Obtaining Joint Distribution from Conditional with Rectilinear Restrictions

Fan Wang

2021-03-28

Contents

1 Obtaining Joint Distribution from Conditional with Rectilinear Restrictions

Go to the **RMD**, **R**, **PDF**, or **HTML** version of this file. Go back to fan's REconTools Package, R Code Examples Repository (bookdown site), or Intro Stats with R Repository (bookdown site).

1.1 2 by 2 Joint from Marginal Probability Mass Functions

We know the $P(U|E)$, the probability of unemployment by educational types. I also know $P(U|A)$, the probability of unemployment by age groups. Additionally, I also know the $P(E, A)$, the mass of individuals at discrete (E, A) combinations. What is the $P(U|E, A)$, the probability of unemployment by age and education groups?

1.1.1 Generate Data Structure

Generate the data structure. First, a function that generates random joint probabilities of mass at different points. This data-structure will work for 2 by 2, 3 by 3, or 4 by 4 problems.

```
ffi_gen_rand_joint_mass <- function(it_seed = 123,
                                    it_Q = 2, it_M = 2,
                                    bl_verbose = FALSE) {

  # set random seed
```

```
  set.seed(it_seed)

  # Generate prob list
  ls_2d <- vector(mode = "list", length = it_Q*it_M)
  dim(ls_2d) <- c(it_Q, it_M)

  # Random joint mass
  ar_rand <- runif(it_Q*it_M)
  ar_rand <- ar_rand/sum(ar_rand)

  # Fill with values
  it_ctr <- 0
  for (it_Q_ctr in seq(1,it_Q)) {
    for (it_M_ctr in seq(1,it_M)) {
      # linear index
      ls_2d[[it_M_ctr, it_Q_ctr]] <- ar_rand[(it_Q_ctr-1)*it_M+it_M_ctr]
    }
  }

  # Replace row names, note rownames does not work
  dimnames(ls_2d)[[1]] <- paste0('E',seq(1,it_M))
  dimnames(ls_2d)[[2]] <- paste0('A',seq(1,it_Q))

  # rename
  ls_prob_joint_E_A <- ls_2d
  mt_prob_joint_E_A <- matrix(unlist(ls_prob_joint_E_A), ncol=it_M, byrow=F)

  if (bl_verbose) {
    print('ls_prob_joint_E_A')
    print(ls_prob_joint_E_A)
    print(mt_prob_joint_E_A)
  }

  # return
  return(list(mt_prob_joint_E_A=mt_prob_joint_E_A,
              ls_prob_joint_E_A=ls_prob_joint_E_A))
}
```

Second, given joint mass at different points, generate conditional unemployment probabilities along each discrete variable's values that works for 2 by 2 case. We have two unique levels for E and for A separately, so these are four points where there is mass. Generate random mass at these points. Then generate given these four additional mass points: $F = P(A_1|E_1)$, $B = P(A_1|E_2)$, $C = P(E_1|A_1)$, $D = P(E_1|A_2)$. Use code from fs_lst_basics.

```
ffi_gen_condi_unemploy_prob_2by2 <- function(mt_prob_joint_E_A,
                                             fl_alpha = 0.25, fl_beta = 0.05,
                                             fl_gamma = 0.31, fl_delta = 0.50,
                                             bl_verbose = FALSE) {

  # From joint probability, generate conditional probabilities
  fl_F <- mt_prob_joint_E_A[1,1]/sum(mt_prob_joint_E_A[1,])
  fl_B <- mt_prob_joint_E_A[2,1]/sum(mt_prob_joint_E_A[2,])
  fl_C <- mt_prob_joint_E_A[1,1]/sum(mt_prob_joint_E_A[,1])
  fl_D <- mt_prob_joint_E_A[1,2]/sum(mt_prob_joint_E_A[,2])
```

```
  if (bl_verbose) {
    print(paste0('fl_F=', fl_F, ',fl_B=',fl_B,',fl_C=',fl_C,',fl_D=',fl_D))
  }

  # Also generate random W X Y Z
  # ar_b <- runif(4)

  # fl_Delta_A_true <- 0.05
  # fl_Delta_E_true <- 0.11

  # fl_alpha_true <- 0.25
  # fl_beta_true <- fl_alpha_true + fl_Delta_A_true
  # fl_gamma_true <- fl_alpha_true + fl_Delta_E_true
  # fl_delta_true <- fl_alpha_true + fl_Delta_E_true + fl_Delta_A_true

  # fl_beta_true <- 0.31
  # fl_gamma_true <- 0.50
  # fl_delta_true <- 0.16

  fl_W <- fl_alpha*fl_F + fl_beta*(1-fl_F)
  fl_X <- fl_gamma*fl_B + fl_delta*(1-fl_B)
  fl_Y <- fl_alpha*fl_C + fl_gamma*(1-fl_C)
  fl_Z <- fl_beta*fl_D + fl_delta*(1-fl_D)
  fl_V <- mt_prob_joint_E_A[1,1]*fl_alpha_true +
    mt_prob_joint_E_A[1,2]*fl_beta_true +
    mt_prob_joint_E_A[2,1]*fl_gamma_true +
    mt_prob_joint_E_A[2,2]*fl_delta_true
  ar_b <- c(fl_W, fl_X, fl_Y, fl_Z)

  if (bl_verbose) {
    print(paste0('ar_b=',ar_b))
    print(paste0('fl_V=',fl_V))
  }

  # return
  return(list(F=fl_F, B=fl_B, C=fl_C, D=fl_D,
    W=fl_W, X=fl_X, Y=fl_Y, Z=fl_Z, V=fl_V))
}
```

1.1.2 Unrestricted Joint 2 by 2 Distribution

Suppose there are two unique states for E and A. For example, suppose we know the unemployment probability for better or worse educated, and also for low and high age groups. We also know the proportion of people who are in each one of the four cells (regardless of unemployment or not). We also know the aggregate proportion of people in the population that is unemployed.

We want to know the joint probability of unemployment for the four types: both better educated and lower age, better educated and higher age, worse educated and lower age, and worse educated and higher age. Then:

$$
\begin{aligned}
P(U|E_1) &= P(U|A_1, E_1) \cdot P(A_1|E_1) + P(U|A_2, E_1) \cdot P(A_2|E_1) \\
P(U|E_2) &= P(U|A_1, E_2) \cdot P(A_1|E_2) + P(U|A_2, E_2) \cdot P(A_2|E_2) \\
P(U|A_1) &= P(U|A_1, E_1) \cdot P(E_1|A_1) + P(U|A_1, E_2) \cdot P(E_2|A_1) \\
P(U|A_2) &= P(U|A_2, E_1) \cdot P(E_1|A_2) + P(U|A_2, E_2) \cdot P(E_2|A_2)
\end{aligned}
$$

And, we also have the extra equation:

$$
P(U) = P(U|A_1, E_1) \cdot P(E_1, A_1) + P(U|A_2, E_1) \cdot P(E_2, A_1) + P(U|A_1, E_2) \cdot P(E_1, A_2) + P(U|A_2, E_2) \cdot P(E_2, A_2)
$$

We know $P(A|E)$, and we know $P(U|E)$ as well as $P(U|A)$. Let Roman letter represent what we know, and greek letters represent what we do not know, then we have four equations and four unknowns in a linear system. We use the letter F because A and E are taken.

$$
\begin{aligned}
W &= \alpha F + \beta(1 - F) \\
X &= \gamma B + \delta(1 - B) \\
Y &= \alpha C + \gamma(1 - C) \\
Z &= \beta D + \delta(1 - D)
\end{aligned}
$$

And we have also:

$$
V = \alpha P(E_1, A_1) + \beta P(E_2, A_1) + \gamma P(E_1, A_2) + \delta P(E_2, A_2)
$$

There are no unique solutions for the linear system. When we write the linear system above in matrix form as shown below, the Ω coefficient matrix is not full rank.

$$
\begin{bmatrix}
F & (1 - F) & 0 & 0 \\
0 & 0 & B & (1 - B) \\
C & 0 & (1 - C) & 0 \\
0 & D & 0 & (1 - D)
\end{bmatrix}
\cdot
\begin{bmatrix}
\alpha \\ \beta \\ \gamma \\ \delta
\end{bmatrix}
=
\begin{bmatrix}
W \\ X \\ Y \\ Z
\end{bmatrix}
$$

$$
\Omega \cdot \mathbb{X} = b
$$

Get data from the data generation functions created prior:

```
# JOint mass of population at all cells
ls_ffi_rand_joint_mass <- ffi_gen_rand_joint_mass(it_seed = 123, it_Q = 2, it_M = 2, bl_verbose = TRUE)
```

```
## [1] "ls_prob_joint_E_A"
##    A1        A2
## E1 0.1214495 0.1727188
## E2 0.3329164 0.3729152
##           [,1]      [,2]
## [1,] 0.1214495 0.1727188
## [2,] 0.3329164 0.3729152
```

```
mt_prob_joint_E_A <- ls_ffi_rand_joint_mass$mt_prob_joint_E_A
# retlinear restrictions for conditional unemployment probabilities
fl_Delta_A_true <- 0.05
fl_Delta_E_true <- 0.11
fl_alpha_true <- 0.25
fl_beta_true <- fl_alpha_true + fl_Delta_A_true
```

```
fl_gamma_true <- fl_alpha_true + fl_Delta_E_true
fl_delta_true <- fl_alpha_true + fl_Delta_E_true + fl_Delta_A_true
ls_FBCDWXYZV <- ffi_gen_condi_unemploy_prob_2by2(mt_prob_joint_E_A,
  fl_alpha = fl_alpha_true, fl_beta = fl_beta_true,
  fl_gamma = fl_gamma_true, fl_delta = fl_delta_true, bl_verbose=TRUE)
```

```
## [1] "fl_F=0.412857205138471,fl_B=0.471665472604598,fl_C=0.267294503388642,fl_D=0.316546995323062"
## [1] "ar_b=0.279357139743076" "ar_b=0.38641672636977"  "ar_b=0.330597604627249" "ar_b=0.3751798305144
## [1] "fl_V=0.354923185445584"
```

```
fl_F <- ls_FBCDWXYZV$F
fl_B <- ls_FBCDWXYZV$B
fl_C <- ls_FBCDWXYZV$C
fl_D <- ls_FBCDWXYZV$D
fl_W <- ls_FBCDWXYZV$W
fl_X <- ls_FBCDWXYZV$X
fl_Y <- ls_FBCDWXYZV$Y
fl_Z <- ls_FBCDWXYZV$Z
fl_V <- ls_FBCDWXYZV$V
```

We can see the rank of a matrix with the qr function (QR decomposition), regardless of the random seed chosen above, the Ω matrix is not full ranked.

```
# does not matter if joint sum is used or the alternative
bl_use_joint_sum <- FALSE
if (bl_use_joint_sum) {
  # Construct The coefficent Matrix
  mt_OMEGA = t(
    matrix(data=c(fl_F, 1-fl_F, 0, 0,
                  0, 0, fl_B, 1-fl_B,
                  fl_C, 0, 1-fl_C, 0,
                  mt_prob_joint_E_A[1,1],  mt_prob_joint_E_A[1,2],
                  mt_prob_joint_E_A[2,1],  mt_prob_joint_E_A[2,2]), nrow=4, ncol=4))
} else {
  # Construct The coefficent Matrix
  mt_OMEGA = t(matrix(data=c(fl_F, 1-fl_F, 0, 0,
                             0, 0, fl_B, 1-fl_B,
                             fl_C, 0, 1-fl_C, 0,
                             0, fl_D, 0, 1-fl_D), nrow=4, ncol=4))

}
# rank Check with the qr function:
print(qr(mt_OMEGA))
```

```
## $qr
##            [,1]       [,2]       [,3]          [,4]
## [1,] -0.4918307 -0.4928650 -0.3982024  0.000000e+00
## [2,]  0.0000000 -0.4494694  0.4366482 -4.813342e-01
## [3,]  0.5434685 -0.7099340 -0.6403896 -7.173303e-01
## [4,]  0.0000000  0.7042682 -0.0385102 -2.081668e-17
##
## $rank
## [1] 3
##
## $qraux
```

```
## [1] 1.839430e+00 1.000000e+00 1.999258e+00 2.081668e-17
##
## $pivot
## [1] 1 2 3 4
##
## attr(,"class")
## [1] "qr"
```

```
# rank Check with the qr function:
```

We cannot solve the linear equations using *solve* because this is NOT full rank.

1.1.3 Rectilinear Restriction on Joint 2 by 2 Distribution

So in the section above, it is demonstrated that it is not possible to uniquely identify the joint probability mass function from marginal probability mass functions.

However, sometimes, we need to find some reasonable joint distribution, when we only observe marginal distributions. This joint distribution might be an input transition matrix in a model we simulate. If we just use one of the infinitely possible joint mass that match up with the marginals, then the model would have infinitely many simulation results depending on our arbitrary choice of joint mass.

Ideally, one should try to obtain data to estimate the underlying joint distribution, when this is not possible, we can impose additional non-parametric restrictions on the structures of the joint probability mass that would lead to unique joint mass from marginals.

Specifically, we will assume the incremental changes across rows and across columns of the joint mass matrix are row or column specific, is this sufficient? (In Some Cases it Will Not be):

$$\Delta_{12}^{E} = P(A_1, E_2) - P(A_1, E_1) = \gamma - \alpha = P(A_2, E_2) - P(A_2, E_1) = \delta - \beta$$
$$\Delta_{12}^{A} = P(A_2, E_1) - P(A_1, E_1) = \beta - \alpha = P(A_2, E_2) - P(A_1, E_2) = \delta - \gamma$$

The assumption is non-parametric. This is effectively an rectilinear assumption on the joint Cumulative Probability Mass Function. The assumption means that if we know Δ_{12}^{E} and Δ_{12}^{A} and say α, we have:

$$\beta = \alpha + \Delta_{12}^{A}\gamma = \alpha + \Delta_{12}^{E}\delta = \gamma\left(\alpha, \Delta_{12}^{E}\right) + \Delta_{12}^{A}$$

Alernatively, we can also say that if we know α, β and γ, then we know δ:

$$\Delta_{12}^{A} = \beta - \alpha\delta = \gamma + \Delta_{12}^{A} = \gamma + \beta - \alpha$$

1.1.4 2 by 2 Problem with Rectlinear Restrictions

We have these three equations, where α, Δ^A and Δ^E are not known:

$$\begin{aligned} W &= \alpha F + \alpha(1-F) + \Delta^A(1-F) \\ X &= \alpha B + \Delta^E B + \alpha(1-B) + \Delta^A(1-B) + \Delta^E(1-B) \\ Y &= \alpha C + \alpha(1-C) + \Delta^E(1-C) \end{aligned}$$

Rewriting a little bit, we have the following linear system:

$$\begin{aligned} W &= \alpha + \Delta^A(1-F) \\ X &= \alpha + \Delta^A(1-B) + \Delta^E \\ Y &= \alpha + \Delta^E(1-C) \\ V &= \alpha + \Delta^A\left(P(E_2, A_1) + P(E_2, A_2)\right) + \Delta^E\left(P(E_1, A_2) + P(E_2, A_2)\right) \end{aligned}$$

Using the First three equations, we have:

$$\begin{bmatrix} 1 & (1-F) & 0 \\ 1 & (1-B) & 1 \\ 1 & 0 & (1-C) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \Delta^A \\ \Delta^E \end{bmatrix} = \begin{bmatrix} W \\ X \\ Y \end{bmatrix}$$

$$\widehat{\Omega} \cdot \widehat{\mathbb{X}} = \widehat{b}$$

1.1.5 Solution Program for 2 by 2 Problem with Rectlinear Restrictions

Using the same F, B and C values obtained from prior, we have now a full ranked 3 by 3 matrix:

```
ffi_solve_2by2_rectilinear <- function(fl_F, fl_B, fl_C,
                                       fl_W, fl_X, fl_Y,
                                       bl_verbose=FALSE) {

  # Construct The coefficent Matrix
  mt_OMEGA_hat = t(matrix(
    data=c(1, 1-fl_F, 0,
           1, 1-fl_B, 1,
           1, 0, 1-fl_C), nrow=3, ncol=3))

  # rank Check with the qr function:
  if (bl_verbose) {
    print(qr(mt_OMEGA_hat))
  }

  # bhat
  ar_b_hat <- c(fl_W, fl_X, fl_Y)

  # solve
  ar_solution <- solve(mt_OMEGA_hat, ar_b_hat)

  # Get alpha and Delta from solution
  fl_alpha <- ar_solution[1]
  fl_Delta_A <- ar_solution[2]
  fl_Delta_E <- ar_solution[3]
  if (bl_verbose) {
    print(paste0('fl_Delta_A=',fl_Delta_A))
    print(paste0('fl_Delta_E=',fl_Delta_E))
  }

  # Get beta gamma , delta
  fl_beta <- fl_alpha + fl_Delta_A
  fl_gamma <- fl_alpha + fl_Delta_E
  fl_delta <- fl_gamma + fl_Delta_A
  if (bl_verbose) {
    print(paste0('fl_alpha=',fl_alpha))
    print(paste0('fl_beta=',fl_beta))
    print(paste0('fl_gamma=',fl_gamma))
    print(paste0('fl_delta=',fl_delta))
    if (abs(ar_b[1] - (fl_alpha*fl_F + fl_beta*(1-fl_F))) < 1e-10) {
      print('W matched')
    }
    if (abs(ar_b[2] - (fl_gamma*fl_B + fl_delta*(1-fl_B))) < 1e-10) {
```

```
      print('X matched')
    }
    if (abs(ar_b[3] - (fl_alpha*fl_C + fl_gamma*(1-fl_C))) < 1e-10) {
      print('Y matched')
    }
    if (abs(ar_b[4] - (fl_beta*fl_D + fl_delta*(1-fl_D))) < 1e-10) {
      print('Z matched')
    }
    if (abs(fl_V - (mt_prob_joint_E_A[1,1]*fl_alpha +
                    mt_prob_joint_E_A[1,2]*fl_beta +
                    mt_prob_joint_E_A[2,1]*fl_gamma +
                    mt_prob_joint_E_A[2,2]*fl_delta)) < 1e-10) {
      print('V matched')
    }
  }

  return(list(Delta_A=fl_Delta_A, Delta_E=fl_Delta_E,
              alpha=fl_alpha, beta=fl_beta,
              gamma=fl_gamma, delta=fl_delta))
}
```

Now solve for $\widehat{\mathbb{X}}$ using *solve* given $\widehat{\Omega}$ and $\widehat{b}$. Solving different assumptions on underlying joint probabilities.

1.1.6 Testing Program for 2 by 2 Problem with Rectlinear Restrictions

Solving first when rectilinear assumption is valid:

```
for (it_i in c(1,2,3)) {

  # JOint mass of population at all cells
  ls_ffi_rand_joint_mass <- ffi_gen_rand_joint_mass(
    it_seed = 123, it_Q = 2, it_M = 2, bl_verbose = FALSE)
  mt_prob_joint_E_A <- ls_ffi_rand_joint_mass$mt_prob_joint_E_A

  # retlinear restrictions for conditional unemployment probabilities
  fl_alpha_true <- 0.25
  if (it_i == 1) {
    fl_Delta_A_true <- 0.05
    fl_Delta_E_true <- 0.11
  } else if (it_i == 2) {
    fl_Delta_A_true <- 0.21
    fl_Delta_E_true <- 0.03
  } else if (it_i == 3) {
    fl_Delta_A_true <- -0.05
    fl_Delta_E_true <- -0.11
  }

  fl_beta_true <- fl_alpha_true + fl_Delta_A_true
  fl_gamma_true <- fl_alpha_true + fl_Delta_E_true
  fl_delta_true <- fl_alpha_true + fl_Delta_E_true + fl_Delta_A_true
  ls_FBCDWXYZV <- ffi_gen_condi_unemploy_prob_2by2(
    mt_prob_joint_E_A,
    fl_alpha = fl_alpha_true, fl_beta = fl_beta_true,
    fl_gamma = fl_gamma_true, fl_delta = fl_delta_true, bl_verbose=FALSE)
```

```
  # call solution function
  fl_F <- ls_FBCDWXYZV$F
  fl_B <- ls_FBCDWXYZV$B
  fl_C <- ls_FBCDWXYZV$C
  fl_D <- ls_FBCDWXYZV$D
  fl_W <- ls_FBCDWXYZV$W
  fl_X <- ls_FBCDWXYZV$X
  fl_Y <- ls_FBCDWXYZV$Y
  fl_Z <- ls_FBCDWXYZV$Z
  fl_V <- ls_FBCDWXYZV$V
  ls_solution <- ffi_solve_2by2_rectilinear(
    fl_F, fl_B, fl_C,
    fl_W, fl_X, fl_Y)

  fl_alpha <- ls_solution$alpha
  fl_beta <- ls_solution$beta
  fl_gamma <- ls_solution$gamma
  fl_delta <- ls_solution$delta

  # check
  print(paste0('it_i=', it_i))
  print(paste0('fl_alpha=', fl_alpha, ', fl_alpha_true=', fl_alpha_true))
  print(paste0('fl_beta=', fl_beta, ', fl_beta_true=', fl_beta_true))
  print(paste0('fl_gamma=', fl_gamma, ', fl_gamma_true=', fl_gamma_true))
  print(paste0('fl_delta=', fl_delta, ', fl_delta_true=', fl_delta_true))
}
```

```
## [1] "it_i=1"
## [1] "fl_alpha=0.25, fl_alpha_true=0.25"
## [1] "fl_beta=0.3, fl_beta_true=0.3"
## [1] "fl_gamma=0.36, fl_gamma_true=0.36"
## [1] "fl_delta=0.41, fl_delta_true=0.41"
## [1] "it_i=2"
## [1] "fl_alpha=0.25, fl_alpha_true=0.25"
## [1] "fl_beta=0.46, fl_beta_true=0.46"
## [1] "fl_gamma=0.28, fl_gamma_true=0.28"
## [1] "fl_delta=0.49, fl_delta_true=0.49"
## [1] "it_i=3"
## [1] "fl_alpha=0.25, fl_alpha_true=0.25"
## [1] "fl_beta=0.2, fl_beta_true=0.2"
## [1] "fl_gamma=0.14, fl_gamma_true=0.14"
## [1] "fl_delta=0.0900000000000001, fl_delta_true=0.09"
```

Solving when rectilinear assumption is NOT valid, results can be approximately correct. However, note that we do not have positive mass in all cases. See case three below, the alpha backed out is negative.

```
for (it_i in c(1,2,3)) {

  # JOint mass of population at all cells
  ls_ffi_rand_joint_mass <- ffi_gen_rand_joint_mass(
    it_seed = 123, it_Q = 2, it_M = 2, bl_verbose = FALSE)
  mt_prob_joint_E_A <- ls_ffi_rand_joint_mass$mt_prob_joint_E_A

  # retlinear restrictions for conditional unemployment probabilities
```

```
  if (it_i == 1) {
    fl_alpha_true <- 0.25
    fl_Delta_A_true <- 0.05
    fl_Delta_E_true <- 0.11
    fl_Delta_AE_true <- 0.10
  } else if (it_i == 2) {
    fl_alpha_true <- 0.55
    fl_Delta_A_true <- 0.21
    fl_Delta_E_true <- 0.03
    fl_Delta_AE_true <- 0.15
  } else if (it_i == 3) {
    fl_alpha_true <- 0.15
    fl_Delta_A_true <- -0.05
    fl_Delta_E_true <- -0.11
    fl_Delta_AE_true <- 0.30
  }

  fl_beta_true <- fl_alpha_true + fl_Delta_A_true
  fl_gamma_true <- fl_alpha_true + fl_Delta_E_true
  fl_delta_true <- fl_alpha_true + fl_Delta_AE_true
  ls_FBCDWXYZV <- ffi_gen_condi_unemploy_prob_2by2(
    mt_prob_joint_E_A,
    fl_alpha = fl_alpha_true, fl_beta = fl_beta_true,
    fl_gamma = fl_gamma_true, fl_delta = fl_delta_true, bl_verbose=FALSE)

  # call solution function
  fl_F <- ls_FBCDWXYZV$F
  fl_B <- ls_FBCDWXYZV$B
  fl_C <- ls_FBCDWXYZV$C
  fl_D <- ls_FBCDWXYZV$D
  fl_W <- ls_FBCDWXYZV$W
  fl_X <- ls_FBCDWXYZV$X
  fl_Y <- ls_FBCDWXYZV$Y
  fl_Z <- ls_FBCDWXYZV$Z
  fl_V <- ls_FBCDWXYZV$V
  ls_solution <- ffi_solve_2by2_rectilinear(
    fl_F, fl_B, fl_C,
    fl_W, fl_X, fl_Y)

  fl_alpha <- ls_solution$alpha
  fl_beta <- ls_solution$beta
  fl_gamma <- ls_solution$gamma
  fl_delta <- ls_solution$delta

  # check
  print(paste0('it_i=', it_i))
  print(paste0('fl_alpha=', fl_alpha, ', fl_alpha_true=', fl_alpha_true))
  print(paste0('fl_beta=', fl_beta, ', fl_beta_true=', fl_beta_true))
  print(paste0('fl_gamma=', fl_gamma, ', fl_gamma_true=', fl_gamma_true))
  print(paste0('fl_delta=', fl_delta, ', fl_delta_true=', fl_delta_true))
}
```

```
## [1] "it_i=1"
## [1] "fl_alpha=0.275066384524696, fl_alpha_true=0.25"
```

```
## [1] "fl_beta=0.282374240902959, fl_beta_true=0.3"
## [1] "fl_gamma=0.350855661880163, fl_gamma_true=0.36"
## [1] "fl_delta=0.358163518258426, fl_delta_true=0.35"
## [1] "it_i=2"
## [1] "fl_alpha=0.587599576787045, fl_alpha_true=0.55"
## [1] "fl_beta=0.733561361354439, fl_beta_true=0.76"
## [1] "fl_gamma=0.566283492820244, fl_gamma_true=0.58"
## [1] "fl_delta=0.712245277387639, fl_delta_true=0.7"
## [1] "it_i=3"
## [1] "fl_alpha=-0.0421756146893388, fl_alpha_true=0.15"
## [1] "fl_beta=0.235130819743977, fl_beta_true=0.1"
## [1] "fl_gamma=0.110106592252085, fl_gamma_true=0.04"
## [1] "fl_delta=0.387413026685401, fl_delta_true=0.45"
```

1.2 3 by 3 Joint from Marginal Probability Mass Functions with Rectilinear Assumptions

1.2.1 The 3 by 3 Problem

We append the 2 by 2 problem to a 3 by 3 problem.

$$
\begin{aligned}
P(U|E_1) &= P(U|A_1,E_1)\cdot P(A_1|E_1) + P(U|A_2,E_1)\cdot P(A_2|E_1) + P(U|A_3,E_1)\cdot P(A_3|E_1)\\
P(U|E_2) &= P(U|A_1,E_2)\cdot P(A_1|E_2) + P(U|A_2,E_2)\cdot P(A_2|E_2) + P(U|A_3,E_2)\cdot P(A_3|E_2)\\
P(U|E_3) &= P(U|A_1,E_3)\cdot P(A_1|E_3) + P(U|A_2,E_3)\cdot P(A_2|E_3) + P(U|A_3,E_3)\cdot P(A_3|E_3)\\
P(U|A_1) &= P(U|A_1,E_1)\cdot P(E_1|A_1) + P(U|A_1,E_2)\cdot P(E_2|A_1) + P(U|A_1,E_3)\cdot P(E_3|A_1)\\
P(U|A_2) &= P(U|A_2,E_1)\cdot P(E_1|A_2) + P(U|A_2,E_2)\cdot P(E_2|A_2) + P(U|A_2,E_3)\cdot P(E_3|A_2)\\
P(U|A_3) &= P(U|A_3,E_1)\cdot P(E_1|A_3) + P(U|A_3,E_2)\cdot P(E_2|A_3) + P(U|A_3,E_3)\cdot P(E_3|A_3)
\end{aligned}
$$

Similar to before, let Roman letter represent what we know, and greek letters represent what we do not know. There are nine potential unknown conditional unemployment probabilities α_{ij}.

$$
\begin{aligned}
V_1 &= \alpha_{11}B_{11} + \alpha_{21}B_{21} + \alpha_{31}B_{31}\\
V_2 &= \alpha_{12}B_{12} + \alpha_{22}B_{22} + \alpha_{32}B_{32}\\
V_3 &= \alpha_{13}B_{13} + \alpha_{23}B_{23} + \alpha_{33}B_{33}\\
W_1 &= \alpha_{11}C_{11} + \alpha_{12}C_{21} + \alpha_{13}C_{31}\\
W_2 &= \alpha_{21}C_{12} + \alpha_{22}C_{22} + \alpha_{23}C_{32}\\
W_3 &= \alpha_{31}C_{13} + \alpha_{32}C_{23} + \alpha_{33}C_{33}
\end{aligned}
$$

1.2.2 3 by 3 Problem with Rectlinear Restrictions

With rectilinear assumptions, we have now rather than nine parameters, five parameters.

$$
\begin{aligned}
V_1 &= \alpha B_{11} + (\alpha+\delta^{21})B_{21} + (\alpha+\delta^{21}+\delta^{31})B_{31}\\
V_2 &= (\alpha+\delta^{12})B_{12} + (\alpha+\delta^{12}+\delta^{21})B_{22} + (\alpha+\delta^{12}+\delta^{21}+\delta^{31})B_{32}\\
V_3 &= (\alpha+\delta^{12}+\delta^{13})B_{13} + (\alpha+\delta^{12}+\delta^{13}+\delta^{21})B_{23} + (\alpha+\delta^{12}+\delta^{13}+\delta^{21}+\delta^{31})B_{33}\\
W_1 &= \alpha C_{11} + (\alpha+\delta^{12})C_{21} + (\alpha+\delta^{12}+\delta^{13})C_{31}\\
W_2 &= (\alpha+\delta^{21})C_{12} + (\alpha+\delta^{12}+\delta^{21})C_{22} + (\alpha+\delta^{12}+\delta^{13}+\delta^{21})C_{32}\\
W_3 &= (\alpha+\delta^{21}+\delta^{31})C_{13} + (\alpha+\delta^{12}+\delta^{21}+\delta^{31})C_{23} + (\alpha+\delta^{12}+\delta^{13}+\delta^{21}+\delta^{31})C_{33}
\end{aligned}
$$

Write these in matrix form:

$$\begin{bmatrix} (B_{11}+B_{21}+B_{31}) & (B_{21}+B_{31}) & (B_{31}) & 0 & 0 \\ (B_{12}+B_{22}+B_{32}) & (B_{22}+B_{32}) & (B_{32}) & (B_{12}+B_{22}+B_{32}) & 0 \\ (B_{13}+B_{23}+B_{33}) & (B_{23}+B_{33}) & (B_{33}) & (B_{13}+B_{23}+B_{33}) & (B_{13}+B_{23}+B_{33}) \\ (C_{11}+C_{21}+C_{31}) & 0 & 0 & (C_{21}+C_{31}) & (C_{31}) \\ (C_{12}+C_{22}+C_{33}) & (C_{12}+C_{22}+C_{32}) & 0 & (C_{22}+C_{32}) & (C_{32}) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \delta^{21} \\ \delta^{31} \\ \delta^{12} \\ \delta^{13} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ W_1 \\ W_2 \end{bmatrix}$$

$$\widehat{\Omega} \cdot \widehat{\mathbb{X}} = \widehat{b}$$

Simplify slightly for where the probabilities sum to 1, which arrives at a matrix form that is similar to the matrix form arrived at for the 2 by 2 problem.

$$\begin{bmatrix} 1 & (B_{21}+B_{31}) & (B_{31}) & 0 & 0 \\ 1 & (B_{22}+B_{32}) & (B_{32}) & 1 & 0 \\ 1 & (B_{23}+B_{33}) & (B_{33}) & 1 & 1 \\ 1 & 0 & 0 & (C_{21}+C_{31}) & (C_{31}) \\ 1 & 1 & 0 & (C_{22}+C_{32}) & (C_{32}) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \delta^{21} \\ \delta^{31} \\ \delta^{12} \\ \delta^{13} \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ W_1 \\ W_2 \end{bmatrix}$$

$$\widehat{\Omega} \cdot \widehat{\mathbb{X}} = \widehat{b}$$

1.2.3 Data Program for 3 by 3 Problem with Rectlinear Restrictions

```
ffi_gen_condi_unemploy_prob_3by3 <- function(mt_prob_joint_E_A, mt_alpha, bl_verbose = FALSE) {

  # From joint probability, generate conditional probabilities
  fl_B21 <- mt_prob_joint_E_A[1,2]/sum(mt_prob_joint_E_A[1,])
  fl_B31 <- mt_prob_joint_E_A[1,3]/sum(mt_prob_joint_E_A[1,])

  fl_B22 <- mt_prob_joint_E_A[2,2]/sum(mt_prob_joint_E_A[2,])
  fl_B32 <- mt_prob_joint_E_A[2,3]/sum(mt_prob_joint_E_A[2,])

  fl_B23 <- mt_prob_joint_E_A[3,2]/sum(mt_prob_joint_E_A[3,])
  fl_B33 <- mt_prob_joint_E_A[3,3]/sum(mt_prob_joint_E_A[3,])

  fl_C21 <- mt_prob_joint_E_A[2,1]/sum(mt_prob_joint_E_A[,1])
  fl_C31 <- mt_prob_joint_E_A[3,1]/sum(mt_prob_joint_E_A[,1])

  fl_C22 <- mt_prob_joint_E_A[2,2]/sum(mt_prob_joint_E_A[,2])
  fl_C32 <- mt_prob_joint_E_A[3,2]/sum(mt_prob_joint_E_A[,2])

  if (bl_verbose) {
    print(paste0('fl_B21=', fl_B21, ',fl_B31=',fl_B31))
    print(paste0('fl_B22=', fl_B22, ',fl_B22=',fl_B22))
    print(paste0('fl_B23=', fl_B23, ',fl_B33=',fl_B33))
    print(paste0('fl_C21=', fl_C21, ',fl_C31=',fl_C31))
    print(paste0('fl_C22=', fl_C22, ',fl_C32=',fl_C32))
  }

  # Generate random W X Y
  fl_V1 <- mt_alpha[1,1]*(1 - fl_B21 - fl_B31) + mt_alpha[2,1]*fl_B21 + mt_alpha[3,1]*fl_B31
  fl_V2 <- mt_alpha[1,2]*(1 - fl_B22 - fl_B32) + mt_alpha[2,2]*fl_B22 + mt_alpha[3,2]*fl_B32
  fl_V3 <- mt_alpha[1,3]*(1 - fl_B23 - fl_B33) + mt_alpha[2,3]*fl_B23 + mt_alpha[3,3]*fl_B33
  fl_W1 <- mt_alpha[1,1]*(1 - fl_C21 - fl_C31) + mt_alpha[1,2]*fl_C21 + mt_alpha[1,3]*fl_C31
```

```
  fl_W2 <- mt_alpha[2,1]*(1 - fl_C22 - fl_C32) + mt_alpha[2,2]*fl_C22 + mt_alpha[2,3]*fl_C32
  ar_b <- c(fl_V1, fl_V2, fl_V3, fl_W1, fl_W2)

  if (bl_verbose) {
    print(paste0('ar_b=',ar_b))
  }

  # return
  return(ar_b)
}
```

1.2.4 Solution Program for 3 by 3 Problem with Rectlinear Restrictions

```
ffi_solve_3by3_rectilinear <- function(mt_prob_joint_E_A, ar_b, bl_verbose=FALSE) {

  # From joint probability, generate conditional probabilities
  fl_B21 <- mt_prob_joint_E_A[1,2]/sum(mt_prob_joint_E_A[1,])
  fl_B31 <- mt_prob_joint_E_A[1,3]/sum(mt_prob_joint_E_A[1,])

  fl_B22 <- mt_prob_joint_E_A[2,2]/sum(mt_prob_joint_E_A[2,])
  fl_B32 <- mt_prob_joint_E_A[2,3]/sum(mt_prob_joint_E_A[2,])

  fl_B23 <- mt_prob_joint_E_A[3,2]/sum(mt_prob_joint_E_A[3,])
  fl_B33 <- mt_prob_joint_E_A[3,3]/sum(mt_prob_joint_E_A[3,])

  fl_C21 <- mt_prob_joint_E_A[2,1]/sum(mt_prob_joint_E_A[,1])
  fl_C31 <- mt_prob_joint_E_A[3,1]/sum(mt_prob_joint_E_A[,1])

  fl_C22 <- mt_prob_joint_E_A[2,2]/sum(mt_prob_joint_E_A[,2])
  fl_C32 <- mt_prob_joint_E_A[3,2]/sum(mt_prob_joint_E_A[,2])

  # Construct The coefficent Matrix
  mt_OMEGA = t(matrix(
    data=c(1, fl_B21+fl_B31, fl_B31, 0, 0,
           1, fl_B22+fl_B32, fl_B32, 1, 0,
           1, fl_B23+fl_B33, fl_B33, 1, 1,
           1, 0, 0, fl_C21+fl_C31, fl_C31,
           1, 1, 0, fl_C22+fl_C32, fl_C32), nrow=5, ncol=5))

  # rank Check with the qr function:
  if (bl_verbose) {
    print(qr(mt_OMEGA_hat))
  }

  # solve
  ar_solution <- solve(mt_OMEGA, ar_b)

  # Get alpha and Delta from solution
  fl_alpha <- ar_solution[1]
  fl_Delta_21 <- ar_solution[2]
  fl_Delta_31 <- ar_solution[3]
  fl_Delta_12 <- ar_solution[4]
  fl_Delta_13 <- ar_solution[5]
```

```
  if (bl_verbose) {
    print(paste0('fl_Delta_21=',fl_Delta_21))
    print(paste0('fl_Delta_31=',fl_Delta_31))
    print(paste0('fl_Delta_12=',fl_Delta_12))
    print(paste0('fl_Delta_13=',fl_Delta_13))
  }

  # Get beta gamma , delta
  mt_alpha = matrix(0, 3, 3)
  for (it_row in c(1,2,3)) {
    for (it_col in c(1,2,3)) {

      fl_current_val <- fl_alpha

      if (it_row == 2) {
        fl_current_val <- fl_current_val + fl_Delta_21
      }
      if (it_row == 3) {
        fl_current_val <- fl_current_val + fl_Delta_21 + fl_Delta_31
      }
      if (it_col == 2) {
        fl_current_val <- fl_current_val + fl_Delta_12
      }
      if (it_col == 3) {
        fl_current_val <- fl_current_val + fl_Delta_12 + fl_Delta_13
      }

      mt_alpha[it_row, it_col] <- fl_current_val
    }
  }


  return(mt_alpha)
}
```

1.2.5 Testing Program for 3 by 3 Problem with Rectlinear Restrictions

Solving first when rectilinear assumption is valid:

```
for (it_i in c(1,2,3)) {

  # JOint mass of population at all cells
  ls_ffi_rand_joint_mass <- ffi_gen_rand_joint_mass(
    it_seed = 123, it_Q = 3, it_M = 3, bl_verbose = FALSE)
  mt_prob_joint_E_A <- ls_ffi_rand_joint_mass$mt_prob_joint_E_A

  # retlinear restrictions for conditional unemployment probabilities
  if (it_i == 1) {
    fl_alpha <- 0.1
    fl_Delta_21 <- 0.05
    fl_Delta_31 <- 0.07
    fl_Delta_12 <- 0.09
    fl_Delta_13 <- 0.03
  } else if (it_i == 2) {
```

```
    fl_alpha <- 0.20
    fl_Delta_21 <- -0.05
    fl_Delta_31 <- +0.07
    fl_Delta_12 <- 0.09
    fl_Delta_13 <- -0.03
  } else if (it_i == 3) {
    fl_alpha <- 0.15
    fl_Delta_21 <- 0.11
    fl_Delta_31 <- -0.01
    fl_Delta_12 <- 0
    fl_Delta_13 <- +0.1
  }

  mt_alpha_true = matrix(0, 3, 3)
  for (it_row in c(1,2,3)) {
    for (it_col in c(1,2,3)) {

      fl_current_val <- fl_alpha

      if (it_row == 2) {
        fl_current_val <- fl_current_val + fl_Delta_21
      }
      if (it_row == 3) {
        fl_current_val <- fl_current_val + fl_Delta_21 + fl_Delta_31
      }
      if (it_col == 2) {
        fl_current_val <- fl_current_val + fl_Delta_12
      }
      if (it_col == 3) {
        fl_current_val <- fl_current_val + fl_Delta_12 + fl_Delta_13
      }

      mt_alpha_true[it_row, it_col] <- fl_current_val
    }
  }

  ar_b <- ffi_gen_condi_unemploy_prob_3by3(mt_prob_joint_E_A, mt_alpha_true, bl_verbose=FALSE)

  # call solution function
  mt_alpha_solu <- ffi_solve_3by3_rectilinear(mt_prob_joint_E_A, ar_b)

  # check
  print(paste0('it_i=', it_i))
  print('mt_alpha_true')
  print(mt_alpha_true)
  print('mt_alpha_solu')
  print(mt_alpha_solu)

}
```

```
## [1] "it_i=1"
## [1] "mt_alpha_true"
##      [,1] [,2] [,3]
## [1,] 0.10 0.19 0.22
```

```
## [2,] 0.15 0.24 0.27
## [3,] 0.22 0.31 0.34
## [1] "mt_alpha_solu"
##      [,1] [,2] [,3]
## [1,] 0.10 0.19 0.22
## [2,] 0.15 0.24 0.27
## [3,] 0.22 0.31 0.34
## [1] "it_i=2"
## [1] "mt_alpha_true"
##      [,1] [,2] [,3]
## [1,] 0.20 0.29 0.26
## [2,] 0.15 0.24 0.21
## [3,] 0.22 0.31 0.28
## [1] "mt_alpha_solu"
##      [,1] [,2] [,3]
## [1,] 0.20 0.29 0.26
## [2,] 0.15 0.24 0.21
## [3,] 0.22 0.31 0.28
## [1] "it_i=3"
## [1] "mt_alpha_true"
##      [,1] [,2] [,3]
## [1,] 0.15 0.15 0.25
## [2,] 0.26 0.26 0.36
## [3,] 0.25 0.25 0.35
## [1] "mt_alpha_solu"
##      [,1] [,2] [,3]
## [1,] 0.15 0.15 0.25
## [2,] 0.26 0.26 0.36
## [3,] 0.25 0.25 0.35
```

Solving first when rectilinear assumption is NOT VALID, same as above, except add randomness at each point. Note that the solution is approximately the same as the actual in the examples here.

```r
# set seed
set.seed(123)
# loop over alternatives
for (it_i in c(1,2,3)) {

  # JOint mass of population at all cells
  ls_ffi_rand_joint_mass <- ffi_gen_rand_joint_mass(
    it_seed = 123, it_Q = 3, it_M = 3, bl_verbose = FALSE)
  mt_prob_joint_E_A <- ls_ffi_rand_joint_mass$mt_prob_joint_E_A

  # retlinear restrictions for conditional unemployment probabilities
  if (it_i == 1) {
    fl_alpha <- 0.1
    fl_Delta_21 <- 0.05
    fl_Delta_31 <- 0.07
    fl_Delta_12 <- 0.09
    fl_Delta_13 <- 0.03
  } else if (it_i == 2) {
    fl_alpha <- 0.20
    fl_Delta_21 <- -0.05
    fl_Delta_31 <- +0.07
    fl_Delta_12 <- 0.09
```

```
    fl_Delta_13 <- -0.03
  } else if (it_i == 3) {
    fl_alpha <- 0.15
    fl_Delta_21 <- 0.11
    fl_Delta_31 <- -0.01
    fl_Delta_12 <- 0
    fl_Delta_13 <- +0.1
  }

  mt_alpha_true = matrix(0, 3, 3)
  for (it_row in c(1,2,3)) {
    for (it_col in c(1,2,3)) {

      fl_current_val <- fl_alpha

      if (it_row == 2) {
        fl_current_val <- fl_current_val + fl_Delta_21 + runif(1)*0.2
      }
      if (it_row == 3) {
        fl_current_val <- fl_current_val + fl_Delta_21 + fl_Delta_31 + runif(1)*0.2
      }
      if (it_col == 2) {
        fl_current_val <- fl_current_val + fl_Delta_12 + runif(1)*0.2
      }
      if (it_col == 3) {
        fl_current_val <- fl_current_val + fl_Delta_12 + fl_Delta_13 + runif(1)*0.2
      }

      mt_alpha_true[it_row, it_col] <- fl_current_val
    }
  }

  ar_b <- ffi_gen_condi_unemploy_prob_3by3(mt_prob_joint_E_A, mt_alpha_true, bl_verbose=FALSE)

  # call solution function
  mt_alpha_solu <- ffi_solve_3by3_rectilinear(mt_prob_joint_E_A, ar_b)

  # check
  print(paste0('it_i=', it_i))
  print('mt_alpha_true')
  print(mt_alpha_true)
  print('mt_alpha_solu')
  print(mt_alpha_solu)

}
```

```
## [1] "it_i=1"
## [1] "mt_alpha_true"
##           [,1]      [,2]      [,3]
## [1,] 0.1000000 0.2813229 0.4113667
## [2,] 0.2406668 0.4900408 0.4705499
## [3,] 0.2692175 0.3839961 0.7088086
## [1] "mt_alpha_solu"
##            [,1]      [,2]      [,3]
```

```
## [1,] 0.06922205 0.2610615 0.4720626
## [2,] 0.26540144 0.4572409 0.6682420
## [3,] 0.24462011 0.4364596 0.6474607
## [1] "it_i=2"
## [1] "mt_alpha_true"
##           [,1]      [,2]      [,3]
## [1,] 0.2000000 0.3813229 0.4513667
## [2,] 0.2406668 0.4900408 0.4105499
## [3,] 0.2692175 0.3839961 0.6488086
## [1] "mt_alpha_solu"
##           [,1]      [,2]      [,3]
## [1,] 0.1692221 0.3610615 0.5120626
## [2,] 0.2654014 0.4572409 0.6082420
## [3,] 0.2446201 0.4364596 0.5874607
## [1] "it_i=3"
## [1] "mt_alpha_true"
##           [,1]      [,2]      [,3]
## [1,] 0.1500000 0.2413229 0.4413667
## [2,] 0.3506668 0.5100408 0.5605499
## [3,] 0.2992175 0.3239961 0.7188086
## [1] "mt_alpha_solu"
##           [,1]      [,2]      [,3]
## [1,] 0.1192221 0.2210615 0.5020626
## [2,] 0.3754014 0.4772409 0.7582420
## [3,] 0.2746201 0.3764596 0.6574607
```