Modèle de Hiérarchie Cognitive

SCAIA Matteo - HACHEM REDA Riwa - GILLET Louison

2025-05-24

Chargement des bibliothèques et données

```
library(readxl)
library(xtable)
library(ggplot2)
library(tidyr)
library(dplyr)

donnees <- read_excel('../data/Exp_rience.xlsx')
donneesQ = donnees[1:63,7:21]
```

```
questions <- c("Q00_16-20 Arad & Rubinstein", "Q00_Mode du groupe", "Q00_ChatGPT",
               "Q00_Prime de 20","Q01_12-20" ,"Q02_Alaoui & Penta", "Q03_Alaoui & Penta
               "Q04_Goeree moderate", "Q05_Goeree extreme", "Q06_Cycle A", "Q07_Cycle C"
               "Q08_Pan A", "Q09_Bear C", "Q10_long pan A", "Q11_long pan C")

actions_par_question <- list(
  "Q00_16-20 Arad & Rubinstein" = 16:20,
  "Q00_Mode du groupe" = 16:20,
  "Q00_ChatGPT" = 16:20,
  "Q00_Prime de 20" = 16:20,
  "Q01_12-20" = c(12,14,16,18,20),
  "Q02_Alaoui & Penta" = c(12,14,16,18,20),
  "Q03_Alaoui & Penta plus" = c(12,14,16,18,20),
  "Q04_Goeree moderate" = c(14,12,18,16,20),
  "Q05_Goeree extreme" = c(18,16,14,12,20),
  "Q06_Cycle A" = c(12, 14, 16, 18, 20),
  "Q07_Cycle C" = c(12, 14, 16, 18, 20),
  "Q08_Pan A" = c(12, 14, 16, 18, 20),
  "Q09_Bear C" = c(12, 14, 16, 18, 20),
  "Q10_long pan A" = c(12, 14, 16, 18, 20),
  "Q11_long pan C" = c(12, 14, 16, 18, 20)
)

bonus_par_question <- list(
  "Q00_16-20 Arad & Rubinstein" = list(gauche = 10, egal = 0),
  "Q00_Mode du groupe" = list(gauche = 10, egal = 0),
  "Q00_ChatGPT" = list(gauche = 10, egal = 0),
  "Q00_Prime de 20" = list(gauche = 20, egal = 0),
  "Q01_12-20" = list(gauche = 20, egal = 0),
  "Q02_Alaoui & Penta" = list(gauche = 20, egal = 10),
  "Q03_Alaoui & Penta plus" = list(gauche = 40, egal = 10),
  "Q04_Goeree moderate" = list(gauche = 20, egal = 0),
  "Q05_Goeree extreme" = list(gauche = 20, egal = 0),
```

```
  "Q06_Cycle A" = list(gauche = 20, egal = 0),
  "Q07_Cycle C" = list(gauche = 20, egal = 0),
  "Q08_Pan A" = list(gauche = 20, egal = 0),
  "Q09_Bear C" = list(gauche = 20, egal = 0),
  "Q10_long pan A" = list(gauche = 20, egal = 0),
  "Q11_long pan C" = list(gauche = 20, egal = 0)
)
```

Algorithme

Fonction stratégie CH bruité

Description : Calcule la distribution de probabilité d'un joueur de niveau k dans un modèle CH (bruité), en supposant que les joueurs de niveau k anticipent les niveaux inférieurs.

```
ch_strategy <- function(mu, k, actions, bonus_gauche, bonus_egal) {

  # Absence de raisonnement stratégique : proba uniforme sur les actions
  if (k == 0) {
    probs <- rep(1 / length(actions), length(actions))
  }

  # k > 0 : simuler une anticipation
  else {

    ## Appel récursif de la fonction pour les niveaux < k
    ## Renvoie une distribution de probabilités pour chaque niveau
    lower_levels <- lapply(0:(k-1), function(l) ch_strategy(mu, l, actions, bonus_gauche

    ## Moyenne des distributions des probabilités de lower_levels
    ## Représente la croyance du joueur k sur le comportement MOYEN des joueurs < k
    avg_lower <- Reduce("+", lower_levels) / length(lower_levels)

    ## Gain espéré en fonction des stratégies inférieures
    expected_payoff <- sapply(seq_along(actions), function(a_idx) {
      a <- actions[a_idx]
      bonus_left <- if (a_idx < length(actions)) avg_lower[a_idx + 1] * bonus_gauche els
      bonus_same <- avg_lower[a_idx] * bonus_egal
      gain <- a + bonus_left + bonus_same
      return(gain)
    })

    ## Introduction du bruit
    probs <- exp(expected_payoff / mu)
```

```
    ## Normalisation -> proba
    probs <- probs / sum(probs)
  }
  return(probs)
}
```

Fonction log-vraisemblance CH bruité

Description : Calcule la log-vraisemblance des réponses d'un individu dans un modèle CH (bruité), pour pouvoir ensuite estimer les paramètres μ et s.

```
log_likelihood_ch <- function(mu, s, reponses, questions, actions_map, k_max = 4) {

  # Initialisation
  total_loglik <- 0

  # Répartition des niveaux de raisonnement selon une loi de Poisson
  poisson_weights <- dpois(0:k_max, lambda = s)

  # Normalisation -> proba
  poisson_weights <- poisson_weights / sum(poisson_weights)

  # Boucle sur les réponses
  for (j in seq_along(reponses)) {

    ## Attribution des réponses, des actions et des bonus pour une question
    r <- reponses[j]
    q <- questions[j]
    actions <- actions_map[[q]]
    bonus <- bonus_par_question[[q]]

    ## Calcul des stratégies pour chaque niveau de raisonnement k (distribution de pro
    ch_levels <- lapply(0:k_max, function(k) {
      ch_strategy(mu, k, actions, bonus$gauche, bonus$egal)
    })

    if (!is.na(r)) {
      idx <- which(actions == r)
      if (length(idx) == 1) {

        ### Calcul de la proba du choix observé
        p_r <- sum(vapply(0:k_max, function(k) poisson_weights[k+1] * ch_levels[[k+1]][
      }
      else {
```

```
        p_r <- 0
      }

      ## Vérifier existence et validité de la proba
      if (p_r > 0 && !is.na(p_r)) {
        total_loglik <- total_loglik + log(p_r)
      }
      ### Sinon, forte pénalité
      else {
        total_loglik <- total_loglik - 1e6
      }
    }
  }
  return(total_loglik)
}
```

Fonctions score de Brier

Description : Calcule le score de Brier pour une question

```
calcul_brier_score <- function(probs, reponse_obs, actions) {
  idx_obs <- which(actions == reponse_obs)
  brier <- sum((probs - ifelse(seq_along(actions) == idx_obs, 1, 0))^2)
  return(brier)
}
```

Description : Calcule un score de Brier total sur l'ensemble des réponses aux questions selon un modèle de Hiérarchie Cognitive (bruité).

```
brier_score_total <- function(mu, s, reponses, questions, actions_map, k_max = 4) {

  # Initialisation
  total_brier <- 0

  # Répartition des niveaux de raisonnement selon une loi de Poisson
  poisson_weights <- dpois(0:k_max, lambda = s)

  # Normalisation -> proba
  poisson_weights <- poisson_weights / sum(poisson_weights)

  # Boucle sur les réponses
  for (j in seq_along(reponses)) {

    ## Attribution des réponses, des actions et des bonus pour une question
    r <- reponses[j]
```

```
    q <- questions[j]
    actions <- actions_map[[q]]
    bonus <- bonus_par_question[[q]]

    ## Calcul des stratégies pour chaque niveau de raisonnement k (distribution de pro
    if (is.na(r) || !(r %in% actions)) next
    ch_levels <- lapply(0:k_max, function(k) {
      ch_strategy(mu, k, actions, bonus$gauche, bonus$egal)
    })

    # Moyenne des stratégies -> donne la distribution des probas
    mix_prob <- Reduce("+", Map("*", poisson_weights, ch_levels))

    # Calcul du score à partir de la fonction calcul_brier_score et des probas obtenue
    total_brier <- total_brier + calcul_brier_score(mix_prob, r, actions)
  }
  return(total_brier/15)
}
```

Estimation CH bruité

Description : Pour chaque individu, on cherche les paramètres μ et s qui maximisent la log-vraisemblance

```
# Initialisation du tableau des résultats
resultats_ch <- data.frame()

# Boucle sur les individus
for (i in 1:(nrow(donnees)-1)) {
  individu <- donnees[i, ]

  # Ne garder que la partie droite pour les écritures "0:20" de la table
  reponses <- sapply(questions, function(q) {
    rep <- individu[[q]]
    if (!is.na(rep) && grepl(":", rep)) {
      val <- as.numeric(trimws(strsplit(rep, ":")[[1]][2]))
    } else {
      val <- as.numeric(trimws(rep))
    }
    return(val)
  })

  tryCatch({
    opt <- optim(
```

```
      par = c(mu = 1, s = 1.5),
      fn = function(par) -log_likelihood_ch(par[1], par[2], reponses, questions, actions
      method = "L-BFGS-B",
      ## Contraintes : bornes sur μ et s
      lower = c(0.01, 0.01), upper = c(10, 10)
    )

    # Récupération des résultats
    mu_estime <- round(opt$par[1],3)
    s_estime <- round(opt$par[2],3)
    max_vraisemblance <- round(-opt$value,3)  # - pour revenir au max
    brier_score <- round(brier_score_total(mu_estime, s_estime, reponses, questions, ac

    # Ajout des résultats au tableau
    resultats_ch <- rbind(resultats_ch, data.frame(id = i,
                                                   mu = mu_estime,
                                                   s = s_estime,
                                                   VraisemblanceMax = max_vraisemblance,
                                                   Brier = brier_score))
  },

  # Gestion des erreurs : ligne de NA si problème
  error = function(e) {
    resultats_ch <- rbind(resultats_ch, data.frame(id = i, mu = NA, s = NA, Vraisemblanc
    cat("Erreur pour l'individu", i, ":", e$message, "\n")
  })
}
```

```
print(resultats_ch)
```

```
##       id     mu      s VraisemblanceMax Brier
## mu     1  0.080  0.907          -11.441 0.397
## mu1    2  2.952 10.000          -19.095 0.656
## mu2    3 10.000  0.010          -24.166 0.801
## mu3    4  0.125  2.404          -13.816 0.442
## mu4    5  0.059  2.765          -12.326 0.457
## mu5    6 10.000  0.010          -22.547 0.747
## mu6    7 10.000  0.010          -24.152 0.800
## mu7    8  0.059  1.056          -15.290 0.551
## mu8    9  0.058  2.469          -14.461 0.549
## mu9   10  0.077  0.779          -20.606 0.699
## mu10  11  5.896  0.770          -23.797 0.791
## mu11  12  0.746  3.141          -19.427 0.702
## mu12  13 10.000  0.010          -24.156 0.800
## mu13  14  0.087  1.345          -16.198 0.592
```

```
## mu14 15  2.207  1.107          -20.525 0.709
## mu15 16 10.000  0.010          -20.929 0.693
## mu16 17  0.058  0.959          -17.088 0.596
## mu17 18  0.085  0.888          -14.400 0.494
## mu18 19  0.072  0.851          -14.364 0.485
## mu19 20  0.316  0.182          -23.466 0.780
## mu20 21  0.661  0.761          -20.673 0.699
## mu21 22 10.000  0.010          -24.143 0.800
## mu22 23  0.060  0.410          -21.686 0.720
## mu23 24 10.000  0.010          -24.142 0.800
## mu24 25  0.058  4.486          -13.991 0.588
## mu25 26  0.072  0.786          -19.537 0.661
## mu26 27  2.290  5.254          -17.824 0.617
## mu27 28 10.000  0.010          -20.940 0.694
## mu28 29  1.335  4.721          -19.140 0.731
## mu29 30  0.275  1.203          -17.335 0.606
## mu30 31  0.058  0.572          -21.178 0.708
## mu31 32 10.000  1.289          -23.979 0.798
## mu32 33  6.947  2.544          -23.474 0.790
## mu33 34 10.000  1.289          -23.979 0.798
## mu34 35  1.208  3.560          -19.407 0.676
## mu35 36 10.000  0.010          -24.171 0.801
## mu36 37  0.450  4.566          -16.288 0.658
## mu37 38  1.512  2.518          -19.860 0.630
## mu38 39  0.059  2.702          -12.752 0.369
## mu39 40  2.401  1.157          -20.689 0.712
## mu40 41 10.000  0.010          -24.167 0.801
## mu41 42 10.000  0.010          -24.142 0.800
## mu42 43 10.000  0.114          -24.140 0.800
## mu43 44  0.138  0.010          -24.145 0.800
## mu44 45  1.786  5.671          -21.883 0.742
## mu45 46  0.096  1.195          -15.291 0.554
## mu46 47 10.000  0.010          -24.144 0.800
## mu47 48 10.000  0.010          -24.159 0.800
## mu48 49  0.058  1.581          -17.072 0.617
## mu49 50  0.987  1.513          -22.316 0.731
## mu50 51  1.784  1.149          -21.122 0.712
## mu51 52 10.000  0.010          -24.146 0.800
## mu52 53  0.206  0.187          -23.445 0.778
## mu53 54  1.007  5.157          -18.182 0.694
## mu54 55  0.244  3.053          -20.815 0.633
## mu55 56  0.066  0.683          -14.947 0.485
## mu56 57  1.951 10.000          -21.621 0.703
## mu57 58  0.058  1.292          -15.638 0.577
## mu58 59  1.052  4.009          -16.729 0.602
```

```
## mu59 60  0.674  1.111          -19.587 0.675
## mu60 61  0.075  0.852          -14.884 0.512
## mu61 62 10.000  1.623          -23.900 0.797
## mu62 63  0.069  0.753          -13.423 0.440
```

Estimation par question

Description : Calcule un score de Brier total sur l'ensemble des réponses aux questions selon un modèle de Hiérarchie Cognitive (bruité)

```r
brier_score_total <- function(mu, s, reponses, questions, actions_map, k_max = 4) {

  # Initialisation
  total_brier <- 0

  # Répartition des niveaux de raisonnement selon une loi de Poisson
  poisson_weights <- dpois(0:k_max, lambda = s)

  # Normalisation -> proba
  poisson_weights <- poisson_weights / sum(poisson_weights)

  valid_n <- 0

  # Boucle sur les réponses
  for (j in seq_along(reponses)) {

    ## Attribution des réponses, des actions et des bonus pour une question
    r <- reponses[j]
    q <- questions[j]
    actions <- actions_map[[q]]
    bonus <- bonus_par_question[[q]]

    ## Calcul des stratégies pour chaque niveau de raisonnement k (distribution de pro
    if (is.na(r) || !(r %in% actions)) next
    ch_levels <- lapply(0:k_max, function(k) {
      ch_strategy(mu, k, actions, bonus$gauche, bonus$egal)
    })
    valid_n <- valid_n + 1


    # Moyenne des stratégies -> donne la distribution des probas
    mix_prob <- Reduce("+", Map("*", poisson_weights, ch_levels))

    # Calcul du score à partir de la fonction calcul_brier_score et des probas obtenue
    total_brier <- total_brier + calcul_brier_score(mix_prob, r, actions)
```

```
  }
  return(total_brier/valid_n)
}

# Initialisation ----
resultats_ch_par_question <- data.frame()

# Boucle sur chaque question ----
for (q in questions) {
  actions <- actions_par_question[[q]]
  bonus <- bonus_par_question[[q]]

  reponses <- sapply(donnees[[q]], function(rep) {
    if (!is.na(rep) && grepl(":", rep)) {
      as.numeric(trimws(strsplit(rep, ":")[[1]][2]))
    } else {
      as.numeric(trimws(rep))
    }
  })

  tryCatch({
    # Optimisation des paramètres mu et s
    opt <- optim(
      par = c(mu = 1.5, s = 1.5),
      fn = function(par) -log_likelihood_ch(par[1], par[2],
                                            reponses = reponses,
                                            questions = rep(q, length(reponses)),
                                            actions_map = actions_par_question),
      method = "L-BFGS-B",
      lower = c(0.01, 0.01), upper = c(10, 10)
    )

    # Résultats estimés
    mu_estime <- round(opt$par[1], 3)
    s_estime <- round(opt$par[2], 3)
    loglik <- round(-opt$value, 3)
    brier <- round(brier_score_total(mu_estime, s_estime,
                                     reponses = reponses,
                                     questions = rep(q, length(reponses)),
                                     actions_map = actions_par_question), 3)

    # Ajout au tableau
    resultats_ch_par_question <- rbind(resultats_ch_par_question,
                                       data.frame(Question = q,
```

```
                                                mu = mu_estime,
                                                s = s_estime,
                                                LogVraisemblance = loglik,
                                                Brier = brier))

  }, error = function(e) {
    resultats_ch_par_question <- rbind(resultats_ch_par_question,
                                       data.frame(Question = q,
                                                mu = NA, s = NA,
                                                LogVraisemblance = NA,
                                                Brier = NA))
    cat("Erreur pour la question", q, ":", e$message, "\n")
  })
}

# Affichage
print(resultats_ch_par_question)
```

```
##                          Question     mu      s LogVraisemblance Brier
## mu   Q00_16-20 Arad & Rubinstein  0.730  0.872          -91.159 0.742
## mu1           Q00_Mode du groupe  3.875 10.000          -99.211 0.786
## mu2                  Q00_ChatGPT  2.424 10.000          -96.088 0.768
## mu3              Q00_Prime de 20  1.333  0.357          -99.766 0.789
## mu4                    Q01_12-20  1.571  0.494          -98.954 0.785
## mu5           Q02_Alaoui & Penta  0.041  0.564          -89.265 0.706
## mu6      Q03_Alaoui & Penta plus  3.454  0.258         -100.727 0.796
## mu7          Q04_Goeree moderate  0.121  0.303          -95.987 0.759
## mu8           Q05_Goeree extreme  1.376  0.744          -90.690 0.721
## mu9                  Q06_Cycle A  7.540  5.198          -98.964 0.787
## mu10                 Q07_Cycle C  8.953 10.000          -99.786 0.791
## mu11                   Q08_Pan A  1.791  0.711          -97.778 0.779
## mu12                  Q09_Bear C  5.051  1.234          -99.015 0.785
## mu13              Q10_long pan A  1.969  1.126          -86.495 0.744
## mu14              Q11_long pan C 10.000  1.009          -94.580 0.798
```

Génération des graphiques

```
library(ggplot2)
library(ggpubr)
library(readxl)
plots <- list()

for (q in 1:15) {
```

```r
actions <- actions_par_question[[q]]
bonus <- bonus_par_question[[q]]

reponses <- sapply(donneesQ[[q]], function(rep) {
  if (!is.na(rep) && grepl(":", rep)) {
    as.numeric(trimws(strsplit(rep, ":")[[1]][2]))
  } else {
    as.numeric(trimws(rep))
  }
})

tryCatch({
  opt <- optim(
    par = c(mu = 1.5, s = 1.5),
    fn = function(par) -log_likelihood_ch(par[1], par[2],
                                          reponses = reponses,
                                          questions = rep(q, length(reponses)),
                                          actions_map = actions_par_question),
    method = "L-BFGS-B",
    lower = c(0.01, 0.01), upper = c(10, 10)
  )

  mu <- opt$par[1]
  s <- opt$par[2]
  k_max <- 4

  poids_k <- dpois(0:k_max, lambda = s)
  poids_k <- poids_k / sum(poids_k)

  strats_k <- lapply(0:k_max, function(k) {
    ch_strategy(mu, k, actions, bonus$gauche, bonus$egal)
  })

  strategie_ch <- Reduce(`+`, Map(`*`, poids_k, strats_k))

  df_pred <- data.frame(
    Action = factor(actions, levels = actions),
    Valeur = strategie_ch,
    Type = "Prévu"
  )
  Q  <- donneesQ[[q]]
  Qr <- table(Q) / length(Q)
  df_obs <- data.frame(
    Action = factor(actions, levels = actions),
```

```
      Valeur = as.numeric(Qr),
      Type = "Observé"
    )
    df_combined <- rbind(df_pred, df_obs)

    p <- ggplot(df_combined, aes(x = Action, y = Valeur, fill = Type)) +
      geom_bar(stat = "identity", position = "dodge") +
      labs(title = paste("Question ", q+1),
           x = "Action possible", y = "Fréquence") +
      scale_fill_manual(values = c("Observé" = "black", "Prévu" = "lightgrey")) +
      theme_minimal(base_size = 10) +
      theme(axis.text.x = element_text(angle = 45, hjust = 1),
            legend.position = "none")

    plots[[length(plots) + 1]] <- p

  }, error = function(e) {
    cat("Erreur lors du traitement de la question", q, ":", e$message, "\n")
  })
}

# Supprimer l'ancien PDF s'il existe
if (file.exists("comparaison_CH_5x3.pdf")) file.remove("comparaison_CH_5x3.pdf")

# Créer un nouveau PDF
pdf("comparaison_CH_5x3.pdf", width = 20, height = 12)
ggarrange(plotlist = plots, ncol = 5, nrow = 3)
dev.off()
```

```
## pdf
##   2
```