplacement où enregistrer l'histogramme du pilote.
puissance	Puissance désirée pour le test d'hypothèse.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (npoints = 15, npilote = 20, meand = 0.1, sd = 0.3,
    runs_bs_pilote = 1000, runs_MC = 1000, taille_max = 100,
    dest_puissance, dest_pilote, puissance = NULL)
{
    alpha = 0.05
    library(gplots)
    pilote = pilote_ttest_normal(npilote, meand, sd, runs_bs_pilote,
        dest_pilote)
    tailles = seq(from = 20, to = taille_max, length.out = npoints)
    longueur = length(tailles)
    puissances = numeric(longueur)
    IC_low_width = numeric(longueur)
    IC_up_width = numeric(longueur)
    for (i in 1:longueur) {
        results = ttest_normal(tailles[i], runs_MC, pilote)
        puissances[i] = results$Puissance_moy_hand
        IC_low_width[i] = puissances[i] - results$IC_Puissance_hand_inf
        IC_up_width[i] = results$IC_Puissance_hand_sup - puissances[i]
    }
    jpeg(dest_puissance)
    plotCI(tailles, puissances, uiw = IC_up_width, liw = IC_low_width,
        type = "o", barcol = "red")
    dev.off()
    results
    return(calcul_n(npoints, puissance, puissances, tailles))
  }
```

`ttest_independants` *Calcul de la puissance du t-Test, dans le cas "Deux Echantillons Independants".*

Description

Cette fonction effectue les calculs de puissance dans le cas "Deux Echantillons Independants", notamment en faisant appel à la fonction MC_ind.

Usage

```
ttest_independants(n1, n2, runs, pilote)
```

Arguments

`n1`	Nombre d'observations du premier echantillon.
`n2`	Nombre d'observations du second echantillon.
`runs`	Nombre d'iteration de la methode de Monte-Carlo.
`pilote`	Etude pilote prealable.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (n1, n2, runs, pilote)
{
    mean = pilote$mean
    conf_mean = pilote$conf_mean
    meaninf = conf_mean[1]
    meansup = conf_mean[2]
    ecart_type = pilote$ecart_type
    conf_sd = pilote$conf_sd
    sdinf = conf_sd[1]
    sdsup = conf_sd[2]
    MC_inf = MC_ind(n1, n2, runs, meaninf, sdsup)
    MC_sup = MC_ind(n1, n2, runs, meansup, sdinf)
    MC_moy = MC_ind(n1, n2, runs, mean, ecart_type)
    IC_Puissance_model = c(MC_inf$p5_hand, MC_sup$p5_hand)
    IC_Puissance_hand = c(MC_inf$p5_model, MC_sup$p5_model)
    IC_Puissance_package = c(MC_inf$p5_package, MC_sup$p5_package)
    Puissance_moy_hand = MC_moy$p5_hand
    Puissance_moy_model = MC_moy$p5_model
    Puissance_moy_package = MC_moy$p5_package
   results = data.frame(n1 = n1, n2 = n2, runs = runs, Puissance_moy_hand = Puissance_moy_hand,
      IC_Puissance_hand_inf = IC_Puissance_hand[1], IC_Puissance_hand_sup = IC_Puissance_hand[2],
      Puissance_moy_model = Puissance_moy_model, IC_Puissance_model_inf = IC_Puissance_model[1],
```

```
    IC_Puissance_model_sup = IC_Puissance_model[2], Puissance_moy_package = MC_moy$p5_package,
    IC_Puissance_package_inf = IC_Puissance_package[1], IC_Puissance_package_sup = IC_Puissance_package[2])
  return(results)
}
```

`ttest_normal` *Calcul de la puissance du t-Test, dans le cas "Un Echantillon".*

Description

Cette fonction effectue les calculs de puissance dans le cas "Un Echantillon", notamment en faisant appel à la fonction MC.

Usage

```
ttest_normal(n, runs, pilote)
```

Arguments

`n`	Nombre d'observations de l'echantillon.
`runs`	Nombre d'iteration de la methode de Monte-Carlo.
`pilote`	Etude pilote prealable.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (n, runs, pilote)
{
    mean = pilote$mean
    conf_sd = pilote$conf_sd
    conf_mean = pilote$conf_mean
    ecart_type = pilote$ecart_type
    meaninf = conf_mean[1]
    meansup = conf_mean[2]
    sdinf = conf_sd[1]
    sdsup = conf_sd[2]
    MC_inf = MC(n, runs, meaninf, sdsup)
    MC_sup = MC(n, runs, meansup, sdinf)
    MC_moy = MC(n, runs, mean, ecart_type)
    IC_Puissance_model = c(MC_inf$p5_model, MC_sup$p5_model)
    IC_Puissance_hand = c(MC_inf$p5_hand, MC_sup$p5_hand)
    IC_Puissance_package = c(MC_inf$p5_package, MC_sup$p5_package)
    Puissance_moy_hand = MC_moy$p5_hand
    Puissance_moy_model = MC_moy$p5_model
    Puissance_moy_package = MC_moy$p5_package
    results = data.frame(n = n, runs = runs, Puissance_moy_hand = Puissance_moy_hand,
```

```
    IC_Puissance_hand_inf = IC_Puissance_hand[1], IC_Puissance_hand_sup = IC_Puissance_hand[2],
    Puissance_moy_model = Puissance_moy_model, IC_Puissance_model_inf = IC_Puissance_model[1],
    IC_Puissance_model_sup = IC_Puissance_model[2], Puissance_moy_package = MC_moy$p5_package,
    IC_Puissance_package_inf = IC_Puissance_package[1], IC_Puissance_package_sup = IC_Puissance_package[2])
  return(results)
}
```

`ttest_paired` *Calcul de la puissance du t-Test, dans le cas "Deux Echantillons Apparies".*

Description

Cette fonction effectue les calculs de puissance dans le cas "Deux Echantillons Apparies", notamment en faisant appel à la fonction MC_ap.

Usage

```
ttest_paired(n, runs, pilote, cf)
```

Arguments

`n`	Nombre d'observations des echantillons.
`runs`	Nombre d'iteration de la methode de Monte-Carlo.
`pilote`	Etude pilote prealable.
`cf`	Facteur de corrélation des normales qui ont genere les echantillons.

Author(s)

Bonjean Gregoire, Crepin Baptiste & Lair Thomas

Examples

```
function (n, runs, pilote, cf)
{
    mean = pilote$mean
    conf_mean = pilote$conf_mean
    meaninf = conf_mean[1]
    meansup = conf_mean[2]
    ecart_type1 = pilote$ecart_type_congruent
    ecart_type2 = pilote$ecart_type_incongruent
    conf_sd1 = pilote$conf_sd1
    conf_sd2 = pilote$conf_sd2
    sd1inf = conf_sd1[1]
    sd1sup = conf_sd1[2]
    sd2inf = conf_sd2[1]
    sd2sup = conf_sd2[2]
    MC_inf = MC_ap(n, runs, meaninf, sd1sup, sd2sup, cf)
```

```
  MC_sup = MC_ap(n, runs, meansup, sd1inf, sd2inf, cf)
  MC_moy = MC_ap(n, runs, mean, ecart_type1, ecart_type2, cf)
  IC_Puissance_model = c(MC_inf$p5_model, MC_sup$p5_model)
  IC_Puissance_hand = c(MC_inf$p5_hand, MC_sup$p5_hand)
  IC_Puissance_package = c(MC_inf$p5_package, MC_sup$p5_package)
  Puissance_moy_hand = MC_moy$p5_hand
  Puissance_moy_model = MC_moy$p5_model
  Puissance_moy_package = MC_moy$p5_package
  results = data.frame(n = n, runs = runs, Puissance_moy_hand = Puissance_moy_hand,
    IC_Puissance_hand_inf = IC_Puissance_hand[1], IC_Puissance_hand_sup = IC_Puissance_hand[2],
    Puissance_moy_model = Puissance_moy_model, IC_Puissance_model_inf = IC_Puissance_model[1],
    IC_Puissance_model_sup = IC_Puissance_model[2], Puissance_moy_package = MC_moy$p5_package,
    IC_Puissance_package_inf = IC_Puissance_package[1], IC_Puissance_package_sup = IC_Puissance_package[2])
  return(results)
}
```

Index