Supplementary materials: Full code base for simulations and figures

Melissa Chapman[a], Serge Wiltshire[a], Timothy Bowles[a], Liz Carlisle[c], Fredrico Castillo[a], Alastair Iles[a], Daniel Karp[d], Claire Kremen[a,1], Jeff Liebert[f], Elissa Olimpi[d], Matthew Ryan[f], Amber Sciligo[g], Jennifer Thompson[a], Hannah Waterhouse[a], Kenzo Eszquivel[a], Sasha Gennet[h], Carl Boettiger[a,]

[a] *Dept. of Environmental Science, Policy, and Management, University of California Berkeley, Berkeley, CA, USA*
[b] *Institute of Resources, Environment and Sustainability, University of British Columbia, Vancouver, BC, Canada*
[c] *Environmental Studies, University of California Santa Barbara, Santa Barbara, CA, USA*
[d] *Wildlife, Fish, and Conservation Biology, University of California Davis, Davis, CA, USA*
[e] *Dept. of Entomology, University of Kentucky, Lexington, KY, USA*
[f] *School of Integrated Plant Science, Cornell University, Cornell, NY, USA*
[g] *The Organic Center, Washington, DC, USA*
[h] *The Nature Conservancy, Arlington, VA, USA*

Abstract

This document gives the full codebase used to define and parameterize the Markov Decision Process (MDP) model, run all simulations, and produce the final plots that appear in the accompanying manuscript. We also provide key equations for the formulation of the MDP model.

Initializations

We rely primarily on the `tidyverse` and `MDPtoolbox` packages for our simulation. We use a a set of additional packages for plotting which can be found in `../R/load-libraries.R`.

```
rm(list = ls(all.names = TRUE)) # clean environment
# data / MDP libraries
library(tidyverse)
library(MDPtoolbox)
source("../R/load-libraries.R")
```

Set MDP parameters

We first define a vector of possible actions, A, and states, S

$$A \in [0, 1)$$
$$S \in [0, 1.5)$$

```
# define action and state space
actions <- seq(0, 1, length = 100) # vector of possible action values
states <- seq(0, 1.5, length = 100) # vector of possible state values
```

And define the set parameters for the MDP model:
b = benefit parameter c = cost parameter σ = ecological stochasticity r = ecological change rate γ = discount factor

```r
# set global mdp parameters
params <- list(
  benefit = 1.57, # benefit param for MDP
  cost = 1, # cost param for MDP
  sigma = 0.1, # ecological change stochasticity
  r = 0.1, # ecological change rate
  discount = 0.97, # myopic discount factor
  Tmax = 20 # max simulation time
  )
```

Define state transition probability and utility functions

We assume a simple model of the farmer's perceived utility $u(s_t, a_t)$ as a function of the difference between the cost c_a associated with diversification practice action a_t, versus expected benefits b_s derived from ecosystem state s_t, at time t, such that

$$u(s_t, a_t) = b_s s_t - c_a a_t$$

Agents' initial ecosystem states were distributed normally around a mean of $s = 0.5$. The ecosystem state is also dynamic, evolving according to the transition probability function $p(s_t, a_t)$, such that

$$P(s_{t+1}|s_t, a_t) = s_t + r\,(a_t - s_t) + \epsilon$$

where $\epsilon \sim N(0, \sigma)$

```r
#' Calculate transition probability based on current state and action.
#'
#' @param s The current state.
#' @param a The current action.
#' @param params Must be a list with element r.
#' @return The state transition probability.
transition_fn <- function(s, a, params) s + params$r * (a - s)
#' Calculate utility based on current state and action.
#'
#' @param s The current state.
#' @param a The current action.
#' @param params Must be a list with elements [benefit, cost].
#' @return The state utility.
utility_fn <- function(s, a, params) params$benefit * s - params$cost * a
```

Calculate transition and utility matrices

We can use the above equation to create both a transition and utility matrices.

```r
#' Calculate transition probability matrix and utility matrix based on MDP parameters.
#'
#' @param states A vector representing the state space.
#' @param actions A vector representing the action space.
#' @param params Must be a list with elements [benefit, cost, sigma].
#' @param transition_fn A function(state, action, params) giving the transition
#' probability to the next state.
```

```r
#' @param utility_fn A function(state, action, params) giving the value of the next state.
#' @return A list containing the transition probability matrix \code{P} and utility
#' matrix
continuous_model <- function(states, actions, params, transition_fn, utility_fn){
  transition_matrix <- function(states, actions, f, params){
    n_s <- length(states)
    n_a <- length(actions)
    transition <- array(0, dim = c(n_s, n_s, n_a))
    for(i in 1:n_a){
      for (k in 1:n_s) {
        nextpop <- transition_fn(states[k], actions[i], params)
        if (nextpop <= 0) {
          x  <- c(1, rep(0, n_s - 1))
        } else {
          x <- truncnorm::dtruncnorm(
            states, 0, max(states), nextpop, params$sigma) # assumes truncated normal error
          if(sum(x) <= 0){
            x  <- c(1, rep(0, n_s - 1))
          } else {
            x <- x / sum(x)
          }
        }
        transition[k, , i] <- x
      }
    }
    if(any(is.na(transition))) stop("error creating transition matrix")
    transition
  }
  utility_matrix <- function(states, actions, utility_fn, params){
    utility <- array(dim=c(length(states), length(actions)))
    for(i in 1:length(states)){
      for(j in 1:length(actions)){
        utility[i,j] <- utility_fn(states[i], actions[j], params)
      }
    }
    utility
  }
  list(P = transition_matrix(states, actions, f, params),
       U = utility_matrix(states, actions, utility_fn, params))
}
```

Define and solve MDP for an infinite time horizon case

We can solve for the optimal action strategy, π^*, which maps a give state, s to an optimal action a, using the a recursive Bellman equation (see Marescot et al, 2013 [@Marescot2013b] for a thorough introduction to these methods).

The Bellman equation can be rewritten as the sum of the utility value at the current state and the sum of the expected future rewards that are the products of transition probabilities and values of all possible next states

$$V_t^{\pi}(s_t) = \left[u(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, \pi(s_t)) V^{\pi}(s_{t+1}) \right]$$

The equation for the optimal action strategy is thus

$$V^{\pi^*}(s) = \max_a u(s, \pi(s)) + \gamma \left[\sum_{s'} P(s'|s, \pi(s)) V^{\pi^*}(s') \right]$$

where π^* is the optimal policy and V^{π^*} refers to the value function of the optimal policy. The equation above describes the reward for taking the action at a given state to get the highest expected return over either an infinite decision horizon or a given finite decision horizon.

More information on solving Markov Decision Process models can be found in Marescot et al, 2013.

```
# compute P and U matrices for the MDP
model <- continuous_model(states, actions, params, transition_fn, utility_fn)
# solve the MDP to find the optimal decision policy
base_soln <- MDPtoolbox::mdp_value_iteration(model$P, model$U, params$discount)
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```
base_soln_df <- tibble(state = states, action = actions[base_soln$policy])
```

Land tenure experiment: Calculating finite time horizon MDP solution

While the above solves for the optimal action strategy using a discounted infinite decision horizon, we can also find optimal action strategy for a discounted finite time horizon. In our main text, we set this time horizon to 10 time steps for this analysis (Figure 5).

Define main simulation function

```
#' Run DFS-MDP simulation for a single agent based on MDP simulation parameters.
#'
#' @param P Transition probability matrix, output from continuous_model().
#' @param U Utility matrix, output from continuous_model().
#' @param policy Optimal decision policy.
#' @param discount Myopic discount factor.
#' @param x0 A vector of initial states.
#' @param Tmax Max time for the simulation.
#' @return A dataframe with time, state, action, and value.
sim_mdp <- function(P, U, policy, discount, x0, Tmax){

  n_states <- dim(P)[1]
  state <- action <- value <- numeric(Tmax+1)
  state[1] <- x0
  tsteps <- 1:(Tmax+1)
  for(t in tsteps){
    # select action, determine value, transition to next state
    action[t] <- policy[state[t]]
    value[t] <- U[state[t], action[t]] * discount^(t-1)
```

```
    state[t+1] <- sample(1:n_states, 1, prob = P[state[t], , action[t]])
  }
  data.frame(time = 0:Tmax, state = state[tsteps],
             action = action[tsteps], value = value[tsteps])
}
```

Set global simulation parameters

```
reps <- 500 # number of replicate simulations to run
init <- truncnorm::rtruncnorm(reps, 0, 1, 0.5, 0.2) %>% # sample init s from norm dist
  map_int(function(x) which.min(abs(x - states))) # map to values in states vector
```

Run base-case simulation
Define timeseries and density plotting functions

```
# timeseries plot of multiple repetitions
sim_plot_ts <- function(sims, title = ggtitle(NULL), tpos = "plot", ytxtoff = FALSE,
                        endcol = pal[1], annotate = FALSE, an_x = NULL, an_lab = NULL,
                        dnmarmod = 0, upmarmod = 0, lmarmod = 0, rmarmod = 0){
  df <- sims %>%
    select(-value) %>% # tidy
    mutate(state = states[state], action = actions[action]) # rescale
  Tmax <- max(sims$time)
  stcol <- col2rgb(endcol)
  stcol <- stcol/5
  stcol <- rgb(t(stcol), maxColorValue=255)
  ytitc <- "black"
  ytxtc = "gray30"
  if (ytxtoff) {
    ytitc <- NA
    ytxtc <- NA
  }
  p <- df %>%
    ggplot(aes(time, state, group = reps, col = time)) +
    geom_path(alpha = 0.1, show.legend = FALSE) +
    title +
    labs(x="Decision cycle", y="Ecosystem service state") +
    scale_x_continuous(limits = c(0,Tmax), breaks = seq(0, Tmax, by=5),
                       expand = c(.01,.01)) +
    scale_y_continuous(limits = c(0,1.5), expand = c(.02,.02)) +
    scale_color_gradient(low=stcol, high=endcol) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_text(size=10, color = ytxtc),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10, color = ytitc),
          plot.title = element_text(size = 10, face = "bold"),
          plot.title.position = tpos,
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5+upmarmod,5+rmarmod,5+dnmarmod,5+lmarmod), "mm"))
  if(annotate) {
```

```r
    p <- p +
      geom_vline(xintercept = an_x, linetype="dashed", color = "red3", size=.5) +
      annotate('label', x = an_x + 2, y = 1.3, label = an_lab, hjust = 0, vjust = .5,
               family = "Roboto", size = 3.25, label.padding = unit(.15, "lines"),
               label.size = 0, alpha = .6) +
      annotate("segment", x = an_x + 1.75, xend = an_x + .5, y = 1.3, yend = 1.3,
               size=.5, arrow=arrow(length = unit(0.22, "cm")))
  }
  p
}
# density plot showing initial ES distribution and final distribution
sim_plot_dens <- function(sims, title = ggtitle(NULL), tpos = "plot", endcol = pal[1],
                          lab_lo_peak = FALSE, lab_hi_peak = FALSE,
                          dnmarmod = 0, upmarmod = 0, lmarmod = 0, rmarmod = 0){
  df <- sims %>%
    mutate(state = states[state]) %>% # rescale
    select(state, time)
  Tmax <- max(sims$time)
  stcol <- col2rgb(endcol)
  stcol <- stcol/5
  stcol <- rgb(t(stcol), maxColorValue=255)
  p <- df %>% filter(time %in% c(0, Tmax))  %>%
    ggplot() + geom_density(aes(state, group = time, fill = time, color = time),
                            alpha=0.8) +
    coord_flip() +
    title +
    labs(x="", y="Density", fill="Time") +
    scale_x_continuous(limits = c(0,1.5), expand = c(.02,.02)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 2)) +
    scale_fill_gradient(low=stcol, high=endcol, guide = guide_colorbar(barwidth = .5),
                        breaks=c(0, Tmax)) +
    scale_color_gradient(low=stcol, high=endcol, guide = NULL) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_blank(),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10),
          legend.text = element_text(size=10),
          legend.title = element_text(size=10),
          legend.box.margin=margin(0,0,0,-5),
          plot.title = element_text(size = 10, face = "bold"),
          plot.title.position = tpos,
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5+upmarmod,5+rmarmod,5+dnmarmod,5+lmarmod), "mm"))
  if(lab_lo_peak) {
    ymax <- ggplot_build(p)$layout$panel_scales_y[[1]]$range$range[2]
    peak_lo <- ggplot_build(p)$data[[1]] %>%
      filter(group == 2, x < .4) %>%
      arrange(desc(y)) %>%
      select(x,y) %>%
      filter(row_number()==1)
    # dotted line
```

```
    p <- p +
      annotate('segment', x = peak_lo$x, xend = peak_lo$x, y = peak_lo$y - .1 * ymax,
               yend = peak_lo$y + .1 * ymax, linetype="dotted")
    # text annotation (decide whether to place left or right of line)
    if(peak_lo$y > .7 * ymax) {
      p <- p +
        annotate('label', x = peak_lo$x, y = peak_lo$y - .32 * ymax,
                 label = sprintf('%.2f', peak_lo$x), hjust = .5, vjust = .5,
                 family = "Roboto Condensed", size = 3,
                 label.padding = unit(.15, "lines"), label.size = 0, alpha = .8)
    } else {
      p <- p +
        annotate('label', x = peak_lo$x, y = peak_lo$y + .26 * ymax,
                 label = sprintf('%.2f', peak_lo$x), hjust = .5, vjust = .5,
                 family = "Roboto Condensed", size = 3,
                 label.padding = unit(.15, "lines"), label.size = 0, alpha = .8)
    }
  }
  if(lab_hi_peak){
    ymax <- ggplot_build(p)$layout$panel_scales_y[[1]]$range$range[2]
    peak_hi <- ggplot_build(p)$data[[1]] %>%
      filter(group == 2, x > .6) %>%
      arrange(desc(y)) %>%
      select(x,y) %>%
      filter(row_number()==1)
    # dotted line
    p <- p +
      annotate('segment', x = peak_hi$x, xend = peak_hi$x, y = peak_hi$y - .1 * ymax,
               yend = peak_hi$y + .1 * ymax, linetype="dotted")
    # text annotation (decide whether to place left or right of line)
    if(peak_hi$y > .7 * ymax) {
      p <- p +
        annotate('label', x = peak_hi$x, y = peak_hi$y - .32 * ymax,
                 label = sprintf('%.2f', peak_hi$x), hjust = .5, vjust = .5,
                 family = "Roboto Condensed", size = 3,
                 label.padding = unit(.15, "lines"), label.size = 0, alpha = .8)
    } else {
      p <- p +
        annotate('label', x = peak_hi$x, y = peak_hi$y + .26 * ymax,
                 label = sprintf('%.2f', peak_hi$x), hjust = .5, vjust = .5,
                 family = "Roboto Condensed", size = 3,
                 label.padding = unit(.15, "lines"), label.size = 0, alpha = .8)
    }
  }
  p
}
```

Define base-base optimal decision strategy plotting function

```
# solution point plot with threshold annotated
soln_plot <- function(soln_df, tpt) {
```

```
  ggplot(soln_df, aes(state,action)) +
    geom_point(size = 1) +
    geom_vline(xintercept = tpt, linetype="dashed", color = "red3", size=.7) +
    annotate('label', x = tpt + .15, y = .375, label = "Tipping point",
             hjust = 0, vjust = .5, family = "Roboto", size = 3.25,
             label.padding = unit(.15, "lines"), label.size = 0, alpha = .65) +
    annotate("segment", x = tpt + .15, xend = tpt + .025, y = .375, yend = .375,
             size=.5, arrow=arrow(length = unit(0.22, "cm"))) +
    labs(x="Ecosystem service state", y="Optimal adoption") +
    scale_x_continuous(limits = c(0,NA), expand = c(.01,.01)) +
    scale_y_continuous(limits = c(0,1), expand = c(.01,.01),
                       labels = scales::percent_format(accuracy = 1)) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_text(size=10),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10),
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5,5,5,5), "mm"))
}
```

Generate optimal policy plot

```
state99th <- sims_baseline %>%
  mutate(state = states[state]) %>%
  pull(state) %>%
  quantile(.99) %>%
  unname()
tpt <- 0
for(si in 1:length(states)-1) {
  if(base_soln_df$action[si] < .1 && base_soln_df$action[si+1] > .9) {
    tpt <- base_soln_df$state[si] + ((base_soln_df$state[2] - base_soln_df$state[1]) / 2)
  }
}
soln_plot(base_soln_df %>% filter(state <= state99th), tpt = tpt)
```

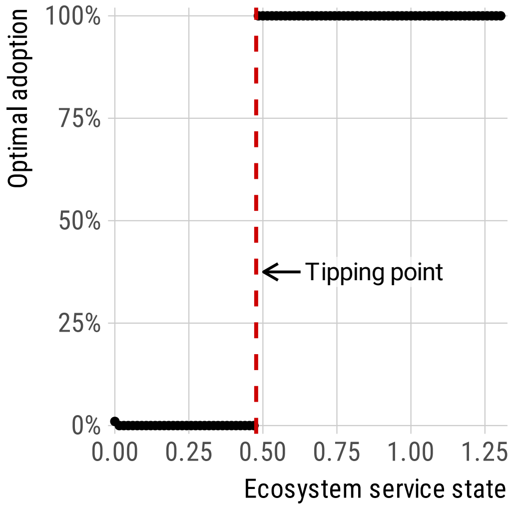


Figure 1: Optimal decision strategy π as a function of ES state, showing a tipping point at $s \approx 0.48$. The upper x axis limit is the 99th percentile of observed states in our simulation results ($s \approx 1.3$).

Generate base-case simulation figure

```
ggarrange(
  soln_plot(base_soln_df %>% filter(state <= state99th), tpt = tpt),
  sim_plot_ts(sims_baseline, title = ggtitle("B. Timeseries"), rmarmod = -5),
  sim_plot_dens(sims_baseline, title = ggtitle("C. State distributions"),
                tpos = "panel", lab_lo_peak = TRUE, lab_hi_peak = TRUE, lmarmod = -5),
  widths = c(3,2.75,2.2), nrow = 1, ncol = 3
)
```

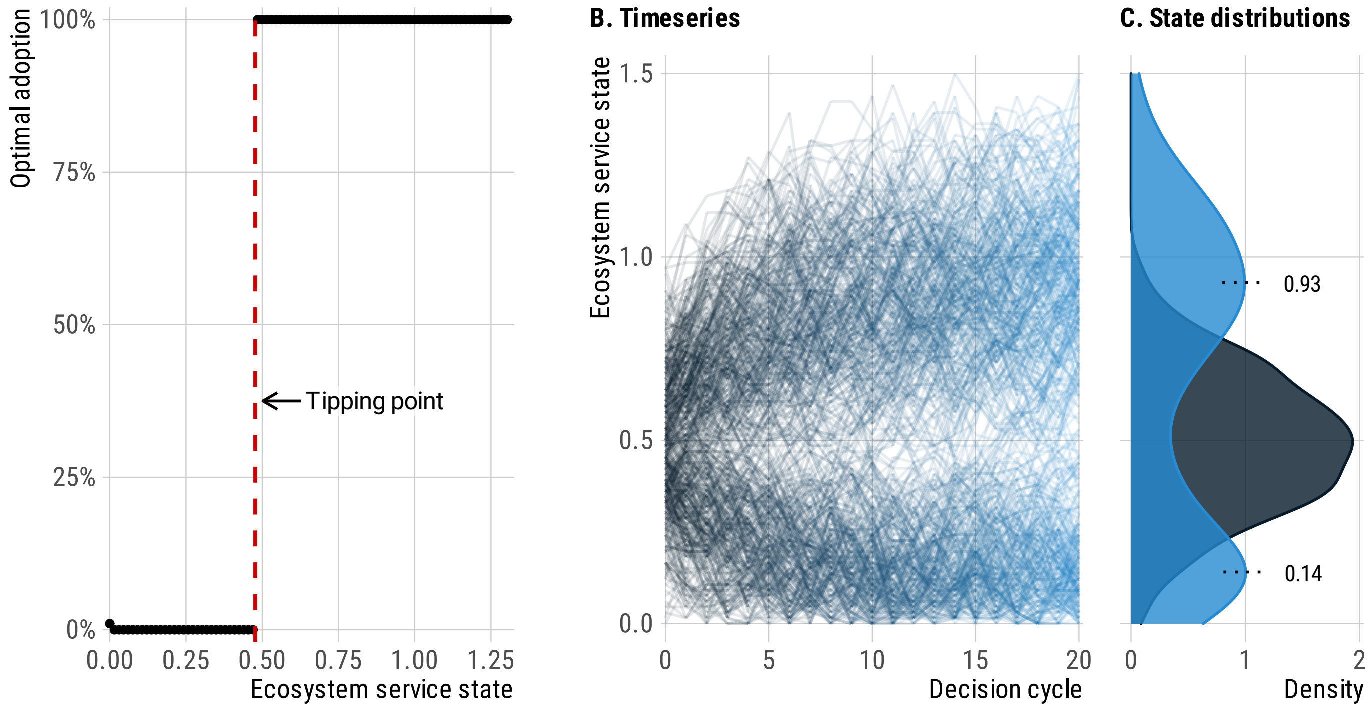


Figure 2: Initial ecosystem states are distributed normally (mean = 0.5; S.D. = 0.2; truncated at [0,1]). Agents follow decision strategy π as shown in Fig 1 until $t = 20$. (A) Ecosystem state of each agent over time (500 simulations). (B) Initial ES distribution (dark blue) and final bimodal distribution at $t = 20$ (light blue).

Simulate short time horizon case

```
sims_short_tenure <- sim(short_tenure_soln)
```

Define density curve comparison plotting function

```
# plot formatting helper functions
# increase vertical spacing between legend keys
# written by @clauswilke
draw_key_polygon3 <- function(data, params, size) {
  lwd <- min(data$size, min(size) / 4)
  grid::rectGrob(
    width = grid::unit(0.3, "npc"),
    height = grid::unit(0.8, "npc"),
    gp = grid::gpar(
      col = data$colour,
      fill = alpha(data$fill, data$alpha),
      lty = data$linetype,
      lwd = lwd * .pt,
      linejoin = "mitre"
    ))
}
```

```r
# register new key drawing function,
# the effect is global & persistent throughout the R session
GeomDensity$draw_key = draw_key_polygon3
# plot comparing final and initial density curves of two simulations
sim_plot_dens_comp <- function(sims1, sims2, sims_base = "init",
                               label1 = "Gp. A Final", label2 = "Gp. B Final",
                               label_base = "Initial", title = ggtitle(NULL),
                               tpos = "plot", cvec = c(pal[2], pal[1], pal[3]),
                               dnmarmod = 0, upmarmod = 0, lmarmod = 0, rmarmod = 0){
  if (sims_base == "init") {
    df_base <- sims1 %>%
      mutate(state = states[state]) %>% # rescale
      filter(time %in% 0) %>%
      select(state) %>%
      add_column(id = label_base)
  } else {
    Tmax_base <- max(sims_base$time)
    df_base <- sims_base %>%
      mutate(state = states[state]) %>% # rescale
      filter(time %in% Tmax_base) %>%
      select(state) %>%
      add_column(id = label_base)
  }
  Tmax1 <- max(sims1$time)
  df1 <- sims1 %>%
    mutate(state = states[state]) %>% # rescale
    filter(time %in% Tmax1) %>%
    select(state) %>%
    add_column(id = label1)
  Tmax2 <- max(sims2$time)
  df2 <- sims2 %>%
    mutate(state = states[state]) %>% # rescale
    filter(time %in% Tmax1) %>%
    select(state) %>%
    add_column(id = label2)
  rbind(df_base, df1, df2) %>% mutate(id = factor(id, unique(id))) %>%
    ggplot() + geom_density(aes(state, group = id, fill = id, color = id), alpha=0.4) +
    coord_flip() +
    title +
    labs(x="", y="Density") +
    scale_x_continuous(limits = c(0,1.5), expand = c(.02,.02)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 2)) +
    scale_fill_manual(values = cvec) +
    scale_color_manual(values = cvec) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_blank(),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10),
          legend.text=element_text(size=9),
          legend.title = element_blank(),
          legend.key.size = unit(12, "mm"),
```

```
          legend.spacing.x = unit(-3, 'mm'),
          legend.box.margin=margin(0,0,0,-18),
          plot.title = element_text(size = 10, face = "bold"),
          plot.title.position = tpos,
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5+upmarmod,5+rmarmod,5+dnmarmod,5+lmarmod), "mm"))
}
```

Generate land tenure experiment plot

```
ggarrange(
  sim_plot_ts(sims_baseline, title = ggtitle("A. Long-tenure"),
            annotate = TRUE,
              rmarmod = -10),
  sim_plot_ts(sims_short_tenure,
              title = ggtitle("B. Short-tenure"),
              endcol = pal[3],
              tpos = "panel", ytxtoff = T,rmarmod = -5, lmarmod = -5),
  sim_plot_dens_comp(sims_baseline, sims_short_tenure,
                     label1 = "Long\ntenure\nfinal",
                     label2 = "Short\ntenure\nfinal",
                     title = ggtitle("C. State distribution"),
                     tpos = "panel", lmarmod = -5),
 nrow = 1, ncol = 3, widths = c(1,1,1))
```

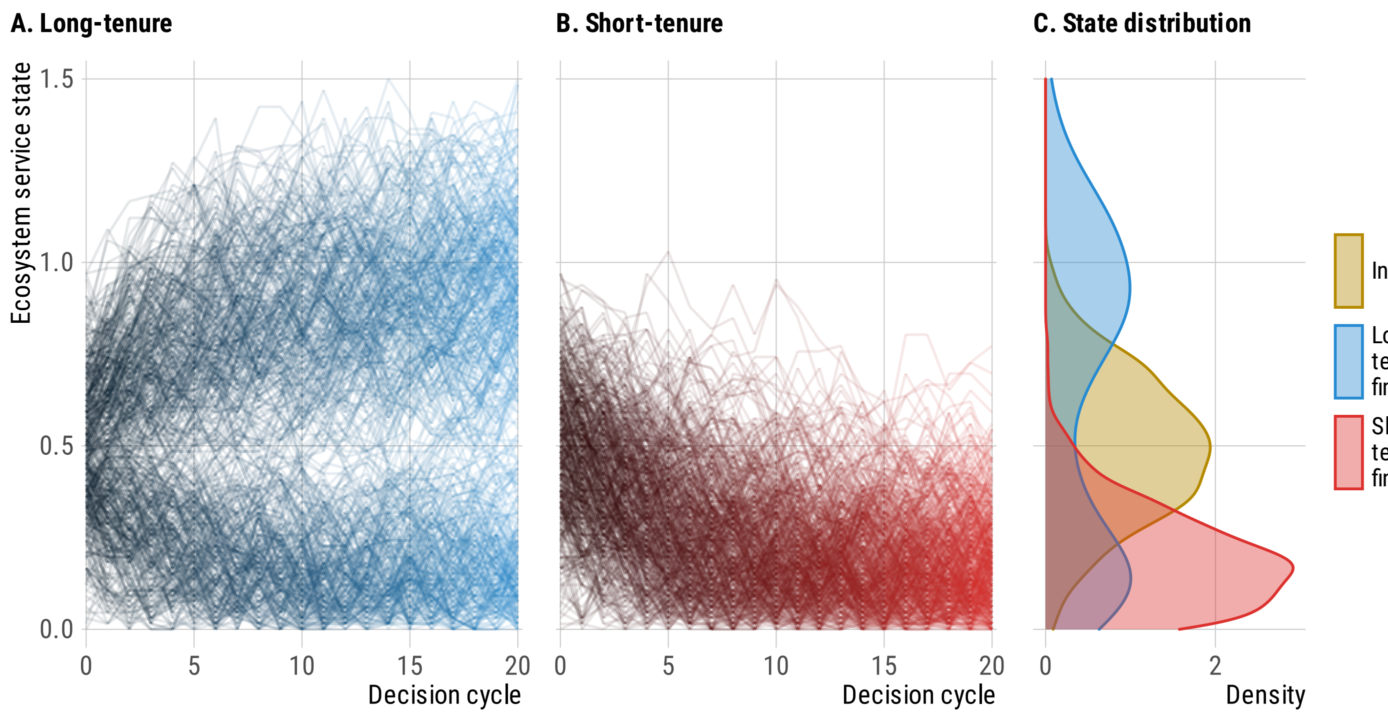


Figure 3: The simulation is identical to that in Fig 2, but the MDP is solved under a finite, 20-year time horizon. (A) Result of short land tenure on ES state over time. (B) Comparison between final state distribution of short- vs. long-tenure model runs.

Incentive duration experiment

Incentive simulation parameters and setup

```
# set incentive simulation parameters
T_incent_1 <- 2
T_incent_2 <- 10
# calc modified cost during incentive = 1 - 2 / T_incent
C_incent_1 <- 1 - 2 / T_incent_1
C_incent_2 <- 1 - 2 / T_incent_2
# incentive simulation fxn
incent_sim <- function(params) {
  incent_model <- continuous_model(states, actions, params,
                                   transition_fn, utility_fn)
  incent_soln <- MDPtoolbox::mdp_value_iteration(incent_model$P,
                                                 incent_model$U, params$discount)
  # simulate incentive period
  start <- sim(incent_soln, Tmax = params$T_incent, x0 = init)
  xi <- start %>% filter(time == params$T_incent) %>% pull(state)
  # pad to Tmax with baseline decision rule
  rest <- sim(base_soln, Tmax = params$Tmax - params$T_incent, x0 = xi) %>%
    filter(time!=0) %>% mutate(time = time + params$T_incent)
```

```r
  bind_rows(start, rest)
}
```

Run incentives simulation

```r
# run incentive simulation
sims_incent_abrupt <- incent_sim(list(benefit = params$benefit,
                                      cost = C_incent_1,
                                      sigma = params$sigma,
                                      r = params$r,
                                      discount = params$discount,
                                      T_incent = T_incent_1, Tmax = params$Tmax))
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```r
sims_incent_sust <- incent_sim(list(benefit = params$benefit,
                                    cost = C_incent_2,
                                    sigma = params$sigma,
                                    r = params$r,
                                    discount = params$discount,
                                    T_incent = T_incent_2, Tmax = params$Tmax))
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

Plot incentive simulation

```r
ggarrange(
  sim_plot_ts(sims_incent_abrupt, title = ggtitle("A. Concentrated incentive"),
              annotate = TRUE, an_x = T_incent_1,
              an_lab = sprintf("%i cycles\n%i%% off", T_incent_1,
                               round((1-C_incent_1)*100)),
              rmarmod = -10, endcol = pal[5]),
  sim_plot_ts(sims_incent_sust, title = ggtitle("B. Sustained incentive"),
              annotate = TRUE, an_x = T_incent_2,
              an_lab = sprintf("%i cycles\n%i%% off", T_incent_2,
                               round((1-C_incent_2)*100)),
              tpos = "panel", ytxtoff = T, endcol = pal[6], rmarmod = -5, lmarmod = -5),
  sim_plot_dens_comp(sims_incent_abrupt, sims_incent_sust, sims_base = sims_baseline,
                     label1 = "Concentrated \nincentive", label2 = "Sustained\nincentive",
                     label_base = "No\nincentive",
                     title = ggtitle("C. Final distribution"), tpos = "panel",
                     cvec = c(pal[2], pal[5], pal[6]), lmarmod = -5, rmarmod = -5),
  nrow = 1, ncol = 3, widths = c(1,1,1)
)
```

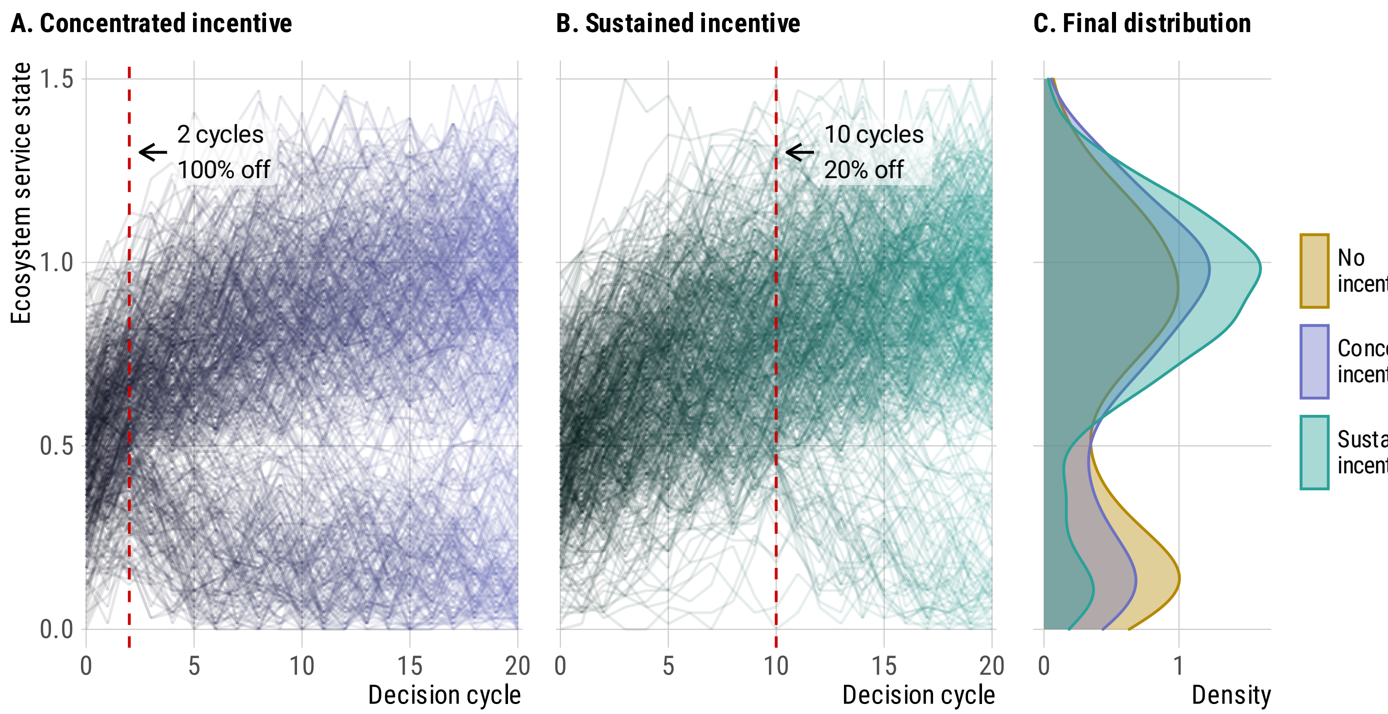


Figure 4: Starting from the same initial states as Fig 2, ES state timeseries are shown for (A) a large, abrupt incentive (100% of adoption expenses are covered for two years) vs. (B) a smaller, more sustained incentive (adoption cost is 80% of baseline for 10 years). Ignoring discounting, both packages have the same total cost to the funder (the equivalent of 2 years' worth of full adoption cost offsets). After the incentive period, agents adjust their decision rules to that of the base case (i.e. no incentive) until $t = 20$. (C) Shows that the sustained incentive ultimately drove more DP adoption.

Temporal dynamics experiment

Define "benefit sweep" simulation function

```
sim_b_sweep <- function(scenarios, inf_fin, t_horiz = 2) {
  sim_b_sweep_scenario <- function(params) {
    model <- continuous_model(states, actions, params, transition_fn, utility_fn)
    if(inf_fin == "infinite") {
      soln <- MDPtoolbox::mdp_value_iteration(model$P, model$U, params$discount)
    } else if(inf_fin == "finite") {
      soln <- MDPtoolbox::mdp_finite_horizon(model$P, model$U, params$discount,
                                             N = t_horiz)
    } else stop('inf_fin must be "infinite" or "finite"')
    sim(soln, Tmax = params$Tmax) %>%
      filter(time == params$Tmax)
  }

  scenarios %>%
    purrr::transpose() %>%
    furrr::future_map_dfr(sim_b_sweep_scenario, .id = "id") %>%
    left_join(scenarios, by = "id") %>%
```

```
    mutate(state = states[state])
}
```

Parameterize and run "benefit-sweep" simulation scenarios

```
chans_scenarios <-
  tidyr::crossing(cbr = seq(.55, .7, length.out = 40),
                  cost = params$cost,
                  sigma = params$sigma,
                  r = params$r,
                  discount = params$discount,
                  Tmax = params$Tmax) %>%
  tibble::rownames_to_column("id") %>%
  mutate(benefit = cost / cbr)
chans_sims <- sim_b_sweep(chans_scenarios, "infinite")
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```
fast_r_scenarios <-
  tidyr::crossing(cbr = seq(.825, .975, length.out = 40),
                  cost = params$cost,
                  sigma = params$sigma,
                  r = 0.95,
                  discount = params$discount,
                  Tmax = params$Tmax) %>%
  tibble::rownames_to_column("id") %>%
  mutate(benefit = cost / cbr)
fast_r_sims <- sim_b_sweep(fast_r_scenarios, "infinite")
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
## [1] "MDP Toolbox: iterations stopped, epsilon-optimal policy found"
```

```
myopic_d_scenarios <-
  tidyr::crossing(cbr = seq(.025, .175, length.out = 40),
                  cost = params$cost,
                  sigma = params$sigma,
                  r = params$r,
                  discount = params$discount,
                  Tmax = params$Tmax) %>%
  tibble::rownames_to_column("id") %>%
  mutate(benefit = cost / cbr)
myopic_d_sims <- sim_b_sweep(myopic_d_scenarios, "finite", t_horiz = 2)
```

```
sim_b_sweep_t <- function(scenarios, inf_fin) {
  sim_b_sweep_scenario <- function(params) {
    model <- continuous_model(states, actions, params, transition_fn, utility_fn)
    if(inf_fin == "infinite") {
      soln <- MDPtoolbox::mdp_value_iteration(model$P, model$U, params$discount)
    } else if(inf_fin == "finite") {
      soln <- MDPtoolbox::mdp_finite_horizon(model$P, model$U, params$discount,
                                              N = params$t_horiz)
    } else stop('inf_fin must be "infinite" or "finite"')
    sim(soln, Tmax = params$Tmax) %>%
      filter(time == params$Tmax)
  }

  scenarios %>%
    purrr::transpose() %>%
    furrr::future_map_dfr(sim_b_sweep_scenario, .id = "id") %>%
    left_join(scenarios, by = "id") %>%
    mutate(state = states[state])
}
```

Define "benefit sweep" simulation helper and plotting functions

```
# find peaks fxn for benefit sweep experiment
b_sweep_peaks <- function(sims, smoothing = 1.5,
                          ythresh = 0.125, output_bizone = FALSE) {

  bimin = 0
  bimax = 0
  peaks <- tibble(cbr = numeric(), peak = numeric())
  for(this_cbr in unique(sims$cbr)) {
    dens <- density((sims %>% filter(cbr == this_cbr))$state, adjust = smoothing)
    dens <- tibble(x = dens$x, y = dens$y) %>% filter(y >= ythresh)
    ps <- dens$x[findPeaks(dens$y)]
    for(p in ps) {
```

```
      peaks <- add_row(peaks, cbr = this_cbr, peak = p)
    }
    if(output_bizone && length(ps) > 1) {
      if(bimin == 0) {
        bimin = this_cbr
      }
      bimax = this_cbr
    }
  }
  if(output_bizone) {
    c(bimin, bimax)
  } else {
    peaks
  }
}
```

```
# plot benefit sweep experiment results
plt_sim_b_sweep <- function(sims, title = ggtitle(NULL), ylab = "",
                            tpos = "plot", col = pal[1],
                            upmarmod = 0, lmarmod = 0, dnmarmod = 0, rmarmod = 0) {
  dkcol <- col2rgb(col)
  dkcol <- dkcol/5
  dkcol <- rgb(t(dkcol), maxColorValue=255)
  sims %>%
    ggplot(aes(x = state, group = cbr, color = cbr)) +
    geom_line(stat='density', size=.5, alpha=.4) +
    title +
    labs(x = "ES state", y = ylab, color = "c:b") +
    scale_x_continuous(limits = c(0, 1.5),
                       breaks = scales::pretty_breaks(n = 3), expand = c(.01,.01)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 2), expand = c(.01,.01)) +
    scale_color_gradient(low = dkcol, high = col, guide = guide_colorbar(barwidth = .5),
                         breaks=c(min(sims$cbr), max(sims$cbr))) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_text(size=10),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10),
          legend.text = element_text(size=10),
          legend.title = element_text(size=10),
          legend.box.margin=margin(0,0,0,-5),
          plot.title = element_text(size = 10, face = "bold"),
          plot.title.position = tpos,
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5+upmarmod,5+rmarmod,5+dnmarmod,5+lmarmod), "mm"))
}
# plot benefit sweep experiment peaks by cbr
plt_b_sweep_peaks <- function(peaks, bizone = NULL, title = ggtitle(NULL), ylab = "",
                              tpos = "plot", col = pal[1],
                              upmarmod = 0, lmarmod = 0, dnmarmod = 0, rmarmod = 0) {
  dkcol <- col2rgb(col)
  dkcol <- dkcol/5
```

```
  dkcol <- rgb(t(dkcol), maxColorValue=255)
  p <- peaks %>%
    ggplot(aes(x = peak, y = cbr, color = cbr)) +
    geom_point(size=1) +
    title +
    labs(x = "state density peak(s)", y = ylab) +
    scale_x_continuous(limits = c(0, 1.5),
                       breaks = scales::pretty_breaks(n = 3), expand = c(.01,.01)) +
    scale_y_continuous(limits = c(min(peaks$cbr), max(peaks$cbr)),
                       breaks = scales::pretty_breaks(n = 3), expand = c(.01,.01)) +
    scale_color_gradient(low = dkcol, high = col, guide = NULL) +
    theme(axis.text.x=element_text(size=10),
          axis.text.y=element_text(size=10),
          axis.title.x=element_text(size=10),
          axis.title.y=element_text(size=10),
          plot.title = element_text(size = 10, face = "bold"),
          plot.title.position = tpos,
          panel.grid.minor = element_blank(),
          plot.margin=grid::unit(c(5+upmarmod,5+rmarmod,5+dnmarmod,5+lmarmod), "mm"))
  if(!is.null(bizone)) {
    p <- p +
      geom_hline(yintercept = bizone[1], linetype="dashed", color = "red3", size=.5) +
      geom_hline(yintercept = bizone[2], linetype="dashed", color = "red3", size=.5) +
      annotate('label', x = .55, y = mean(bizone), label = "Bimodal\nregion",
               hjust = .5, vjust = .5, family = "Roboto", size = 3,
               label.padding = unit(.15, "lines"), label.size = 0, alpha = .65)
  }
  p
}
```

Generate "benefit sweep" simulation plot

```
ggarrange(
  ggarrange(
    plt_sim_b_sweep(chans_sims,
                    title = ggtitle("A. Forward-looking decision-maker
                                    \n     and adaptive ecological system"),
                    ylab = "Density", col = pal[1], rmarmod = -7, dnmarmod = -2.75) +
      theme(legend.position="none"),
    plt_b_sweep_peaks(b_sweep_peaks(chans_sims),
                      bizone = b_sweep_peaks(chans_sims, output_bizone = TRUE),
                      ylab = "Cost/benefit ratio", col = pal[1],
                      rmarmod = -7, upmarmod = -2.75),
    nrow = 2, ncol = 1, heights = c(1.1,1), align = "v"
  ),
  ggarrange(
    plt_sim_b_sweep(myopic_d_sims,
                    title = ggtitle("B. Short-term
                                    \n     decision strategy"),
                    tpos = "panel", col = pal[4],
                    lmarmod = -3.5, rmarmod = -3.5, dnmarmod = -2.75) +
```

```
      theme(legend.position="none"),
    plt_b_sweep_peaks(b_sweep_peaks(myopic_d_sims),
                      col = pal[4], lmarmod = -3.5, rmarmod = -3.5, upmarmod = -2.75),
    nrow = 2, ncol = 1, heights = c(1.1,1), align = "v"
  ),
  ggarrange(
    plt_sim_b_sweep(fast_r_sims,
                    title = ggtitle("C. fast
                                    \n     ecological change rate"),
                    tpos = "panel", col = pal[7], lmarmod = -7, dnmarmod = -2.75) +
      theme(legend.position="none"),
    plt_b_sweep_peaks(b_sweep_peaks(fast_r_sims),
                      col = pal[7], lmarmod = -7, upmarmod = -2.75),
    nrow = 2, ncol = 1, heights = c(1.1,1), align = "v"
  ),
  nrow = 1, ncol = 3, widths = c(1,1,1)
)
```

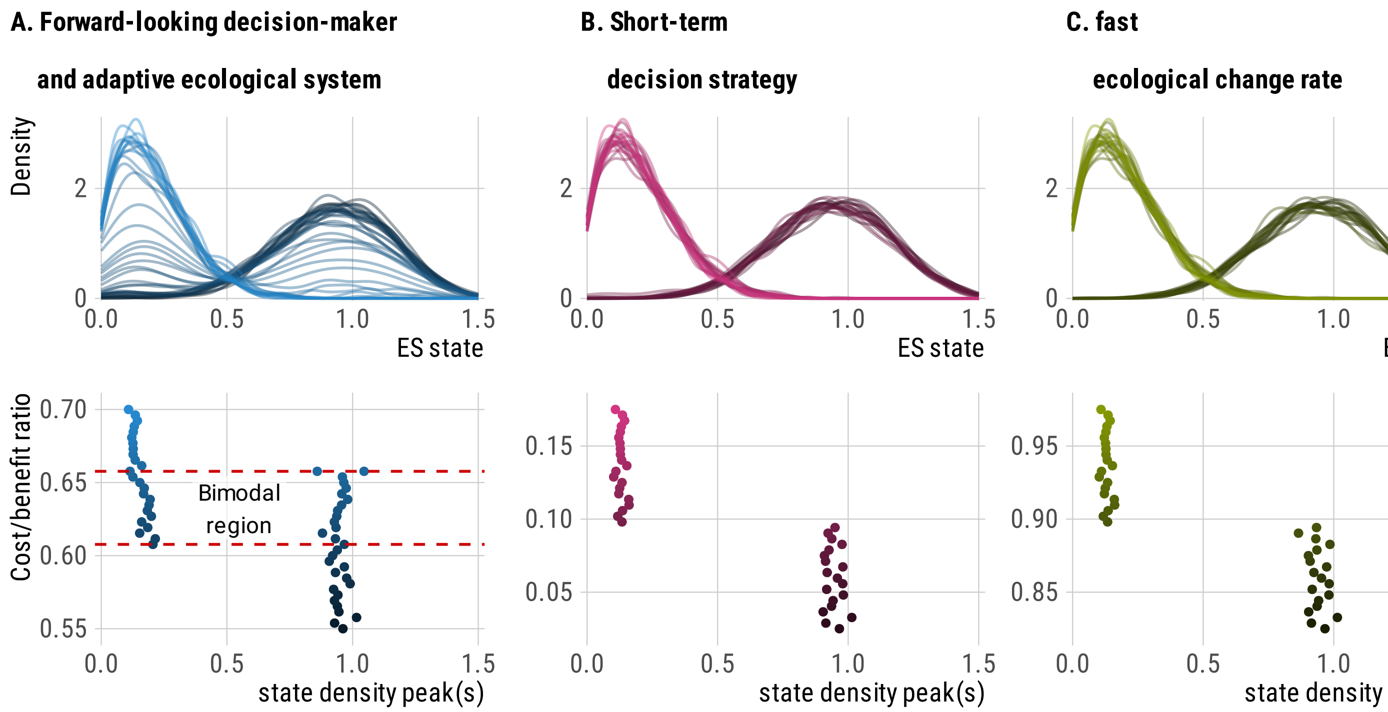


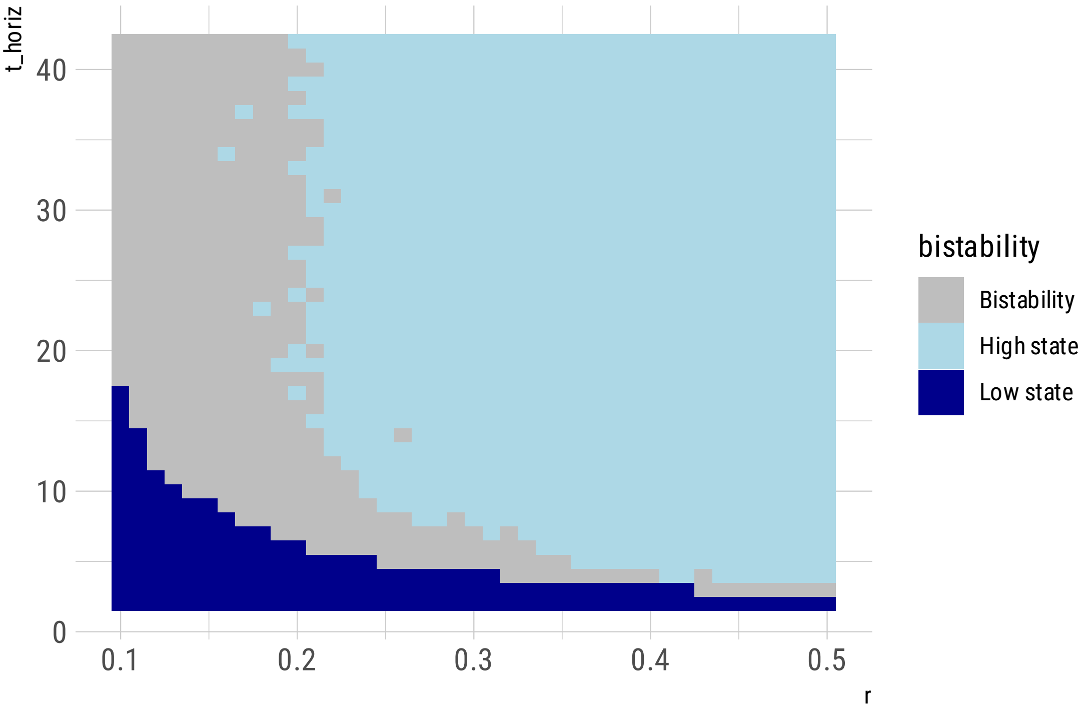
Figure 5: For three scenarios (coupled human/natural system, overly-myopic decision maker, and overly-fast ecological change), cost/benefit ratio was varied incrementally over 40 values, indicated by color shade, across a $c : b$ range of width 0.15, encompassing the transition between a "never invest" to an "always invest" policy. For each $c : b$, 500 replicate simulations were conducted as in Fig 2. Upper plots show distribution of ES state at $t = 20$ for each $c : b$. Lower plots show density curve peak(s). (A) By coupling a forward-looking decision-maker and a slowly-adapting environment, complex dynamics like alternate stable states can emerge. However, with (B) a short-term decision strategy (solving the MDP over a 2-year time horizon), or (C) a fast ecological change rate ($r = 0.95$), no bimodality is observed.

```r
grid_scenarios <-
  tidyr::crossing(benefit = params$benefit,
                  cost = params$cost,
                  sigma = params$sigma,
                  r = seq(0.1,0.50, length.out = 41),
                  discount = params$discount,
                  Tmax = params$Tmax,
                  t_horiz = seq(2,42, length.out = 41)) %>%
  tibble::rownames_to_column("id")

if (!file.exists("../manuscript/grid_sims.rds")) {
  grid_sims <- sim_b_sweep_t(grid_scenarios, inf_fin = "finite")
  write_rds(grid_sims, "../manuscript/grid_sims.rds")
}

grid_sims<- readRDS("../manuscript/grid_sims.rds")
```

```r
grid_sims %>%
  group_by(r, t_horiz) %>%
  summarise(mean_action = mean(action, na.rm = TRUE),
            mean_state = mean(state, na.rm = TRUE),
            min_action = min(action, na.rm = TRUE),
            max_action = max(action, na.rm = TRUE)) %>%
  mutate(action_var = max_action - min_action) %>%
  mutate(bistability = ifelse(action_var > 20, "Bistability",
                              ifelse(min_action < 10, "Low state", "High state"))) %>%
  ggplot(aes(x= r, y = t_horiz, fill = bistability)) + geom_tile() +
  scale_fill_manual(values=c("grey", "lightblue", "darkblue"))
```


t_horiz
40
30
20
10
0
0.1
0.2
0.3
0.4
0.5
r
bistability
Bistability
High state
Low state