Ecological cycling

This update of lifecycles can do batch runs and feedbacks nutrients to the pool

Here batchruns tells us how many realisations of the cycling to run

```
In[•]:= batchruns = 100;
```

Initialisation

In this section we provide definitions required for proper colours, number of strains, environments to be used

```
In[•]:= speciesproperties = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}};
        base = {🟦, 🟥, 🟦, 🟧};
        possibleenvs = Tuples[{0, 1}, 4];
        carryingcapacity = 10;
        inoculumsize = 5;
        negativegrowthrate = -2;(*If we put this to 0 then we get persistence*)
        tmax = 20;
        tpool =.;
```

```
In[•]:= base
Out[•]=
        {🟦, 🟥, 🟦, 🟧}
```

```
In[•]:= List @@@ base
Out[•]=
        {{0.172549, 0.482353, 0.713725}, {0.843137, 0.29, 0.29},
         {0.670588, 0.85098, 1}, {0.992157, 0.682353, 0.380392}}
```

```
In[•]:= base
Out[•]=
        {🟦, 🟥, 🟦, 🟧}
```

The equations for growth in wells

In[•]:=

$$\texttt{xdot[i\_, grrates\_]} := \left(\texttt{grrates}[\![i]\!]\left(1-\frac{\sum_{k=1}^{4} x_k[t]}{\texttt{carryingcapacity}}\right)\right) x_i[t];$$

Growth rates of the strains

```
In[•]:= expgr = {1.028, 0.282, 1.564, 1.103};
```

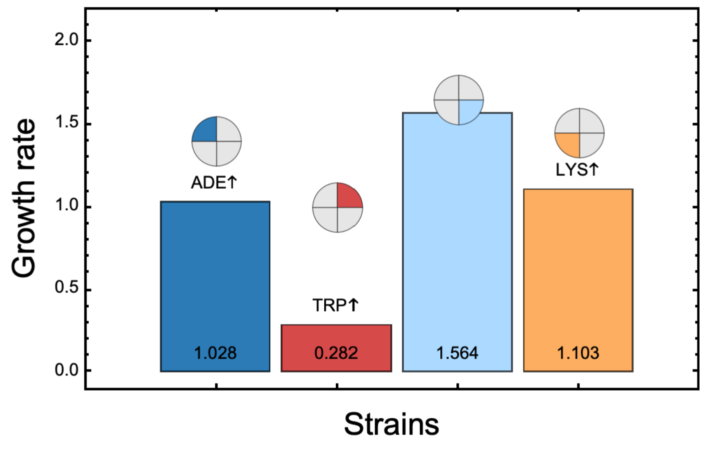
When we want to change the basic growth rates by a certain percent use the following code

In[ ]:=
```
(*deltas=Table[(Mean[expgr]-expgr⟦i⟧) 0.5,{i,Length[expgr]}];
 expgr=expgr+deltas;*)
```

Now we make the figure for the growth rates.

In[ ]:=
```
avggrowthrates = Flatten[Join[{0, expgr}]];
legends = {"NP", "ADE↑", "TRP↑", "HIS↑", "LYS↑"};
charts =
  PieChart[{1, 1, 1, 1}, ChartStyle → {If[#⟦1⟧ == 1, base⟦1⟧, GrayLevel[0.9]],
       If[#⟦2⟧ == 1, base⟦2⟧, GrayLevel[0.9]], If[#⟦3⟧ == 1, base⟦3⟧, GrayLevel[
         0.9]], If[#⟦4⟧ == 1, base⟦4⟧, GrayLevel[0.9]]}] & /@ speciesproperties;
BarChart[avggrowthrates⟦2 ;;⟧, ChartLabels → Placed[legends⟦2 ;;⟧, Above],
 ChartStyle → base, Frame → True, FrameStyle → Directive[Black, Thickness[0.003]],
 FrameLabel → {Style["Strains", Black, 16], Style["Growth rate", Black, 16]},
 PlotRange → {Automatic, {-0.1, 2.2}},
 Epilog → {(*Inset[charts⟦1⟧,{1,1.4},{0,0},0.5],*)
   Inset[charts⟦1⟧, {1, 1.4}, {0, 0}, 0.5], Inset[charts⟦2⟧, {2, 1.0}, {0, 0}, 0.5],
   Inset[charts⟦3⟧, {3, 1.65}, {0, 0}, 0.5], Inset[expgr⟦1⟧, {1, 0.1}],
   Inset[expgr⟦2⟧, {2, 0.1}], Inset[expgr⟦3⟧, {3, 0.1}],
   Inset[expgr⟦4⟧, {4, 0.1}], Inset[charts⟦4⟧, {4, 1.45}, {0, 0}, 0.5]}]
```

Out[ ]=



Growth rate equations for the pool

In[ ]:=
```
poolgrrate = expgr;
ydot[i_, poolgrrate_] := (poolgrrate⟦i⟧) yᵢ[t]
```

Then we write the functions that numerically integrate the wells and the pool phases

```
In[ ]:= rundynamics[initcond_, grrates_] := {eqs =.; sol =.; t =.;
           NDSolve[Join[Table[xᵢ'[t] == xdot[i, grrates], {i, 1, 4}],
             Table[xᵢ[0] == initcond[[i]], {i, 1, 4}]], {x1, x2, x3, x4}, {t, 0, tmax}]
         };
        pooldynamics[initcond_, tpool_] := {eqs =.; poolsol =.; t =.;
           NDSolve[Join[Table[yᵢ'[t] == ydot[i, poolgrrate], {i, 1, 4}], Table[
                yᵢ[0] == initcond[[i]], {i, 1, 4}]], {y1, y2, y3, y4}, {t, 0, tpool}] // Quiet
         };
```

Forming groups

```
In[ ]:= groupsize = 4;
        grouped = Tuples[speciesproperties, groupsize];
```

Importing fitness matrix for env and community

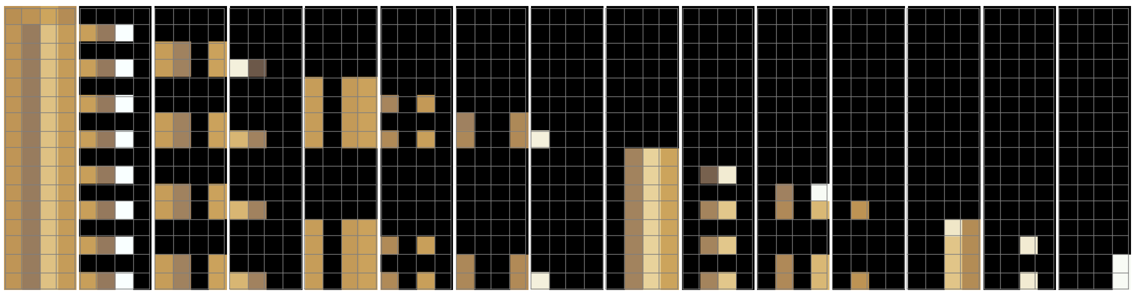
Here we import the fitnesses as extrapolated from the experiments. For each strain the growth rate can be different if the metabolite it requires but cannot produces comes from another strain or the environment.

```
In[ ]:= rawfit = Import[NotebookDirectory[] <> "fitnesstoimport_avg.xlsx"];
        rawfit = rawfit[[1]];
        environments = ToExpression[rawfit[[1, 2 ;;]]];
        communities = ToExpression[rawfit[[All, 1]][[2 ;;]]];
        grs = ToExpression[rawfit[[2 ;;, 2 ;;]]];
        fulldict = Association[];
        fulldict = {};
        For[j = 1, j ≤ Length[environments], j++,
         For[i = 1, i ≤ Length[communities], i++,
          AppendTo[fulldict, {environments[[j]], communities[[i]]} → grs[[i, j]]]
         ]
        ]
```

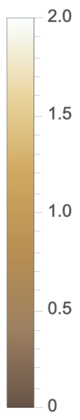
In the following we make the figure showing the fitnesses along with the colour code

In[∘]:=
```
GraphicsRow[Table[MatrixPlot[grs⟦i⟧ /. {Null → {0, 0, 0, 0}}, ColorFunction →
     (If[# == 0, Black, ColorData["CoffeeTones"][Rescale[#, {0, 2}]]] &),
   ColorFunctionScaling → False, Mesh → True, Frame → False,
   MeshStyle → Directive[{Gray, Thickness[0.01]}]],
  {i, Length[grs]}], Spacings → 0]
BarLegend[{"CoffeeTones", {0, 2}}]
```

Out[∘]=



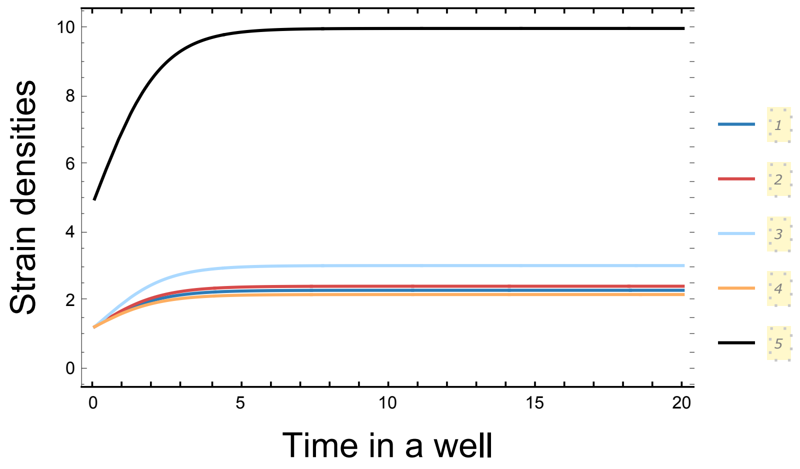
Out[∘]=



Using the above fitnesses and the dynamics equations we show how the growth looks like in a single well.

In[ ]:=
```
Plot[Evaluate[{x₁[t], x₂[t], x₃[t], x₄[t], x₁[t] + x₂[t] + x₃[t] + x₄[t]} /.
    rundynamics[{0.25, 0.25, 0.25, 0.25} 5, Values[fulldict]⟦1⟧]],
  {t, 0, 20}, PlotStyle → Flatten[Join[{base, Black}]], Frame → True,
  FrameStyle → {Black, Thickness[0.003]}, PlotLegends → Automatic, FrameLabel →
   {Style["Time in a well", Black, 18], Style["Strain densities", Black, 18]}]
```

Out[ ]=



To make the process efficient we create a dictionary for the growth rates

In[ ]:=
```
allpossiblecombinations = Tuples[Tuples[{0, 1}, 4], 2];
notvalid = 0;
valid = {};
For[i = 1, i ≤ Length[allpossiblecombinations], i++,
  If[MemberQ[Total[allpossiblecombinations⟦i⟧], 0] == True ||
    allpossiblecombinations⟦i, 2⟧ == {0, 0, 0, 0},
   notvalid++, AppendTo[valid, i]]];
allvalidcombinations = allpossiblecombinations⟦#⟧ & /@ valid;
```

If choosing random fitnesses use the follow block

In[ ]:=
```
(*randomfitnesses=Table[allvalidcombinations⟦i,2⟧ RandomReal[2,4],
  {i,1,Length[allvalidcombinations]}];
envcommunityfitnessmatrix=Table[allvalidcombinations⟦i⟧→randomfitnesses⟦i⟧,
  {i,1,Length[allvalidcombinations]}];*)
```

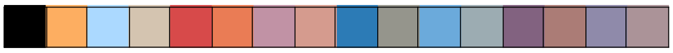
else use this

In[ ]:=
```
envcommunityfitnessmatrix = Table[
   allvalidcombinations⟦i⟧ → Lookup[fulldict, {allvalidcombinations⟦i⟧}]⟦1⟧,
   {i, 1, Length[allvalidcombinations]}];
```

Distribution of the environments

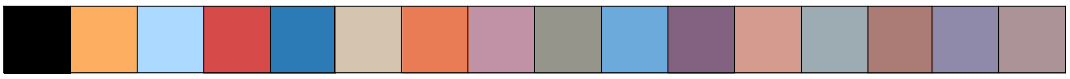
How are the environments chosen? at random or do they follow a certain distribution?

```
In[∘]:= envcols = Blend[{🟦, 🟥, 🟦, 🟧}, #] & /@ possibleenvs
        ArrayPlot[{envcols}, Mesh → True, MeshStyle → Black]
Out[∘]=
        {⬛, 🟧, 🟦, 🟫, 🟥, 🟧, 🟪, 🟫, 🟦, ⬜, 🟦, ⬜, 🟪, 🟫, 🟪, 🟫}
Out[∘]=
```



However these are not sorted by the level of richness... below we will sort the environments according to their richness level and then reorder the colours.

```
In[∘]:= ones = Count[#, 1] & /@ possibleenvs;
        possibleenvssorted =
          SortBy[Partition[Riffle[possibleenvs, ones], 2], Last][[All, 1]];
        envcolssorted = Blend[{🟦, 🟥, 🟦, 🟧}, #] & /@ possibleenvssorted;

In[∘]:= ArrayPlot[{envcolssorted}, Mesh → True, MeshStyle → Black, ImageSize → Large]
Out[∘]=
```

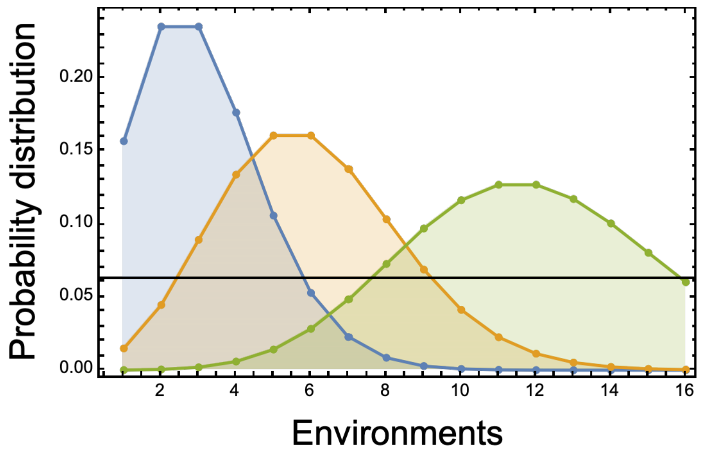


Now we derive different form of environment posteriors
When the distribution is uniform then we use *flat* or else we draw from a Poisson distribution (in *weightfun*) or we can also modify the environments to select only a few to be represented as in *depleted*

```
In[∘]:= flat = ConstantArray[1 / 16 // N, 16];

In[∘]:= rans =.;
        rans[λ_] := Table[PDF[PoissonDistribution[λ], k] // Evaluate, {k, 16}] // N
         (*RandomReal[1,16]*);
        weightfun[param_] := Module[{λ = param}, randomlist = rans[λ];
           randomlist / Total[randomlist]];
```

In[∘]:=
```
ListPlot[Table[weightfun[λ], {λ, {3, 6, 12}}], Filling → Axis,
 Mesh → Full, Joined → True, PlotRange → All, Frame → True,
 FrameStyle → Directive[Black, Thickness[0.003]], GridLines → {None, {1 / 16}},
 GridLinesStyle → Directive[Black, Thickness[0.003]],
 Method → {"GridLinesInFront" → True},
 FrameLabel → {Style["Environments", Black, 18],
   Style["Probability distribution", Black, 18]}]
```

Out[∘]=

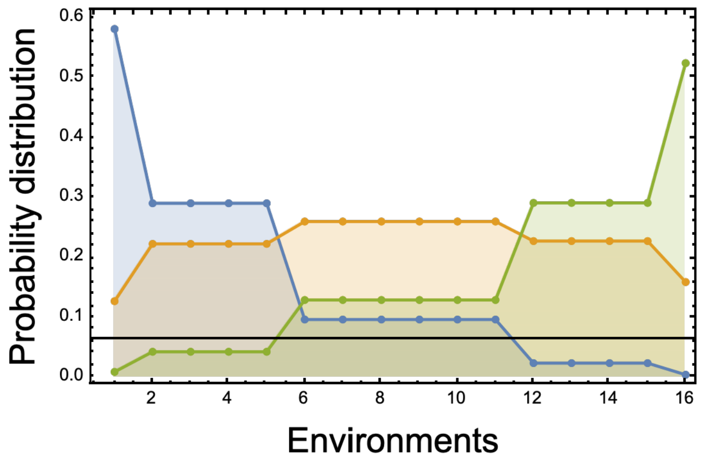


If we want to choose the richness and then choose the environment accordingly then we pick the poor (0) to rich (4) environments as per a Poisson

In[∘]:=
```
rans =.;
poorrich[λ_] := Table[PDF[PoissonDistribution[λ], k] // Evaluate, {k, 5}] // N
  (*RandomReal[1,16]*);
poorrichweightfun[param_] := Module[{λ = param}, randomlist = poorrich[λ];
   randomlist / Total[randomlist]];
```

In[ ]:=
```
ListPlot[Table[{poorrichweightfun[λ]⟦1⟧, poorrichweightfun[λ]⟦2⟧,
    poorrichweightfun[λ]⟦2⟧, poorrichweightfun[λ]⟦2⟧, poorrichweightfun[λ]⟦2⟧,
    poorrichweightfun[λ]⟦3⟧, poorrichweightfun[λ]⟦3⟧, poorrichweightfun[λ]⟦3⟧,
    poorrichweightfun[λ]⟦3⟧, poorrichweightfun[λ]⟦3⟧, poorrichweightfun[λ]⟦3⟧,
    poorrichweightfun[λ]⟦4⟧, poorrichweightfun[λ]⟦4⟧, poorrichweightfun[λ]⟦4⟧,
    poorrichweightfun[λ]⟦4⟧, poorrichweightfun[λ]⟦5⟧}, {λ, {1, 3.5, 9}}],
 Filling → Axis, Mesh → Full, Joined → True, PlotRange → All,
 Frame → True, FrameStyle → Directive[Black, Thickness[0.003]],
 GridLines → {None, {1 / 16}},
 GridLinesStyle → Directive[Black, Thickness[0.003]],
 Method → {"GridLinesInFront" → True},
 FrameLabel → {Style["Environments", Black, 18],
   Style["Probability distribution", Black, 18]}]
```

Out[ ]=



If we want a specific metabolite to be depleted from the environment and not others.. here is an example where the TRP is depleted and all the environments having TRP are then removed.

In[ ]:=
```
trpdepleted = {};
Table[If[possibleenvssorted⟦i⟧⟦2⟧ == 1, AppendTo[trpdepleted, 0],
   AppendTo[trpdepleted, flat⟦i⟧]], {i, 1, Length[possibleenvssorted], 1}];
depleted = Normalize[trpdepleted, Total[#, ∞] &];
```

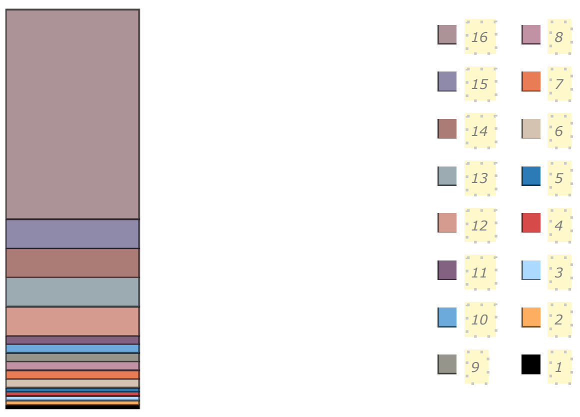
Use the appropriate distribution to finally choose the weights for the environments
Flat for Uniform
depleted for a certain metabolite removed
or weightfun which draws a Poisson Distribution with a given degree.

In[∘]:=
```
weightdistribution = Table[{poorrichweightfun[λ]〚1〛, poorrichweightfun[λ]〚2〛/4,
    poorrichweightfun[λ]〚2〛/4, poorrichweightfun[λ]〚2〛/4,
    poorrichweightfun[λ]〚2〛/4, poorrichweightfun[λ]〚3〛/6,
    poorrichweightfun[λ]〚3〛/6, poorrichweightfun[λ]〚3〛/6,
    poorrichweightfun[λ]〚3〛/6, poorrichweightfun[λ]〚3〛/6,
    poorrichweightfun[λ]〚3〛/6, poorrichweightfun[λ]〚4〛/4,
    poorrichweightfun[λ]〚4〛/4, poorrichweightfun[λ]〚4〛/4,
    poorrichweightfun[λ]〚4〛/4, poorrichweightfun[λ]〚5〛}, {λ, {9}}]〚1〛
  (*depleted*)(*flat*)(*weightfun[12]*);
```

Visualise the distribution to be used

In[∘]:=
```
BarChart[weightdistribution, ChartLayout → "Stacked",
 ChartStyle → envcolssorted, ChartLegends → Automatic, Axes → False]
```

Out[∘]=



Lifecycle

First define the global pool of strains

Initially in the global pool all strain are in equal frequency = 0.25 and hence all communities are equally probable. We sample on 96 for the experiment.

The time in the wells $t_{\text{well}} = 20$ is already set so we can choose the time spent in the pool phase relative to it.

In[∘]:=
```
pooltimes = RecurrenceTable[{a[n + 1] == 2 a[n], a[1] == 0.02}, a, {n, 1, 10}]
  (*Table[2/i,{i,{100,10,1,0.1}}]//N*);
```

```
In[∘]:= counterlist = pooltimes(*{1.28}*)
          (*use a specific time relative to t_well = 20 or the above vector pooltimes*);
        startfeedbackat = 12;
```

startfeedbackat is the cycle number where we start the feedback of nutrients in the next pool. Currently the number is set to be larger than cycles so no feedback occurs. Try 0, 5, 9 to generate results for the parameters in the manuscript

```
In[∘]:= meanfrequenciesofstrains = {};
        meandeath = {};
        storeenvs = {};
        comparingextinctions = {};
        For[counter = 1, counter ≤ Length[counterlist], counter++,
         tpool = counterlist⟦counter⟧;
         storegraphics = {};
         storebarchartdata = {};
         extinction = {};
         For[batchrun = 1, batchrun ≤ batchruns, batchrun++,
          fourrans = RandomReal[1, 4];
          distribution = If[Length[counterlist] == 1,
            {0.25, 0.25, 0.25, 0.25}, fourrans / Total[fourrans] // N];
          totalcycles = 10;
          cyclegraphic = {};
          finalnumbers = {};
          finalpool = {distribution};
          countdeath = {};
          For[numberofcycle = 0, numberofcycle < totalcycles, numberofcycle++,
           If[distribution ≠ {0, 0, 0, 0}, {
              sampledcommunities =
               Table[RandomChoice[distribution → speciesproperties, 4], 96];
              production = Total[#] & /@ sampledcommunities;
              prodreduced = production /. {2 → 1, 3 → 1, 4 → 1};
              weights = If[numberofcycle < startfeedbackat, weightdistribution,
                Table[Times @@ Table[If[possibleenvssorted⟦j⟧⟦i⟧ == 0,
                     1 - distribution⟦i⟧, distribution⟦i⟧], {i, 4}], {j, 16}]];
              randenvs = RandomChoice[weights → possibleenvssorted, 96];

              If[batchruns == 1 && Length[counterlist] == 1, AppendTo[storeenvs, weights]];
              mix = randenvs + prodreduced;
              partitionedmixes = Partition[Riffle[randenvs, prodreduced], 2];

              validmix = {};
              validcols = {};
              For[i = 1, i ≤ Length[mix], i++, If[MemberQ[mix⟦i⟧, 0] == True,
                AppendTo[validcols, White], AppendTo[validcols, Gray];
                AppendTo[validmix, i]]];
```

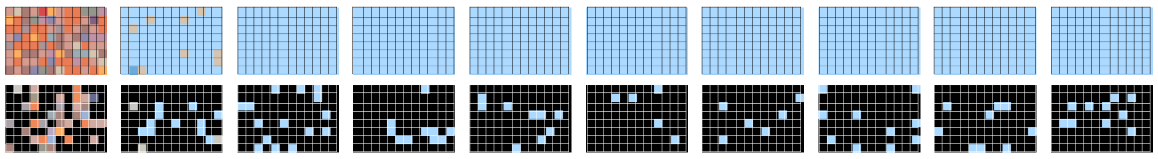
```
 AppendTo[countdeath, 96 - Length[validmix]];

 totrack = partitionedmixes[[#]] & /@ validmix;
 positionstochoosefromdictionary = Table[Position[
     allvalidcombinations, totrack[[i]]][[1, 1]], {i, 1, Length[totrack]}];
 validgrs =
  Table[envcommunityfitnessmatrix[[positionstochoosefromdictionary[[i]]]][[2]],
   {i, 1, Length[positionstochoosefromdictionary]}];

 initconds =
  Table[prodreduced[[i]] x /. Solve[Total[{x, x, x, x} prodreduced[[i]]] ==
        inoculumsize, x][[1]] // N , {i, 1, Length[prodreduced]}];
 initcolors = Blend[base, #] & /@ initconds;
 allgrrates = ReplacePart[ConstantArray[negativegrowthrate, {96, 4}],
   Table[validmix[[i]] → validgrs[[i]], {i, 1, Length[validmix], 1}]];
 finalvalues = {};
 For[i = 1, i ≤ Length[initconds],
  i ++, sol = rundynamics[initconds[[i]], allgrrates[[i]]][[1]];
  AppendTo[finalvalues,
   Evaluate[{x1[tmax], x2[tmax], x3[tmax], x4[tmax]} /. sol] // Chop]];

 finalcolors = Table[
   Blend[base, finalvalues[[i, 1]] // Chop], {i, 1, Length[finalvalues]}];

 AppendTo[cyclegraphic, {ArrayPlot[Partition[initcolors, 12],
    Mesh → True, MeshStyle → Directive[Thickness[0.001], Black]],
   ArrayPlot[Partition[finalcolors, 12], Mesh → True,
    MeshStyle → Directive[Thickness[0.001], White]]}];

 final = Chop[finalvalues, 10^-7];
 finalamounts = {final[[All, 1, 1]] // Total, final[[All, 1, 2]] // Total,
   final[[All, 1, 3]] // Total, final[[All, 1, 4]] // Total};
 finalfractions = If[finalamounts == {0, 0, 0, 0}, {0, 0, 0, 0},
   Chop[finalamounts / Total[finalamounts], 10^-7] // Quiet];
 AppendTo[finalnumbers, finalfractions];
 If[finalfractions == {0, 0, 0, 0}, AppendTo[extinction, numberofcycle]];
  poolsol = pooldynamics[finalfractions, tpool];
 afterppolgrowth = Evaluate[
    {y1[tpool], y2[tpool], y3[tpool], y4[tpool]} /. poolsol][[1, 1]] // Chop;
 distribution = If[afterppolgrowth == {0, 0, 0, 0}, {0, 0, 0, 0},
   afterppolgrowth / Total[afterppolgrowth] // Chop];
 (*Print[distribution];*)
 AppendTo[finalpool, distribution];},
AppendTo[finalnumbers, {0, 0, 0, 0}];
AppendTo[finalpool, {0, 0, 0, 0}];
AppendTo[countdeath, 96];
(*Print["here"<>ToString[distribution]];*)
```

```
      ]
    ];
    AppendTo[storegraphics, Transpose[cyclegraphic]];
    AppendTo[storebarchartdata,
     {finalnumbers, finalpool, Riffle[finalpool, finalnumbers], countdeath / 96}];
   ];
   AppendTo[comparingextinctions, extinction];
   AppendTo[meanfrequenciesofstrains,
    Transpose[Table[Table[Mean[DeleteCases[#, 0.] & /@
          Transpose[Table[Transpose[storebarchartdata⟦i, 3⟧][[j]] // N,
             {i, Range[batchruns]}]]⟦k⟧], {k, 1, 2 totalcycles + 1, 1}], {j, 1, 4, 1}]]
     (*Table[Mean[Table[Transpose[storebarchartdata⟦i,3⟧][[j]]//N,
       {i,Range[batchruns]}]],{j,1,4,1}]*)];
   meandeath = AppendTo[meandeath,
     Mean[Table[storebarchartdata⟦i, 4⟧, {i, Range[batchruns]}]]];
  ]
```

Plotting

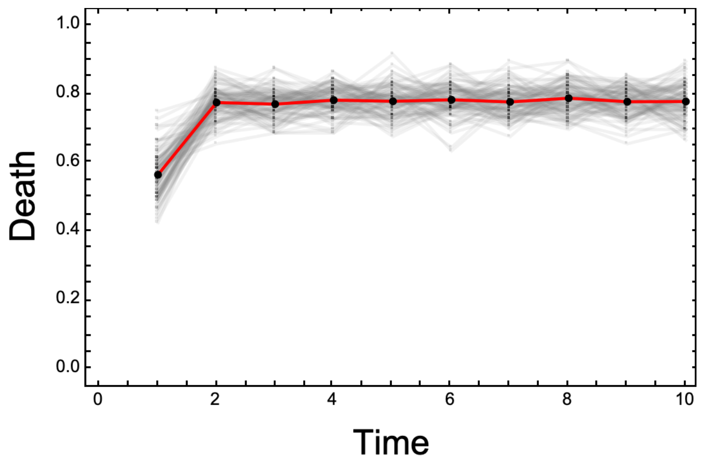
The above lifecycle code can be run in a number of different ways to get the results in the manuscript.
For batchruns = 1 we store the graphics pertaining to each cycle of the the single run.
We can also change the pool times, strain growth rates and the startfeedbacktime to get the relevant figures for single exemplary runs as shown in the manuscript.
When batchruns > 1 (for example when set to 100) and done for multiple pool times the figures for death and mean death will be relevant.
The averaging of the batch runs per t_{pool}will be done automatically for the plots.

```
In[ ]:= GraphicsGrid[storegraphics⟦1⟧]
Out[ ]=
```



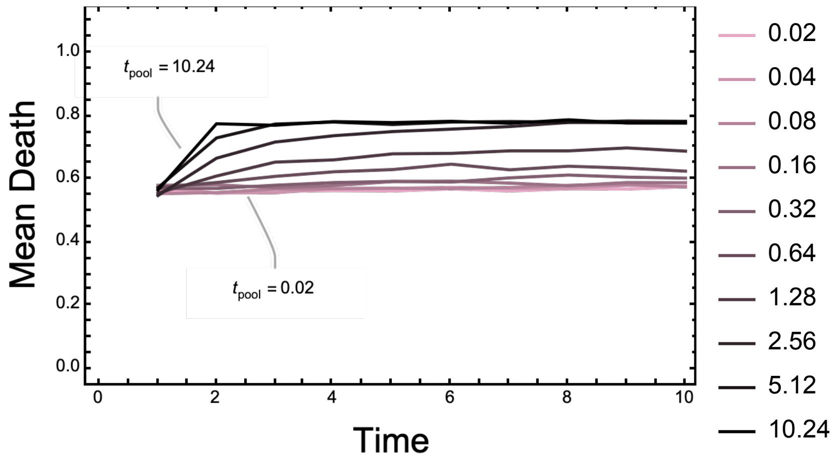
For batchruns > 1

```
In[ ]:= deathstoplot = Table[storebarchartdata⟦i, 4⟧, {i, Range[batchruns]}];
        Show[
         {ListPlot[deathstoplot, Joined → True, PlotStyle → Directive[Gray, Opacity[0.1]],
           PlotRange → {-0.01, 1.01}, Mesh → All, MeshStyle → {Black, Thickness[0.002]},
           Frame → True, FrameStyle → Directive[Black, Thickness[0.002]],
           FrameLabel → {Style["Time", Black, 18], Style["Death", Black, 18]}],
          ListPlot[deathstoplot // Mean, Joined → True, PlotStyle → Red,
           PlotRange → {-0.01, 1.01}, Mesh → All, MeshStyle → {Black, Thickness[0.002]},
           Frame → True, FrameStyle → Directive[Black, Thickness[0.002]],
           FrameLabel → {Style["Time", Black, 18], Style["Death", Black, 18]}]},
         PlotRange → {-0.05, 1.05}]
Out[ ]=
```



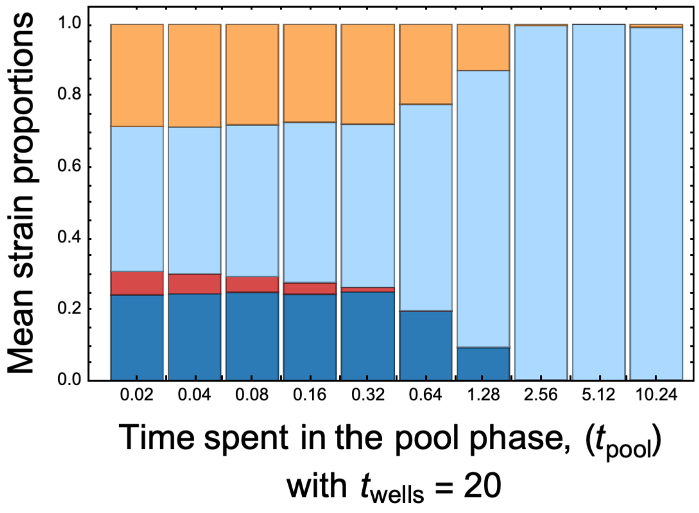
In[ ]:=
```
ListPlot[Table[If[i == 1 || i == Length[counterlist],
   Callout[meandeath⟦i⟧, "t_pool = " <> ToString[counterlist⟦i⟧],
    If[i == 1, {3, 0.38}, {1, 0.8}]], meandeath⟦i⟧],
  {i, 1, Length[meandeath]}], PlotLegends → counterlist⟦1 ;;⟧,
 PlotStyle → Table[{Darker[Hue[0.92, 0.27, 1.], i/Length[counterlist]],
    i/Length[counterlist]}, {i, 1, Length[counterlist]}]
   (*{{Red,Full},{Red,Dashed},{Red,DotDashed}}*), PlotRange → {-0.05, 1.05},
 Joined → True, MeshStyle → {Black, Thickness[0.002]},
 Frame → True, FrameStyle → Directive[Black, Thickness[0.003]],
 FrameLabel → {Style["Time", Black, 18], Style["Mean Death", Black, 18]}(*,
 GridLines→{None, {0.875}},GridLinesStyle→Directive[Red, Dashed,Thick]*)]
```

Out[ ]=



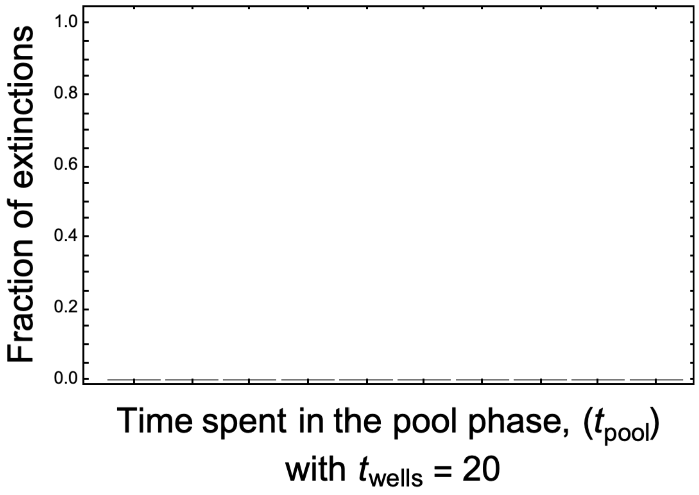
```
In[•]:= BarChart[Normalize[#, Total[#, ∞] &] & /@ Table[
            meanfrequenciesofstrains⟦i⟧ // Last, {i, Length[meanfrequenciesofstrains]}],
          ChartLayout → "Stacked", ChartStyle → base, Frame → True,
          FrameStyle → Directive[Black, Thickness[0.002]],
          FrameLabel → {Style["Time spent in the pool phase, (tpool)\n with twells = " <>
              ToString[tmax], Black, 18], Style["Mean strain proportions", Black, 18]},
          ChartLabels → {counterlist, None} (*,
          PlotLabel→Style["Equilibrium proportions",Black,24]*)]
```

Out[•]=

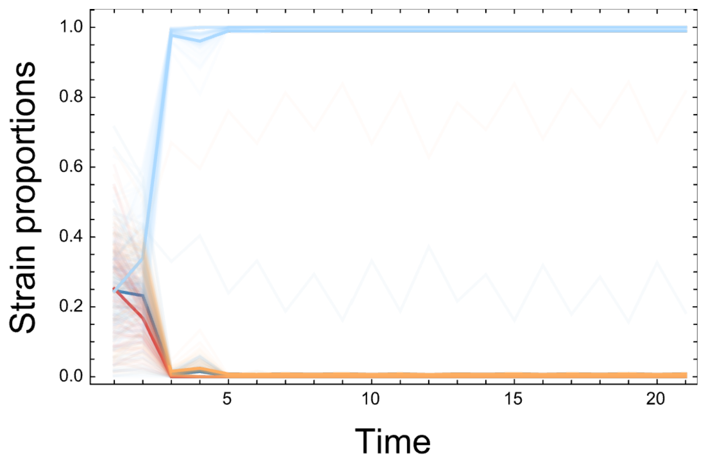


```
In[•]:= BarChart[Length[#]/batchruns & /@ comparingextinctions, ChartStyle → Darker[Red],
          Frame → True, FrameStyle → Directive[Black, Thickness[0.002]],
          FrameLabel → {Style["Time spent in the pool phase, (tpool)\n with twells = " <>
              ToString[tmax], Black, 18], Style["Fraction of extinctions", Black, 18]},
          ChartLabels → {counterlist, None} (*,
          PlotLabel→Style["Equilibrium proportions",Black,24]*),
          PlotRange → {All, {-0.01, 1.05}}]
```

Out[•]=



```
In[ ]:= Show[
         {ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[1]], {i, Range[batchruns]}],
           Joined → True, PlotStyle → Directive[base[[1]], Opacity[0.03]],
           PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[1]], {i, Range[batchruns]}] //
            Mean, Joined → True, PlotStyle → base[[1]], PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[2]], {i, Range[batchruns]}],
           Joined → True, PlotStyle → Directive[base[[2]], Opacity[0.03]]],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[2]], {i, Range[batchruns]}] //
            Mean, Joined → True, PlotStyle → base[[2]], PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[3]], {i, Range[batchruns]}],
           Joined → True, PlotStyle → Directive[base[[3]], Opacity[0.03]],
           PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[3]], {i, Range[batchruns]}] //
            Mean, Joined → True, PlotStyle → base[[3]], PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[4]], {i, Range[batchruns]}],
           Joined → True, PlotStyle → Directive[base[[4]], Opacity[0.03]],
           PlotRange → All],
          ListPlot[Table[Transpose[storebarchartdata[[i, 3]]][[4]], {i, Range[batchruns]}] //
            Mean, Joined → True, PlotStyle → base[[4]]]}, PlotRange → All,
         Frame → True, FrameStyle → Directive[Black, Thickness[0.002]], FrameLabel →
          {Style["Time", Black, 18], Style["Strain proportions", Black, 18]},
         PlotRange → {{0, 25}, {-0.01, 1.01}}]
Out[ ]=
```

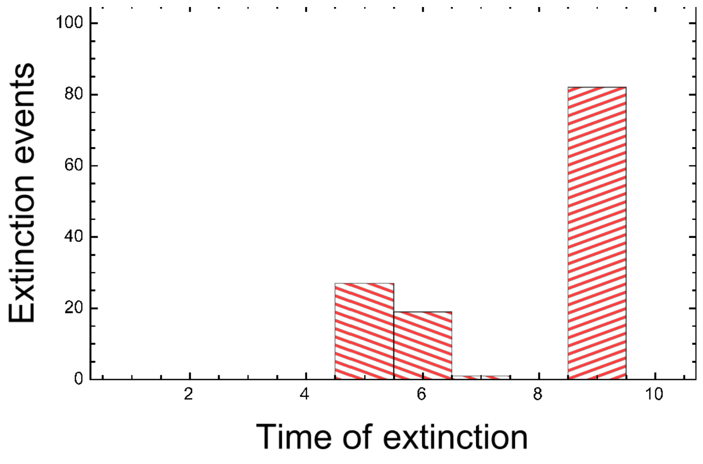


Run the above code multiple times by changing the startfeedback at for batchsize>1 and store the results in the following variables to generate the histogram and the piechart

```
In[ ]:= feedbackat9 = extinction;

In[ ]:= feedbackat5 = extinction;

In[ ]:= feedbackat0 = extinction;
```

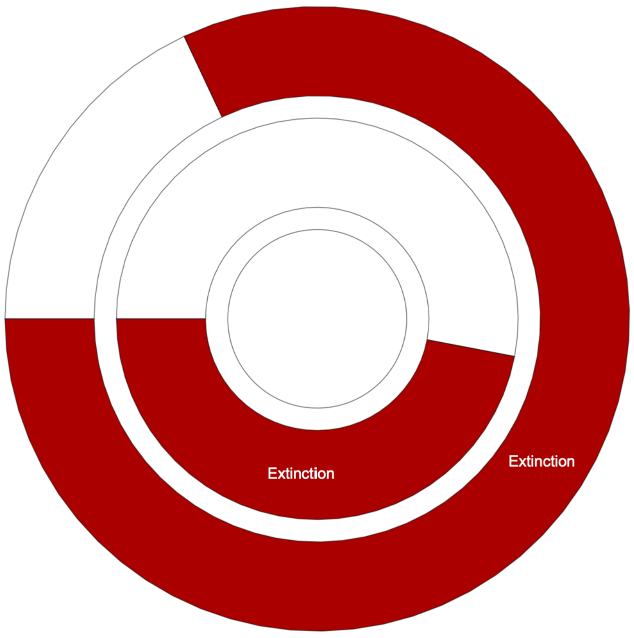
In[ ]:=
```
Histogram[{feedbackat0, feedbackat5, feedbackat9},
 Frame → True, FrameStyle → Directive[Black, Thickness[0.003]],
 FrameLabel → {Style["Time of extinction", Black, 18],
   Style["Extinction events", Black, 18]}, PlotRange → {{0.5, 10.5}, {0, 100}},
 ChartStyle → {{HatchFilling[90 Degree, 1, 4], HatchFilling[-20 Degree, 1, 4],
    HatchFilling[20 Degree, 1, 4]}, {Darker[Red], Darker[Red], Darker[Red]}}]
```

Out[ ]=



In[ ]:=
```
PieChart[{{100 - Length[feedbackat0], Length[feedbackat0]},
  {100 - Length[feedbackat5], Length[feedbackat5]},
  {100 - Length[feedbackat9], Length[feedbackat9]}},
 ChartLabels → {"", Style["Extinction", White, 8]},
 ChartStyle → {White, Darker[Red]}, Background → White]
```

Out[ ]=

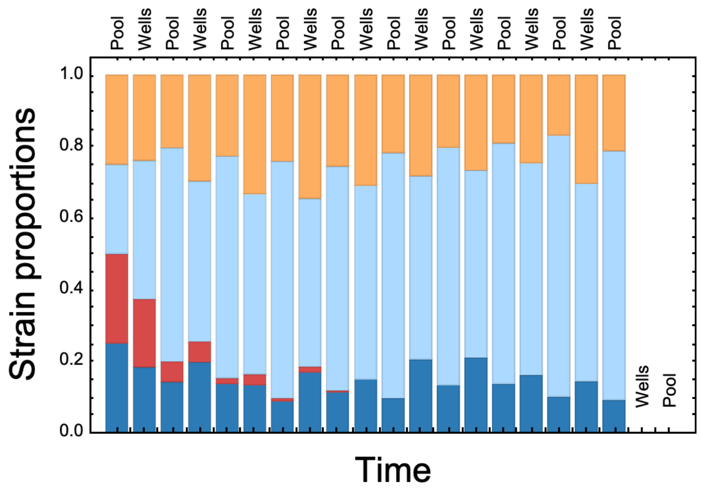


Single run (for batchruns=1)

For a single run the following is a time series of the strains and the environments over the cycles.

```
labels =
  Quiet[AppendTo[ConstantArray[{"\ \ \ Pool", "\ \ \ Wells"}, 10] // Flatten,
     {"\ \ \ Pool"}] // Flatten];
BarChart[storebarchartdata⟦1, 3⟧,
 ChartLayout → "Stacked", ChartStyle → {🟦, 🟥, 🟦, 🟧}, Frame → True,
 FrameStyle → Directive[Black, Thickness[0.002]], FrameLabel →
  {Style["Time", Black, 18], Style["Strain proportions", Black, 18]},
 ChartLabels → {Placed[labels, Above, Rotate[#, 90 Degree] &], None},
 PlotRange → {{0, 22}, Automatic}, Background → White]
```

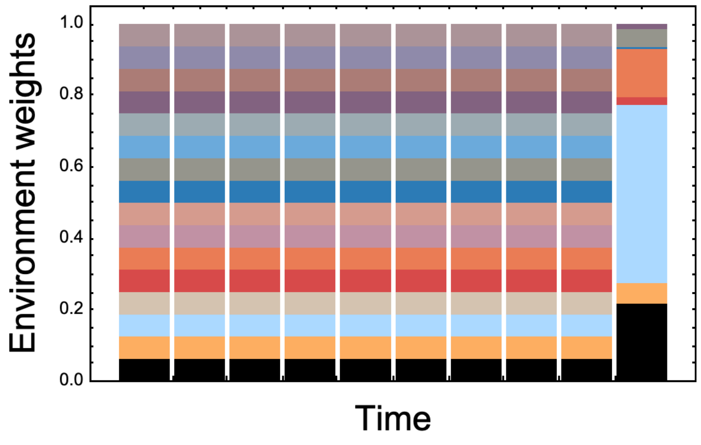
Out[∘]=



In[∘]:=

```
BarChart[storeenvs, ChartLayout → "Stacked", ChartStyle → envcols,
 Frame → True, FrameStyle → Directive[Black, Thickness[0.002]], FrameLabel →
  {Style["Time", Black, 18], Style["Environment weights", Black, 18]},
 PlotRange → {{0, 11}, Automatic}, Background → White]
```

Out[∘]=


Environment weights
Time