Section 8.2: Dicrete models

Created by Tomas de-Camino-Beck
tomasd@math.ualberta.ca

Simple Population Models

■ Exponnetial growth

Solving for the equation $x_{t+1} = r\, x_t$ where x_t is the population at time t, and r is population growth rate. Define the equation as follows:

```
eqn = x[t + 1] == r x[t];
```

Altough the solution is simple, to solve the equation we use the command `RSolve`

```
solution = RSolve[eqn, x[t], t]
```

$\{\{x[t] \to r^{-1+t}\, C[1]\}\}$

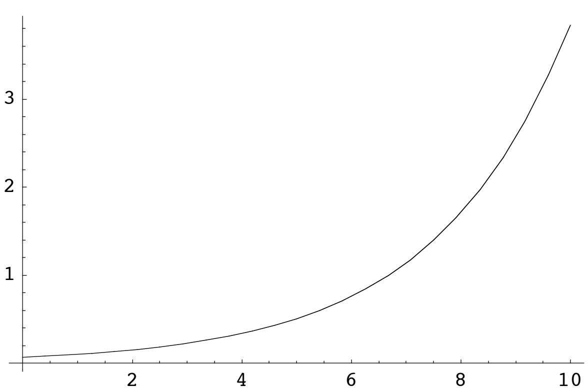
Lets evaluate the solution at 2,

```
x[2] /. solution
```

```
{r C[1]}
```

Generate a Plot for $r = 1.5$ and initial population size $C[1] = 0.1$

```
Plot[x[t] /. solution /. {r → 1.5, C[1] → 0.1}, {t, 0, 10}]
```



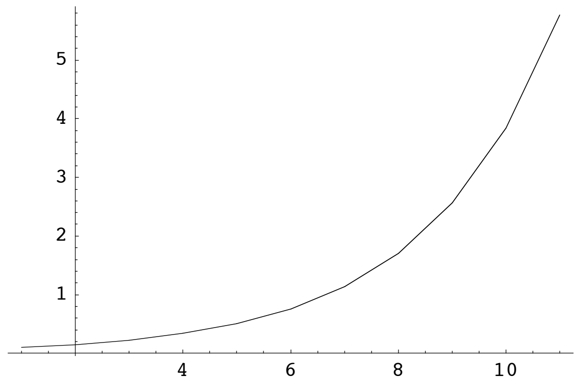

```
- Graphics -
```

Another way is using `NestList` directly using pure functions

```
r = 1.5;
l = NestList[r # &, 0.1, 10]
```

```
{0.1, 0.15, 0.225, 0.3375, 0.50625, 0.759375, 1.13906, 1.70859, 2.56289, 3.84434, 5.7665}
```

```
ListPlot[l, PlotJoined → True];
```



Note that the second option requires les code, but the plot is less smoother, and it takes more time to execute than using the algebraic solution.

■ Cobwebbing

The follwoing code builds a cobweb of a function *f*,

```
CobWeb[f_, n0_, steps_] := Module[{temps = n0, res},
   res := {{n0, 0}, {n0, f[n0]}};
   Do[res = Join[res, {{temps, f[temps]}, {f[temps], f[temps]}}];
    temps = f[temps],
    {steps}];
   Return[ListPlot[res, PlotJoined → True, DisplayFunction → Identity]];
  ];
```

■ Example

lets use the non-dimensinal logistic equation,

```
r = 3.2;
```

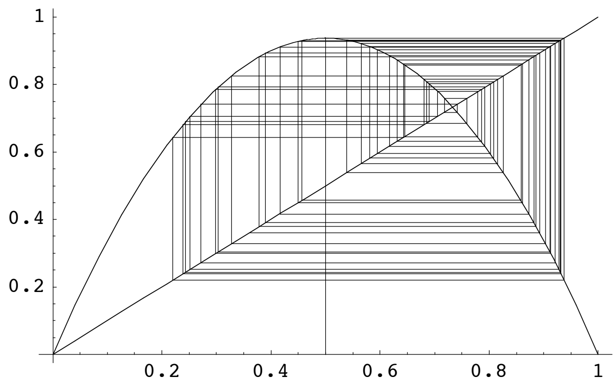
```
f[u_] := r u (1 - u);
```

Plotting the function, and assigning it to p.

Building the cobweb. The parameters for the cobweb CobWeb[f, n_0, steps] function are: f :Map to be evaluated, n_0: the initial population and steps: nummber of iterations. In this example, the two plots are combined using the Show function.

```
r = 3.75;
p = Plot[Evaluate[{f[x], x}], {x, 0, 1}, DisplayFunction → Identity]
neweps = Show[CobWeb[f, 0.5, 50], p, DisplayFunction → $DisplayFunction]
```

- Graphics -



- Graphics -

■ Bifurcation Diagram

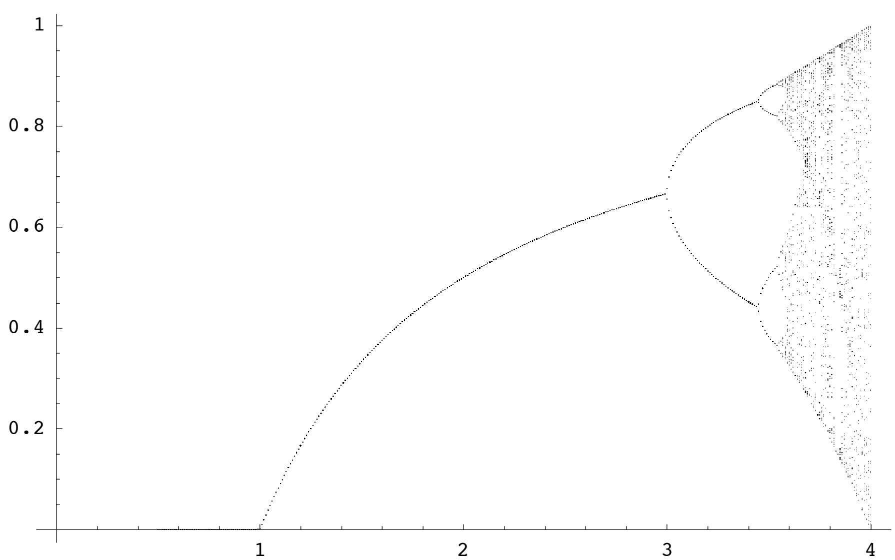
```
r = 2.5;
f[u_] := r u (1 - u);
```

A fast way of doing a biffurcation diagram:

```
bifdiagram = Table[{r, #} & /@ Take[NestList[f, 0.01, 500], -50], {r, 0.5, 4, 0.01}];
```

THis code simulates 500 points usign the logistic map, and takes the last fifty of them (using the `Take` function), and does this for different values of *r*

```
ListPlot[Flatten[bifdiagram, 1], PlotStyle → PointSize[0.001]]
```



- Graphics -

Of course, it is always better to build a general module that builds a bifurcation diagram for any discrete map, this is left as excercise. Note however, that in *Mathematica* it is always easy to create code for fast solutions.

Structured Population Models (not in course book)

■ Matrix Population Models

Killer Whales example (see Caswell 2001)
Caswell H. 2001. *Matrix population models : construction, analysis, and interpretation*, 2nd edn. Sinauer Associates, Sunderland, Mass

$$A = \begin{pmatrix} 0 & 0.0043 & 0.1132 & 0 \\ 0.9775 & 0.9111 & 0 & 0 \\ 0 & 0.0736 & 0.9534 & 0 \\ 0 & 0 & 0.0452 & 0.9804 \end{pmatrix};$$

The population growth rate is calculated from the dominant (largest) eigenvalue of this matrix:

```
Eigenvalues[A]
```

```
{1.02544, 0.9804, 0.834223, 0.0048357}
```

It can be obtained directly using the `Max` function:

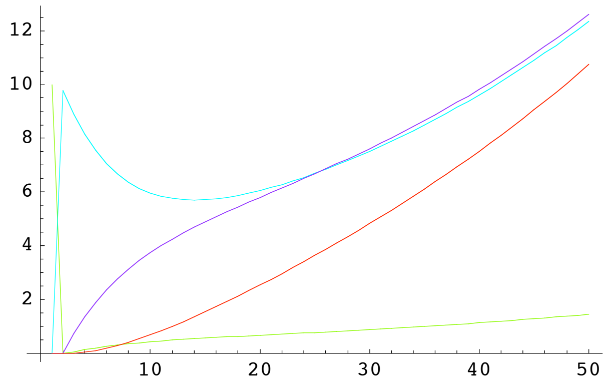
```
Max[Eigenvalues[A]]
```

```
1.02544
```

The following code allows us to do projections of all stages in the population and generate a plot.

```
PopulationProject[B_, t_, n0_] := MatrixPower[B, t].Partition[n0, 1];
ListPopulation[B_, t_, n0_] :=
  Transpose[Array[Flatten[PopulationProject[B, #, Partition[n0, 1]]] &, t, 0]];
PopulationPlot[B_, t_, n0_, plotoptions___] := Module[{l = ListPopulation[B, t, n0], plt},
   plt = Show[Sequence @@ Array[ListPlot[l[[#]], PlotJoined → True,
         PlotStyle → Hue[# / Length[l]], DisplayFunction → Identity] &, {Length[l]}],
     DisplayFunction → $DisplayFunction, plotoptions];
   Return[
    plt];];
```

```
PopulationPlot[A, 50, {{10}, {0}, {0}, {0}}]
```




```
- Graphics -
```

■ Sensitivity and Elasticity of Matrix Models

This code allows us to calculate the sensitivity and elasticity matrix (see Caswell 2001 book)

```
LeftEigenvector[B_] := Conjugate[Eigenvectors[Transpose[B]]];
RightEigenvector[B_] := Eigenvectors[B];
SensitivityMatrix[B_] :=
  Table[((Conjugate[LeftEigenvector[B]])[[1, i]] RightEigenvector[B][[1, j]]) /
    (Inner[Times, LeftEigenvector[B][[1]], RightEigenvector[B][[1]], Plus]),
   {i, Length[B]}, {j, Length[B]}];
ElasticityMatrix[B_] := Table[B[[i, j]] / Eigenvalues[B][[1]]
    ((Conjugate[LeftEigenvector[B]])[[1, i]] RightEigenvector[B][[1, j]]) /
     (Inner[Times, LeftEigenvector[B][[1]], RightEigenvector[B][[1]], Plus]),
   {i, Length[B]}, {j, Length[B]}];
```

From the previous example:

```
SensitivityMatrix[A] // MatrixForm
```

$$\begin{pmatrix} 0.0422083 & 0.360837 & 0.368644 & 0.369943 \\ 0.0442784 & 0.378534 & 0.386724 & 0.388086 \\ 0.0663227 & 0.56699 & 0.579258 & 0.581298 \\ 0. & 0. & 0. & 0. \end{pmatrix}$$

```
ElasticityMatrix[A] // MatrixForm
```

$$\begin{pmatrix} 0 & 0.0015131 & 0.0406952 & 0 \\ 0.0422083 & 0.336326 & 0 & 0 \\ 0 & 0.0406952 & 0.538563 & 0 \\ 0 & 0 & 0. & 0. \end{pmatrix}$$