Iterative Maps

\section*{Bifurcation maps}

The \texttt{Bifurcation} command generates two-dimensional images of an \textit{iterated map}. Small changes in values of some parameters in the governing equations can give rise to qualitative changes in the evolving solution of the system, referred to as a \textit{bifurcation}. For example, the evolving solution of a system of differential equations can switch between being periodic or chaotic in nature.

Images or \textit{bifurcation diagrams} can illustrate the nature of the solution as the bifurcation parameters change value.

\begin{verbatim}
> with(IterativeMaps):
  with(ImageTools):

The \texttt{Gauss map} is an example of such a system. Its bifurcation diagram can be computed using the following iterated map of a single real variable.

\begin{equation}
x_{\text{new}} = r + e^{k(x - 1)^2}
\end{equation}

This map is sometimes referred to as the \textit{mouse map} since for some parameter values the bifurcation diagram looks like a mouse.

In the following generated image the horizontal direction corresponds to \(r\) and the vertical direction corresponds to \(x\).

\begin{verbatim}
> GaussMapImage := Bifurcation([x], [r + e^{-4.8(x - 1)^2}], [0.5],
  0.2, 1.7, xmin = 0.7, xmax = 2.1,
  iterations = 10000 ) :

> ArrayTools:-Dimensions( GaussMapImage )
[1 .. 500, 1 .. 500, 1 .. 3 ]

> ColouringProcedures:-HueToRGB(GaussMapImage) :
\end{verbatim}
The `Explore` command can be used to obtain an application with which to interact with the bifurcation diagram.

This allows a convenient mechanism for exploring the behavior of the iterative process as the number of iterations or the parameter value change.

The following `appliable module` uses a local member `img` in order to take advantage of the in-place computational functionality of the `Bifurcation` command. The following procedure for `GaussMap` generates an image according to its arguments which specify the parameter `k` and the number of iterations of the map.
GaussMap := module() local ModuleApply, img;

    ModuleApply := proc(k, logiter)
        if not type(img, Array) then
            img := Array(1..400, 1..400, 1..3,
                             datatype=float[8], order=C_order);
        else
            ArrayTools:-Fill( 0.0, img );
        end if;
        IterativeMaps:-Bifurcation( height = 400, width = 400,
                                     [x],[r+exp(-k*(x-1)^2)], [0.5],
                                     0.2, 1.0+0.5*(8.0-k)/(8.0-3.9),
                                     xmin = 0.5, xmax = 2.1,
                                     outputimage = img,
                                     iterations = 1+ceil(10^logiter) );
        ImageTools:-ColouringProcedures:-HueToRGB(img);
        return img;
    end proc:
end module:

> Explore(GaussMap(k, logiter),
           parameters = [[k = 3.7..9.0, animate=false], [logiter = 0.0..4.0, 
                          label = `log[10](iterations)`]],
           initialvalues = [k = 4.96, logiter = 4.0],
           placement = left, orientation = vertical,
           animate, title = "Gauss map")
Attractor maps

The Attractor command generates two-dimensional images of the iterated or evolving solution of a system of equations. The command gets its name from the fact that certain systems have the property that if the solution becomes close enough to attaining some special state then it remains close.

For example, the special state may be the situation of being close to a particular point, curve, or surface which is then referred to as an attractor.

The Rossler attractor appears in the solution of the following set of differential equations:

\[
\begin{align*}
\frac{dx(t)}{dt} &= -y(t) - z(t) \\
\frac{dy(t)}{dt} &= x(t) + ay(t) \\
\frac{dz(t)}{dt} &= b + z(t)(x(t) - c)
\end{align*}
\]

A diagram for this system can be computed using an iterative map which effectively approximates a solution using a fixed step-size \( dt \).

A temporary variable \( tx \) is added to the list of variables, so that values for variables \( x, y, \) and \( z \) can be iterated in unison.
The behavior of this iterated map depends on the values of the parameters $a$, $b$, and $c$. Using the following values, the computed orbit exhibits chaotic behavior around a fixed attractor. For other values of the parameter $c$, the evolving solution becomes periodic in nature.

```plaintext
> a, b, c := 0.1, 0.1, 14.0 :
> RoesslerImage, xrng, yrng := Attractor(height = 400, width = 400,
[ x, y, z, tx ],
[ xnew, ynew, znew, x ],
[ 1, 1, 1, 1 ],
xmin = -24, xmax = 24, ymin = -24, ymax
= 22, fixview,
iterations = 300000 ) :
> ArrayTools:-Dimensions( RoesslerImage )
[ 1..400, 1..400, 1..3 ] (2.1)
> ColouringProcedures:-HueToRGB(RoesslerImage) :
ImageTools:-Embed(RoesslerImage) :
```
The previous image has a view onto the \( x \) and \( y \) variables' space. An oblique view can be attained by introducing two additional variables and a projection. Here, a projection is made using a rotation about the first axis. A fixed viewing range is used.

The speed of computation is due to internal use of the Maple Compiler to apply the iterative map.

\[
g := -\frac{\pi}{8} : 
\]

\[
\text{RosslerImage, xrng, yrng} := \text{CodeTools:-Usage}(\text{Attractor}(\text{height} = 400, \text{width} = 400, \text{[xview, yview, x, y, z, tx],} \\
\quad x, (28 - y) \sin(\text{ang}) + z \sin\left(\frac{\pi}{2} - \text{ang}\right), xnew, ynew, znew, x), [1, 1, 1, 1, 1], \text{xmin} = -24, \text{xmax} = 24, \text{ymin} = -20, \text{ymax} = 28, \text{fixview, iterations}}
\]
The `Explore` command can be used to obtain an application which can be used to interact with the Attractor diagram.

This allows a convenient mechanism for exploring the behavior of the solution as the number of iterations or the parameters change. The tilt angle can also be made interactive.

The following **applicable module** uses a local member `img` in order to take advantage of
the in-place computational functionality of the Attractor command. The following procedure, **Rossler**, will generate an image according to its arguments which specify the parameter $c$, the number of iterations of the map, and the angle of tilt in the projected view.

The procedure **Rossler** can be passed to the **Explore** command to provide a convenient interface for varying the values of its arguments.

```plaintext
Rossler := module() local ModuleApply, img;
    ModuleApply := proc(C, logiter, ang)
        local znew;
        znew := z + dt*(b + z*(tx - C));
        if not type(img, Array) then
            img := Array(1..400, 1..400, 1..3,
                          datatype=float[8], order=C_order);
        else
            ArrayTools:-Fill( 0.0, img );
        end if;
        IterativeMaps:-Attractor( height=400, width=400,
                                   [xview, yview, x, y, z, tx],
                                   [x, 1/2*((24-y)*sin(ang)+z*sin(Pi/2-ang)),
                                    xnew, ynew, znew, x],
                                   [1, 1, 1, 1, 1],
                                   xmin=-24, xmax=24, ymin=-24, ymax=24, #fixview,
                                   outputimage = img,
                                   iterations = 1+ceil(10^logiter) );
        ImageTools:-ColouringProcedures:-HueToRGB(img);
        return img;
    end proc:
end module:
```

> Explore \( \text{Rossler}(C, \text{logiter}, \text{angle}) \),
Escape maps

The *Escape* command generates images from iterated maps involving several variables. The dimensions of the resulting image correspond to the initial values of the first two specified variables of the map. The iterated computation can involve several variables, and operates by updating the value of each two-dimensional image using the previous value and (optionally) the values of any additional variables.

The iteration continues until either the supplied iteration limit is exceeded or a supplied *escape* condition is met.

One well-known application of such iterative maps is the generation of images for which the approximation of the boundaries of the regions that escape are fractal in nature.
\[ \text{expr} := (zr + zr I)^3 \]

\[ \text{expr} := (zr + 1zr I)^3 \quad (3.1) \]

\[ fzi := \text{evalc}(\Re(e_{xpr})) \]

\[ fzi := -zr^3 + 3 zr zr^2 \quad (3.2) \]

\[ fze := \text{evalc}(\Re(\text{subs}(zr = zit, expr))) \]

\[ fze := -3 zit^2 zr + zr^3 \quad (3.3) \]

\[ \text{mandelbroid} := \text{Escape}([\text{zi}, \text{zr}, \text{zit}, \text{zrsqr}, \text{zisqr}], \]

\[ [fzi + y, fze + x, zr, zr^2, zr^2], \]

\[ [y, x, x^2, y^2], \]

\[ 4 < zrsqr + zisqr, \]

\[ -1.2, 1.2, -1.2, 1.2 \] : \]

\[ \text{ArrayTools:-Dimensions( mandelbroid )} \]

\[ [1 .. 500, 1 .. 500, 1 .. 3] \quad (3.4) \]

\[ \text{ColouringProcedures:-HueToRGB(mandelbroid)} : \]

\[ \text{Embed(mandelbroid)} \]