Adaptive Geometry of Resource Competition

On these pages I am investigating the evolution of interactions in the MacArthur/Levins resource competition model.

This page explains the adaptive dynamics concepts and questions I'm using, and includes Sage source code for the models.

Child pages use this theory and code to investigate specific cases of the models:

- Adaptive Geometry of Resource Competition: One Species, One Resource
- Adaptive Geometry of Resource Competition: Two Species, Two Resources

MacArthur-Levins population dynamics

We start with a population-resource model:

$$\frac{dX_i}{dt} = b_i X_i (\sum_\ell c_{i\ell} w_\ell R_\ell - m_i)$$

$$\frac{dR_\ell}{dt} = r_\ell (K_\ell - R_\ell) - \sum_i c_{i\ell} X_i$$

where X_i is the density of population i, R_ℓ is the abundance of resource ℓ and the parameters are b_i, an intrinsic population growth rate, m_i, mortality rate, $c_{i\ell}$, the rate at which population i captures resource ℓ, w_ℓ, the amount a unit of resource ℓ contributes to population growth, r_ℓ, the resupply rate of resource ℓ, and K_ℓ its maximum possible abundance.

This is a fine model on its own, but I am interested in Lotka-Volterra models, which are models in which there is a simple number a_{ij} describing how populations i and j interact with each other. Using a Lotka-Volterra model will let us look at how these interaction terms a_{ij} change, telling us how evolution drives these populations to become more competitive, antagonistic, or mutualistic.

In the above model the populations i interact indirectly by taking resources from each other, but we can make the interactions direct by making simplifying assumptions, and this is what MacArthur and Levins did.

We do this by assuming that the resources come to equilibrium very quickly compared to the populations. Under this assumption, we can hold the population sizes X_i fixed and solve the second equation above for R_ℓ when $dR_\ell/dt = 0$:

$$\hat{R}_\ell = K_\ell - \frac{1}{r_\ell} \sum_i c_{i\ell} X_i.$$

Then we simply use that value of R_ℓ in the first equation, and we have a system of population sizes only:

$$\frac{dX_i}{dt} = b_i X_i \left(\sum_\ell c_{i\ell} w_\ell \left(K_\ell - \frac{1}{r_\ell} \sum_j c_{j\ell} X_j \right) - m_i \right).$$

We can rearrange the terms of this to have the standard Lotka-Volterra form:

$$\frac{dX_i}{dt} = k_i X_i + \sum_j a_{ij} X_i X_j,$$

where

$$k_i = b_i \left(\sum_\ell c_{i\ell} w_\ell K_\ell - m_1 \right) \text{ (ordinarily I'd call this } r_i\text{, but the name is already in use)},$$

$$a_{ij} = -b_i \sum_\ell \sum_j \frac{c_{i\ell} c_{j\ell} w_\ell}{r_\ell}.$$

I'll be interested in how the interaction terms a_{ij} change as the populations coevolve, because this expresses whether competition becomes stronger or weaker. When two coevolving populations compete for resources, we expect them to differentiate from each other and lessen the competition, but I'd like to look at both that and other cases, and do some detailed analysis about when it lessens and when it greatens.

Adaptive dynamics in the Macarthur-Levins model

And now here's where we look at how that single population "evolves" in the Mac-Lev model. I won't review the details of how adaptive dynamics is done, but in summary: suppose that certain aspects of the population dynamics depend on the characteristics of the population, and those characteristics are able to mutate. Then over time, mutants will arise, and if they are better able to thrive in the environment they encounter than their forefathers, they will gradually replace them, and the characteristics of the population will slowly change.

In this model, it's b_i, $c_{i\ell}$, and m_i that have to do with the populations, so we introduce a ''phenotype" variable u_i representing the characteristics of the population and suppose that b_i, $c_{i\ell}$ and m_i are determined by the value of u. Then we can find out how u changes in time in response to the conditions created by the population dynamics, and also how the other values such as X_i, R_ℓ, a_{ij}, k_i, change as u evolves.

The change in u, to make a long story short, is driven by how the population growth rate of a rare mutant varies with the mutant's phenotype u. That is, there's an "invasion speed" $\mathcal{I}$ that is closely related to the population growth rate $\frac{dX_i}{dt}$, and the change in u (that is, $\frac{du}{dt}$) is in proportion to $\frac{\partial \mathcal{I}}{\partial u}$.

To be precise:

$$\mathcal{I}(u_i|E) = \lim_{X_i \to 0} \frac{1}{X_i} \frac{dX_i}{dt}$$

defines the invasion speed (where E is the environment that an individual population i experiences, including the rest of population i and all other populations), and then

$$\frac{du_i}{dt} = \gamma \hat{X}_i \frac{\partial \mathcal{I}(v|u_1, \ldots, u_n)}{\partial v}\bigg|_{v=u_i},$$

where γ is a coefficient accounting for the frequency and size of available mutations. In simple cases, it's a constant, while when available mutations are not quite so simple and uniform, it may not be (as we will see).

Adaptive geometry of ecological parameters

Now let's go deeper into what happens in that evolutionary change. We're considering several different ways to describe the evolving population, and now I want to give them clearer notation:

- $\mathbf{u}_i$, the "underlying phenotype". I'm going to be using a scalar u_i, and in fact just one of them, so I can call it u, but in general it's a vector of numbers, which is what I mean by writing it in boldface.
- $\mathbf{p}(\mathbf{u}_i) = \begin{pmatrix} b_i(\mathbf{u}_i) \\ m_i(\mathbf{u}_i) \\ c_{i1}(\mathbf{u}_i) \\ \vdots \\ c_{im}(\mathbf{u}_i) \end{pmatrix}$, the "ecological phenotype"
- $\mathbf{A}(\mathbf{p}(\mathbf{u}_i), \mathbf{p}(\mathbf{u}_1), \ldots, \mathbf{p}(\mathbf{u}_n)) = \begin{pmatrix} k_i(\mathbf{p}(\mathbf{u}_i)) \\ a_{i1}(\mathbf{p}(\mathbf{u}_i), \mathbf{p}(\mathbf{u}_1)) \\ \vdots \\ a_{in}(\mathbf{p}(\mathbf{u}_i), \mathbf{p}(\mathbf{u}_n)) \end{pmatrix}$, the "interaction phenotype"

These are vectors that are functions of one another. Using a suitable vector notation in our calculus, we can write the adaptive dynamics of $\mathbf{u}$ directly, ignoring the intermediate variables $\mathbf{p}$ and $\mathbf{A}$ for the moment:

$$\begin{aligned} \frac{d\mathbf{u}_i}{dt} &= \gamma \hat{X}_i \frac{\partial \mathcal{I}(\mathbf{v}|\mathbf{u}_1, \ldots, \mathbf{u}_n)}{\partial \mathbf{v}}\bigg|_{\mathbf{v}=\mathbf{u}_i} \\ &= \gamma \hat{X}_i \partial_1 \mathcal{I}(\mathbf{u}_i)^T. \end{aligned}$$

The T sign is for vector or matrix transposition: with the notation I'm using, $\mathbf{u}_i$ is a column vector - a point in a vector space of values - and $\frac{\partial \mathcal{I}}{\partial \mathbf{u}_i}$ is a row vector - a thing that operates on points. Since $\frac{d\mathbf{u}_i}{dt}$ is a column, we need to transpose the derivative of $\mathcal{I}$ to make it match. The ∂_1 sign is for the partial derivative with respect to the first argument.

Now we can use the chain rule to see how $\mathbf{p}$ changes:

$$\begin{aligned} \frac{d\mathbf{p}(\mathbf{u}_i)}{dt} &= \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i} \frac{d\mathbf{u}_i}{dt} \\ &= \gamma \hat{X}_i \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i} \partial_1 \mathcal{I}(\mathbf{u}_i)^T \\ &= \gamma \hat{X}_i \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i} (\partial_1 \mathcal{I}(\mathbf{p}(\mathbf{u}_i)) \frac{d\mathbf{p}}{d\mathbf{u}_i})^T \\ &= \gamma \hat{X}_i \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i} \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i}^T \partial_1 \mathcal{I}(\mathbf{p}(\mathbf{u}_i))^T. \end{aligned}$$

Here we see the usefulness of this row-and-column-vector notation, because it lets us use the chain rule in a natural way: when we write $\frac{d\mathbf{p}(\mathbf{u}_i)}{dt} = \frac{\partial \mathbf{p}}{\partial \mathbf{u}_i} \frac{d\mathbf{u}_i}{dt}$, we're

multiplying a matrix by a column vector to get another column vector, in just the way we want.

I like to write $S(\mathbf{u}) = \partial_1 \mathcal{I}(\mathbf{u})^T$ for the "selection gradient" of $\mathbf{u}$ – the direction of increasing fitness in the space of possible $\mathbf{u}$ values. I'm very interested in the role of this selection gradient in various spaces, as we'll see. Notice that in the derivation above I switched from $S(\mathbf{u}_i) = \left(\frac{\partial \mathcal{I}(\mathbf{v}|\mathbf{u}_1,\ldots,\mathbf{u}_n)}{\partial \mathbf{v}}\right)_{\mathbf{v}=\mathbf{u}_i}$ to $S(\mathbf{p}(\mathbf{u}_i)) = \begin{pmatrix} \frac{\partial \mathcal{I}(\mathbf{v}|\mathbf{u}_1,\ldots,\mathbf{u}_n)}{\partial b(\mathbf{v})} \\ \vdots \\ \frac{\partial \mathcal{I}(\mathbf{v}|\mathbf{u}_1,\ldots,\mathbf{u}_n)}{\partial c_{im}(\mathbf{v})} \end{pmatrix}_{\mathbf{v}=\mathbf{u}_i}$. These different selection gradients are very different objects: they're vectors expressing the direction of selection in different spaces – here, the space of $\mathbf{u}$ values "vs." the space of $\mathbf{p}$ values. Soon enough we'll also be looking at the selection gradient in $\mathbf{A}$ space.

Using the above chain rule manipulations, we can write

$$\frac{d\mathbf{u}_i}{dt} = \gamma \hat{X}_i S(\mathbf{u}_i)$$

$$\frac{d\mathbf{p}(\mathbf{u}_i)}{dt} = \gamma \hat{X}_i \frac{\partial \mathbf{p}}{\partial \mathbf{u}} \frac{\partial \mathbf{p}}{\partial \mathbf{u}}^T S(\mathbf{p}).$$

Like $\mathbf{u}$ or whatever other vector, $\mathbf{p}$ would evolve in the direction of $S(\mathbf{p})$ if it were to mutate in all directions equally. However, it doesn't do that: since $\mathbf{p}$ is a function of $\mathbf{u}$, it only mutates in directions given by mutations in $\mathbf{u}$. The matrix multiplying $S(\mathbf{p})$ in the dynamics of $\mathbf{p}$ is a projection matrix, restricting the motion of $\mathbf{p}$ to directions allowed by the mapping between $\mathbf{u}$ and $\mathbf{p}$. (To be precise, $\frac{\partial \mathbf{p}}{\partial \mathbf{u}} \frac{\partial \mathbf{p}}{\partial \mathbf{u}}^T S(\mathbf{p})$ is not precisely the projection of $S(\mathbf{p})$ onto the subspace parametrized by $\mathbf{u}$ unless the columns of $\frac{\partial \mathbf{p}}{\partial \mathbf{u}}$ are unit-length; otherwise there's some scaling by positive numbers involved. But it definitely transforms the vector into a direction allowed by the restriction to points parametrized by $\mathbf{u}$.)

The motion of $\mathbf{A}$ is the same way, except that it is influenced by all the populations in the system, not just one, because two phenotypes are involved in each a_{ij} value. For that reason, the dynamics of $\mathbf{A}$ has some extra terms...

But before we go into that, let's look at the selection gradient and the dynamics of $\mathbf{p}(\mathbf{u})$.

Adaptive geometry of interaction terms

Now, let's get to how a_{ij} and k_i "would like to evolve" and why they move in the directions they do.

Expanding out the dynamics of the $\mathbf{A}$ vector has more involved than doing it for the $\mathbf{p}$ vector, because $\mathbf{A}$ depends on all the different phenotypes $\mathbf{u}_j$, not just one we're concerned with. Also, we have to keep careful track of $\mathbf{u}_i$, because it appears in $\mathbf{A}$ in two different ways: on the left-hand side of each $\mathbf{a}_{ij} = a(\mathbf{u}_i, \mathbf{u}_j)$ term, which describes how population i (the "patient") is affected by an encounter with population j (the "agent"); and also on the right-hand side of the $a_{ii} = a(\mathbf{u}_i, \mathbf{u}_i)$ term, as the "agent" in an encounter between i and i. This distinction is important because every encounter has two effects, the effect on oneself and the effect on the other, and we'll see that selection treats these two effects very differently.

So to be clear, we'll work with the notation

$$\mathbf{A}_i = \mathbf{A}(\mathbf{v}, \mathbf{u}_1, \ldots, \mathbf{u}_n)|_{\mathbf{v}=\mathbf{u}_i} = \begin{pmatrix} \mathbf{a}(\mathbf{v}, \mathbf{u}_1) \\ \vdots \\ \mathbf{a}(\mathbf{v}, \mathbf{u}_n) \\ k(\mathbf{v}) \end{pmatrix}_{\mathbf{v}=\mathbf{u}_i},$$

to distinguish the two different roles of $\mathbf{u}_i$ (suppressing the intermediate variable $\mathbf{p}$), and we'll refer to the two different partial derivatives that relate to $\mathbf{u}_i$ as $\partial_1 \mathbf{A}_i(\mathbf{u}_i) = \partial \mathbf{A}_i / \partial \mathbf{v}|_{\mathbf{v}=\mathbf{u}_i}$ and $\partial_2 \mathbf{A}_i(\mathbf{u}_i) = \partial \mathbf{A}_i / \partial \mathbf{u}_i|_{\mathbf{v}=\mathbf{u}_i}$.

So first of all, if we use $\mathbf{A}_i$ as an intermediate variable, the dynamics of $\mathbf{u}_i$ is

$$\begin{aligned} \frac{d\mathbf{u}_i}{dt} &= \gamma \hat{X}_i \partial_1 \mathscr{I}(\mathbf{u}_i)^T \\ &= \gamma \hat{X}_i (\partial_1 \mathscr{I}(\mathbf{A}_i) \partial_1 \mathbf{A}_i(\mathbf{u}_i))^T \\ &= \gamma \hat{X}_i \partial_1 \mathbf{A}_i(\mathbf{u}_i)^T \partial_1 \mathscr{I}(\mathbf{A}_i)^T. \end{aligned}$$

Then

$$\begin{aligned} \frac{d\mathbf{A}_i}{dt} &= \partial_1 \mathbf{A}_i(\mathbf{u}_i) \frac{d\mathbf{u}_i}{dt} + \sum_{j=1}^{n} \partial_2 \mathbf{A}_i(\mathbf{u}_j) \frac{d\mathbf{u}_j}{dt} \\ &= \gamma \hat{X}_i \partial_1 \mathbf{A}_i(\mathbf{u}_i) \partial_1 \mathbf{A}_i(\mathbf{u}_i)^T \partial_1 \mathcal{I}(\mathbf{A}_i)^T \\ &\quad + \sum_{j=1}^{n} \gamma \hat{X}_j \partial_2 \mathbf{A}_i(\mathbf{u}_j) \partial_1 \mathbf{A}_j(\mathbf{u}_j)^T \partial_1 \mathcal{I}(\mathbf{A}_j)^T \\ &= \gamma \hat{X}_i \partial_1 \mathbf{A}_i(\mathbf{u}_i) \partial_1 \mathbf{A}_i(\mathbf{u}_i)^T S(\mathbf{A}_i) + \sum_{j=1}^{n} \gamma \hat{X}_j \partial_2 \mathbf{A}_i(\mathbf{u}_j) \partial_1 \mathbf{A}_j(\mathbf{u}_j)^T S(\mathbf{A}_j). \end{aligned}$$

This expression is made up of two somewhat complex terms. The first term, the $S(\mathbf{A}_i)$ term, expresses the change in the interactions experienced by population

i due to change in population i in its patient role. I refer to this as the **direct effect** of selection on population i. The second term, the sum of $S(\mathbf{A}_j)$ vectors, expresses the change in population i's interactions due to change in all the different agents it encounters, including population i itself in its agent role. I refer to this as an **indirect effect** of selection on the various populations.

What would happen if there were no constraints on $\mathbf{A}_i$? That is, if all the terms just mutate independently without being constrained by dependence on $\mathbf{u}$ variables. In this case, we would simply have

$$\frac{d\mathbf{A}_i}{dt} = \gamma \hat{X}_i S(\mathbf{A}_i).$$

In the constrained case, as with $\mathbf{p}$, the motion due to $S(\mathbf{A}_i)$ is modified by a transformation matrix. Here, though, there's also another term dealing with the effect of changing $\mathbf{u}$ values as the second argument to $a(\mathbf{u}_i, \mathbf{u}_j)$. Let's expand that out one step further:

$$\begin{aligned}\frac{d\mathbf{A}_i}{dt} &= \gamma \hat{X}_i \partial_1 \mathbf{A}_i(\mathbf{u}_i) \partial_1 \mathbf{A}_i(\mathbf{u}_i)^T S(\mathbf{A}_i) + \sum_{j=1}^{n} \gamma \hat{X}_j \partial_2 \mathbf{A}_i(\mathbf{u}_j) \partial_1 \mathbf{A}_j(\mathbf{u}_j)^T S(\mathbf{A}_j) \\ &= D(\mathbf{u}_i) + I(\mathbf{u}_i | \mathbf{u}_1, \ldots, \mathbf{u}_n).\end{aligned}$$

Here I'm using colors to distinguish three different vector quantities, which I'll soon plot in the same colors:

- $\frac{d\mathbf{A}_i}{dt}$, the resultant change in $\mathbf{A}_i$, blue.
- $D(\mathbf{u}_i)$, the direct effect of change in $\mathbf{u}_i$ as patient, green.
- $I(\mathbf{u}_i | \mathbf{u}_1, \ldots, \mathbf{u}_n)$, the indirect effect of change in all agents $\mathbf{u}_j$ including $\mathbf{u}_i$, purple.

Supplementary materials

Here are the Sage classes that do the work for the Mac-Lev models. above. They use generalized Sage machinery that's stored at [[SageDynamics]].

Now here's the MacArthur-Levins resource competition model.

```
from sage.all import *

from dynamicalsystems import *

from sage.symbolic.relation import solve
from sage.symbolic.function_factory import function

# functional forms used in the definitions of ResourceCompetitionModel
```

```
# and MacArthurLevinsModel

def c_var(i, l):
    return SR.var( 'c_%s_%s'%(i,l), latex_name='c_{%s%s}'%(i,l) )

def b_var(i):
    return SR.var( 'b_%s'%i )

def m_var(i):
    return SR.var( 'm_%s'%i )

class ResourceCompetitionModel(PopulationDynamicsSystem):
    """The general resource competition model, with some number of
    populations and some number of resources.  The various parameters
    b_i, c_i_l, etc. may or may not be bound to numerical values.
    """
    def __init__(self, x_indices=None, r_indices=None,
        X = indexer('X'),
        R = indexer('R'),
        b = indexer(b_var),
        m = indexer(m_var),
        w = indexer('w'),
        r = indexer('r'),
        K = indexer('K'),
        c = indexer_2d('c'),
        bindings=Bindings()):
        """How to specify an instance of this model?

        x_indices: might, for instance, be [1, 2, 3], or might not.
        r_indices: likewise.
        X, R, b, etc: some kind of things that can implement operations
            such as, for example, X[i] where i is in x_indices.
These
            should return variables, functions, numbers, or other
            things that can be used in an expression.
        """
        self._r_indices = r_indices
        self._indexers = dict( X=X, R=R, b=b, m=m, c=c, w=w, r=r, K=K )
        super(ResourceCompetitionModel, self).__init__(
            [R[l] for l in r_indices], x_indices, X,
            bindings = bindings )
    def flow(self):
        return self.make_flow(**self._indexers)
    def make_flow(self, X, R, b, c, m, r, w, K):
        X_flows = [ (X[i],
            b[i]*X[i]*(sum(c[i][l]*w[l]*R[l] for l in self._r_indices) - m[i]))
            for i in self._population_indices ]
        R_flows = [ (R[l],
            r[l]*(K[l] - R[l]) -
              sum(c[i][l]*X[i] for i in self._population_indices))
            for l in self._r_indices ]
        return dict(X_flows + R_flows)
    def plot_R_ZNGIs( self, with_perpendicular=True, **options ):
        """This probably doesn't do anything helpful when the number of
        resources or populations isn't 2"""
        R_0, R_1 = ( self._indexers['R'][i] for i in (0,1) )
        X_0, X_1 = ( self._population_indexer[i] for i in (0,1) )
```

```
        P = Graphics()
        if with_perpendicular:
            P += point( self._bindings( vector( [ self._indexers['K'][i] for i in (0,1) ] ) ),
        # aux. line crossing R*
        for x_in in self.population_vars():
            zngi = solve( self._flow[x_in]/x_in == 0, R_1, solution_dict=True )[0][R_1]
            zngi_options = dict( options, color='gray', thickness=2, fill=True, fillcolor='gra
            del zngi_options['filename']
            r0max = solve( zngi, R_0, solution_dict=True )[0][R_0]
            #print 'plot zngi:', zngi
            #sys.stdout.flush()
            P += plot( zngi, (R_0, 0, r0max), **zngi_options )
            if with_perpendicular:
                elim_x = Bindings( { x_out: 0 for x_out in self.population_vars() if x_out !=
                # todo: plot parametrized by x_in, if needed for vertical case
                zri_soln = solve( [
                    elim_x( self._flow[R_0] == 0 ),
                    elim_x( self._flow[R_1] == 0 ) ],
                    [ x_in, R_1 ], solution_dict=True )[0]
                zri_r1 = zri_soln[R_1]
                #print 'plot zri:', zri_r1
                #sys.stdout.flush()
                zri_options = dict( options, thickness=1,
                   color = { X_0:'blue', X_1:'red' }[x_in] )
                del zri_options['filename']
                P += plot( zri_r1, (R_0,0,4), **zri_options )
                rstar_soln = solve( [ R_1 == zri_r1, R_1 == zngi ], [ R_0, R_1 ], solution_dic
                rstar = Bindings( rstar_soln )( vector( [ R_0, R_1 ] ) )
                P += point( rstar, color='gray', size=30 )
        if 'filename' in options:
            P.save( **options )
        return P

class MacArthurLevinsModel(PopulationDynamicsSystem):
    """The MacArthur-Levins model is constructed by separating timescales
    to remove the R_l variables from the dynamics"""
    def __init__(self, rescomp_model=None, **named_args):
        """Give me a ResourceCompetitionModel with nx populations and
        nr resources, I will crunch it down into a MacArthurLevinsModel
        with nx populations."""
        #print 'MacArthurLevinsModel', (rescomp_model, named_args)
        if rescomp_model is None:
            rescomp_model = ResourceCompetitionModel(**named_args)
        self._rescomp_model = rescomp_model
        self._r_indices = rescomp_model._r_indices
        super(MacArthurLevinsModel, self).__init__(
          [],
          rescomp_model._population_indices,
          rescomp_model._population_indexer,
          bindings = rescomp_model._bindings )
    def flow(self):
        # find the equilibrium values of R_l given the current values X_i
        R = [self._rescomp_model._indexers['R'][l] for l in self._r_indices]
        Rsolns = { R_l:self.solve_for_R(R_l) for R_l in R }
        #print 'R solns:', Rsolns
        # now substitute those values of R into the X equations
        reduced_flow = dict( (X_i,
```

```
            self._rescomp_model._flow[X_i].substitute_expression(Rsolns) )
          for X_i in self.population_vars() )
        # that's the Macarthur-Levins model.
        return reduced_flow
    def solve_for_R(self, R_l):
        """solve for the equilibrium value of one R_l given the X_i values"""
        Rhat_soln = solve( 0 == self._rescomp_model._flow[R_l], R_l, solution_dict=True )
        #print 'solve for', R_l, 'in', [ 0 == self._rescomp_model._flow[R_l] ], ' => ', Rhat_s
        # there should be just one solution
        assert len(Rhat_soln) == 1
        #print "R-hat: %s" % Rhat_soln[0][R_l]
        # put it in the bindings in case we want to plot R_l or something
        self._bindings[R_l] = Rhat_soln[0][R_l]
        # it's also its own equilibrium
        self._bindings[hat(R_l)] = self._rescomp_model.add_hats( Rhat_soln[0][R_l] )
        return Rhat_soln[0][R_l]
    def set_population_indices(self,xi):
        # need to pass this through to the res-comp system
        self._rescomp_model.set_population_indices(xi)
        super(MacArthurLevinsModel, self).set_population_indices(xi)
```