Adaptive Geometry and Resource Competition: One Species, One Resource

This page investigates the adaptive dynamics of the Macarthur-Levins resource competition model in the special case of one species and one resource.

- See Selection Gradients/MacLev.html for the general model, and the adaptive dynamics formulation we're using here, including the specific questions about the a and k of the Lotka-Volterra framework.

Before we do evolution, we start by defining the basic population dynamics. Here I'm just going to consider one population and one resource. This configuration is well suited to visualizing in two-dimensional plots.

The model equations, according to Sage: The original resource competition system:

$$\frac{dR_0}{dt} = -X_0 c_{00} + (K_0 - R_0) r_0$$

$$\frac{dX_0}{dt} = (R_0 c_{00} w_0 - m_0) X_0 b_0$$

And the Mac-Lev system derived from it:

$$\frac{dX_0}{dt} = -\left(\frac{(X_0 c_{00} - K_0 r_0) c_{00} w_0}{r_0} + m_0\right) X_0 b_0$$

Adaptive dynamics of u

```
# requires: maclevmodels.py maclev-1-1.sobj
# produces: maclev-1-1-adap.sage.out.tex maclev-1-1-adap.sobj
# produces: maclev-1-1-c.png maclev-1-1-X.png maclev-1-1-ak.png
# produces: maclev-1-1-R.png maclev-1-1-a.png maclev-1-1-k.png
# produces: maclev-1-1-R.svg maclev-1-1-a.svg maclev-1-1-k.svg maclev-1-1-c.svg
from sage.all import *
from sage.misc.latex import _latex_file_
from sage.misc.latex import latex
latex.add_to_preamble('\\usepackage{amsmath}')
```

```
from maclevmodels import *
from dynamicalsystems import latex_output

# create variables with custom latex names because load_session
# creates them wrong
# http://trac.sagemath.org/ticket/17559
for x in ('X_0', 'X_i', 'R_0'): hat(SR.symbol(x))
SR.symbol( 'c_0_0', latex_name='c_{00}' )
SR.symbol( 'c_i_0', latex_name='c_{i0}' )

load_session('maclev-1-1')

ltx = latex_output( 'maclev-1-1-adap.sage.out.tex' )

u_indexer = indexer('u')
u_0 = u_indexer[0]

evol_c_bindings = Bindings( reduce( operator.add, (
    [ ( rescomp._indexers['c'][i][0], u_indexer[i] ),
      ( rescomp._indexers['b'][i], 1 ),
      ( rescomp._indexers['m'][i], 1 )
    ] for i in (0,1) ) ) )

maclev_adap = AdaptiveDynamicsModel(maclev,
    [ indexer('u') ], early_bindings=evol_c_bindings )

print 'when c_00 == u, the adaptive dynamics is ', maclev_adap

ltx.write( 'Adaptive dynamics of the Mac-Lev model with $b_i=m_i=1$ and $c_{i0} = u_i$:' )
ltx.write( maclev_adap )

numeric_params = Bindings(
  { SR.var('r_0'): 1, SR.var('w_0'): 1,
    SR.var('K_0'): 2, SR.var('gamma'): 1 } )
maclev_adap_bound = maclev_adap.bind( numeric_params )

ltx.write( 'and the bound adaptive dynamics is:', latex( maclev_adap_bound ) )

ltx.write( 'its bindings:', latex(maclev_adap_bound._bindings) )
initial_u = 2/3
c_evolution = maclev_adap_bound.solve( [initial_u], end_time=10 )

# and plot.
t = maclev_adap.time_variable()
c_evolution.plot( t, u_indexer[0], filename='maclev-1-1-c.png', ymin=0, ylabel='$c_{00}$', fig
c_evolution.plot( t, u_indexer[0], filename='maclev-1-1-c.svg', ymin=0, ylabel='$c_{11}$', fig
# Xhat increases.
Xhat = hat( maclev_adap_bound._popdyn_model._population_indexer[0] )
c_evolution.plot( t, evol_c_bindings( maclev_adap_bound._bindings( Xhat ) ), filename="maclev-
Rhat = hat( maclev_adap_bound._popdyn_model._rescomp_model._indexers['R'][0] )
c_evolution.plot( t, evol_c_bindings( maclev_adap_bound._bindings( Rhat ) ), filename="maclev-
c_evolution.plot( t, evol_c_bindings( maclev_adap_bound._bindings( Rhat ) ), filename="maclev-
# and let's also plot k_0 vs. a_00.
# these are the coefficients if you rewrite the maclev dynamics
# as dX_0/dt = X_0(k_0 + a_00 X_0)
# and why not get sage to do that, rather than type in the definitions
```

```
X_0 = maclev._population_indexer[0]
rhsc = maclev._flow[X_0].collect(X_0)
k_0 = rhsc.coeff(X_0,1)
a_00 = rhsc.coeff(X_0,2)
ltx.write( 'With and without these bindings, the Lotka-Volterra coefficients $k_0$ and $a_{00}
ltx.write_equality( SR.var('k_0'), k_0, numeric_params( maclev_adap_bound._early_bindings(k_0)
ltx.write_equality( SR.var('a_0_0', latex_name='a_{00}' ), a_00, numeric_params( maclev_adap_b

ltx.write( 'And the equilibrium values $\\hat X$ and $\\hat R$:',
    '\\begin{align*}\n  \\hat X_0 &\\to ',
    latex( maclev_adap_bound._equilibrium( Xhat ).expand() ), ' = ',
    latex( maclev_adap_bound._bindings( Xhat ).expand() ), '\\\\\n',
    '  \\hat R_0 &\\to ',
    latex( maclev_adap_bound._equilibrium( maclev_adap_bound._popdyn_model._bindings( Rhat ) )
    latex( maclev_adap_bound._bindings( Rhat ).expand() ), '\n',
    '\\end{align*}\n' )

c_evolution.plot( t, maclev_adap_bound._early_bindings( k_0 ), filename="maclev-1-1-k.svg",
  ymin=0, ylabel='$k_1$', figsize=(2,2) )
c_evolution.plot( t, maclev_adap_bound._early_bindings( k_0 ), filename="maclev-1-1-k.png",
  figsize=(4,4) )

c_evolution.plot( t, maclev_adap_bound._early_bindings( a_00 ), filename="maclev-1-1-a.svg",
  ymax=0, ylabel='$a_{11}$', figsize=(2,2) )
c_evolution.plot( t, maclev_adap_bound._early_bindings( a_00 ), filename="maclev-1-1-a.png",
  ymax=0, ylabel='$a_{11}$', figsize=(4,4) )

c_evolution.plot( maclev_adap_bound._early_bindings( a_00 ), maclev_adap_bound._early_bindings
  xlabel = SR.var('a_00'), ylabel = SR.var('k_0'), figsize=(4,4) )

ltx.close()

save_session('maclev-1-1-adap')
```

Here we derive the adaptive dynamics equations when it is the uptake rate c_{00} that is evolving – that is, when $c_{00}(u) = u$, $b_0 = 1$, and $m_0 = 1$; and then we bind the remaining parameters to get a realization of the system that we can integrate and plot.

Adaptive dynamics of the Mac-Lev model with $b_i = m_i = 1$ and $c_{i0} = u_i$:

$$\frac{du_0}{dt} = \frac{(K_0 r_0 u_0 w_0 - r_0)\left(K_0 r_0 w_0 - \frac{K_0 r_0 u_0 w_0 - r_0}{u_0}\right)\gamma}{r_0 u_0^2 w_0}$$

and the bound adaptive dynamics is:

$$\frac{du_0}{dt} = -\frac{(2\,u_0 - 1)\left(\frac{2\,u_0 - 1}{u_0} - 2\right)}{u_0^2}$$

its bindings:`{Xhat_0 -> (2*u_0 - 1)/u_0^2, m_0 -> 1, R_0 -> -X_0*u_0 + 2, b_1 -> 1, K_0 -> 2, r`
and without these bindings, the Lotka-Volterra coefficients k_0 and a_{00} are:

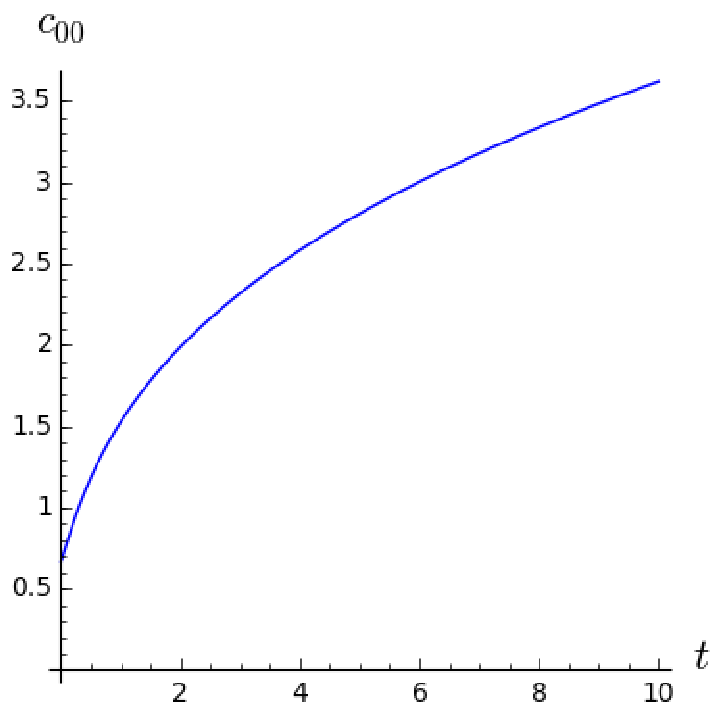
$$k_0 = K_0 b_0 c_{00} w_0 - b_0 m_0 = 2\,u_0 - 1$$

$$a_{00} = -\frac{b_0 c_{00}^2 w_0}{r_0} = -u_0^2$$

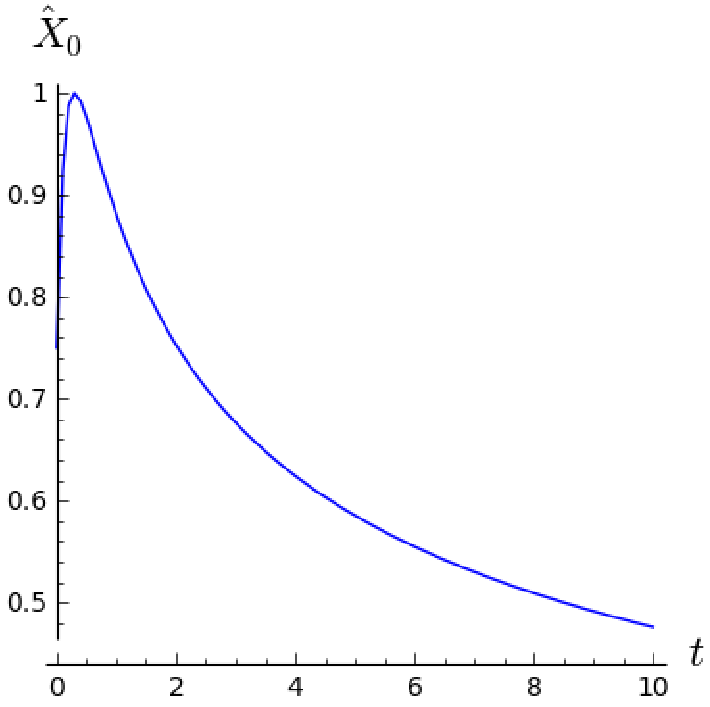
And the equilibrium values $\hat{X}$ and $\hat{R}$:

$$\hat{X}_0 \to \frac{K_0 r_0}{c_{00}} - \frac{m_0 r_0}{c_{00}^2 w_0} = \frac{2}{u_0} - \frac{1}{u_0^2}$$
$$\hat{R}_0 \to -K_0 r_0 + \frac{m_0 r_0}{c_{00} w_0} + 2 = \frac{1}{u_0}$$

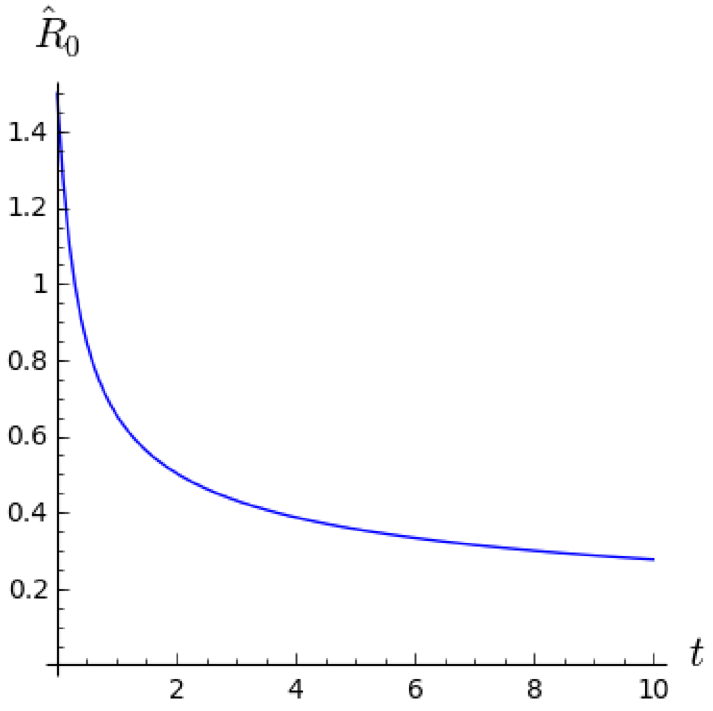
With these parameters, the uptake rate c_{00} $(= u_0)$ increases as long as it starts above $\frac{1}{2}$. This is the only relevant range, because it has to be above $\frac{1}{2}$ in order for the population to be viable $(\hat{X}_0 > 0)$.



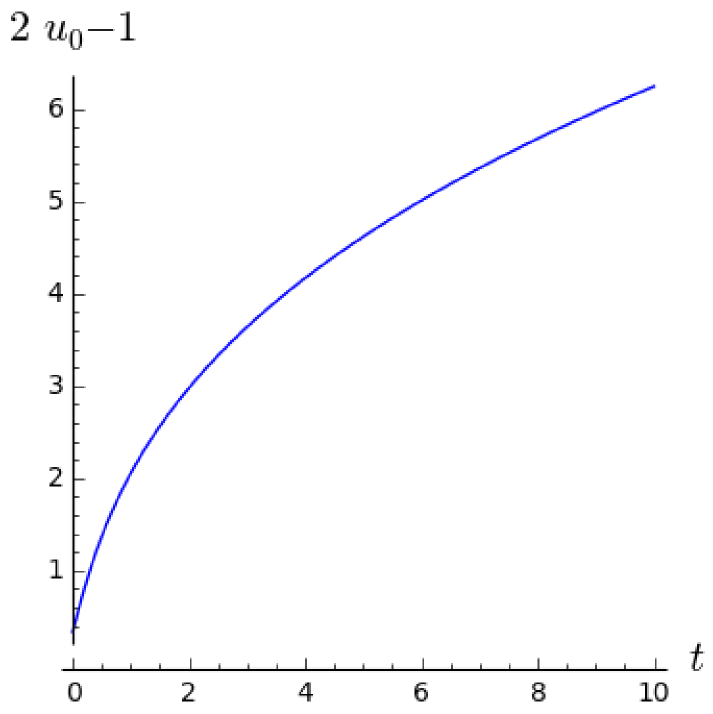
The equilibrium population size $\hat{X}_0$ increases to 1 and then decreases.

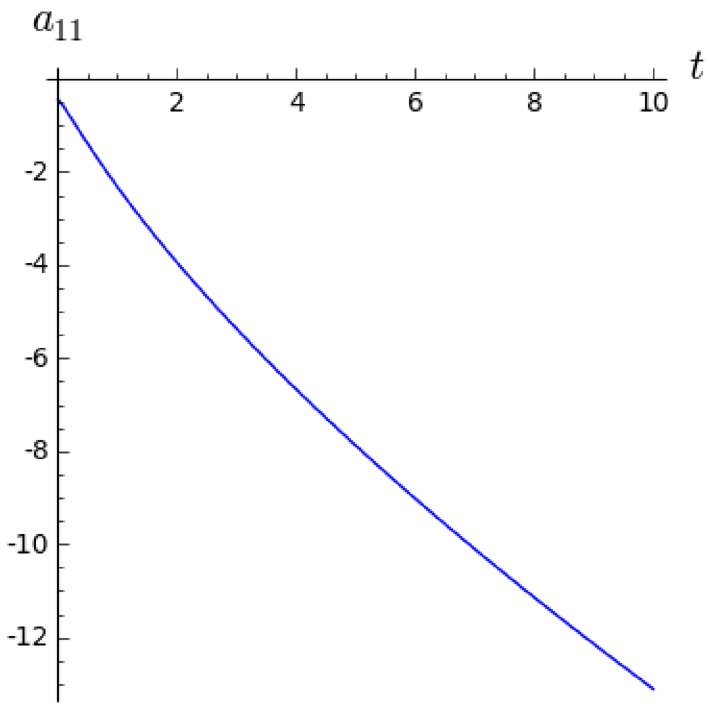


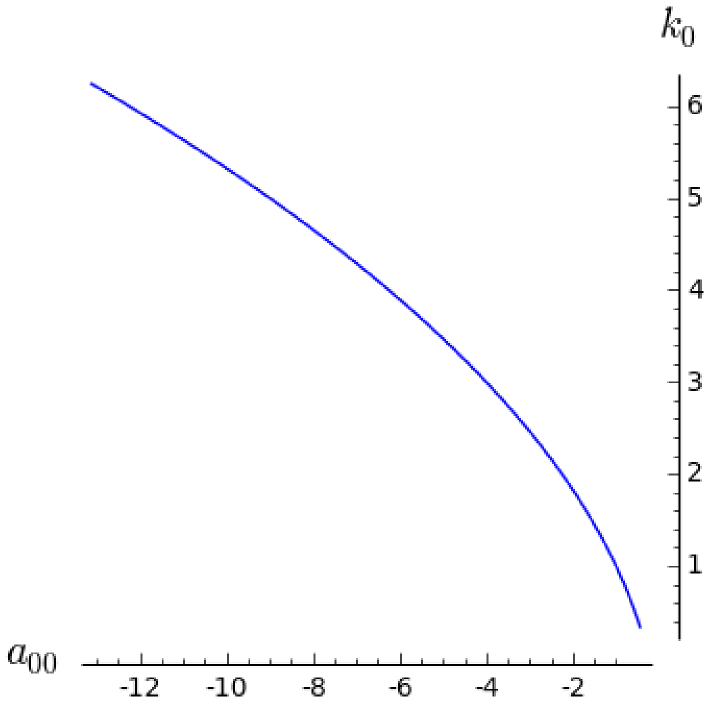
The availability of the resource $\hat{R}_0$ decreases.



And the Lotka-Volterra coefficients k_0, a_{00} both move away from zero. Since a_{00} is negative, this indicates an intensification of competition.


$2\,u_0-1$
t
0
1
2
3
4
5
6
0
2
4
6
8
10


a_{11}
t
2
4
6
8
10
-2
-4
-6
-8
-10
-12



Adaptive dynamics of c and m

```
# requires: maclevmodels.py maclev-1-1-adap.sobj
# produces: maclev-1-1-mc-adap-geom.sobj maclev-1-1-mc.png maclev-1-1-mc-adap-geom.sage.out.te
from sage.all import *

from maclevmodels import *
from dynamicalsystems import latex_output

from sage.symbolic.function_factory import function

load_session("maclev-1-1-adap")

# Notes: whoo, boy, I need to abstract out more of this and make this
# file simpler!

ltx = latex_output( 'maclev-1-1-mc-adap-geom.sage.out.tex' )

# Now: project the p vector into two dimensions, for the 1-species
# model (e.g. plot b_1 and c_11).  Plot the curve defined by p(u) (for
# one-dimensional u), together with S(p) and the projection of S(p) onto
# the curve.

# This is the adaptive dynamics of unconstrained p, and its rhs is gamma X S(p)
maclev_p = MacArthurLevinsModel( x_indices = [0], r_indices = [0] )
```

```
maclev_adap_p = AdaptiveDynamicsModel( maclev_p,
    [ indexer('b'), indexer('m'), indexer(lambda i:'c_%s_0'%i) ])

b_0, m_0, c_0_0 = SR.var('b_0 m_0 c_0_0')
p = vector( (b_0, m_0, c_0_0) )

p_symb = SR.var( 'p', latex_name='\mathbf p' )
S_symb = function( 'S', print_latex_func=lambda f, v : r'\mathbf S(%s)' % latex(v) )

# now we recast b_0, m_0, c_0_0 to be functions of u_0, the phenotype

change_p_to_functions = Bindings( { 'c_0_0':function('c_0')(u_0), 'm_0':function('m')(u_0), 'b

flow_p_func = vector( change_p_to_functions(maclev_adap_p._flow[v]) for v in p )
p_func = vector( change_p_to_functions(v) for v in p )
#dp_du = diff(p_func, u_0)
#print 'dp/du =', dp_du

pf_symb = function( 'p', print_latex_func=lambda f, v : r'\mathbf p(%s)' % latex(v) )
#ltx.write( 'Now we change these 3 characters to be functions of $u$:\n\\[ ',
#  latex( pf_symb( u_0 ) ), ' = ',
#  latex( column_vector( p_func ) ), ', \\]\n',
#  '\\[ ', latex( S_symb( pf_symb( u_0 ) ) ), ' = ',
#  latex( column_vector( flow_p_func ) ), ' \\]\n' )

#print 'S(u) =', (S_p_func.dot_product(dp_du))

# Now the version with p constrained to (0, 0, u)
#identity(x) = x
#const_one(x) = 1
maclev_c00 = MacArthurLevinsModel( x_indices = [0], r_indices = [0],
       b = indexer(lambda i: 1),
       m = indexer(lambda i: 1),
       c = indexer(lambda i: indexer(lambda l: 'u_%s'%i)) )

#ltx.write( 'maclev_c00:' )
#ltx.write( maclev_c00 )

maclev_adap_c00 = AdaptiveDynamicsModel(maclev_c00,
    [ indexer('u') ])

#ltx.write( 'Returning to the adaptive dynamics, with $c_0(u_0)=u_0$, $b(u_0) = m(u_0) = 1$:'
#ltx.write( maclev_adap_c00 )

maclev_adap_c00_bound = maclev_adap_c00.bind( Bindings(
  { SR.var('r_0'): 1, SR.var('w_0'): 1,
    SR.var('K_0'): 2, SR.var('gamma'): 1 } ) )

#ltx.write( 'And with parameters bound:' )
#ltx.write( maclev_adap_c00_bound )

c00_evolution = maclev_adap_c00_bound.solve( [2] )

c00_bindings = c00_evolution._system._bindings.merge( FunctionBindings( {
    function('c_0'):u_0.function(u_0), function('m'):SR('1').function(u_0), function('b'):SR('
flow_p_bound = vector( (c00_bindings(s_i) for s_i in flow_p_func) )

S_p = vector( maclev_adap_p._S[ pi ] for pi in p )

#ltx.write( 'c00_bindings is ', latex( c00_bindings ) )

ltx.write( 'With $c_{00}=u_0$, the vector $\mathbf p$ is ' )
ltx.write_equality(  p_symb, column_vector( p ),
  column_vector( c00_bindings( change_p_to_functions( pi ) ) for pi in p ) )
ltx.write( 'And the selection gradient of $\mathbf p$ is ' )
ltx.write_equality( S_symb( p_symb ),
  column_vector( S_p ),
  column_vector( c00_bindings( change_p_to_functions( si ) ) for si in S_p ) )

#ltx.write( 'And with these bindings,\n\\[ ',
#  latex( S_symb( pf_symb( u_0 ) ) ),
#  ' = ', latex( column_vector( flow_p_bound ) ), ' \\]\n' )

dp_du_bound = vector( (diff(c00_bindings(p_i),u_0) for p_i in p_func) )

ltx.write( 'The derivative of $\mathbf p$ is \n\\[\n  ',
  '\\frac{d%s}{d%s}' % ( latex( p_symb ), latex( u_0 ) ), ' = ',
  latex( column_vector( dp_du_bound ) ), ', \\]\n' )
ltx.write( 'and we can recover the motion of $u_0$: \n\\[ ',
  latex( S_symb( u_0 ) ),
  ' = \\frac{d%s}{d%s}^{\mathrm{T}} %s = ' % ( latex(p_symb), latex(u_0), latex( S_symb( pf_sym
  latex( c00_bindings( change_p_to_functions( dp_du_bound.dot_product( S_p ) ) ) ), ' \\]\n' )
ltx.write_equality( wrap_latex( '\\frac{du_0}{dt}' ),
  wrap_latex( '\\gamma\\hat X_0 %s' %  latex( S_symb( u_0 ) ) ),
  latex( c00_bindings( change_p_to_functions( SR('gamma') * X_0 * dp_du_bound.dot_product( S_p

ltx.write( 'and the motion of $\\mathbf p(u_0)$:' )
ltx.write_equality( wrap_latex( '\\frac{d\\mathbf p(u_0)}{dt}' ),
  wrap_latex( '\\gamma\\hat X_0 \\frac{d\\mathbf p}{du_0} \\frac{d\\mathbf p}{du_0}^{\\mathrm{
latex( S_symb( pf_symb( u_0 ) ) ) ),
  latex( column_vector( c00_bindings( change_p_to_functions( z ) ) for z in SR('gamma') * X_0

# OK, so that seems to be right, so we ought to be able to plot S(p)
# and the actual motion of p on the constraint curve defined by p(u),
# on the b-c plane (letting m be fixed).

#mc_points = c00_evolution.plot( c00_bindings(p_func[1]), c00_bindings(p_func[2]),
#  xlabel='m_0', ylabel='c_{00}' )
initcond_bindings = Bindings( {u_0: initial_u} )
p_0 = vector( (initcond_bindings(c00_bindings(p_i)) for p_i in p_func) )
flow_p_0 = vector( (initcond_bindings(f_i) for f_i in flow_p_bound) )

ltx.write( 'So now we can get a look at the geometry of $%s$.' % latex( p_symb ),
  '  At $u_0=', latex(initial_u), '$,\n\\[ ',
  latex( p_symb ), ' = ', latex( column_vector( p_0 ) ), ', \\]\n\\[ ',
  latex( S_symb( p_symb ) ), ' = ', latex( column_vector( initcond_bindings( c00_bindings( cha

if 1:
    flow_p_arrow = arrow( (p_0[1], p_0[2]), (p_0[1]+flow_p_0[1], p_0[2]+flow_p_0[2]),
      color=(1,0,0), width=1 )
    p_func_bound = vector( (c00_bindings(f) for f in p_func) )
    ltx.write( 'The constraint curve for $%s$:' % latex( pf_symb( u_0 ) ),
      ' is \n\\[ ', latex( column_vector( p_func ) ), ' = ',
       latex( column_vector( p_func_bound ) ), ' \\]\n' )
    mc_constraint = parametric_plot( (p_func_bound[1], p_func_bound[2]),
     (u_0, initial_u - 0.25, initial_u + 1.5) )
    dp_du_0 = vector( (initcond_bindings(d_i) for d_i in dp_du_bound) )
    du_dt_0 = initcond_bindings(c00_bindings( maclev_adap_c00._flow[u_0] ) )
    dp_dt_0 = vector( (initcond_bindings(c00_bindings(
      d_i*du_dt_0 )) for d_i in dp_du_0) )
    ltx.write( 'And on that curve, \n\\[ ',
      '\\left.\\frac{d%s}{dt}\\right|_{u_0=%s} = ' % ( latex( pf_symb( u_0 ) ), latex( initial
      latex( column_vector( dp_dt_0 ) ), ' \\]\n' )
    dp_dt_arrow = arrow( (p_0[1], p_0[2]), (p_0[1]+dp_dt_0[1], p_0[2]+dp_dt_0[2]),
      width=1 )
    # The arrow heads may not look like they're at the same y position but they are
    mc_geom = flow_p_arrow + mc_constraint + dp_dt_arrow # + mc_points
    mc_geom.set_axes_range( 0, 2, initial_u - 0.25, initial_u + 1.75 )
    mc_geom.axes_labels( ['$m_0$', '$c_{00}$'] )
    save(mc_geom, filename="maclev-1-1-mc.png", figsize=(4,4))

ltx.close()

save_session("maclev-1-1-mc-adap-geom")
```

With $c_{00} = u_0$, the vector $\mathbf{p}$ is

$$\mathbf{p} = \left(\begin{array}{c} b_0 \\ m_0 \\ c_{00} \end{array}\right) = \left(\begin{array}{c} 1 \\ 1 \\ u_0 \end{array}\right)$$

And the selection gradient of $\mathbf{p}$ is

$$\mathbf{S}(\mathbf{p}) = \left(\begin{array}{c} -\frac{\hat{X}_0 c_{00}^2 w_0 - K_0 c_{00} r_0 w_0 + m_0 r_0}{r_0} \\ -b_0 \\ -\frac{\hat{X}_0 b_0 c_{00} w_0 - K_0 b_0 r_0 w_0}{r_0} \end{array}\right) = \left(\begin{array}{c} 0 \\ -1 \\ -\frac{2\,u_0 - 1}{u_0} + 2 \end{array}\right)$$

The derivative of $\mathbf{p}$ is

$$\frac{d\mathbf{p}}{du_0} = \left(\begin{array}{c} 0 \\ 0 \\ 1 \end{array}\right),$$

and we can recover the motion of u_0:

$$\mathbf{S}(u_0) = \frac{d\mathbf{p}}{du_0}^{\mathrm{T}} \mathbf{S}(\mathbf{p}(u_0)) = -\frac{2\,u_0 - 1}{u_0} + 2$$

$$\frac{du_0}{dt} = \gamma \hat{X}_0 \mathbf{S}(u_0) = -X_0 \left(\frac{2\,u_0 - 1}{u_0} - 2\right)$$

and the motion of $\mathbf{p}(u_0)$:

$$\frac{d\mathbf{p}(u_0)}{dt} = \gamma \hat{X}_0 \frac{d\mathbf{p}}{du_0} \frac{d\mathbf{p}}{du_0}^{\mathrm{T}} \mathbf{S}(\mathbf{p}(u_0)) = \left(\begin{array}{c} 0 \\ 0 \\ -X_0 \left(\frac{2\,u_0 - 1}{u_0} - 2\right) \end{array}\right)$$

So now we can get a look at the geometry of $\mathbf{p}$. At $u_0 = \frac{2}{3}$,

$$\mathbf{p} = \begin{pmatrix} 1 \\ 1 \\ \frac{2}{3} \end{pmatrix},$$

$$\mathbf{S}(\mathbf{p}) = \begin{pmatrix} 0 \\ -1 \\ \frac{3}{2} \end{pmatrix}$$

The constraint curve for $\mathbf{p}(u_0)$: is

$$\begin{pmatrix} b\,(u_0) \\ m\,(u_0) \\ c_0\,(u_0) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ u_0 \end{pmatrix}$$

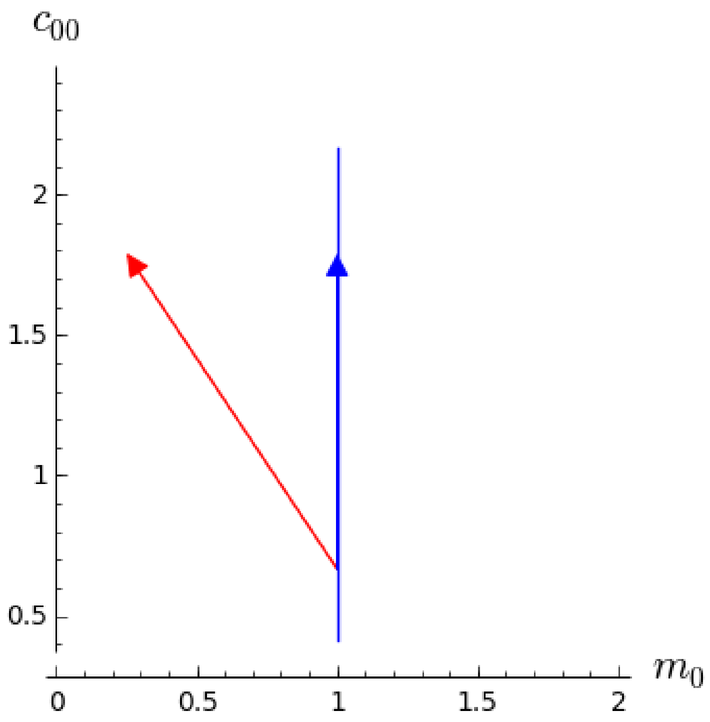
And on that curve,

$$\left.\frac{d\mathbf{p}(u_0)}{dt}\right|_{u_0=\frac{2}{3}} = \begin{pmatrix} 0 \\ 0 \\ \frac{9}{8} \end{pmatrix}$$

And here in the graphics is the part I really care about: the geometry of $\mathbf{p}$ and $\mathbf{A}$. Starting with $\mathbf{p}$.

Here we see the evolution of c_{00}, which gets smaller, and m_0, which is constant. Sticking to two dimensions, we don't plot b_0, which is also constant. We already knew what these three things do when we only allow c_{00} to change, which is what we're doing here. But here we can see it in a different way. The red arrow is the relevant two dimensions of $S(\mathbf{p})$, and the blue arrow is the same view of $\frac{d\mathbf{p}}{dt} = \frac{\partial \mathbf{p}}{\partial u}\frac{du}{dt}$.

The red arrow shows how $\mathbf{p}$ "would like to evolve" if all its components were allowed to change, and the blue arrow how it actually does evolve when constrained to the vertical line $m_0 = 1$. The red arrow is equal to the blue arrow plus another vector (not shown) perpendicular to the blue one. That is, the blue arrow is the ''projection" of the red one onto the vertical line that is the constraint curve defined by the mapping $\mathbf{p} = \mathbf{p}(u)$.



Adaptive dynamics of a and k

```
# requires: maclevmodels.py maclev-1-1-mc-adap-geom.sobj
# produces: maclev-1-1-ak-adap-geom.png maclev-1-1-ak-adap-geom.sage.out.tex
from sage.all import *
from sage.misc.latex import _latex_file_
from sage.misc.latex import latex

from maclevmodels import *
from lotkavolterra import *
from dynamicalsystems import latex_output

# create variables with custom latex names because load_session
# creates them wrong
# http://trac.sagemath.org/ticket/17559
for x in ('X_0', 'X_i', 'R_0'): hat(SR.symbol(x))
SR.symbol( 'c_0_0', latex_name='c_{00}' )
SR.symbol( 'c_i_0', latex_name='c_{i0}' )

load_session("maclev-1-1-mc-adap-geom")

ltx = latex_output( 'maclev-1-1-ak-adap-geom.sage.out.tex' )

# and now: do the same for the A vector.  For this, plot S(A), the projection
# of S(A) onto the curve, the indirect effect of evolution, and the resultant
# motion, which is not generally restricted to the curve because of the
# indirect component.

maclev_adap_lv = LotkaVolterraAdaptiveDynamics( maclev_adap_bound, r_name='k' )

ltx.write( 'As before, the adaptive dynamics of $u$ is' )
ltx.write_equality( wrap_latex('\\mathbf S' + latex( maclev_adap_lv.A(0) )),
    maclev_adap_lv.S( maclev_adap_lv.A(0) ),
    maclev_adap_lv._lv_model.interior_equilibrium_bindings()( maclev_adap_lv.S( maclev_adap_lv
ltx.write( maclev_adap_lv._adaptivedynamics )

save_session( 'maclev-1-1-ak-adap-geom' )
ltx.close()
exit()

# Here is the Lotka-Volterra population model we will use, with one population,
# with simple parameters a_ij, k_i
maclev_lv = GeneralizedLotkaVolterraModel( [0], r = indexer('k') )

ltx.write( "Abstract Lotka-Volterra model corresponding to the 1-population Mac-Lev model:" )
ltx.write( maclev_adap_lv )

# Here is the adaptive dynamics of the Lotka-Volterra model
# We get the general S(A) using this unparametrized lv system; we'll add
# the dependence on u later
maclev_adap_lv = AdaptiveDynamicsModel( maclev_lv,
  [ indexer('k'), indexer(lambda i: 'a_%s_0'%i) ] )

A = cvector( ( SR.var('a_0_0'), SR.var('k_0') ) )
S_A = A.apply_map( lambda a: maclev_adap_lv._S[a] )

#print 'adaptive dynamics of %s:\n' % A, maclev_adap_lv
#print 'S_A:', S_A

#ltx.write( 'Generating the adaptive dynamics of $a$ and $k$:\n\\begin{quote}\n',
#    maclev_adap_lv._debug_output._output._str, '\n\\end{quote}\n' )

# now we derive the values of the L-V coefficients k_i and a_ij
# from the mac-lev dynamics.

# this only works for a single variable - it would have to be more
# sophisticated to find the coefficients of multiple X's.  but it works
# fine with just X_0 and X_0^2.

def get_coeff( expr, vars, power ):
    for c, p in expr.coefficients( vars ):
        if p == power:
            return c

# Here is the population dynamics, with parameters as a function of u.
maclev_p_ij = MacArthurLevinsModel( x_indices = ['i','j'], r_indices = [0],
  b = indexer(lambda i:'b(u_%s)'%i),
  m = indexer(lambda i:'m(u_%s)'%i),
  c = indexer(lambda i: indexer(lambda l: function('c_%s'%l)(SR('u_%s'%i)) ) ) )

lvi = maclev_p_ij._flow[SR('X_i')].expand()
#ltx.write( 'In the $i,j$ LV system\n\\[ \\frac{dX_i}{dt} = ', latex(lvi), ' \\]\n\n' );
```

```
from sage.symbolic.function_factory import function
af = function('a')
kf = function ('k')
A_bindings = FunctionBindings(
  { af: get_coeff( get_coeff( lvi, SR('X_i'), 1), SR('X_j'), 1).function(SR('u_i'), SR('u_j')),
    kf: get_coeff( get_coeff( lvi, SR('X_i'), 1), SR('X_j'), 0).function(SR('u_i')) } )

# now do the adaptive dynamics with a, k bound to
# specific functions of u

# first make a L.V. model with a, k as functions of u
maclev_lv_u = GeneralizedLotkaVolterraModel( [0],
  r = indexer(lambda i: kf(SR('u_%s'%i))),
  a = indexer(lambda i: indexer(lambda j: af(SR('u_%s'%i),SR('u_%s'%j)))) )

#ltx.write( 'now with $a$ and $k$ as functions of $u$:' )
#ltx.write( maclev_lv_u )

#ltx.write( 'with coefficients:' )
#ltx.write( A_bindings )

# then bind a, k to specific functions of u, to be precise, with
# c_0i = u_i, b_i = m_1 = 1
maclev_lv = maclev_lv_u.bind( A_bindings )
#ltx.write( 'becomes:' )
#ltx.write( maclev_lv )

#ltx.write( '(a 2-variable system would be ' )
#ltx.write( GeneralizedLotkaVolterraModel( [0, 1],
#  k = indexer(lambda i:'k(u_%s)'%i),
#  a = indexer(lambda i: indexer(lambda j: 'a(u_%s,u_%s)'%(i,j))),
#  bindings = A_bindings ) )
#ltx.write( ')... ' )

maclev_lv_c00 = maclev_lv.bind( FunctionBindings( {
    function('c_0'): u_0.function(u_0),
    function('b'): symbolic_expression('1').function(u_0),
    function('m'): symbolic_expression('1').function(u_0) } ) )

ltx.write( 'Now we bind $c, m, b$:' )
ltx.write( maclev_lv_c00 )
ltx.write( 'maclev_lv_c00._bindings:' )
ltx.write( maclev_lv_c00._bindings )

# assign constant values to the rest of the parameters
maclev_lv_c00_bound = maclev_lv_c00.bind( Bindings(
  { SR.var('r_0'): 1, SR.var('w_0'): 1,
    SR.var('K_0'): 2, SR.var('gamma'): 1 } ) )

ltx.write( 'With all the bindings works out to:' )
ltx.write( maclev_lv_c00_bound )

# and derive the adaptive dynamics of u_0 under those assumptions,
# i.e. the adaptive dynamics of c_00 with all else held fixed.
maclev_adap_lv_c00_bound = AdaptiveDynamicsModel( maclev_lv_c00_bound,
  [ indexer('u') ], bindings = Bindings( gamma=1 ) )
```

```
#ltx.write( 'Generating the adaptive dynamics of $u$:\n\\begin{quote}\n',
#    maclev_adap_lv_c00_bound._debug_output._output._str, '\n\\end{quote}\n' )

ltx.write( 'As before, the adaptive dynamics of $u$ is' )
ltx.write_equality( wrap_latex('\\mathbf S' + latex( cvector( [ u_j for u_j in maclev_adap_lv_
            cvector( [ dI_duj for dI_duj in maclev_adap_lv_c00_bound._S.values() ] ) )
ltx.write( maclev_adap_lv_c00_bound )

ltx.write( 'and' )
ltx.write_equality( X_0, maclev_adap_lv_c00_bound._bindings( X_0 ) )

# Initial condition for the adaptive dynamics of u_0
initcond_bindings = Bindings({u_0: initial_u})

# integrate the adaptive dynamics and get a numeric trajectory
lv_c00_evolution = maclev_adap_lv_c00_bound.solve( [initcond_bindings(u_0)] )

c00_bindings = maclev_adap_lv_c00_bound._bindings
#print 'c00_bindings:', c00_bindings

A_u = cvector( [ symbolic_expression('a(u_0,u_0)'), symbolic_expression('k(u_0)') ] )
A_u_binding = Bindings( a_0_0 = 'a(u_0,u_0)', k_0 = 'k(u_0)' )
S_A_u = S_A.apply_map( lambda s : A_u_binding(s) )
A_bound = A_u.apply_map( lambda a : c00_bindings(a) )
S_A_bound = S_A_u.apply_map( lambda s : c00_bindings(s) )
ltx.write( 'And now we can compare the different vector derivatives...' )
ltx.write_equality( SR('A'), A_u, A_bound )
ltx.write_equality( SR('S(A)'), S_A_u, S_A_bound )

A_0 = A_u.apply_map( lambda a : initcond_bindings(c00_bindings(a)) )
S_A_0 = S_A_u.apply_map( lambda s : initcond_bindings(c00_bindings(s)) )

A_c00 = c00_bindings(A_u)
#ltx.write_equality( SR('A(u_0,u_0)'), A_c00 )

# dA/du is different from the projection of S(A) into the first variable.
dA_du_bound = diff(A_c00,u_0)
dA_du_0 = initcond_bindings(dA_du_bound)

# now the direct and indirect parts.
# until I improve the AdaptiveDynamicsModel class, I have to redo some of
# what it does here.
maclev_u_ij = MacArthurLevinsModel(
  x_indices=['i','j'], r_indices=[0],
  b = indexer(lambda i:'b(u_%s)'%i),
  m = indexer(lambda i:'m(u_%s)'%i),
  c = indexer(lambda i: indexer(lambda l: function('c_%s'%l)(SR('u_%s'%i)))) )
X_i = SR.var('X_i')
X_j = SR.var('X_j')
f_j = maclev_u_ij._flow[X_j].collect(X_i)
#print 'f_i:', f_i
# there are better ways to do this, involving multivariate coefficients
f_over = limit((f_j/X_j).collect(X_i), X_j=0)
#print 'f_i/X_i:', f_over
A_j = cvector( [ f_over.coeff(X_i,1), f_over.coeff(X_i,0) ] )
A_j = c00_bindings( A_j )
```

```
#v = SR.var('v')
#A_i = A_i.apply_map( lambda a : a.substitute( { SR.var('u_i') : v } ) )
u_0 = SR.var('u_0')
u_i = SR.var('u_i')
u_j = SR.var('u_j')

ltx.write_equality( SR('A_j(u_i,u_j)'), A_j )

#ltx.write_equality( SR('A_0(u_0,u_0)'), A_j.substitute( { u_i: u_0, u_j: u_0 } ) )

# So that's A.  Now we can plot its derivatives.

# first d1A = first partial of A
d1A = diff(A_j,u_j).substitute( { u_i: u_0, u_j: u_0 } )
ltx.write( 'So' )
ltx.write_equality( wrap_latex('\\partial_1\mathbf A(u_i,u_j)'), d1A )

# now d2A = second partial
d2A = diff(A_j,u_i).substitute( { u_i: u_0, u_j: u_0 } )
ltx.write_equality( wrap_latex('\\partial_2\mathbf A(u_i,u_j)'), d2A )

#ltx.write_equality( wrap_latex('\\partial_1\mathbf A \\partial_1\mathbf A^T'), (dAdv * dAdv.t

d1ATS = ( d1A.transpose() * S_A_bound ).apply_map( lambda i : i.rational_simplify() )
ltx.write_equality( wrap_latex('\\partial_1\mathbf A^T S(A)'), d1ATS )

ltx.write( 'so' )

ltx.write_equality( wrap_latex('\\partial_1\mathbf A\\partial_1\mathbf A^T S(\mathbf A)'), d1A

d1A_curve = A_j.substitute( { u_j : initial_u } )
#ltx.write( 'and the mysterious curve (for $v\in[%s,%s]$) is' %(initial_u-2,initial_u+2) )
#ltx.write( d1A_curve )

d1A_constraint = parametric_plot( (d1A_curve[0], d1A_curve[1]),
  (u_i, initial_u-2, initial_u+2), color='green' )
#dAdv   = dAdv.substitute({u_0: initial_u, v: initial_u})
#dAdu_0 = dAdu_0.substitute({u_0: initial_u, v: initial_u})

#ltx.write( 'dA(v,u_0)/dv:\n' )
#ltx.write( dAdv )
#ltx.write( 'dA(v,u_0)/du_0:\n' )
#ltx.write( dAdu_0 )

dAdt_unconstrained = A.apply_map( lambda a :  c00_bindings(maclev_adap_lv._flow[a].substitute(

gX = c00_bindings(SR('gamma * X_0'))
D = gX * d1A * d1ATS
D = D.apply_map( lambda d : d.rational_simplify() )

ltx.write( 'thus' )
ltx.write_equality( wrap_latex('\\gamma \hat X_0'), gX )
ltx.write_equality( SR('D'), wrap_latex('\\gamma X_0 \\partial_1 \mathbf A \\partial_1 \mathbf

#I_circle = gX * dAdv * dAdu_0.transpose() * S_A_bound

#ltx.write( 'I_circle(%g) ='%initial_u )
```

```
#ltx.write_equality( wrap_latex('I^\circ'), wrap_latex('\\gamma X_0 \\frac{\\partial A}{\\part

I = gX * d2A * d1ATS
I = I.apply_map( lambda i : i.rational_simplify() )

ltx.write_equality( SR.var('I'), wrap_latex('\\gamma X_0 \\partial_2 \mathbf A \\partial_1 \ma

ltx.write_equality( wrap_latex('D + I'), (D+I).apply_map( lambda di : di.rational_simplify() )
ltx.write( "and to check, " )
ltx.write( 'the total derivative of $A$ is ' )
ltx.write_equality( wrap_latex('\\frac{dA(u_0,u_0)}{du_0}'), dA_du_bound )

dudt = maclev_adap_c00_bound._flow[u_0]

ltx.write( 'and' )
ltx.write_equality( wrap_latex('\\frac{du_0}{dt}'), dudt )
ltx.write( 'so that' )
ltx.write_equality( wrap_latex('\\frac{dA}{dt}'),
  wrap_latex('\\frac{dA(u_0,u_0)}{du_0}\\frac{du_0}{dt}'),
  dA_du_bound * dudt )

dudt_0 = initcond_bindings(maclev_adap_c00_bound._flow[u_0])
dA_du_0c = dA_du_0.transpose()
#ltx.write( 'dudt_0 =' )
#ltx.write( dudt_0 )
#ltx.write( 'dA_du_0 =' )
#ltx.write( dA_du_0c )

dAdt_constrained = dA_du_0c * dudt_0
#ltx.write( 'and dAdt_constrained =' )
#ltx.write( dAdt_constrained )

def arrow_from_cvectors(tail, length, **kargs):
    return arrow( (tail[0][0], tail[1][0]),
        (tail[0][0] + length[0][0], tail[1][0] + length[1][0]), **kargs)

if 1:
    ak_points = lv_c00_evolution.plot( c00_bindings(A_u[0][0]), c00_bindings(A_u[1][0]),
      xlabel='a_{00}', ylabel='k_0', color='black' )
    S_A_arrow = arrow_from_cvectors( A_0, dAdt_unconstrained,
      color='red', width=1 )
    ak_constraint = plot( (A_c00[0][0], A_c00[1][0]),
      (u_0,initial_u-0.5,initial_u+0.5), color='light blue' )
    D_arrow = arrow_from_cvectors( initcond_bindings( c00_bindings( A_0 ) ), initcond_bindings
    I_arrow = arrow_from_cvectors( initcond_bindings( c00_bindings( A_0 + D ) ), initcond_bind
    dA_du_arrow = arrow_from_cvectors( A_0, initcond_bindings( c00_bindings( dA_du_bound * dud
    ak_geom = ak_points + dA_du_arrow + S_A_arrow + d1A_constraint + D_arrow + I_arrow
    # + ak_constraint
    ak_geom.set_axes_range( -8, 3, -2, 7 )
    ak_geom.axes_labels( ['$a_{00}$', '$k_0$'] )
    save(ak_geom, filename="maclev-1-1-ak-adap-geom.png", figsize=(4,4))

ltx.close()
```

As before, the adaptive dynamics of u is

$$\mathbf{S}\left(k_0,\, a_{00}\right) = \left(1,\, \hat{X}_0\right) = \left(1,\, -\frac{k_0}{a_{00}}\right)$$

$$\frac{du_0}{dt} = -\frac{\left(2\,u_0-1\right)\left(\frac{2\,u_0-1}{u_0}-2\right)}{u_0^2}$$

This is a work in progress.

This figure includes $S(A)$ (red) and $\frac{dA}{dt}$ (blue), but also the projection of $S(A)$ onto the constraint curve $A(u, u_0)$ – the "direct effect", that is (green) – and the "indirect effect", which is the terms (purple) that combine with the direct effect to give $\frac{dA}{dt}$. The direct effect expresses how the population is "tempted" to change, and the indirect effect expresses the "unintended consequences" of that change.

```
[WorkingWiki encountered errors:  Error:  File ‘maclev-1-1-ak-adap-geom.png’
not found in working directory.  ]
```

Biological meaning of these quantities

The interaction terms, A

We made this model by using an equilibrium resource quantity

$$\hat{R}_0 = K_0 - \frac{c_{00}X_0}{r_0}$$

given population size X_0.

Here K_0 is the resource "set point" - the level it'll return to if consumption stops - and $\frac{c_{00}X_0}{r_0}$ is the "draw-down" of the resource - the amount removed to bring R to the level where replacement and consumption balance.

The population dynamics is

$\frac{dX_0}{dt} = (b_0c_{00}w_0R_0 - b_0m_0)X_0,$

with a growth term driven by resource availability and a constant per-capita mortality term.

So after we substitute $R_0 \to \hat{R}_0$ and partition out the linear and quadratic terms to make a_{00} and k_0, we have

$$\begin{aligned}\frac{dX_0}{dt} &= (b_0c_{00}w_0R_0 - b_0m_0)X_0\\ &= (b_0c_{00}w_0(K_0 - \frac{c_{00}X_0}{r_0}) - b_0m_0)X_0\\ &= b_0c_{00}w_0K_0X_0 - b_0m_0X_0 - b_0c_{00}w_0\frac{c_{00}}{r_0}X_0^2.\end{aligned}$$

So in a nutshell,

$$k_0 = b_0c_{00}w_0K_0 - b_0m_0$$

is what the net reproduction rate ‘'would be” if R were at its set point, and

$$a_{00} = -b_0c_{00}w_0\frac{c_{00}}{r_0}$$

encompasses the loss of reproduction due to draw-down of R.

The agent-patient distinction is that we consider the fitness of population 0 (as patient) in the environment created by population 0 (as agent). Adaptive dynamics is driven by the difference of fitness as the patient population varies while the agent population stays fixed. So the green c_{00} in the expression for $\hat{R}$ is the agent population's consumption rate - it shows up in a_{00} - and the purple b_0c_{00} and red b_0m_0 in the X equation - the other population parameters in a and k - are the patient population's characteristics.

To summarize: the interaction vector A is
$A = \begin{pmatrix} a_{00} \\ k_0 \end{pmatrix} = \begin{pmatrix} -b_0c_{00}w_0\frac{c_{00}}{r_0} \\ b_0c_{00}w_0K_0 - b_0m_0 \end{pmatrix}.$

The selection gradient $S(A)$

Any evolvable traits of this population must be expressed in its ecological parameters b_0, c_{00}, and m_0. Under the adaptive dynamics assumptions, selection occurs through the marginal reproductive fitness of a variant invader population i in the resident population 0's environment. That works out to mean the fitness conferred by the variant's b, c, and m, when i is the ‘'patient'‘in the environment of'‘agent” 0: the derivative of

$$\begin{aligned}\mathcal{I} &= k_i + a_{i0}\hat{X}_0\\ &= (b_ic_{i0}w_0K_0 - b_im_i) - b_ic_{i0}w_0\frac{c_{00}}{r_0}\hat{X}_0\end{aligned}$$

with respect to the characteristics of population i.

Because we want to keep track of k and a, we take that (partial) derivative in two steps: the derivative of $\mathcal{I}$ with respect to a_{i0} and k_i, and then the derivative of those with respect to b_i, c_{i0} and m_i.

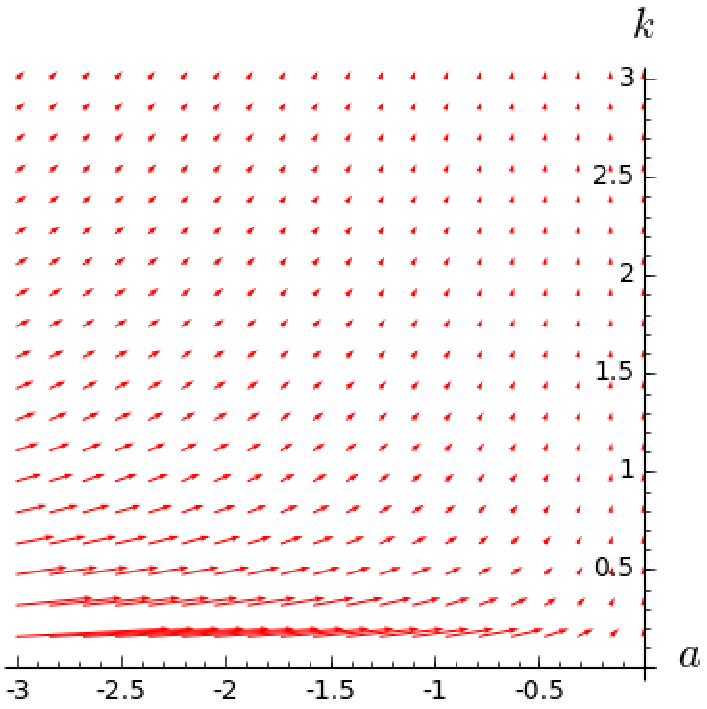
The derivatives of $\mathcal{I}$ with respect to a and k is the interaction selection gradient $S(A)$, and its value is simple and depends not at all on the specifics of the model:

$S(A) = \begin{pmatrix} \frac{\partial \mathcal{I}}{\partial a_{i0}} \\ \frac{\partial \mathcal{I}}{\partial k_i} \end{pmatrix} = \begin{pmatrix} \hat{X}_0 \\ 1 \end{pmatrix}.$

Actually we can write the equilibrium population size directly without regard to what a and k are:

$\hat{X}_0 = -\frac{k_0}{a_{00}}.$

- Let's digress for a moment: it follows from this that the vector field $S(A)$ in a one-population LV model is always the same, no matter what the model is. It looks like this:



But maybe it's helpful to expand out the value of $\hat{X}_0$?

$$\hat{X}_0 = \left(K_0 - \frac{m_0}{c_{00} w_0}\right) \frac{r_0}{c_{00}}.$$

Anyway, it's directly proportional to the population's growth rate in a sterile world, and inversely proportional to the drawdown: the more X draws down the resource at equilibrium, the smaller X is.

[And while we're at it:

$$\begin{aligned}\hat{R}_0 &= K_0 - \frac{c_{00}}{r_0}\hat{X}_0 \\ &= K_0 - \frac{c_{00}}{r_0}(K_0 - \frac{m_0}{c_{00}w_0})\frac{r_0}{c_{00}} \\ &= \frac{m_0}{c_{00}w_0}.]\end{aligned}$$

For reference, the expanded-out value of the gradient is

$$S(A) = \begin{pmatrix} (K_0 - \frac{m_0}{c_{00}w_0})\frac{r_0}{c_{00}} \\ 1 \end{pmatrix}.$$

The direct effect

The direct effect,

$D = \gamma\hat{X}_i\partial_1\mathbf{A}_i(\mathbf{u}_i)\partial_1\mathbf{A}_i(\mathbf{u}_i)^T S(\mathbf{A}_i)$,

is the selection gradient vector projected on to the constrained available direction of motion. Since $\partial_1\mathbf{A}_i(\mathbf{u}_i)^T S(\mathbf{A}_i)$ is a scalar (it's the dot product of $\partial_1\mathbf{A}_i(\mathbf{u}_i)$ and $S(\mathbf{A}_i)$), the vector D is a multiple of $\partial_1\mathbf{A}_i(\mathbf{u}_i)$, which is the direction that A can change by varying the invading population (the patient) and holding the resident (agent) fixed.

[Mathematical note: Since $\partial_1\mathbf{A}_i(\mathbf{u}_i)^T$ isn't necessarily unit-length, this isn't necessarily the exactly the projection of $S(\mathbf{A}_i)$ onto the line generated by $\partial_1\mathbf{A}_i(\mathbf{u}_i)$, but it's a multiple of it. This reflects the relationship between $\mathbf{u}_i$ and $\mathbf{A}_i$. The selection gradient is a description of how $\mathbf{A}_i$ would move if it were mutating directly, and the projection is a description of how it moves when it's $\mathbf{u}_i$ that mutates. If $\mathbf{A}_i$ changes a lot in response to a small change in $\mathbf{u}_i$, the projected vector D will be longer, and if it only changes a little per change in $\mathbf{u}_i$, D will be shorter.]

Let's examine D more closely in the Mac-Lev model.

To review, the interaction vector (or whatever we call it) is

$$A = \begin{pmatrix} a_{00} \\ k_0 \end{pmatrix} = \begin{pmatrix} -\frac{b_0 {c_{00}}^2 w_0}{r_0} \\ K_0 b_0 c_{00} w_0 - b_0 m_0 \end{pmatrix}$$

but to get the adaptive dynamics right we have to distinguish agents from patients: the terms of this are

$$A_i = \begin{pmatrix} -\frac{b_i c_{i0} c_{j0} w_0}{r_0} \\ K_0 b_i c_{i0} w_0 - b_i m_i \end{pmatrix}$$

and indexes i and j are the patient and agent, respectively, even if they're both 0 in this simple example, and we need to keep them straight in order to get the first and second partial derivatives.

Here we only need the first partial, the derivative with respect to the patient.

The derivatives with respect to b_i, c_{i0}, and m_i are

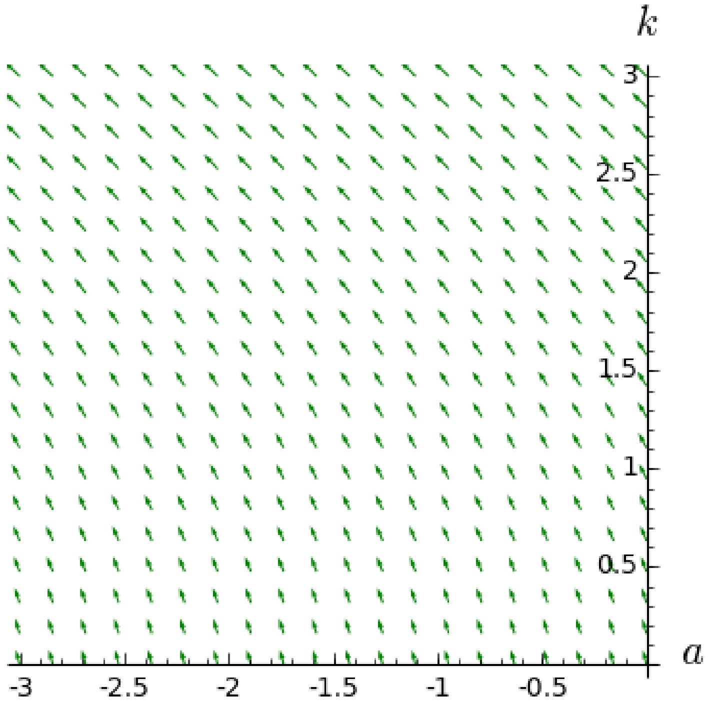
$$\partial_{1b} A = \begin{pmatrix} -\frac{c_{i0} c_{j0} w_0}{r_0} \\ K_0 c_{i0} w_0 - m_i \end{pmatrix} = \begin{pmatrix} -\frac{c_{00}^2 w_0}{r_0} \\ K_0 c_{00} w_0 - m_0 \end{pmatrix},$$

$$\partial_{1c} A = \begin{pmatrix} -\frac{b_i c_{j0} w_0}{r_0} \\ K_0 b_i w_0 \end{pmatrix} = \begin{pmatrix} -\frac{b_0 c_{00} w_0}{r_0} \\ K_0 b_0 w_0 \end{pmatrix},$$

$$\partial_{1m} A = \begin{pmatrix} 0 \\ -b_i \end{pmatrix} = \begin{pmatrix} 0 \\ -b_0 \end{pmatrix}.$$

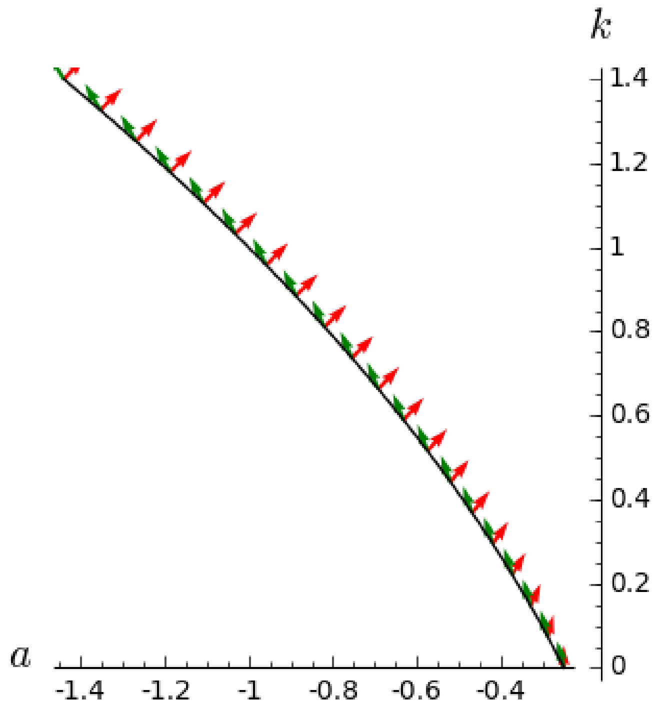
We've been focusing on varying only c, so let's keep working with that for now. From these equations we can see that the partial derivative vector $\partial_{1c} A$ always points up and to the left (as does $\partial_{1b} A$, while $\partial_{1m} A$ points straight down).

For the fixed values of b, m, etc. that we've been using, the vector field $\partial_{1c} A$ looks like this:



But actually those vectors are only meaningful at (a, k) combinations that can actually be achieved for some value of c. Here's $S(A)$ and $\partial_{1c}A$ on that subset of (a, k):

[making of this disabled due to code needing to be redone]



The main part of D is

$\partial_1 A \partial_1 A^T S(A)$,

which is a scalar product times that green vector $\partial_1 A$. The scalar product, for the varying-c model, is

$\partial_1 A^T S(A) = -\frac{b_0 c_{00} w_0}{r_0} \cdot (K_0 - \frac{m_0}{c_{00} w_0}) \frac{r_0}{c_{00}} + K_0 b_0 w_0 \cdot 1$

$= \frac{b_0 m_0}{c_{00}}$.

Without going too far into interpreting this quantity, we can note that it's always positive. Thus the projection will always be in the direction of $\partial_1 A$, not its negative.

So the "main part" is

$\partial_1 A \partial_1 A^T S(A) = \frac{b_0 m_0}{c_{00}} \begin{pmatrix} -\frac{b_0 c_{00} w_0}{r_0} \\ K_0 b_0 w_0 \end{pmatrix} = b_0^2 m_0 w_0 \begin{pmatrix} -\frac{1}{r_0} \\ \frac{K_0}{c_{00}} \end{pmatrix}$,

and the full value of D is

$D = \gamma \hat{X}_0 \partial_1 A \partial_1 A^T S(A) = \gamma (K_0 - \frac{m_0}{c_{00} w_0}) \frac{r_0}{c_{00}} b_0^2 m_0 w_0 \begin{pmatrix} -\frac{1}{r_0} \\ \frac{K_0}{c_{00}} \end{pmatrix}$

$= \gamma (K_0 w_0 - \frac{m_0}{c_{00}}) \frac{b_0^2 m_0}{c_{00}} \begin{pmatrix} -1 \\ \frac{r_0 K_0}{c_{00}} \end{pmatrix}$

So we know that the ‘‘direct effect” D is a vector up and to the left.

Big questions of interest to me are

- What makes D - a projection of $S(A)$ - point in the direction of increasing competition: to the left, while $S(A)$ points to the right?
- What makes I contribute more competition to what D is already piling on?

Here are $S(A)$ and D on the c curve:

```
[WorkingWiki encountered errors: Error: <ww-make-failed: maclev-1-1-S-and-D-on-curve.png
/Selection_Gradients/maclev-1-1-S-and-D-on-curve.png.make.log> ]
```

I'd like to have more perspective on this, but I don't want to get too bogged down. Here's what I see here at present:

- Separating a from k breaks the population dynamics into 3 parts:
  - in k: growth that would obtain at the maximum resource availability, and death (density-independent); and
  - in a: reduction in growth due to drawdown of the resource by the population(s).
- Both growth terms vary directly with the consumption parameter c, so that:
  - Selection increases c because the net growth is positive
  - Increasing c increases the magnitude of a.
- This is why c increases, and why k becomes more positive and a becomes more negative as c increases.