In `secr` you can use non-Euclidian (or "eclological") distances by specifying a distance calculation function that uses a mask covariate called `noneuc`, and how this `noneuc` covariate is to be constructed. The `secr` funcrion `secr.fit` is passed the distance calculation function via the `userdist` component of the `details` argument of `secr.fit`. The variable `noneuc` and the `userdist` function are described in more detail below:

`noneuc`: Program `secr` constructs this variable as a function of mask covariates, using (by default) a log link function and linear predictor. You tell it how to do this in the same way that you tell it how to make density D depend on explanatory variables or σ depend on explanatory variables. For example, suppose that you have covariates called `z1` and `z2` on your mask, and you want "ecological distance' to depend on them. If you include the following text in the list passed to the `model` argument of `secr.fit`:

```
noneuc ~ z1 + z2
```

this creates a new (temporary) covariate on the mesh, called `noneuc` which is equal to $\exp\{\beta_0 + \beta_1 \mathrm{z1} + \beta_2 \mathrm{z2}\}$. And if you include the same text, but with a "`-1`" to remove the intercept parameter:

```
noneuc ~ z1 + z2 -1
```

this creates a new (temporary) covariate on the mesh, called `noneuc` which is equal to $\exp\{\beta_1 \mathrm{z1} + \beta_2 \mathrm{z2}\}$. (See below for the reason you might want the "`-1`".)

`userdist`: This element of the `details` list argument of `secr.fit` must be passed a function that returns all the distances between two sets of points (data frames or matrices `xy1` and `xy2`) using the covariate `noneuc` to calculate the distances, as appropriate. The covariate `noneuc` is constructed and temporarily attached to the mask by `secr` (invisibly to the user – see above).

If you want "ecological distances" that are least-cost distances, you can use functions in the package `gdistance` to calculate them. The key steps to doing this are

(1) Use the function `transition` to calculate the "conductance" between all pairs of points, by passing it a `transitionFunction` that tells it how "conductance" is to be calculated from `noneuc`.
(2) Use the function `costDistance` to calculate least-cost paths between all pairs of points, given their locations and the "conductances" between them.

The "**conductance**" is the thing that divides the Euclidian distance in an `secr` detection function or encounter rate function. If conductance is the same everywhere, we denote it σ and in this case, the distances used by `secr` are the usual Euclidian distances. When conductance depends on spatial covariates, and so changes in space, we denote it $\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ between mask points $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$. Conductance is the inverse of movement cost: high conductance implies low movement cost, and *vice-versa*.

Note that the `transitionFunction` that you create must calculate the conductance and although it is the conductances that must be passed to `costDistance`, it is the cost-weighted distances that are returned by `costDistance`.

Here is an example function, taken (with minor modifications) from the `secr` vignette "secr.noneuclidean.pdf": The `gdistance` function `transition` does (1) above, using a transitionFunction that you specify, and the gdistance function costDistance operates on the output of transition to do (2) above. (Function geoCorrection is used to refine the distances output by transition before these are passed to costDistance).

```
# Create a user-dfined tranistion function:
myConductanceFun = function(x) exp(mean(log(x)))

# Create a user-defined function to calculate least-cost distances, from the
# mask covariate "noneuc" with the user-defined function "myConductanceFun"
mydistFun = function (xy1, xy2, mask) {
  if (missing(xy1)) return("noneuc") # required by secr.fit
  if (!require(gdistance))
    stop ("install␣package␣gdistance␣to␣use␣this␣function")
  # Make raster from mask, because transition() requires raster
  # The mask must contain a covariate called "noneuc", this
  # is added invisibly by secr.fit:
  Sraster <- raster(mask, "noneuc")
  # Calculate the conductances between all points in xy1 and all in xy2,
  # using a user-defined function called myConductanceFun.
  # (See help for "transition" for more on transitionFunctions.)
  tr <- transition(Sraster, transitionFunction=myConductanceFun, directions=16)
  # Correction to get more accurate distances:
  # (See help for "geoCorrection" for more on this function.)
  tr <- geoCorrection(tr, type = "c", multpl = FALSE)
  # Pass the object containing the conductances, to costDistance,
  # which uses their inverse to calculate least-cost distances
  # by means of Dijkstra's Algorithm.
  costDistance(tr, as.matrix(xy1), as.matrix(xy2))
}
```

Here is the algebra that shows how the above code calculates the conductance and its inverse (which will be called the "cost-rate function") when you specify noneuc as

```
noneuc ~ z1 + z2.
```

In this case, the values of the mesh covariate noneuc at at $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$ that are constructed by secr are

$$\text{noneuc}_m = \exp\{\beta_0 + \beta_1 z_1(\boldsymbol{s}_m) + \beta_2 z_2(\boldsymbol{s}_m)\} \tag{1}$$

$$\text{noneuc}_{m^*} = \exp\{\beta_0 + \beta_1 z_1(\boldsymbol{s}_{m^*}) + \beta_2 z_2(\boldsymbol{s}_{m^*})\} \tag{2}$$

where $z_1(\boldsymbol{s}_m)$, $z_1(\boldsymbol{s}_{m^*})$, $z_2(\boldsymbol{s}_m)$, $z_2(\boldsymbol{s}_{m^*})$ are the values of the mesh covariates z1 and z2 at $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$.

Applying the user-defined myConductanceFun to noneuc_m and noneuc_{m^*} yields the conductance between $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$

$$\begin{aligned}
\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) &= \exp\left\{\frac{\log(\text{noneuc}_m) + \log(\text{noneuc}_{m^*})}{2}\right\} \\
&= \exp\left\{\sum_{c=1}^{2} \frac{\beta_0 + \beta_c[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]}{2}\right\} \\
&= \exp\left\{\beta_0 + \sum_{c=1}^{2} \beta_c[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]/2\right\} \qquad (3)
\end{aligned}$$

and hence the cost-rate function

$$c(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = \frac{1}{\sigma(\boldsymbol{s}_{m^*}, \boldsymbol{s}_{m^*})} = \exp\left\{-\beta_0 - \sum_{c=1}^{2} \beta_c \frac{[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]}{2}\right\} \tag{4}$$

and the "ecological distance"

$$d^{(c)}(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = c(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})d(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = \frac{d(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})}{\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})} \tag{5}$$

With the above transition function `myConductanceFun`, the interpretation of the β parameters is as follows:

- β_0 is the log of the "baseline" σ. If all the β_cs are zero then we have the usual Euclidian distance metric with $d^{(c)}(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = d(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})/\sigma_0$, where $\sigma_0 = \exp\{\beta_0\}$. (The "$(c)$" superscript on the d of $d^{(c)}(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ is to show that it is a **c**ost-based distance, or a **c**onductance-based distance, as opposed to a Euclidian distance, $d(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$.)

  Note that when the intercept term β_0 is included in `noneuc`, this implicitly includes σ_0 in `noneuc`. If when fitting this model, you also try to estimate the `secr.fit` parameter `sigma`, inference will fail. This is because your range function now looks like this

$$\begin{aligned} \sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) &= \sigma^* \exp\left\{\beta_0 + \sum_{c=1}^{2} \beta_c[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]/2\right\} \\ &= \exp\left\{\beta_0^* + \beta_0 + \sum_{c=1}^{2} \beta_c[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]/2\right\}. \end{aligned} \tag{6}$$

  where $\sigma^* = e^{\beta_0^*}$ is `sigma`.

  Here β_0^* and β_0 can't both be estimated: if you add any constant K to one and subtract it from the other, $\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ is unchanged. So there are an infinite number of β_0^*s and β_0s that give exactly the same likelihood.

  So if you specify

```
noneuc ~ z1 + z2
```

  you must also tell `secr.fit` not to estimate `sigma` separately, by passing

```
fixed=list(sigma=1)
```

  to `secr.fit`. Alternatively, you could remove the intercept term β_0 from `noneuc`, by adding "`-1`" to its linear predictor, and allow estimation of σ, as follows:

```
noneuc ~ z1 + z2 -1
```

  without fixing `sigma`. This will lead to the same inference and give you the same parameter estimates (although what was reported as the intercept parameter of `noneuc` will be reported as the intercept parameter of `sigma`).

- β_c quantifies the strength and direction of the effect of the covariate z_c on σ, on the log scale. For example, if in Equation (3) we write $[z_c(\boldsymbol{s}_m) + z_c(\boldsymbol{s}_{m^*})]/2$ as $f_{m,m^*}(z_c)$, we can rewrite $\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ as

$$\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = \sigma_0 e^{\beta_1 f_{m,m^*}(z_1)} e^{\beta_2 f_{m,m^*}(z_2)} \tag{7}$$

  Suppose that $f_{m,m^*}(z_c)$ is positive. Then if β_c ($c = 1, 2$) is positive, $e^{\beta_c f_{m,m^*}(z_c)}$ is greater than 1 and $\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ is increased by a multiplicative factor of $e^{\beta_c f_{m,m^*}(z_c)}$ between $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$. If β_c is negative then $e^{\beta_c f_{m,m^*}(z_c)}$ is less than 1 and $\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*})$ is decreased by a multiplicative factor of $e^{\beta_1 f_{m,m^*}(z_1)}$ between $\boldsymbol{s}_m$ and $\boldsymbol{s}_{m^*}$. The larger the absolute value of β_c, the greater the effect (positive or negative).

With positive $f_{m,m^*}(z_c)$, positive β_c implies that the larger the value of $f_{m,m^*}(z_c)$, the easier is movement or propogation, while negative β_c implies that the larger the value of $f_{m,m^*}(z_1)$, the more difficult is movement or propogation.

With negative $f_{m,m^*}(z_c)$, the effect is reversed. Positive β_c implies that the larger the value of $f_{m,m^*}(z_c)$, the more difficult is movement or propogation, while negative β_c implies that the larger the value of $f_{m,m^*}(z_1)$, the easier is movement or propogation.

An aside

Notice that use of `myConductanceFun` and `mydistFun` as defined above, and using one mask covariate, implements the least cost distance metric proposed by Royle *et al.*(2013) and Sutherland *et al.* (2015). What if you have a single covariate (z_1 say) but you use these `transitionFunctions` instead?:

```
myConductanceFun = function(x) 1/mean(x)
myConductanceFun = function(x) mean(x)
```

With the first of these functions we have

$$\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = e^{\beta_0} \frac{2}{e^{\beta_1 z_1(\boldsymbol{s}_m)} + e^{\beta_1 z_1(\boldsymbol{s}_{m^*})}} \tag{8}$$

so that positive β_1 implies that as z_1 incereases, the range of animal movement decreases (or conversely, the cost of movement increases). That is, z_1 inhibits movement.

With the second of the above functions we have

$$\sigma(\boldsymbol{s}_m, \boldsymbol{s}_{m^*}) = e^{\beta_0} \frac{e^{\beta_1 z_1(\boldsymbol{s}_m)} + e^{\beta_1 z_1(\boldsymbol{s}_{m^*})}}{2} \tag{9}$$

so that positive β_1 implies that as z_1 incereases, the range of animal movement increases (or conversely, the cost of movement decreases). That is, z_1 facilitates movement.

Both of these seem reasonable enough (depending on the context of course), but they do not use distance metrics that have appeared in the literature.