Life-history tradeoff example runs

Karthik Ram

Wed May 23 12:26:40 2012

```
# Tradeoff Functions
# TOC
# Setting up tradeoffs
#     sJ.from.m
#     sA.from.Fec
#     Fec.from.m
# dem.model (basic martix version)
# tradeoff (calculates basic tradeoff)
# vd_tradeoff (calculates tradeoff using varDev)
#     run_vdm
#     run_vdm_jg
#     run_vdm_corr
# param_combs (generates parameter combinations)
# param_combs_jg (same as above with juvenile growth)
# param_combs_corr (same as above with corrleation between stages)
# tradeoff_plot
# do_tradeoff (wrapper for tradeoff)
# do_vd_tradeoff (wrapper for do_tradeoff)
# ‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘
# --------------------------------------------------
sJ.from.m <- function(m, a, b) {
    return(a + b * m)
}
# --------------------------------------------------
dem.model <- function(m, sJ, sA, Fec) {
    mat <- matrix(c((1 - m) * sJ, m * sJ, Fec, sA), nrow = 2)
    return(max(eigen(mat)$values))
}
 # --------------------------------------------------
tradeoff <- function(a, b, sA, Fec, m = NULL) {
    if (is.null(m))
        m <- seq(0.01, 0.99, length = 20)
    sJ <- sJ.from.m(m, a, b)
    df <- as.matrix(data.frame(m = m, sJ = sJ, sA = sA, Fec = Fec))
    lambda <- apply(df, 1, function(x) dem.model(x[1], x[2], x[3], x[4]))
    type <- "simple"
    return((data.frame(m, sJ, lambda, type)))
}
# =------------------------------------------
do_tradeoff <- function(tlist) {
    toff <- tradeoff(tlist$a, tlist$b, tlist$Fec, tlist$sA, m = NULL)
    basic_result <- list(data = toff, params = tlist)
    return(basic_result)
}

# ---------------------------------------------------
run_vdm <- function(m, sJ, sA, Fec) {
    vdmodel <- suppressMessages(VD.model(num.stages = 2, marginal.durations = list(VD.dist("
        list(prob = m)), VD.dist("geomp1", list(prob = (1 - sA)))), marginal.death.times = l
        list(prob = (1 - sJ))), VD.dist("infinite")), fecundity = Fec))
    VDS <- VD.run(vdmodel)
    dev.table <- compile.dev.table(VDS)
    mean.fec <- calc.average.surv.rep.by.age(dev.table, F = Fec)
    r <- VD.solve.euler(mean.fec)
    return(exp(r))
}  # end run_vdm
# -----------------------------------------
run_vdm_jg <- function(m, sA, sJ, Fec, juvshape) {
    lambdaJ <- -log(1 - m)  ## prob of not maturing for one time step is exp(-lambdaJ)
    meanjuv <- 1/lambdaJ  ## mean of the exponential
                # we need the scale parameter. mean = shape * scale. sd = sqrt(shape) * sca
    juvscale <- meanjuv/juvshape
    vdmodel <- suppressMessages(VD.model(2, marginal.durations = list(VD.dist("gamma",
        list(shape = juvshape, scale = juvscale)), VD.dist("geomp1", list(prob = (1 -
        sA)))), marginal.death.times = list(VD.dist("geomp1", list(prob = (1 -
        sJ))), VD.dist("infinite")), fecundity = Fec))
    VDS <- VD.run(vdmodel)
    dev.table <- compile.dev.table(VDS)
    mean.fec <- calc.average.surv.rep.by.age(dev.table, F = Fec)
    r <- VD.solve.euler(mean.fec)
    return(exp(r))
}  # end run_vdm_jg
# -----------------------------------------
run_vdm_corr <- function(m, sA, sJ, Fec, corr) {
    my.gauss.cov <- matrix(c(1, corr, corr, 1), nrow = 2)
    vdmodel <- suppressMessages(VD.model(2, marginal.durations = list(VD.dist("geomp1",
        list(prob = m)), VD.dist("geomp1", list(prob = (1 - sA)))), marginal.death.times = l
        list(prob = (1 - sJ))), VD.dist("infinite")), gauss.cov = my.gauss.cov,
        fecundity = Fec))
    VDS <- VD.run(vdmodel)
    dev.table <- compile.dev.table(VDS)
    mean.fec <- calc.average.surv.rep.by.age(dev.table, F = Fec)
    r <- VD.solve.euler(mean.fec)
    return(exp(r))
}  # end run_vdm_corr
# ---------------------------------------------------
vd_tradeoff <- function(a, b, sA, Fec, m = NULL, corr = NULL, juvshape = NULL,
    version = "t1") {
    if (is.null(m))
        m <- seq(0.01, 0.99, length = 20)
    sJ <- sJ.from.m(m, a, b)
    if (is.null(juvshape) && is.null(corr)) {
                        # message(’running run_vdm()....\n’)
        df <- as.matrix(data.frame(m = m, sJ = sJ, sA = sA, Fec = Fec))
        lambda <- apply(df, 1, function(x) run_vdm(x[1], x[2], x[3], x[4]))
        df <- data.frame(m, sJ, lambda)
        df$type <- "juv_tradeoff"
    }
    if (!is.null(juvshape) && is.null(corr)) {
                        # message(’running with run_vdm_jg()....\n’)
        df <- as.matrix(data.frame(m = m, sJ = sJ, sA = sA, Fec = Fec, juvshape = juvshape))
        lambda <- apply(df, 1, function(x) run_vdm_jg(x[1], x[2], x[3], x[4],
            x[5]))
        df <- data.frame(m, sJ, lambda)
        df$type <- "juvshape"
    }
    if (is.null(juvshape) && !is.null(corr)) {
                        # message(’running with run_vdm_corr()....\n’)
        df <- as.matrix(data.frame(m = m, sJ = sJ, sA = sA, Fec = Fec, corr = corr))
        lambda <- apply(df, 1, function(x) run_vdm_corr(x[1], x[2], x[3], x[4],
            x[5]))
        df <- data.frame(m, sJ, lambda)
        df$type <- "corr"
    }
    return(df)
}
# =-----------------------------------------
# The wrapper for the above 3 functions. The right function to call is determined by the nu
do_vd_tradeoff <- function(tlist) {
    if (length(tlist) == 5)
        vd_toff <- vd_tradeoff(tlist$a, tlist$b, tlist$sA, tlist$Fec, m = NULL)
    if (length(tlist) == 6 && names(tlist)[5] == "juvshape")
        vd_toff <- vd_tradeoff(tlist$a, tlist$b, tlist$sA, tlist$Fec, juvshape = tlist$juvsh
            m = NULL)
    if (length(tlist) == 6 && names(tlist)[5] == "corr")
        vd_toff <- vd_tradeoff(tlist$a, tlist$b, tlist$sA, tlist$Fec, tlist$corr,
            m = NULL)
    vd_result <- list(data = vd_toff, params = tlist)
    return(vd_result)
}

# =============================================
# checks for tradeoff combinations that might result in unrealistic mortality rates.
validity <- function(aa, bb) {
    mm <- seq(0.01, 0.99, length = 20)
    sJ <- sJ.from.m(mm, aa, bb)
    if (min(sJ) < 0) {
        return(FALSE)
    } else {
        return(TRUE)
    }
}
# After checking for invalid tradeoffs, this function removes duplicates.
param_check <- function(ax, bx) {
    para2 <- expand.grid(ax, bx)
    para2$rr <- apply(para2, 1, function(x) validity(x[1], x[2]))
    clear <- as.data.table(para2[which(para2$rr), ])
    res <- data.frame(unique(clear[, list(Var1, Var2)]))
    final <- list(a = res$Var1, b = res$Var2)
    return(final)
}
# Generates a list of tradeoff combinations that allow all combinations to be pushed throug
param_combs <- function(a, b, sA, Fec) {
    valid <- param_check(a, b)
    base <- data.frame(a = valid[[1]], b = valid[[2]])
    others <- expand.grid(sA = sA, Fec = Fec)
    parameters <- ddply(others, .(sA, Fec), function(x) data.frame(base$a, base$b,
        sA = rep(x$sA, dim(base)[1]), Fec = rep(x$Fec, dim(base)[1])))
    parameters$sim_id <- paste0("S", 1:dim(parameters)[1])
    params <- mapply(list, a = parameters[, 1], b = parameters[, 2], sA = parameters[,
        3], Fec = parameters[, 4], sim_id = parameters[, 5], SIMPLIFY = F)
    return(params)
}
param_combs_jg <- function(a, b, sA, Fec, juvshape) {
    valid <- param_check(a, b)
    base <- data.frame(a = valid[[1]], b = valid[[2]])
    others <- expand.grid(sA = sA, Fec = Fec, juvshape = juvshape)
    parameters <- ddply(others, .(sA, Fec), function(x) data.frame(base$a, base$b,
        sA = rep(x$sA, dim(base)[1]), Fec = rep(x$Fec, dim(base)[1]), juvshape = rep(x$juvsh
            dim(base)[1])))
    parameters$sim_id <- paste0("JG", 1:dim(parameters)[1])
    params <- mapply(list, a = parameters[, 1], b = parameters[, 2], sA = parameters[,
        3], Fec = parameters[, 4], juvshape = parameters[, 5], sim_id = parameters[,
        6], SIMPLIFY = F)
    return(params)
}
param_combs_corr <- function(a, b, sA, Fec, corr) {
    valid <- param_check(a, b)
    base <- data.frame(a = valid[[1]], b = valid[[2]])
    others <- expand.grid(sA = sA, Fec = Fec, corr = corr)
    parameters <- ddply(others, .(sA, Fec), function(x) data.frame(base$a, base$b,
        sA = rep(x$sA, dim(base)[1]), Fec = rep(x$Fec, dim(base)[1]), corr = rep(x$corr,
            dim(base)[1])))
    parameters$sim_id <- paste0("CO", 1:dim(parameters)[1])
    params <- mapply(list, a = parameters[, 1], b = parameters[, 2], sA = parameters[,
        3], Fec = parameters[, 4], corr = parameters[, 5], sim_id = parameters[,
        6], SIMPLIFY = F)
    return(params)
}
# ---------------------------------------

# A simple spline to smooth simulated points and find the max.
arg_max <- function(data) {
    smoothed <- smooth.spline(data$m, data$lambda, df = 10)
    maxy <- max(smoothed$y)
    maxx <- smoothed$x[which(smoothed$y == maxy)]
    return(list(maxx, maxy))
}

# These are the plotting functions.
tradeoff.plot <- function(data, ptitle = "") {
              # Make colors red, blue, and gold.
              # make shapes also 3 different kinds.
    if (length(unique(data$type)) > 3)
        stop("Function not set up to deal with more than 3 categories at the moment")
    base_colours <- c("#a8ddb5", "#43a2ca", "#e0f3db")
    colours <- base_colours[1:length(unique(data$type))]
    tplot <- ggplot(data, aes(m, lambda, colour = type)) + geom_point(size = 2.8,
        shape = 16) + opts(title = ptitle) + scale_color_manual(values = colours) +
        theme_complete_bw()
    return(tplot)
}
# ------------------------------------------------------------------
# assembles plots in the same fig.
assemble_plots <- function(dat) {
    s1 <- as.numeric(dat$sim_id)
    s2 <- as.numeric(dat$sim_juv)
    s3 <- as.numeric(dat$sim_cor)
    title <- sprintf("a:%s, b:%s, sA:%s, Fec:%s, juvshape:%s, corr:%s", dat$a,
        dat$b, dat$sA, dat$Fec, dat$juvshape, dat$corr)
    plot_data <- rbind(t1_vd[[s1]]$data, t1_juvshape[[s2]]$data, t1_corr[[s3]]$data)
    tradeoff.plot(plot_data, title)
}
# -------------------------------------------------------------------
# The ggplot2 theme to keep the plot simple.
theme_complete_bw <- function(base_size = 12) {
    structure(list(axis.line = theme_blank(), axis.text.x = theme_text(size = base_size *
        0.8, lineheight = 0.9, colour = "black", vjust = 1), axis.text.y = theme_text(size =
        0.8, lineheight = 0.9, colour = "black", hjust = 1), axis.ticks = theme_segment(colo
        axis.title.x = theme_text(size = base_size, vjust = 0.5), axis.title.y = theme_text(
            angle = 90, vjust = 0.5), axis.ticks.length = unit(0.15, "cm"),
        axis.ticks.margin = unit(0.1, "cm"), legend.background = theme_rect(colour = NA),
        legend.key = theme_rect(fill = NA, colour = "black", size = 0.25), legend.key.size =
            "lines"), legend.text = theme_text(size = base_size * 0.8), legend.title = theme
            0.8, face = "bold", hjust = 0), legend.position = "right", panel.background = th
            colour = "black", size = 0.25), panel.border = theme_blank(), panel.grid.major =
            size = 0.05), panel.grid.minor = theme_line(colour = "black", size = 0.05),
        panel.margin = unit(0.25, "lines"), strip.background = theme_rect(fill = NA,
            colour = NA), strip.text.x = theme_text(colour = "black", size = base_size *
            0.8), strip.text.y = theme_text(colour = "black", size = base_size *
            0.8, angle = -90), plot.background = theme_rect(colour = NA, fill = "white"),
        plot.title = theme_text(size = base_size * 0.8), plot.margin = unit(c(1,
            1, 0.5, 0.5), "lines")), class = "options")
}
```