A Eastern Grey Kangaroo population simulation example

Casey Visintin

2018-11-29

Below is an example using the eastern grey kangaroo (EGK) - a large marsupial native to Australia.

First we setup the stage based transition matricies. The first matrix represents survival and fecundity life-stage transition probabilities for kangaroos. The second matrix describes the uncertainty around the transition probabilities and is used to simulate environmental stochasticity - zeros indicate no uncertainty. Note, there are three life-stages and the matrices are symmetrical in their row and column numbers. Also, names are added to the columns and rows to identify the different life-stages of the kangaroo. These two objects are available in the steps package ("egk_mat" and "egk_mat_stoch").

```
egk_mat <- matrix(c(0.00,0.00,1.00,
                    0.50,0.00,0.00,
                    0.00,0.85,0.85),
                  nrow = 3,
                  ncol = 3,
                  byrow = TRUE)
colnames(egk_mat) <- rownames(egk_mat) <- c('juvenile','subadult','adult')

egk_mat_stoch <- matrix(c(0.00,0.00,0.20,
                          0.05,0.00,0.00,
                          0.00,0.10,0.05),
                  nrow = 3,
                  ncol = 3,
                  byrow = TRUE)
colnames(egk_mat_stoch) <- rownames(egk_mat_stoch) <- c('juvenile','subadult','adult')
```

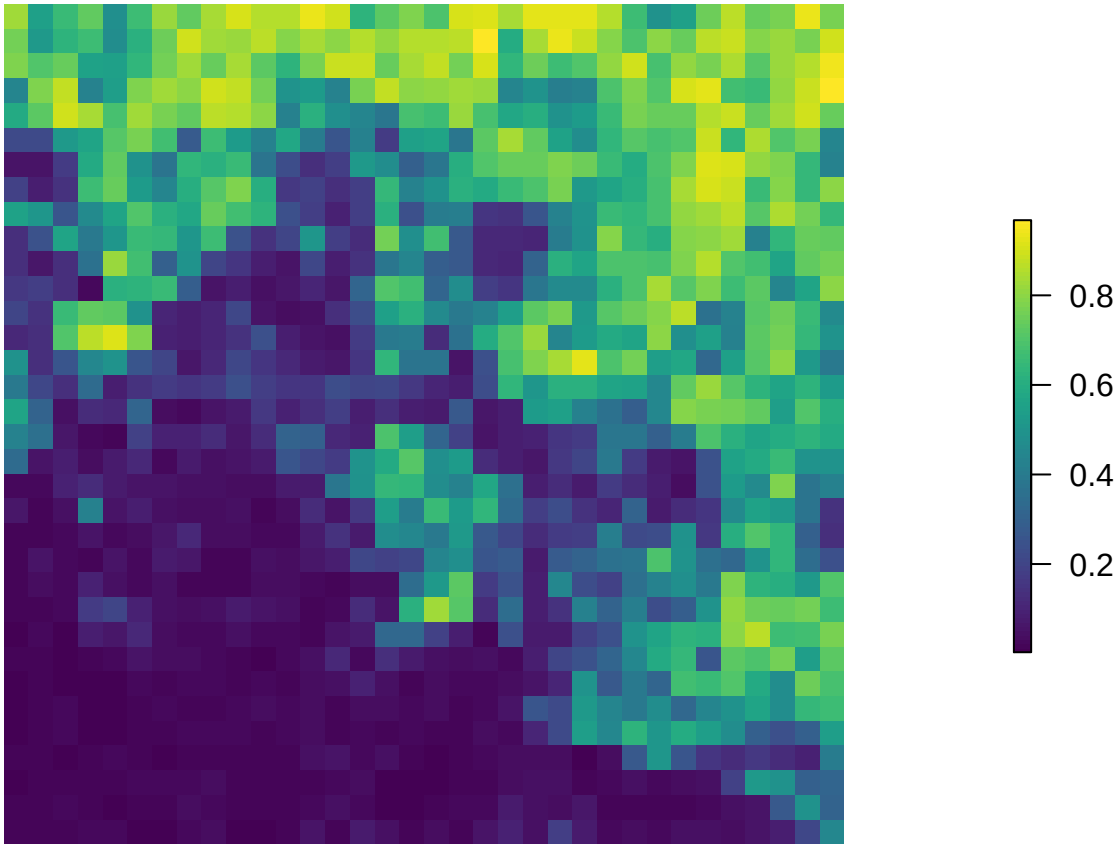
Read in spatial inputs to be used for the simulations. A 17.5km x 18km spatial grid with a resolution of 500m2 is used as the landscape for the meta-population of kangaroos. Each cell represents a patch that kangaroos can move between - dependent upon its unique attributes.

A habitat suitability layer describes the relative likelihood of the species occurring in each cell and should contain values between 0 (not inhabited) and 1 (inhabited). If the original values are not in this range, they should be rescaled accordingly. This example spatial data is available in the steps package ("egk_hab"), however, any raster can be used in its place.

```
egk_hab
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : EGK_habitat_500
## values      : 0.003082677, 0.9678264  (min, max)
```

```
par(mar=c(0,0,0,0), oma=c(0,0,0,0))
plot(egk_hab, box = FALSE, axes = FALSE, col = viridis(100))
```

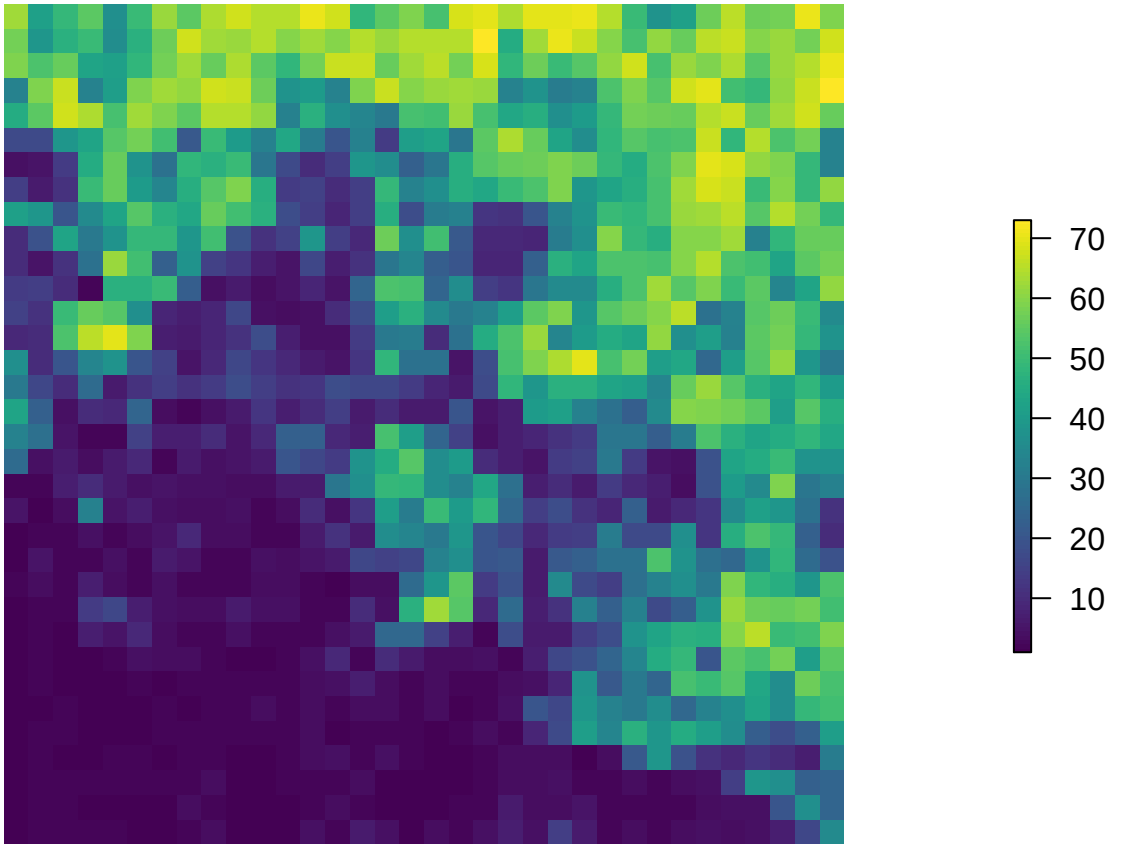


The carrying capacity layer describes the total number of species that may occur in each cell and contains either zeros or positive integer values. This example spatial data is available in the steps package ("egk_k"), however, any raster can be used in its place.

```
egk_k
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : layer
## values      : 1, 73  (min, max)
```

```
par(mar=c(0,0,0,0), oma=c(0,0,0,0))
plot(egk_k, box = FALSE, axes = FALSE, col = viridis(100))
```

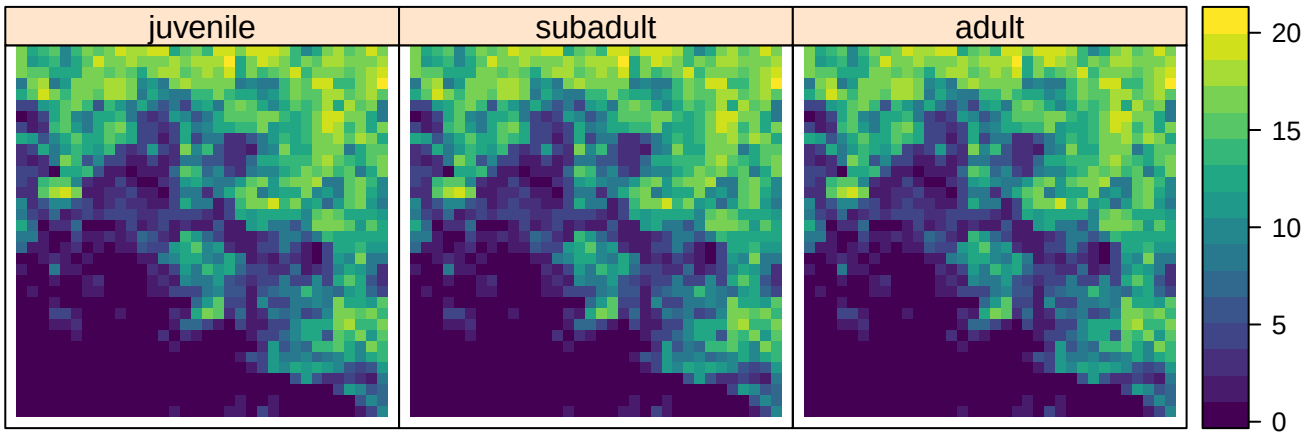


Populations are represented as a stack of rasters that describes the total number of individuals that occur in each cell for each life-stage. In the kangaroo example data, there are three life-stages and thus three individual raster layers in the stack. The values are either zeros or positive integers. This example spatial data is available in the steps package ("egk_pop"), however, any raster stack can be used in its place.

```
egk_pop
```

```
## class       : RasterStack
## dimensions  : 35, 36, 1260, 3  (nrow, ncol, ncell, nlayers)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## names       : juvenile, subadult, adult
## min values  :        1,        1,     1
## max values  :       20,       20,    20
```

```
par(mar=c(0,0,0,0), oma=c(0,0,0,0))
spplot(egk_pop, col.regions = viridis(100))
```



All three of these spatial components define a "landscape" object, however, only the population data is required to complete a simulation. The landscape object is modified at each timestep in a single simulation based on habitat and population dynamics (described in subsequent sections below).

```
egk_landscape <- landscape(population = egk_pop,
                           suitability = NULL,
                           carrying_capacity = NULL)
```

These are the only data input requirements for a simple population simulation, however, we also need to specify population dynamics at a minimum. Dynamic functions are used to modify populations or habitats in a landscape at each timestep in a simulation. They can be selected as 'off-the-shelf' functions - included in the 'steps' package - or custom defined functions created by the user. In its most basic form, only population growth will be applied to the landscape. As shown in the following code, we pass in a predefined growth function - with a transition matrix - and leave all of the other parameters at their default values (NULL).

```
egk_pop_dynamics <- population_dynamics(change = growth(transition_matrix = egk_mat),
                                        dispersal = NULL,
                                        modification = NULL,
                                        density_dependence = NULL)
```

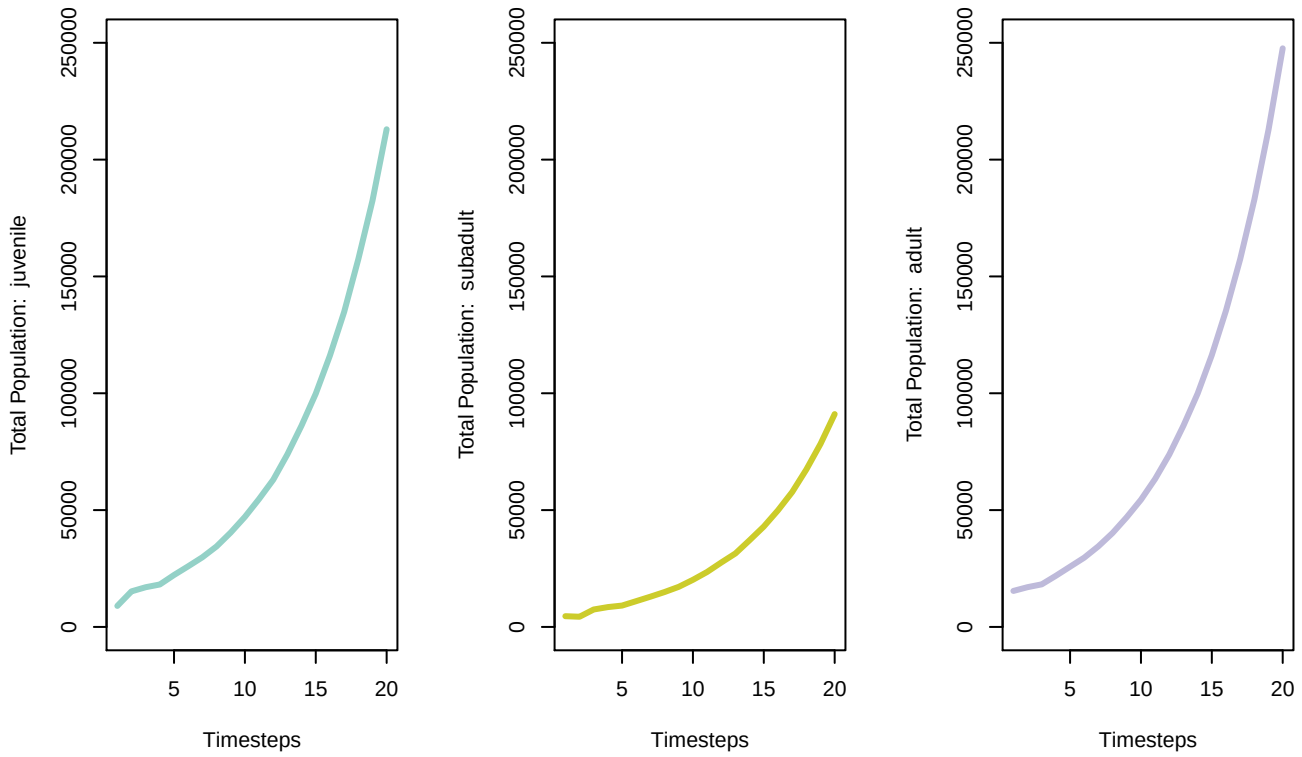
Now that we have constructed the landscape object and defined a dynamic object, we can run a single simulation (i.e replicates = 1, default). We simulate changes to the kangaroo population over twenty timesteps. Runtime will depend on the complexity of the landscape object and the configuration of the dynamic object(s).

```
egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 1,
                          verbose = FALSE)
```

Once a simulation has been run, we can plot spatially-explicit and temporally-explicit information.

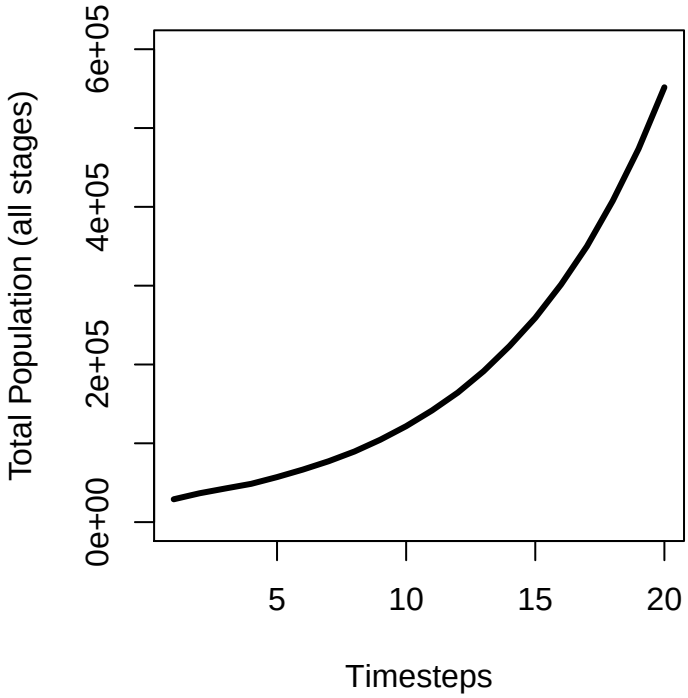
For this example, we can view the kangaroo population trajectories of each life-stage:

```
plot(egk_results)
```



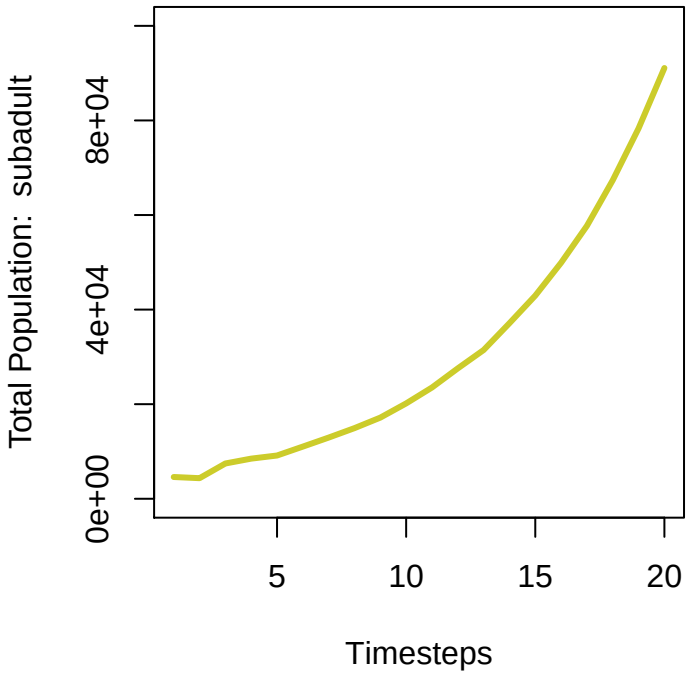
Or the total kangaroo population trajectory:

```
plot(egk_results, stage = 0)
```



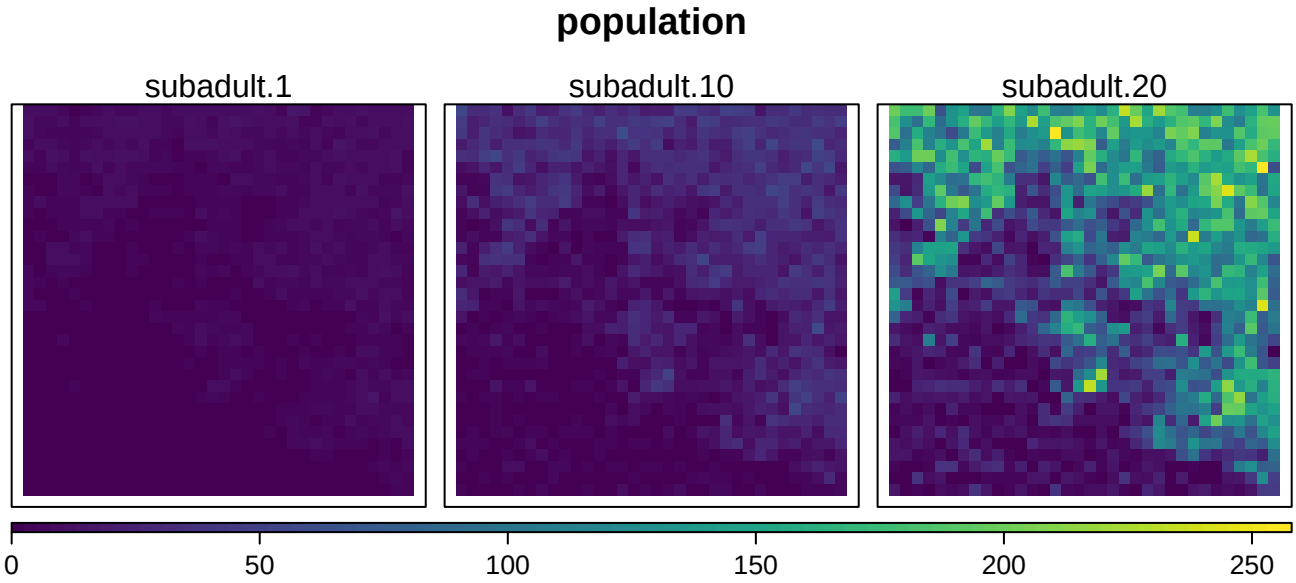
Or the kangaroo population trajectory for a single life-stage:

```
plot(egk_results, stage = 2, newplot = TRUE)
```



We can also view the population distribution over the landscape for a single life-stage (only timesteps one, ten, and twenty shown):

```
plot(egk_results, type = "raster", stage = 2, timesteps = c(1, 10, 20), panels = c(3, 1))
```

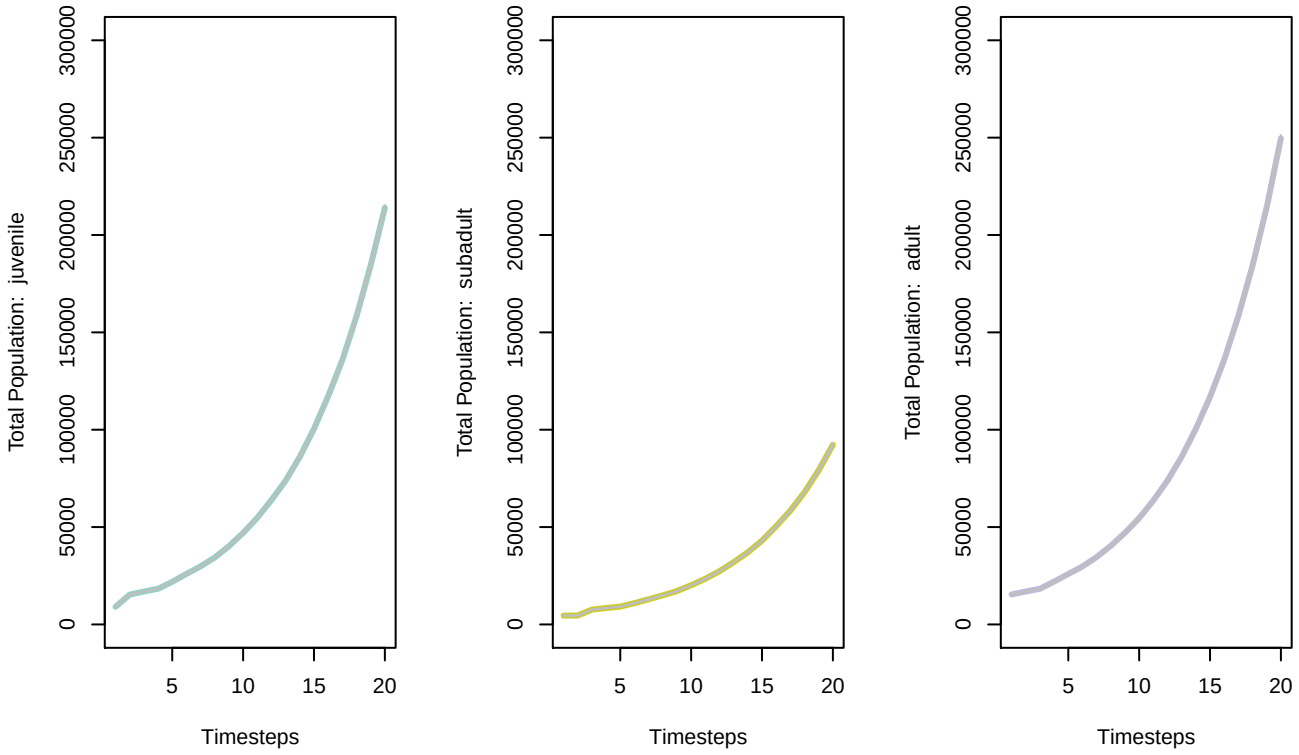


All of the rasters may also be plotted as animations - see package help for more information.

We can also perform multiple simulations. For the kangaroo, we specify three replicates of a twenty timestep simulation. The replicates are run sequentially by default, however, to improve computation time, we have included the ability to run all three replicates in parallel - each on a different processor.

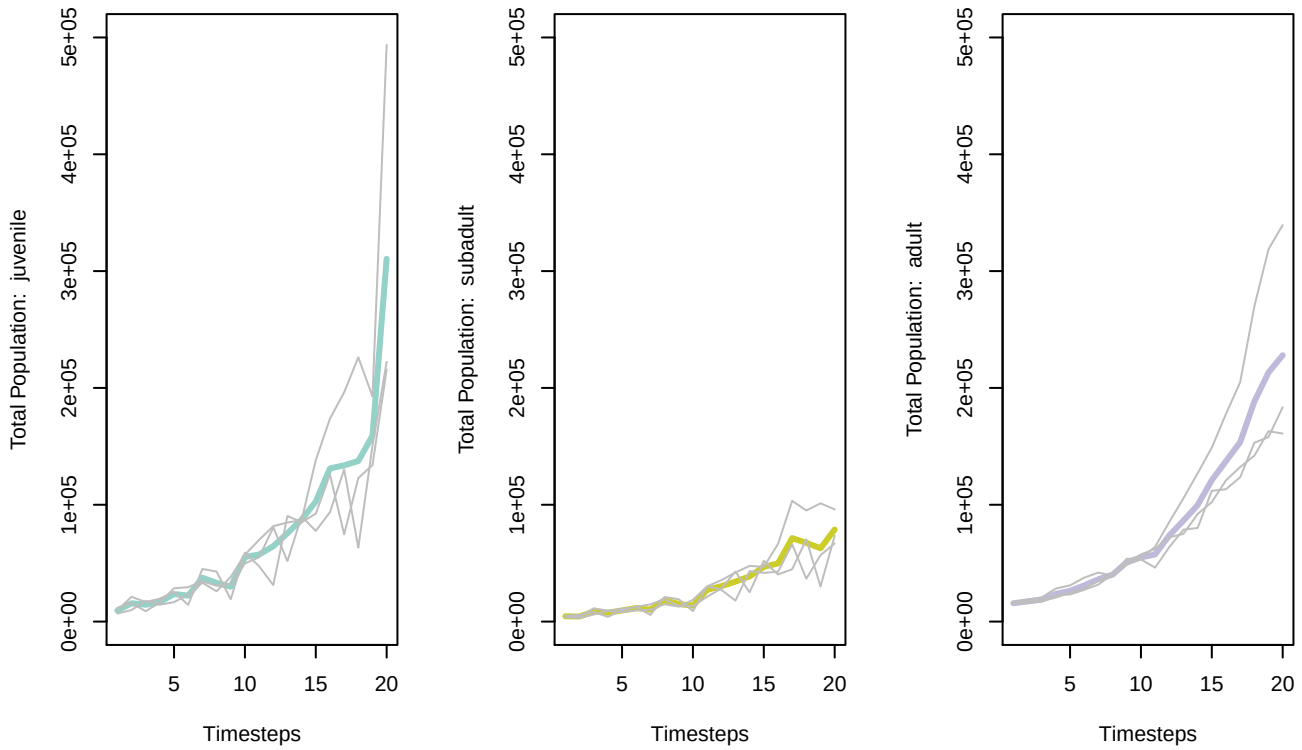
```
egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)
```

```
plot(egk_results)
```



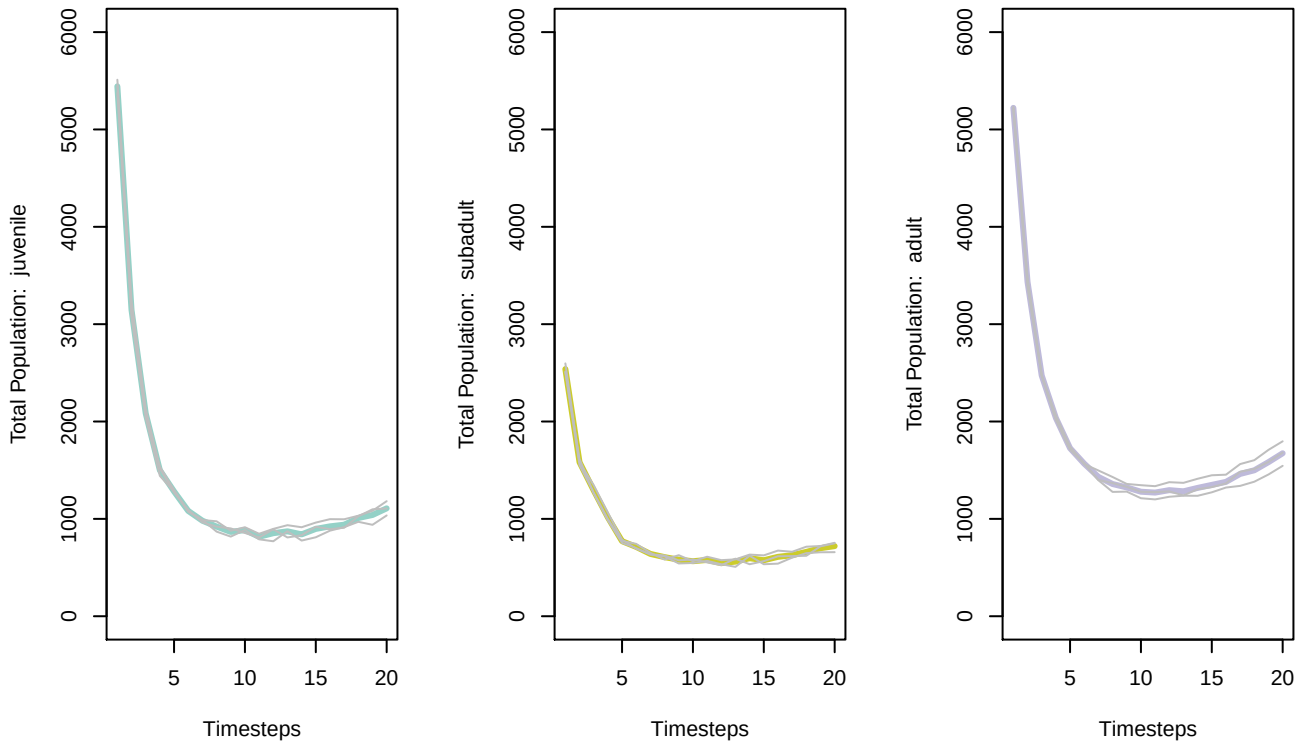
Note that the plot above looks very similar to the single replicate simulation. This is because we have not added any major stochastic dynamics to the landscape (but demographic stochasticity IS on by default in the growth function). Let's try adding some globally acting environmental stochasticity to the growth function:

```
egk_pop_dynamics <- population_dynamics(change = growth(transition_matrix = egk_mat,
                                                        global_stochasticity = egk_mat_stoch),
                                        dispersal = NULL,
                                        modification = NULL,
                                        density_dependence = NULL)

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



We can also specify how survival and fecundity values are influenced by spatial layers in the landscape object (e.g. habitat suitability or carrying capacity). Here we specify a transition function in the growth function that uses the habitat suitability to modify both the survival and fecundity values in the transition matrix at each timestep. Note, we must now add a habitat suitabilty layer to the landscape for the transition function to work:

```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = NULL)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  transition_function = modified_transition(egk_mat,
                                                            survival_layer = "suitability",
                                                            fecundity_layer = "suitability")),
  dispersal = NULL,
  modification = NULL,
  density_dependence = NULL)

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



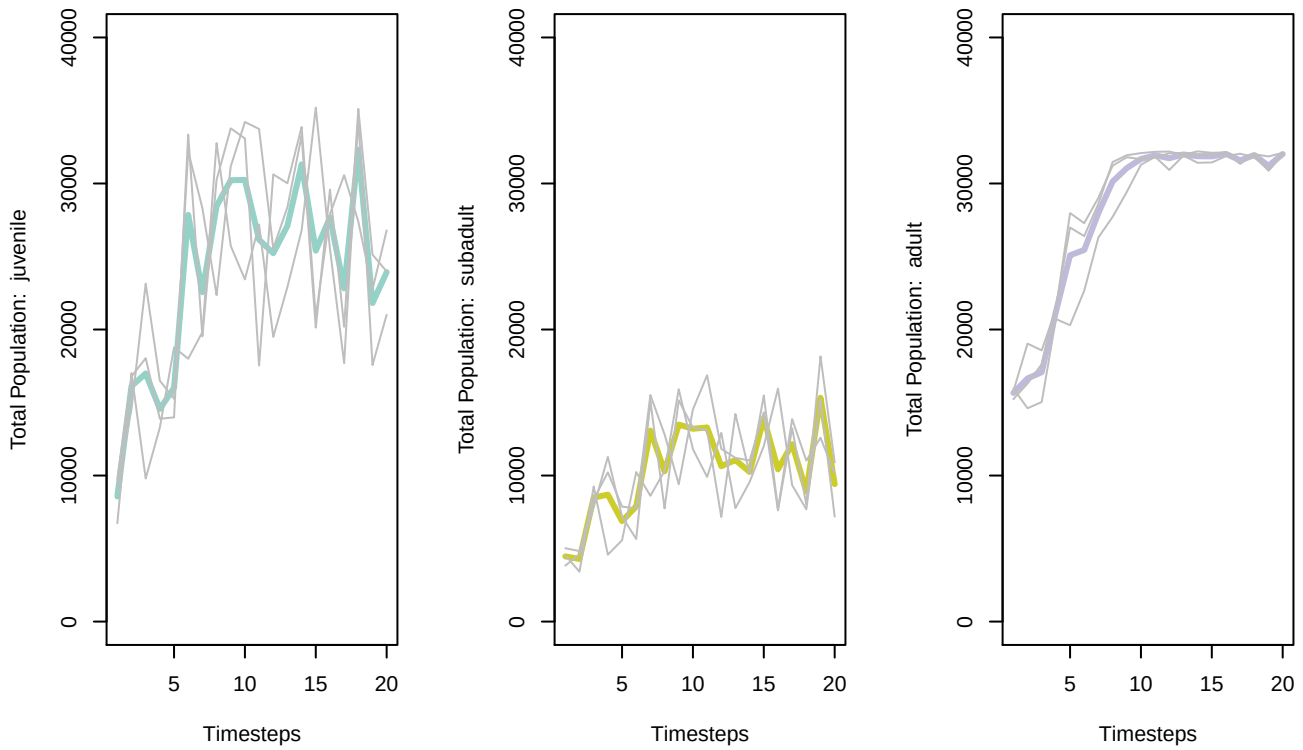
A user may also specify a custom function that operates on or defines survival and fecundity in the transition matrix at each timestep. The function requires a transition matrix, the landscape object, and a timestep as input parameters, and must return an array of transition matrices that matches the dimensions of the original transition matrix and number of cells in the landscape. As an example, we have created a custom function below that reads in spatial data to define the survival and fecundities at each timestep:

```
deterministic_transitions <- function(transition_matrix) {

  dim <- nrow(transition_matrix)

  function (landscape, timestep) {

    #### This assumes that the files are named with a particular convention
    #### and will not work for all cases. User must change code below accordingly.

      # get metrics and constructor info
      cell_idx <- which(!is.na(raster::getValues(landscape$population[[1]])))
      current_timestep <- sprintf("%02i", timestep)
      n_cells <- length(which(!is.na(raster::getValues(landscape$population[[1]]))))

      # get relevant rasters - note, your working directory will be different
      files <- list.files("../working/rasters", pattern = paste0("_", current_timestep, "_"))

      #initialise array
      transition_array <- array(0, dim = c(dim, dim, n_cells))

      # populate array:
      for (file in files) {
        r <- as.integer(substr(substr(file, nchar(file) - (8-1), nchar(file)), 1, 2))
        c <- as.integer(substr(substr(file, nchar(file) - (6-1), nchar(file)), 1, 2))
        #note, your working directory will be different
        transition_array[r, c, ] <- raster::raster(paste0("../working/rasters/",
                                                          file))[cell_idx]
```

```
    }

    #### Return array with required dimensions
    transition_array
  }
}

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  transition_function = deterministic_transitions(transition_matrix)),
  dispersal = NULL,
  modification = NULL,
  density_dependence = NULL)
```

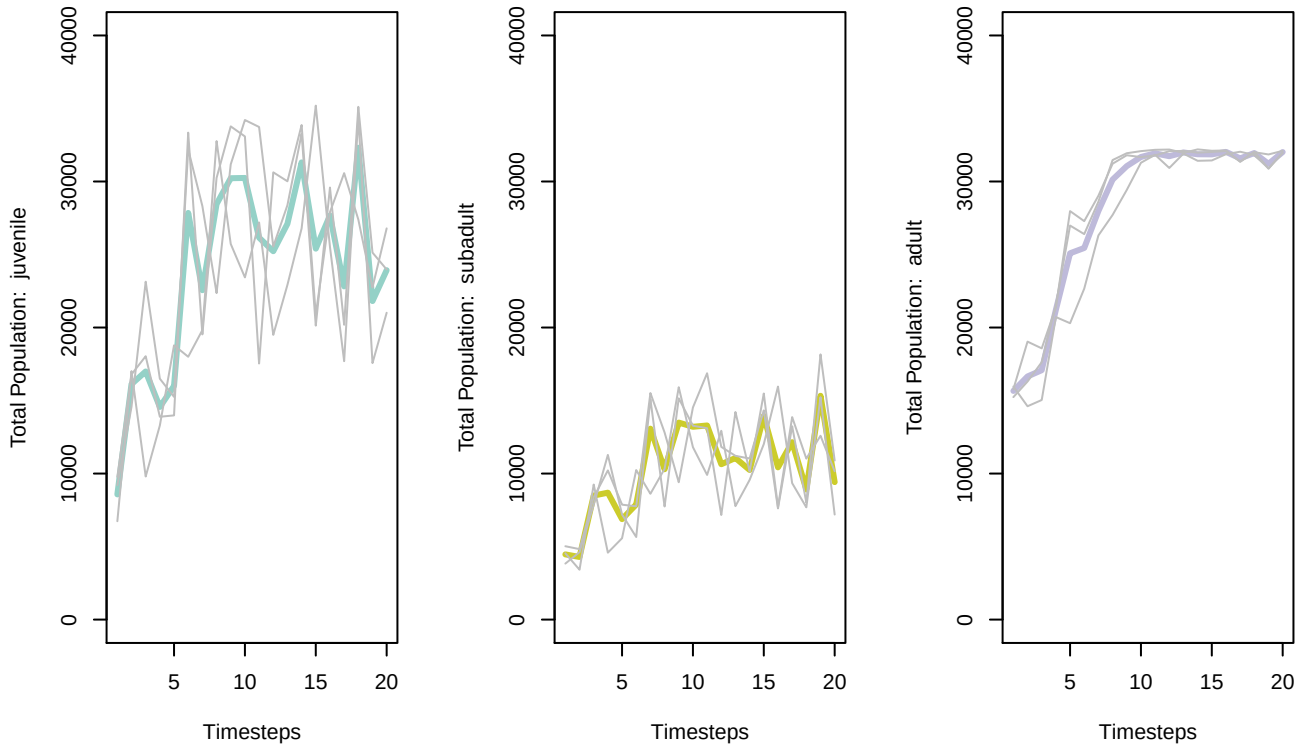
Now let's add some other population dynamics. We can specify a ceiling density dependence (population_cap function) but must provide a carrying capacity layer in the landscape for this to work. By default all life-stages contribute to density dependence, however, we specify the adults only - see the help for more information.

```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k)

egk_pop_dynamics <- population_dynamics(change = growth(transition_matrix = egk_mat,
                                                        global_stochasticity = egk_mat_stoch),
                                        dispersal = NULL,
                                        modification = NULL,
                                        density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



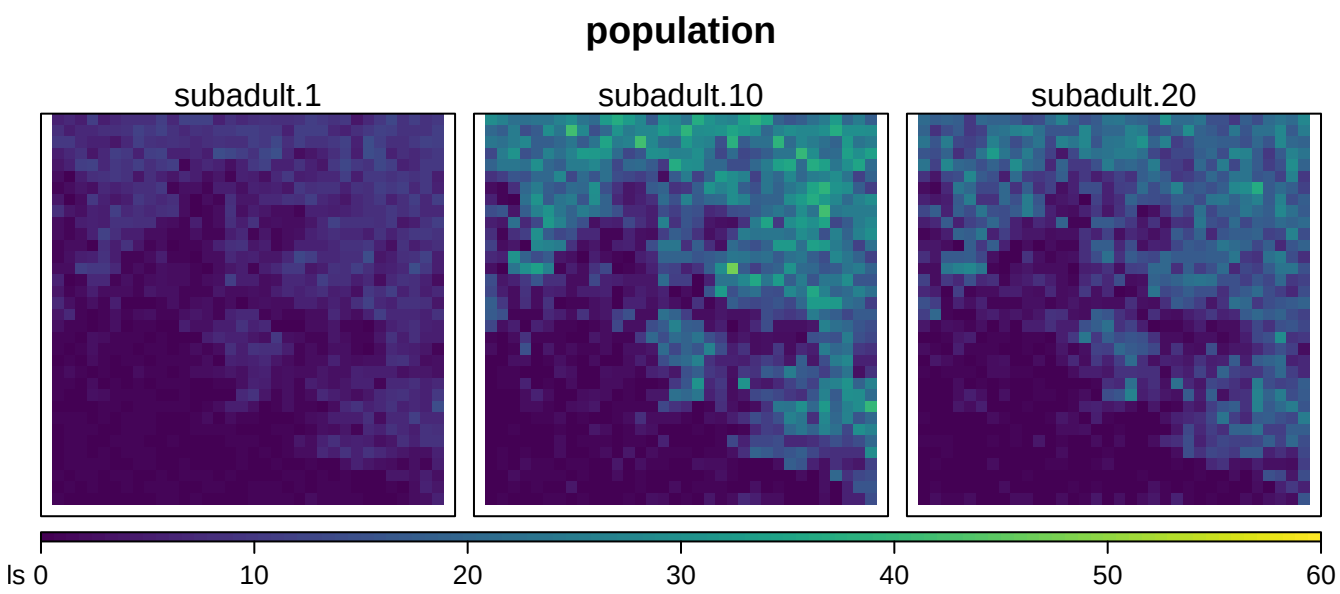
The default plot will show all of the simulation relicates and the mean population trajectory:

```
plot(egk_results)
```



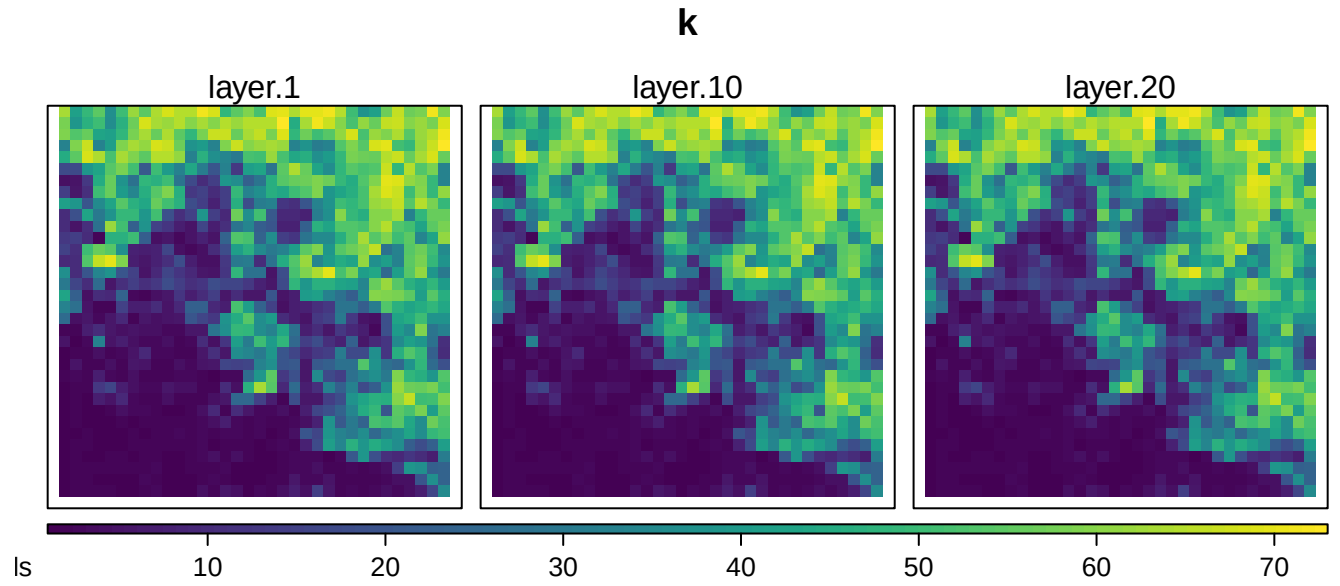
Note, if several replicates of a simulation are run, you must explicitly specify which one to plot for rasters (i.e. egk_results[1]):

```
plot(egk_results[1], type = "raster", stage = 2, timesteps = c(1, 10, 20), panels = c(3, 1))
```



The landscape object sub-components are stored for each timestep (if they are initially specified) so it is possible to also view the carrying capacity throughout a single simulation (i.e. egk_results[1]). Note, only timesteps one, ten, and twenty are shown - this is controlled by specifying the 'panels' parameter in the plot call:

```
plot(egk_results[1], object = "carrying_capacity", timesteps = c(1, 10, 20), panels = c(3, 1))
```



The simulation results object is a list of lists containing spatial objects at the termini and can be understood by the following tree diagram:

- Replicate (**a**)
  - Timestep (**b**)
    - Population Raster Stack (**c**)
      - Life-Stage Raster (**d**)
    - Habitat Suitability Raster (**c**)
    - Carrying Capacity Raster (**c**)
    - Other Raster Stack (**c**)
      - Raster (**d**)

To access the different components from the results object the following syntax is used:

result_object[[**a**]][[**b**]][[**c**]][[**d**]]

where **a** through **d** represent numbers denoting replicate, timestep, or object indices. All values will equal one or a larger positive integer depending on the simulation setup. The landscape object (denoted by **c**) will always be structured in the same order where position 1 is the population, position 2 is the habitat suitability, and position 3 is the carrying capacity. Positions beyond these are reserved for additional spatial data added to the initial landscape object.

For example, to return the population raster for the first life-stage in timestep five of the second replicate the following syntax would be used:

```
egk_results[[2]][[5]][[1]][[1]]
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : juvenile
## values      : 0, 63  (min, max)
```

To return the habitat suitability for the third timestep of the second replicate (note d is not used in this case):

```
egk_results[[2]][[3]][[2]]
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : EGK_habitat_500
## values      : 0.003082677, 0.9678264  (min, max)
```

Alternatively, the names of the raster objects can be used:

```
egk_results[[2]][[3]][["carrying_capacity"]]
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : layer
## values      : 1, 73  (min, max)
```

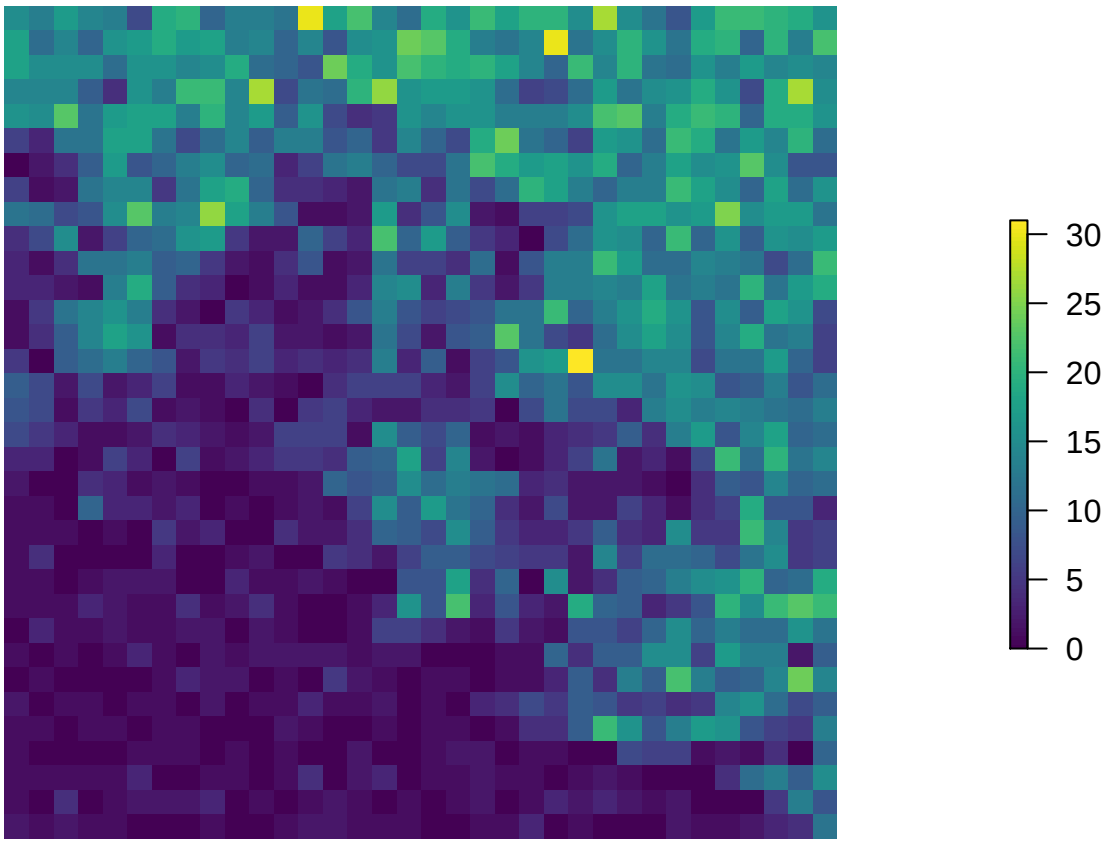
Based on this structure, we have also provided an extraction function to return components of a simulation results object. For example, to return the same object as in the previous example:

```
extract_results(egk_results, replicate = 2, timestep = 3, landscape_object = "carrying_capacity")
```

```
## class       : RasterLayer
## dimensions  : 35, 36, 1260  (nrow, ncol, ncell)
## resolution  : 500, 500  (x, y)
## extent      : 319000, 337000, 5817000, 5834500  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
## data source : in memory
## names       : layer
## values      : 1, 73  (min, max)
```

By default, the extract function will return a population raster (landscape object = 1 or landscape object = "population"), for the first life-stage (stage = 1), for the first iteration (timestep = 1), for the first simulation run (replicate = 1). These values can be specified accordingly to return the spatial object of your choosing. Here, we plot the default object from the extract function:

```
par(mar = c(0.1, 0.1, 0.1, 0.1))
plot(extract_results(egk_results), box = FALSE, axes = FALSE, col = viridis(100))
```

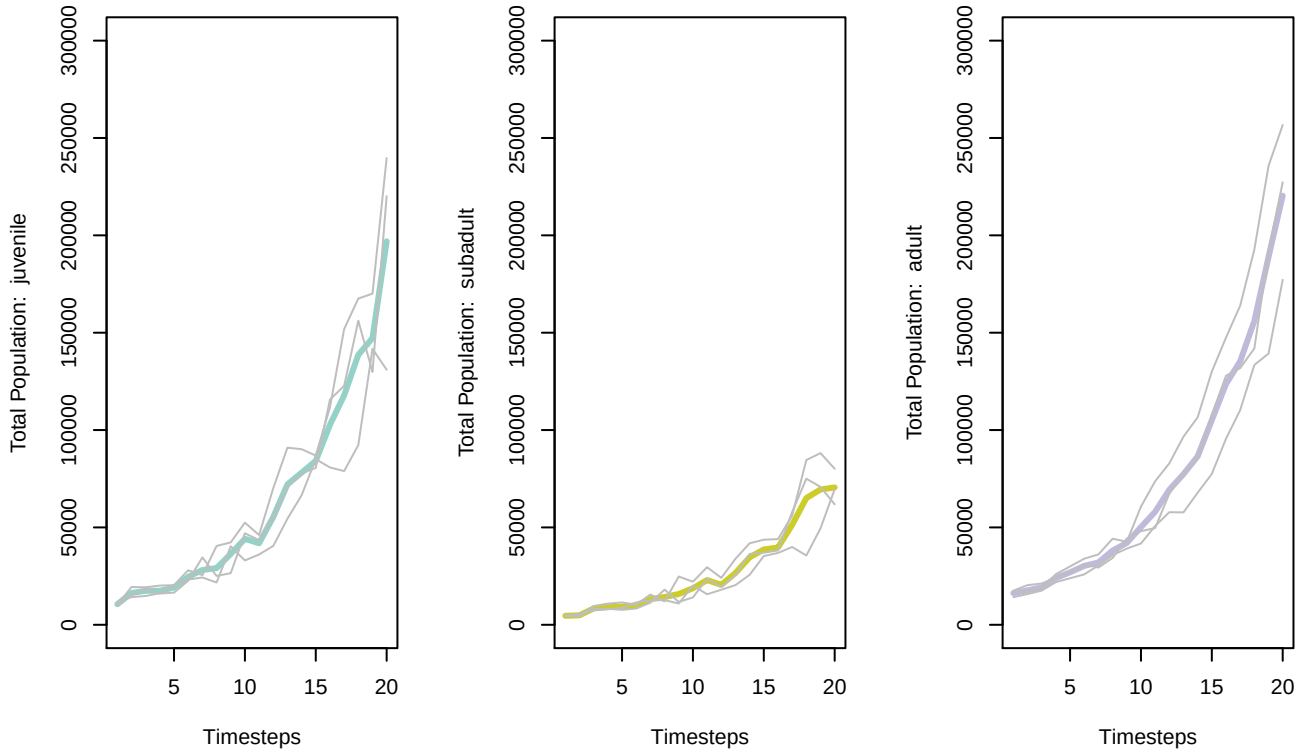


So far, the populations are only changing within their cells without any movement between cells. We can either specify assisted species movements (i.e. translocations or reintroductions) or natural species movements (dispersal) in the model simulations.

To characterise a translocation, we specify two raster layers that map the locations (and counts) of where we will take individuals from and where we will place individuals in the landscape. Source and sink spatial layers are stored in the landscape object and referenced by name. We use a built-in translocation function to modify the population(s) in a simulation at each specified timestep. The function requires the name of source and sink layers stored in the landscape object ("source" and "sink"), the life-stages to modify (default all - NULL), and the timesteps at which to modify the population (default is 1). In this example, we provide the kangaroo source ("egk_source") and sink ("egk_sink") layers, a life-stage of 3 (only adults affected), and timesteps one, five, ten and fifteen - meaning each translocation will occur every five years in each simulation replicate. Note, if the number of individuals are not available in the landscape when the translocation is applied, a warning will be generated and only the maximum available individuals will be used.

```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k,
                           source = egk_source,
                           sink = egk_sink)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
```

```
                  global_stochasticity = egk_mat_stoch),
  modification = translocation(source_layer = "source",
                               sink_layer = "sink",
                               stages = 3,
                               effect_timesteps = c(1, 5, 10, 15)),
  density_dependence = NULL)

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



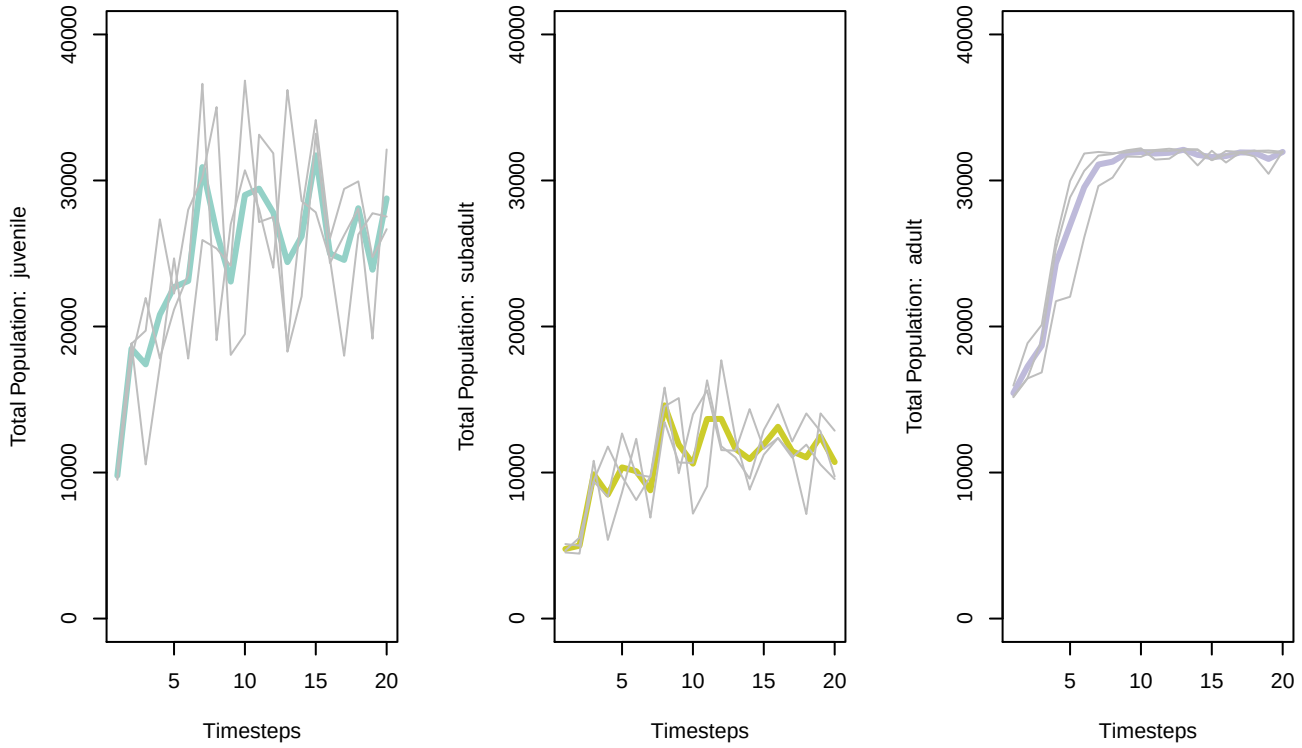
We can also add dispersal to allow kangaroos to move throughout the landscape. There are several built-in dispersal functions and most of them utilise habitat suitability, carrying capacity, or both to determine arrival probabilities - see package help for more information. Thus we will need to provide the appropriate layer in the intial landscape. Below, we use kernel-based dispersal and habitat suitability to determine the arrival probabilities:

```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
  dispersal = kernel_dispersal(arrival_probability = "suitability"),
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
```

```
                        habitat_dynamics = NULL,
                        timesteps = 20,
                        replicates = 3,
                        verbose = FALSE)
```

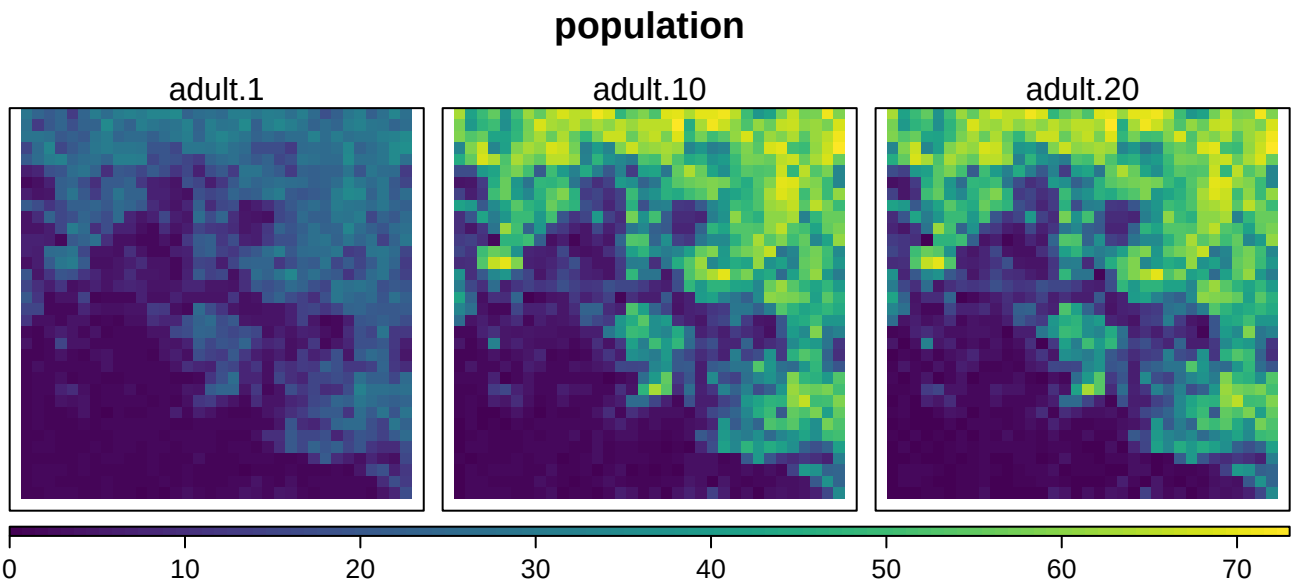
```
##      3 life stages exist but 1 dispersal proportion(s) were specified. Is this what was intended?     3
```

```
plot(egk_results)
```



Let's have a look at how the adult population is changing in the first replicate of the simulation:

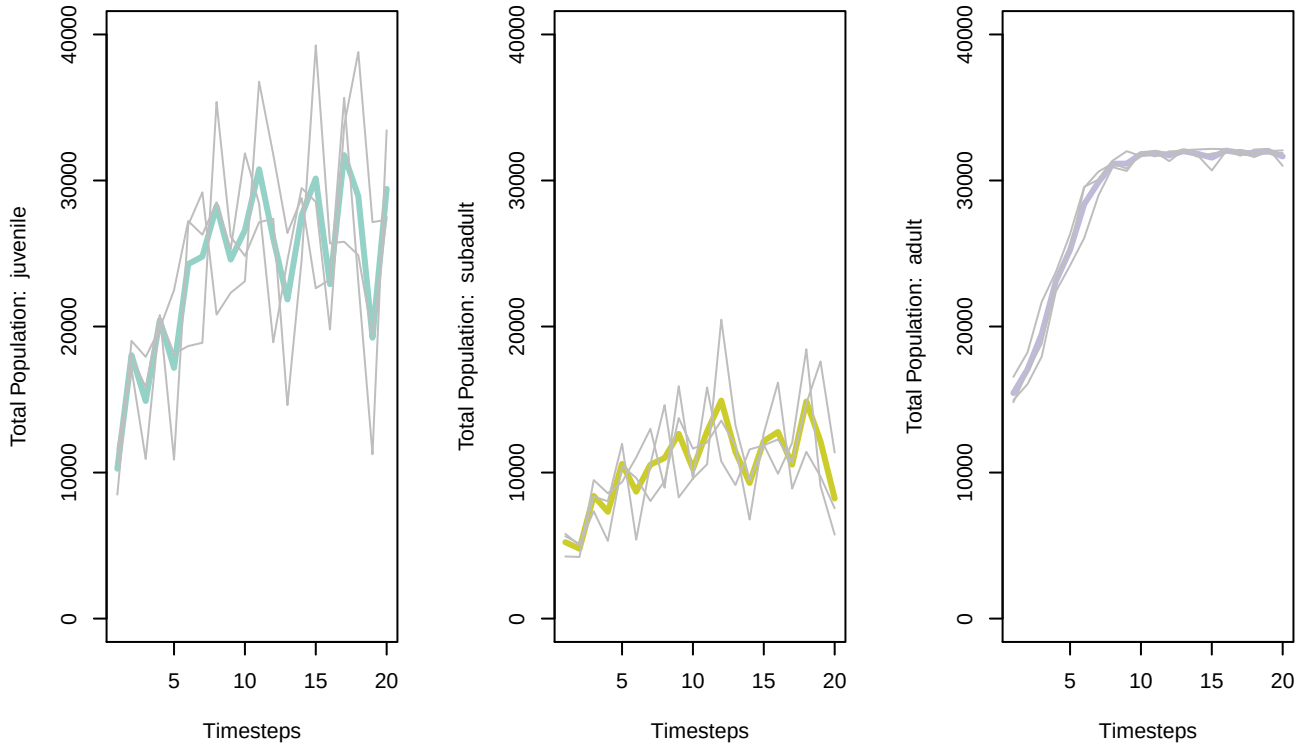
```
plot(egk_results[1], type = "raster", stage = 3, timesteps = c(1, 10, 20), panels = c(3, 1))
```



Proportion of individuals dispersing can also be specified in the dispersal functions. If dispersal proportions are not specified, the default behaviour is to disperse all individuals in all life-stages. If proportions are specified as a single number, then all life-stages disperse with that proportion, however, a vector of proportions (equal in length to the

number of life-stages) can also be specified. In the example below, no juveniles (stage 1) disperse, only fifty-percent of subadults (stage 2) disperse, and all adults (stage 3) disperse:

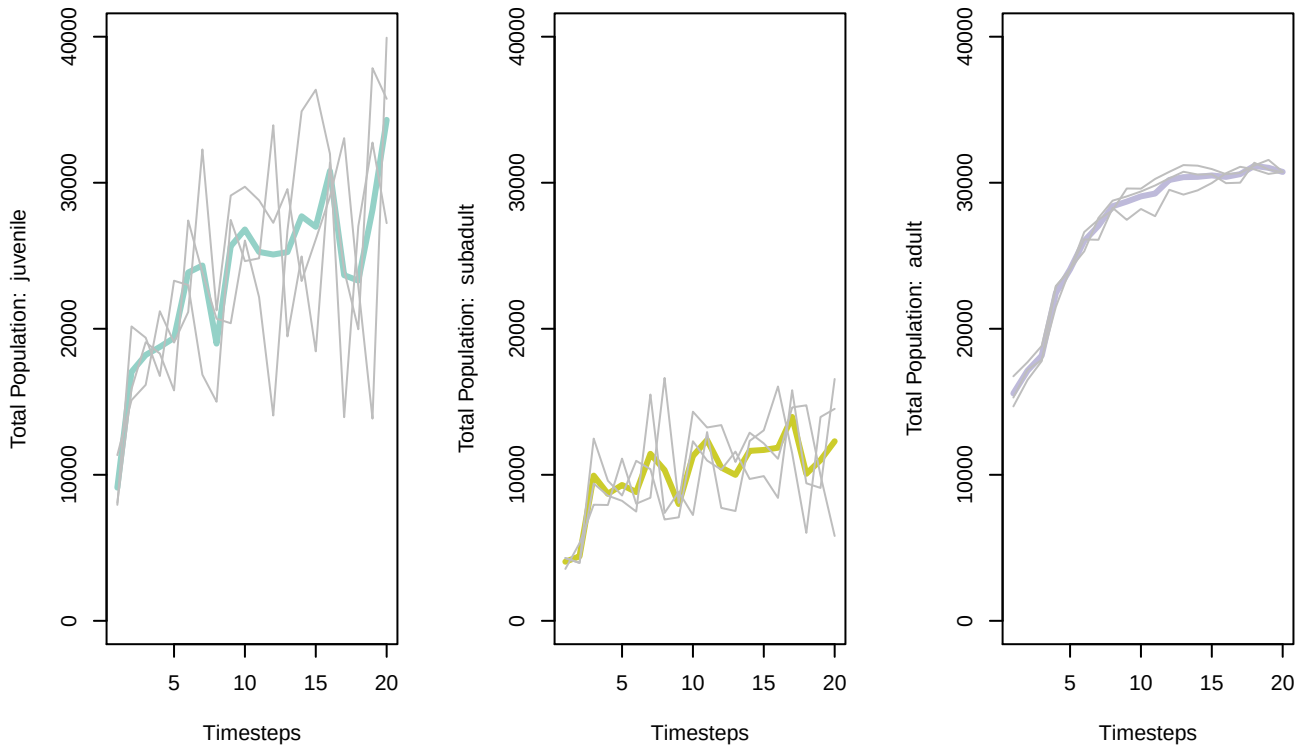
```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
  dispersal = kernel_dispersal(arrival_probability = "suitability",
                               dispersal_proportion = c(0, 0.5, 1)),
  modification = NULL,
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



Dispersal distances can also be specified. Distances are expressed as number of cells which correspond to Euclidean distances based on the resolution of the raster layers. For example, the kangaroo rasters are 500 meter resolution so a distance of 3 (cells) would correspond to 1500 meters. Maximum distances (in number of cells) are explicitly defined for the cellular automata dispersal function.

Note, as with the proportions, a single number will indicate dispersal distances for all life-stages whilst a vector of distances (equal in length to the number of life-stages) can also be specified. In the example below, no juveniles (stage 1) disperse, subadults (stage 2) only disperse up to 5 cells (2500 meters), and adults (stage 3) disperse up to 10 cells (5000 meters):

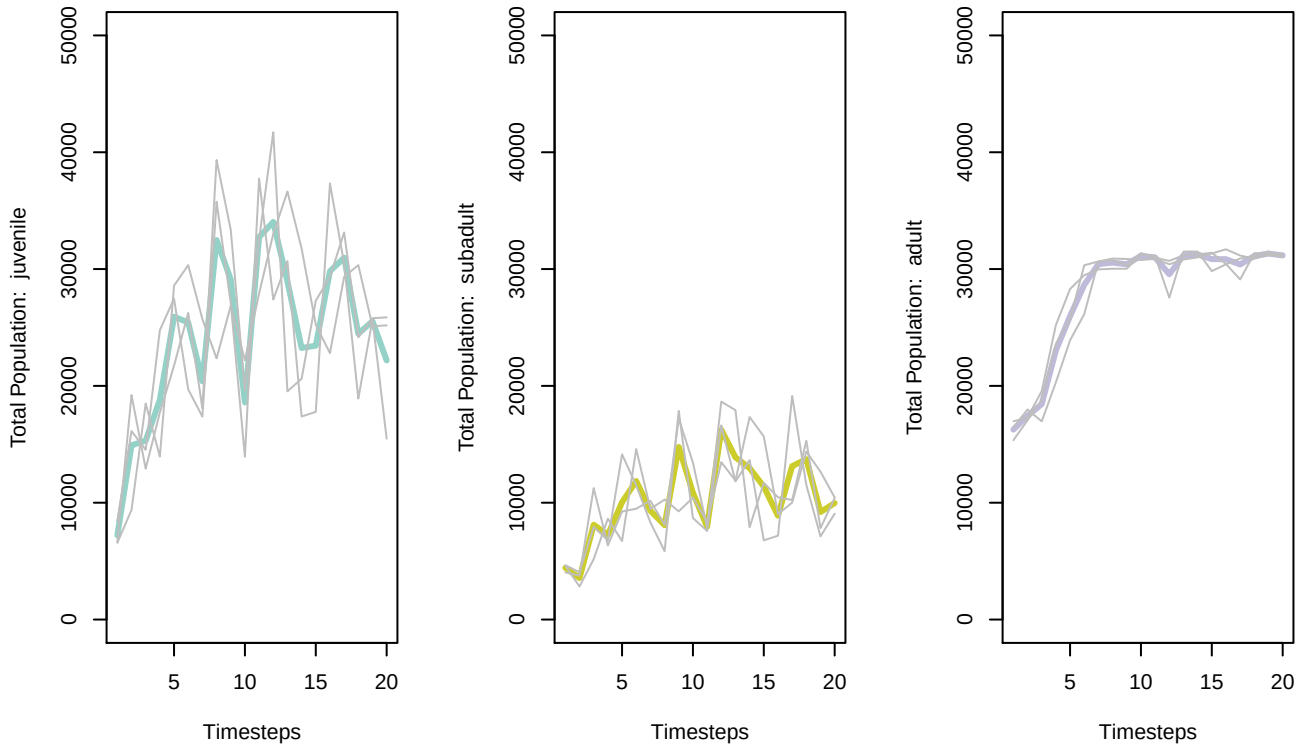
```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
  dispersal = cellular_automata_dispersal(dispersal_distance = c(0, 5, 10),
                                          dispersal_proportion = c(0, 0.5, 1)),
  modification = NULL,
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



The 'number of cells' representation of distance should be considered when specifying custom distance kernel functions, for example:

```
power_law_dispersal_kernel <- function (r) 0.8*(1 + (r/0.9))^-2.1

egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
```

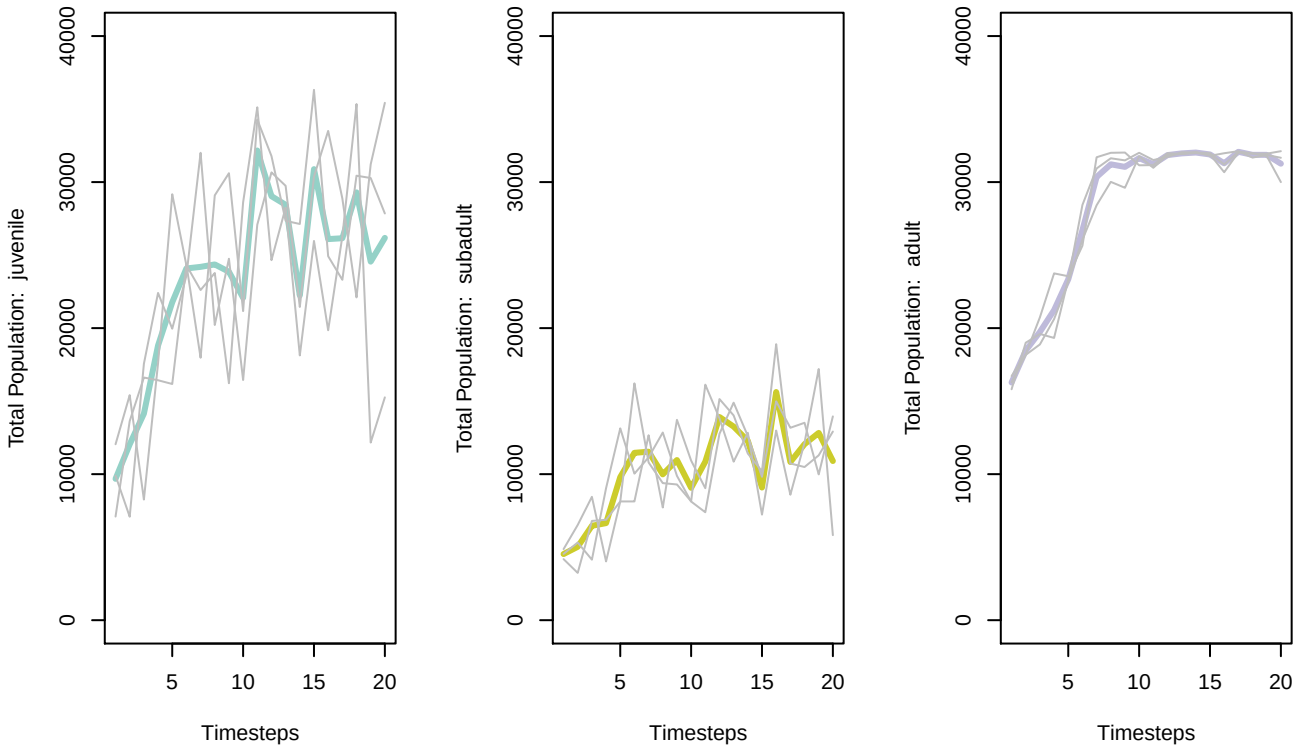
```
  dispersal = kernel_dispersal(arrival_probability = "suitability",
                               dispersal_kernel = power_law_dispersal_kernel,
                               dispersal_proportion = c(0, 0.5, 1)),
  modification = NULL,
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = NULL,
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



So far the kangaroo population is the only thing to be dynamically changing in the landscape, however, often habitat suitability also changes due to disturbances. To characterise habitat disturbance, we can use a series of raster layers that map the locations and severity of disturbances. Each disturbance raster will be multiplied with the original habitat suitability layer and thus contain values that represent appropriate numerical modifications. Note, the number of disturbance layers must match the intended number of timesteps in a single simulation. There is an existing spatial dataset of fires in the steps package ("egk_dist") which we store in the landscape object. We use a pre-defined disturbance function to modify the habitat suitability in the simulation at each timestep. The function requires the name of disturbance layers stored in the landscape object, and an effect time which specifies the number of timesteps that each disturbance layer acts on the habitat suitability. In this example, we provide the kangaroo fire disturbance input name ("fires"), and an effect time of five - meaning each fire layer will affect the habitat suitability (in decreasing intensity) for five timesteps in each simulation replicate. All functions that act on the habitat must be passed in as a list in the simulation call:

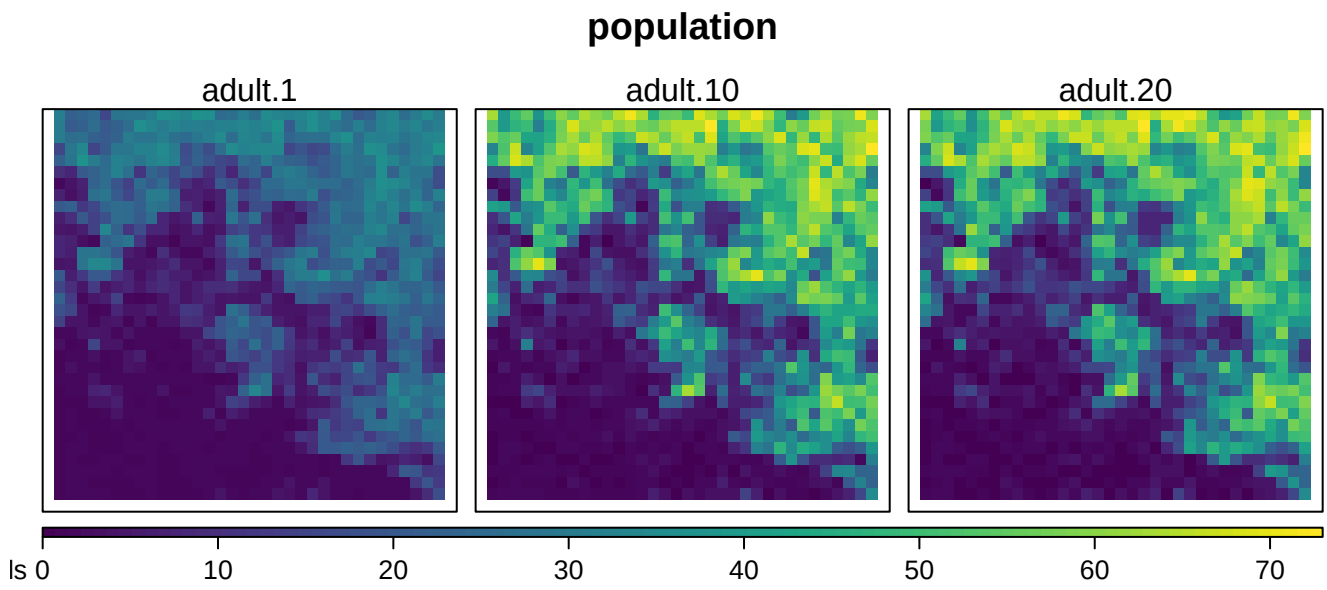
```
egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = egk_k,
                           fires = egk_dist)
```

```
egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
  dispersal = kernel_dispersal(arrival_probability = "suitability",
                               dispersal_proportion = 0.5),
  modification = NULL,
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = list(disturbance(disturbance_layers = "fires",
                                                              effect_time = 5)),
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results)
```



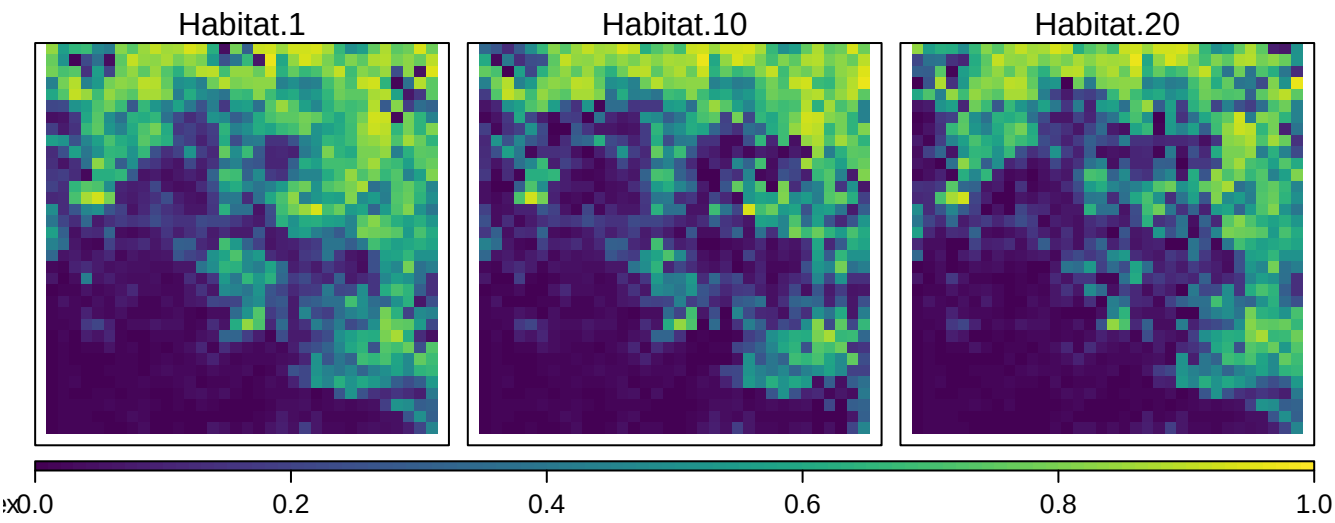
And now let's have a look at how the adult population is changing with fires occurring in the landscape:

```
plot(egk_results[1], type = "raster", stage = 3, timesteps = c(1, 10, 20), panels = c(3, 1))
```



Since we are using habitat dynamics we may want to have a look at how the habitat suitability changes in a landscape. Similar to carrying capacity - as shown earlier - it is possible to view the habitat suitability throughout a single simulation:

```
plot(egk_results[1], object = "suitability", timesteps = c(1, 10, 20), panels = c(3, 1))
```



In all of the examples above, carrying capacity has remained unchanged throughout a simulation. However, carrying capacity is often derived from dynamically changing habitat suitabilities. A function of this relationship can be specified in place of the carrying_capacity raster layer when constructing the landscape object:

```
carrying_cap_fun <- function (x) 75 - round(75 * dlogis(x, scale = 0.27), 0)

egk_landscape <- landscape(population = egk_pop,
                           suitability = egk_hab,
                           carrying_capacity = carrying_cap_fun,
                           fires = egk_dist)

egk_pop_dynamics <- population_dynamics(
  change = growth(transition_matrix = egk_mat,
                  global_stochasticity = egk_mat_stoch),
```

```
  dispersal = kernel_dispersal(arrival_probability = "suitability",
                               dispersal_proportion = 0.5),
  modification = NULL,
  density_dependence = population_cap(stages = 3))

egk_results <- simulation(landscape = egk_landscape,
                          population_dynamics = egk_pop_dynamics,
                          habitat_dynamics = list(disturbance(disturbance_layers = "fires",
                                                              effect_time = 5)),
                          timesteps = 20,
                          replicates = 3,
                          verbose = FALSE)

plot(egk_results[1], object = "carrying_capacity", timesteps = c(1, 10, 20), panels = c(3, 1))
```