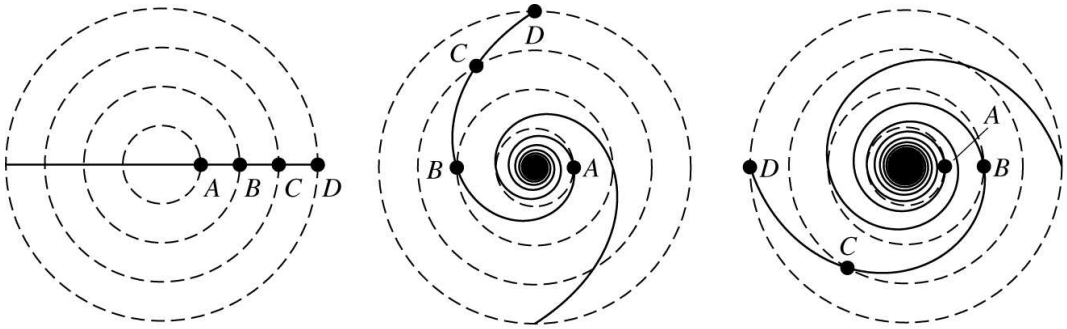
Spiral Galaxies with Maple (sll --- v1.0, March 2012)

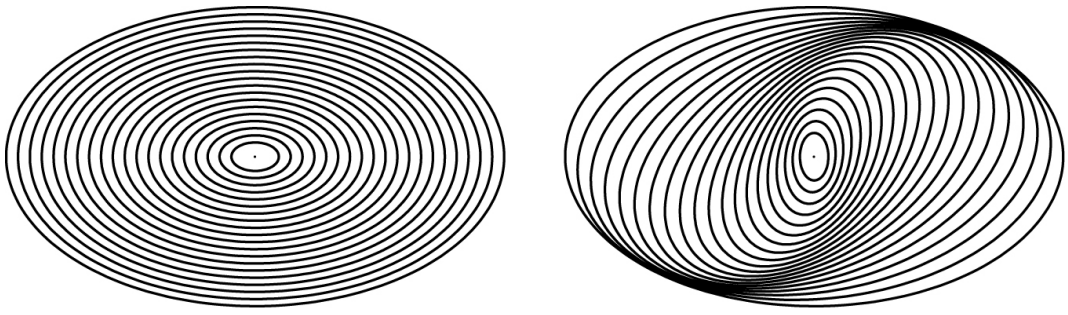
Galaxy Arms and Density Waves

The origin of spiral arms is a matter of intense interest in astrophysics. They are believed to be dynamical structures, the result of *density waves* that are set up in the stellar distribution as the galaxy rotates.

Kepler III tells us that stars at different radii from the center of the galaxy orbit at different speeds. This has the effect of "winding" the galaxy up, as shown in the figure below (from BOB).



The idea that density waves are responsible for the appearance of spiral galaxies was first proposed in the 1960s by Frank Shu and C.C. Lin. A density wave is a pattern of overdensity that appears in the galaxy due to the differential rotation of stars. Stars bunch up together and the overdensity is easily recognized as a spiral arm. Like stars, gas is overdense in the spiral arms which leads to increased star formation, a generic feature of most spiral arms (that is why spiral arms generally appear blue in images). A classic example of the emergence of spiral structure from a density wave is shown below by imagining a series of ovals of different sizes rotating at different speeds depending on the size of the oval.



How do you reconcile the idea of constant winding with the fact that most spiral arms are only wrapped around a galaxy by a minimal amount? The resolution is that stars and gas are not statically related to the spiral arm; they are "passing through" in much the same way cars pass through a traffic jam on the interstate. There is a region of overdensity, where cars bunch up. New cars come in at the back, cars leave at the front, but the jam persists.

The mathematical formalism of density wave theory (some of which is covered in CH 25 of BOB) is complex and still not well understood. For our purposes here, we are interested in the morphology of galaxies. As it turns out, we can simulate the spiral structure of galaxies using a simple analytical model that treats the arms as distorted geometrical objects.

A Toy Model: Distorted Geometric Beams[1]

To your eye, spiral arms are simply linear structures that have been wound around the core of the galaxy. We will describe the arms using polar coordinates $\{r, \phi\}$ that depend on two *phenomenological parameters*. Phenomenological parameters are functional parameters that reproduce with high fidelity the physical state of a system, but are not necessarily derived from proper physical considerations. Ultimately they must be related to physical parameters if they illustrate the correct state of the system, but they are used because they are motivated by simple mathematical considerations or by fits to the data.

The two phenomenological parameters are called α and β. Experimenting with the toy model, you can discover the effect of each of these parameters on the appearance of the galaxy. In particular:

- ▷ β is related to the winding number — the number of times a spiral arm wraps around the galaxy. It is not exactly the winding number, but usually within ~ 1 depending on where you start your counting.
- ▷ α controls the strength of the bar in the model. For $\alpha \ll \beta$ the bar is minimal; for $\alpha \sim \beta$ the bar is strong.
- ▷ For $\alpha < \beta$ there is a smooth transition from the end of the bar to the trailing spiral arms. For $\alpha > \beta$ there is a kink (not unheard of in galactic morphology)

In addition, there are several physical parameters that we use to characterize the appearance of the model. These are:

- ▷ r_{bar} a central region of the galaxy where the beams are rigid and co-rotate with the galaxy. We are not modeling true physical distances, but rather modeling geometric proportions and appearances. As a general rule, I set $r_{bar} \equiv 1$ and then adjust the other parameters to control the appearance of my galaxy.
- ▷ r_{max} characterizes the overall size of the galaxy. If I am trying to model a particular galaxy (say one I have in an image) then I generally set r_{max} by looking/measuring off the image the overall size of the galaxy compared to the bar (remember in this model I always have $r_{bar} = 1$). This is not perfect, since the appearance of the bar is also controlled by my choice of the parameter α, but this usually gets me in the ballpark such that tinkering will produce a good model.
- ▷ ω is the orientation of the galaxy to the horizontal on your image/page/computer screen. It is the angle measured clockwise from the positive $x-$axis to the bar of the galaxy. For those of you familiar with the study of visual binaries or orbits projected onto other planes, this angle is similar to the *longitude of the ascending node*.
- ▷ i is the inclination of the galaxy to the line of sight. Face on spiral galaxies (often called

[1] Our toy model follows a model described by Roger Sinnott in the Dec 1990 *Astronomical Computing* column in Sky & Telescope. Note that based on output from our `Maple` model, I think the parameters in some of his figures, particularly on the third page, are incorrect.

grand-design spirals) have equal diameters in all directions, and $i = 0°$. As a galaxy is inclined to the line of sight, pivoted around the x-axis, a circular galaxy of radius a appears as an ellipse with semi-major axis a and semi-minor axis b. You can measure the axes of a galaxy in an image and determine the inclination from

$$i = \cos^{-1}(b/a)$$

- ▷ c_w is a parameter that I use to control the handedness of the spiral. $c_w = 1$ gives a right-handed spiral, and $c_w = -1$ gives a left-handed spiral. There is some minor opportunity for confusion here, because the other parameters can cause arms to do odd turns and loops, so the handedness can be hard to define. As a general rule, if I need the spiral to go the other way, I simply flip the sign of c_w in my model!

Lastly, there are some model parameters that we use to control how `Maple` sets up and computes the model.

- ▷ `nR` is the number of data points we will calculate along each arm. This affects smoothness of the model when it is plotted, and also the amount of memory that `Maple` uses! Increase this number if your arms don't look smooth.
- ▷ $dR = r_{max}/\texttt{nR}$ is the stepsize along each of the arms that data points are calculated.

The parameters r measures our distance along the beam. The beam is assumed to have two parts: an inner part $r < r_{bar}$ that rotates rigidly with the galaxy, and an outer part $r > r_{bar}$ that gets wound around the galaxy. Depending on the value of r we define the polar angle ϕ to be

$$r < r_{bar} \qquad \rightarrow \qquad \phi = (\alpha - \beta)r$$

$$r > r_{bar} \qquad \rightarrow \qquad \phi = \alpha(1 + \ln r) - \beta r$$

From these, we can compute the $\{x, y\}$ coordinates along the arm (the easiest thing to use if we want `Maple` to draw the spiral arms by connecting dots with lines) using the usual polar $\rightarrow$ Cartesian conversions:

$$x = r \cdot \cos\theta \qquad \rightarrow \qquad x = r \cdot \cos(\phi + \omega + \phi_o)$$

$$y = r \cdot \sin\theta \qquad \rightarrow \qquad y = r \cdot \sin(\phi + \omega + \phi_o) \cdot \cos(i)$$

The constant ϕ_o allows us to use the same equations for each of the two arms in our model. One arm is given by $\phi_o = 0$, and the other arm is given by $\phi_o = \pi$.

Putting the Model into Maple

We're going to be using some of `Maple`'s primitive drawing ability (like drawing lines between specified coordinates) so we need to load the full graphics packages:

```
[> with(plots):

[> with(plottools):
```

In order to specify the points that Maple is going to connect for the spiral arms in our model, we have to compute and then store the data points. The most convenient way to do this is to set up arrays. First, we define the parameters that will control the size and range of the data:

```
[> rMax := 5.0;

[> nR := 100;

[> dR := rMax/nR;
```

Now I define some arrays to work with; I define four arrays for the x and y position data in each of the two arms. These commands simply fill out some dummy arrays to the appropriate length.

```
[> x1 := Array([seq(1,ii=1..nR)]):

[> x2 := Array([seq(1,ii=1..nR)]):

[> y1 := Array([seq(1,ii=1..nR)]):

[> y2 := Array([seq(1,ii=1..nR)]):
```

I also define an array myRadius that holds the r values where the model will be computed. Note the use of the stepsize dR here in the calculation:

```
[> myRadius := Array([seq(evalf(ii*dR),ii=1..nR)]):
```

Next, I put in the parameters that control my model. This is the place where most of the tinkering happens when I am trying to match the model to a particular galaxy image. Note that I've set up the formulae for the angles i and ω such that I can enter a number in degrees, but they are immediately converted to *radians* before Maple does any calculations with them!

```
[> rBar := 1.0;

[> inc := 60*Pi/180;

[> w := 0*Pi/180;

[> alpha := 0.5;

[> beta := 3.0;

[> CW := -1;
```

The Heavy Lifting: Calculating the Arms

Now comes a bit of new, complicated Maple. We have a series of r values (stored in myRadius) for which we'd like to know the corresponding value of ϕ, which can then be used to calculate the {x,y} values along each arm. In the past, we've done these sorts of calculations through studious use of the seq() command to build new arrays of numbers.

In this case, that would be difficult because there are different values of ϕ depending on the current value of r. Fortunately, `Maple` has some built in commands to help with just this sort of situation. Those of you familiar with programming languages will recognize these commands as the `Maple` equivalent of *for-do loops* and *if-then conditionals*.

Some coding know-how ▶

One of the difficulties of using programs like `Maple` and `Mathematica` is that the commands get long and messy and are hard to read. Just as with ordinary computer-language programming, you can make your life a lot easier by adopting some useful coding habits. Here is some advice:

- ▷ `Maple` reads commands by starting immediately after the `[>` prompt, and reading until it sees a `;` or `:` As such, you can make your life easier by splitting commands up across multiple lines, breaking the lines so that blocks of the command that are related are on a single line together. *To make Maple start a newline without evaluating the command, type* ***shift-return***.
- ▷ Indentation is your friend. The commands we are about to learn (the *for-do* and the *if-then* enclose other `Maple` commands. In order to be clear what is and is not contained within the purview of these programming structures, we use indentation.
- ▷ It is worthwhile to adopt some standardized naming habits when choosing and using variables. This facilitates understanding what is going on in your code with a quick glance. My basic rules are as follows (you are free to adopt them, or stylize to your own way of thinking):
  - When you are choosing names for variables, choose something useful if possible; give the variable a name that will mean something if you read it. Instead of stuffing a whole bunch of stuff in `A`, how about calling it `Density`? In many instances, you can mimic what you might write in algebra; for instance the temperature at the core of a star, T_{core}, is well represented by `Tcore`. You can often make good variables by intercapping (using lowercase letters for words with capital letters to distinguish words), like `dustDensity` and `gasDensity`.
  - For numbers that are constants in a problem, I use capital letters for the bulk of their name. For instance `nDAT` might be the number of data points, and `kBOLTZ` might be the Boltzmann constant.
  - You often have iterative counters, particularly in applications like *for-do loops*. In order to always be clear that I have a counter, I label such variables with a repeated lower case letter, like `ii`, `kk`, or `qq`.

The for-do Loop ▶

In many applications you have a calculation that needs to be repeated a fixed number of times. In those instances, a *for-do loop* is the right hammer for the job. The syntax of a *for-do loop* in `Maple` is (note I'll use blue text for example commands not directly part of our galaxy simulation)

```
[> for counter from initial value to final value do Maple statements end do;
```

As the author of the *for-do loop*, you have to specify the iterative counter and the initial and final values it is counting between. Note that the value of the iterative counter can be used inside the *for-do loop*. Let's do an example.

Suppose you wanted to check to see if numbers are Mersenne primes — prime numbers that can be written as $p = 2^n - 1$ where n is an integer. You wanted to check all the integers from $n = 1$ to $n = 20$ to check if the numbers p are prime. You might create a *for-do loop* to generate the numbers as follows

```
[> nCHECK := 20:

[> pMersenne := Array([seq(1,ii=1..nCHECK)]):

[> for ii from 1 to nCHECK do pMersenne[ii] := 2^ii - 1:  end do;
```

So this loop executes `nCHECK` times. The first time through, `ii = 1`, the second time `ii = 2` and so on. The loop's only activity is to make a calculation of the next number in the Mersenne sequence, and store it in the `ii`th element of the array `pMersenne`. Note that the `Maple` command inside the *for-do loop* ends with a `:` If it did not, each time the loop executed, it would print out the result of the calculation. Try putting a `;` in and see what happens.

There are other, sophisticated things you can do, like increment by some amount other than 1 or count backwards. Check the `Maple` help for details. There are also other kinds of loop structures, which are also covered in the `Maple` help.

The if-then Conditional ▶ .....
Conditional statements execute a series of `Maple` commands only when certain conditions are true. The *if-then-else conditional* is a structure of this type. The syntax in `Maple` has the following form:

```
[> if condition then Maple statements else Maple statements end if;
```

The `else` part of this statement is optional; it could just as easily been written as

```
[> if condition then Maple statements end if;
```

There are all manner of conditions that can be constructed. You use *relational operators* to write them. The operators `Maple` will accept are summarized in the table below.

Conditional	Name	Example
<	less than	`a < 42.0`
>	greater than	`density > 3.14e15`
<=	less than or equal	`nPLANETS <= 9`
>=	greater than or equal	`radius >= rEARTH`
=	equal	`Tcore = Tsun`
<>	not equal	`atomicNumber <> 26`

Table 1: Conditional operators in `Maple`.

This allows you to make calculations only some of the time. For instance, imagine that calculating $\sin\theta$ was harder than it is (it *used* to be hard, before the advent of electronic calculators). You know that for small angles $\sin\theta \sim \theta$. When this is valid is a matter of personal choice, so supposed you are willing to use this as an approximate for $\sin\theta$ up to $\theta = 10^\circ = 0.1745$ rad. If the angle is greater than this, you use the normal built in sin function. Such a `Maple` statement might be

```
[> if (theta < 0.1745) then mySin := theta:  else mySin := sin(theta):  end if;
```

I could have just as easily written the condition in the opposite way and used the statement

```
[> if (theta > 0.1745) then mySin := sin(theta):  else mySin := theta:  end if;
```

Again, note the use of `:` instead of `;` to suppress the `Maple` output.

Loop to Build the Galaxy Arms ▶ .....
Let's use our new found knowledge of the *for-do loop* and *if-then conditional* to do our calculations of the galaxy arms.

We will use a *for-do loop* to calculate {x,y} for each of the arms at each of the nR data points in myRadius. The loop will run from ii = 1 to ii = nR.

We will use and *if-then conditional* to decide which expression for ϕ to use, based on what the current value of r in myRadius is. The conditional will compare myRadius[ii] < rBar

The loop will contain multiple statements (all separated by a :). In order to simply the appearance of the formulae for the {x,y} positions along each arm, we define several auxiliary variables cos1, cos2, sin1 and sin2, the trig evaluations for each arm.

Enter the code below; remember that to start a new line without evaluating the expression, use ***shift-return***.

```
[> for ii from 1 to nR do
     if myRadius[ii] < rBar
        then phi := (alpha - beta)*myRadius[ii]:
        else phi := alpha*(1.0 + log(myRadius[ii])) - beta*myRadius[ii]:
     end if;
     cos1 := cos(phi + w):
     cos2 := cos(phi + w + Pi):
     sin1 := sin(phi + w):
     sin2 := sin(phi + w + Pi):
     x1[ii] := evalf(myRadius[ii]*cos1):
     x2[ii] := evalf(myRadius[ii]*cos2):
     y1[ii] := evalf(myRadius[ii]*CW*sin1*cos(inc)):
     y2[ii] := evalf(myRadius[ii]*CW*sin2*cos(inc)):
   end do:
```

Drawing the Galaxy ▶ .....
At this point we've calculated all the coordinates of a series of points along each of the two arms, and stored them in the arrays {x1,y1} and {x2,y2}. Maple can draw lines between any pair of coordinates using the line() command. The syntax to draw a line from $\{x_a, y_a\}$ to $\{x_b, y_b\}$ is

```
[> line([xa,ya],[xb,yb],color=red);
```

We have a lot of points we would like to draw lines between. We could enter repeated line() commands and draw them one at a time, but the easiest approach here is to create an array whose elements are all the individual line() commands; the entire sequence can then be plotted together using the display() command, which we've used in the past to overlay multiple plots.

First, we set up two arrays, one for each arm. We draw the arms different colors, to facilitate understanding the structure of the spiral pattern when Maple plots it. To fill the array, we use the familiar seq() command:

```
[> Arm1 := [seq(line([x1[ii],y1[ii]],[x1[ii+1],y1[ii+1]],color=red),ii=1..nR-1)]:
```

```
[> Arm2 := [seq(line([x2[ii],y2[ii]],[x2[ii+1],y2[ii+1]],color=green),ii=1..nR-1)]:
```

Note that because `line()` needs two points, we only generate `nR - 1` data points; the last data point in the line does not have a partner for us to draw a line to!

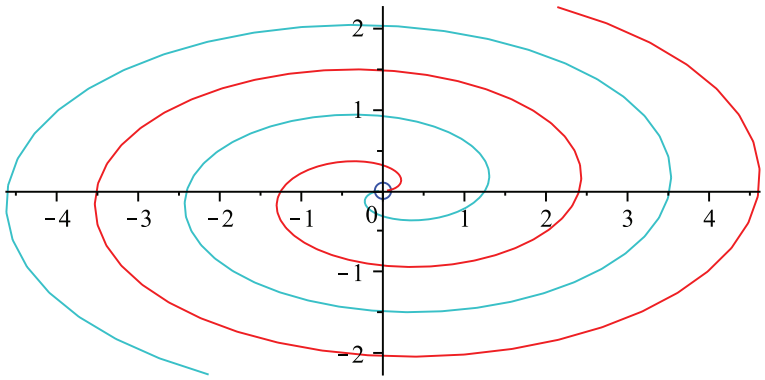
When we draw the galaxy, it is nice to have a small indicator of the center, so we draw a small circle there, which we name `core`

```
[> core := circle([0,0],0.1,color=blue):
```

Now, at last we are ready to display our galaxy model!

```
[> display(core,Arm1,Arm2,scaling=constrained);
```

If you have followed along my example, you should see something like the figure below.



Capturing images ▶ .....

We're going to run several examples where you change the parameters each time. When you get a model you like (or that I ask for!) you'll want to capture the image so you can print it out. In `Maple` you capture an image by right-clicking on the plot (***control-click*** on the Macintosh – true for Windows?). A menu will pop up, with one of the choices being `Export`; you can choose any output format that is agreeable to you.

Modeling Real Galaxies

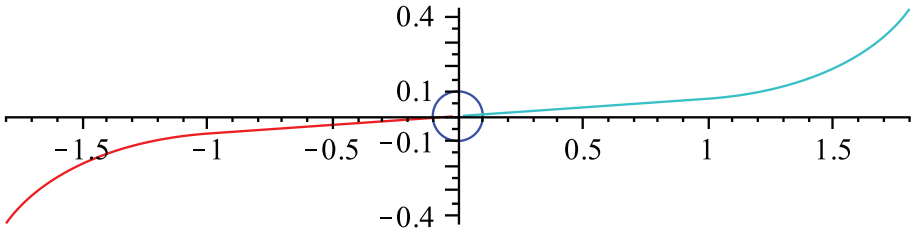
We can use images of real galaxies to determine the model parameters that best describe their appearance. For any given image, you can make measurements of the image (either in a software program, or with a ruler and protractor on a printed image. The measurements you can make are:

- ▷ The semi-major and semi-minor axes, a and b. From these, compute the inclination i
- ▷ The relative size of the bar (if any) compared to the overall size of the galaxy.
- ▷ The orientation angle ω of the bar (or the arm anchor points at the core), measured clockwise from the x-axis.
- ▷ The number of windings the arms make determines the size of β, and the strength of the bar determines the size of α relative to β. As we'll see, these are the parameters that require the most fiddling.

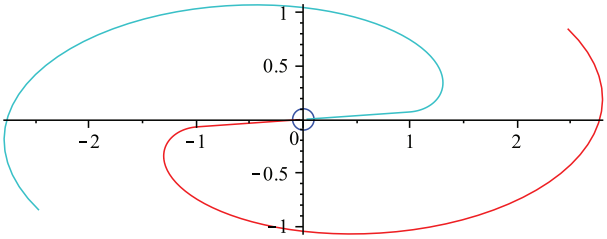
Let's do a real example together, then you can do two more on your own.

NGC 1300 ▶ .....

NGC 1300 is an SBc galaxy located in the constellation Eridanus. Making my own measurements off the image below, I settle on parameters: $i = 60°$, $\omega = 172°$, and $r_{max}/r_{bar} \sim 3$. The arms only make one circuit of the core, so I choose $\beta = 1.0$ as my initial guess. Since the bar is strongly present, I choose $\alpha = \beta = 1.0$.



My first attempt doesn't look so hot. What's going on? β most closely approximates the winding number when $r_{max} \gg r_{bar}$; here, since we've made $r_{max} \sim 3r_{bar}$, we have to increase the value of the parameters β and α dramatically to get the full winding. If I set $\beta = \alpha = 4$, keeping my other choice of parameters, then I get a good looking approximation to NGC 1300. `:-)`



Note that if you try and fit NGC 1300 on your own, you may come up with slightly different parameters, depending on precisely how you make your measurements, the accuracy of your measurements, and how *you* gauge the morphological importance of certain features (where are the edges of the disk when you measure a and b? How strong is the bar when you assign the value of α? etc.).

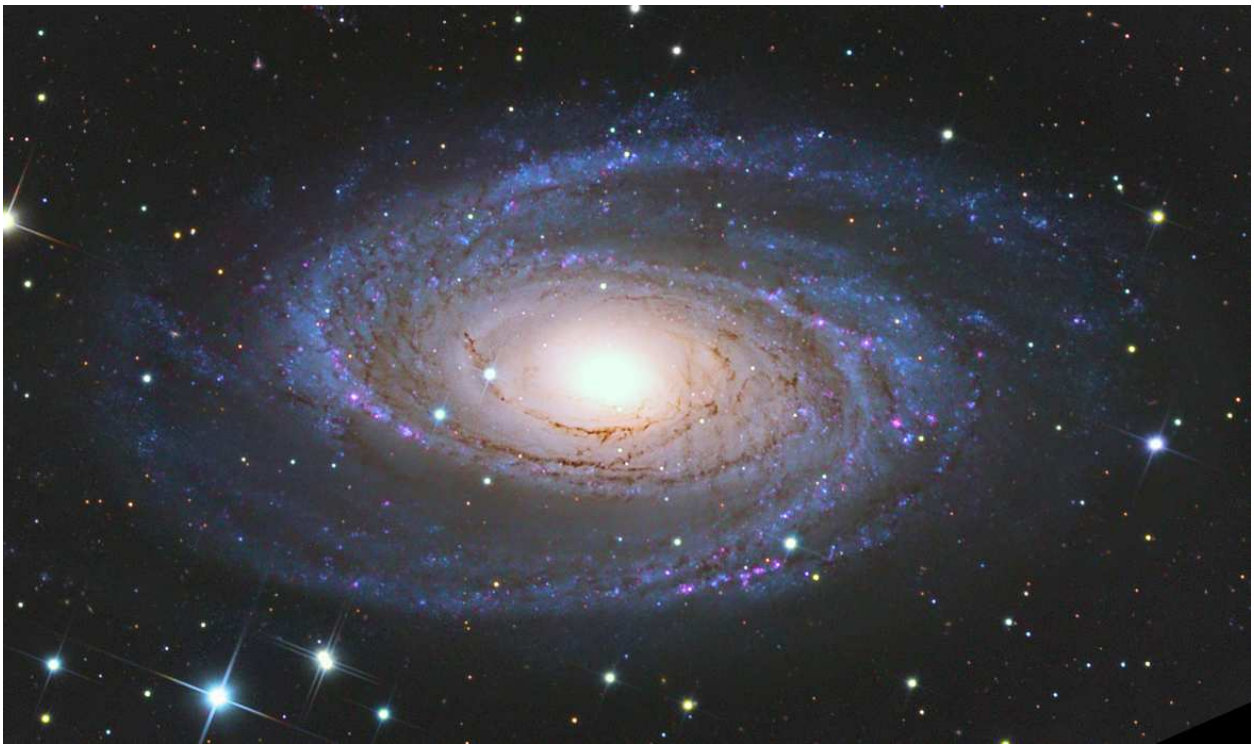
Galaxies on your own ▶ .....

On the next pages are three galaxies for you to do on your own. I've provided entry lines for you to report your model parameters (note I've assumed you will leave $r_{bar} = 1.0$).

Turn in the following: a printout of your `Maple` notebook, a printout of your generated model galaxies, and the record pages with your final parameter choices.

M81 (Bode's Galaxy) ▶ .....

M81 is a spiral galaxy located on the outskirts of Ursa Major. It has an active nucleus, powered by a 70 million solar mass black hole!



Parameter	Model Value	Parameter	Model Value
rMax		w	
a		CW	
b		alpha	
i (calculated)		beta	

NGC 1365 ▶ .....

NGC 1365 is a type SBb spiral galaxy located in the constellation Fornax. It's bar structure is prominent, and it is sometimes known as the *Great Barred Spiral Galaxy*.

Parameter	Model Value	Parameter	Model Value
rMax		w	
a		CW	
b		alpha	
i (calculated)		beta	

M51 — Whirlpool Galaxy ▶ .....

M51 is a grand-design spiral galaxy located Canes Venatici, just below the handle of the Big Dipper. It is famous for being the first galaxy in which spiral structure was detected (in 1845, by Lord Rosse).

Parameter	Model Value	Parameter	Model Value
rMax		w	
a		CW	
b		alpha	
i (calculated)		beta	