

Plop User's Manual

David Lewis, Mar 2021

Introduction

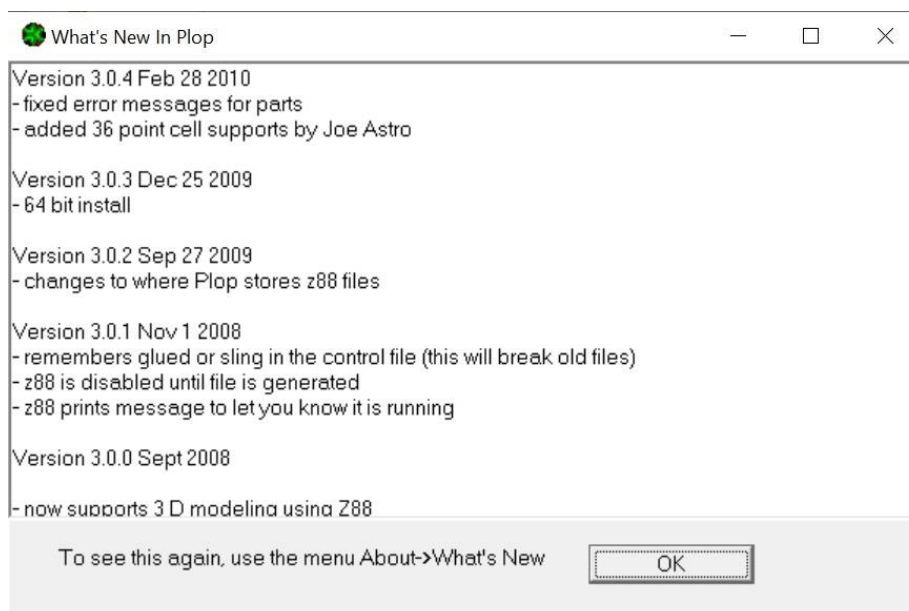
Plop is a GUI-based analysis and optimization program for mirror cells. It is based on the Plate program by Toshimi Taki, and enhanced to run hundreds of times faster, so that it can support optimization of the mirror cell by adjusting the design to minimize the wavefront error. It also supports the FEM program z88 (<https://en.z88.de/>) which is a powerful and excellent program for FEM analysis written by Prof. Dr.-Ing. Frank Rieg and his team. Plop uses a very old version of Z88 (v12) which is adequate for its purposes. Due to the complexity of Z88, I have not been able to perform the same level of speedup needed to perform optimization, so Z88 is only for analysis, not optimization. However, being a 3D model, its results are more accurate than Plate, and it can also analyze mirror cells that are tilted with respect to the ground, while Plate only performs analysis of horizontal mirror cells.

What exactly does analysis and optimization mean? Analysis means that Plop will perform a calculation of the mirror to determine the wavefront error at each point on the mirror due to gravity acting on it. Optimization allows Plop to move around the positions of the supports, including both the radius and angle, if specified, and even the relative amount of force on each point, to minimize the wavefront error. Plop can produce pictures with the dimensions of each of the cell components and their locations.

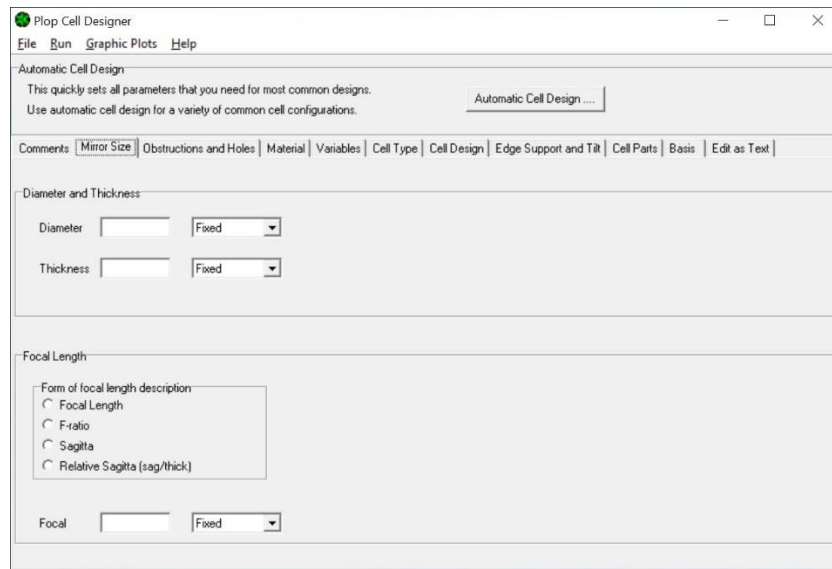
Plop contains a very general way of describing the mirror cell, but also has a design wizard that will handle almost all conventional cell designs.

Quick Start

After downloading and installing Plop, you should see it in your start menu under Gui Plop. The first time you start it you will see the What's New box. You can close this, and see it any time later under the Help menu item on the main page.



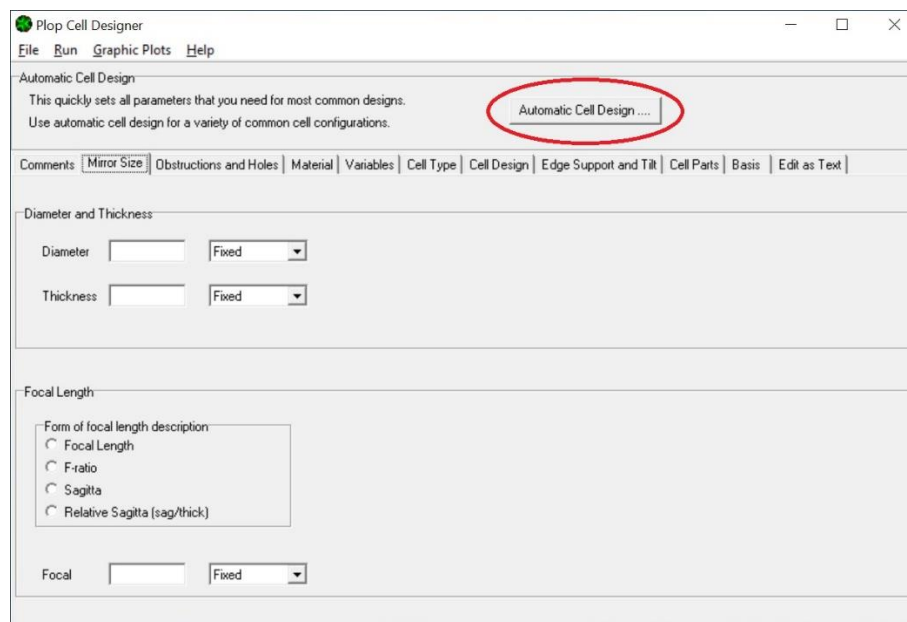
After closing this, you now see the main design form.



The screenshot shows the 'Plop Cell Designer' window. The title bar includes 'File', 'Run', 'Graphic Plots', and 'Help'. Below the title bar is a section for 'Automatic Cell Design' with a description and an 'Automatic Cell Design' button. A tabbed interface follows, with 'Mirror Size' selected. The main area contains two sections: 'Diameter and Thickness' with input fields for 'Diameter' and 'Thickness', each with a 'Fixed' dropdown; and 'Focal Length' with a radio button group for 'Form of focal length description' (Focal Length, F-ratio, Sagitta, Relative Sagitta (sag/thick)) and a 'Focal' input field with a 'Fixed' dropdown.

This form is the primary form for describing the mirror and starting other forms to run analysis or display results.

The majority of ATMs will be content with the Automatic Cell Design. This calls up a form that lets you select from a variety of commonly used cell designs.



This screenshot is identical to the previous one, but the 'Automatic Cell Design' button in the top right of the 'Automatic Cell Design' section is circled in red.

Now you will see the automatic cell design form. This lets you enter the basic parameters of mirror diameter, thickness, focal length, and size of the central hole, if any. Enter all of these parameters and click Next. In this example we'll use a typical 300mm f/5 mirror. Note that all dimensions are in mm.

Plop Automatic Cell Designer

Diameter of the mirror in millimeters: 300

Thickness of the mirror in millimeters: 50

Focal length of the mirror in millimeters: 1500

Diameter of the secondary mirror in millimeters: 60

Diameter of central hole, if any, else leave blank:

Cancel Next >>

Now you can select the type of cell that you want to use from the list. Lets use a 6 point cell which is completely adequate for this 300mm diameter mirror. There also is a check button if you want to allow the angles of the cell supports to be changed as part of the optimization. For simple cells such as 3,6,9 points, all of the angles must be multiples of 120 degrees, so leave this unchecked, but for more complex cells it is possible to allow the angles to vary. So leave it unchecked and continue.

Plop Automatic Cell Designer

Number of Points in the Cell

☐ 3 points

☒ 6 points

☐ 9 points

☐ 18 points

☐ 27 points

☐ 36 points

☐ 54 points

☐ Allow angles to vary

Select the cell style you want.

Check if you want to allow the angles to be adjusted.

<< Back Cancel Done

When you click Done, the Plop run form will appear. This is also accessed from the main form under the Run menu item. The run form contains a number of check boxes, which you should leave alone for most optimizations.

Plop Run Controls

Graphic Plots

Plop P-V Error Num Trials Z88 P-V Error

Plop RMS Error Step Size Z88 RMS Error

Refocus change in focal length

Matrix Solution Progress

Run Plop | Zernike Polynomials | Monte Carlo | Trace

☒ Update pictures as optimization progresses

☒ Refocus Error Calculation

☐ Refocus Includes Tilt in Mirror

☐ Use P-V error for optimization (almost always a bad idea)

☒ Use Basis (automatic if none specified in edit sheet)

☐ Generate Z88 input after running Plop

Z88 Run Controls

Run Z88

Run Plop Controls

Start Plop Pause Resume Abort

Just click Start Plop. Or, you can select one or more of the graphic plots from the Graphic Plots menu before you start it, and watch Plop do the optimization. It will only take a second or two for a simple cell to be run, and you will see a popup when it is done. Click it to go back to the run form.

Plop Execution Finished

Plop Execution Completed

OK

Plop Run Controls

Graphic Plots

Plop P-V Error Num Trials Z88 P-V Error

Plop RMS Error Step Size Z88 RMS Error

Refocus change in focal length

Matrix Solution Progress

Run Plop | Zernike Polynomials | Monte Carlo | Trace

☒ Update pictures as optimization progresses

☒ Refocus Error Calculation

☐ Refocus Includes Tilt in Mirror

☐ Use P-V error for optimization (almost always a bad idea)

☒ Use Basis (automatic if none specified in edit sheet)

☐ Generate Z88 input after running Plop

Z88 Run Controls

Run Z88

Run Plop Controls

Start Plop Pause Resume Abort

Error and Refocus Mirror Calculation

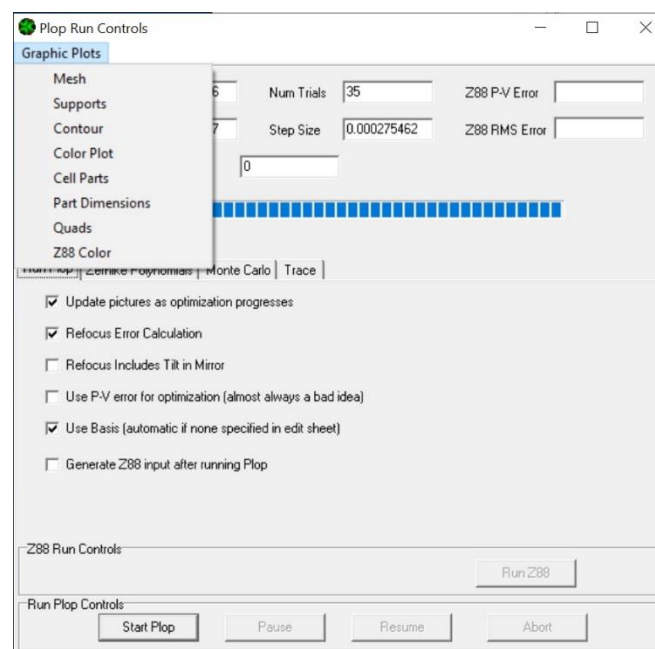
The run form now tells you the error on the mirror surface. Double this to get wavefront error. You can see that it has both RMS and P-V errors.

The one box in the run form that deserves some explanation is the Refocus Error Calculation. Refocus means that when Plop calculates the wavefront error, it will assume that the eyepiece is adjusted to best focus. This is because in many cases, especially for mirrors with a small number of supports, the mirror will sag in a way that is in part paraboloidal. The visual effects of this sag can be removed by the user adjusting the focus of the mirror slightly. Plop will take this into account when performing the analysis and optimization if this box is checked. The run form has a box that will tell you the amount of this refocus, but I removed the code to calculate it at some point while restructuring the code. This is fixed in the newest version, and if you look at the run form above you can see it has changed the focal length to 1499.9907. You don't want to include tilt in mirror unless you are running Z88.

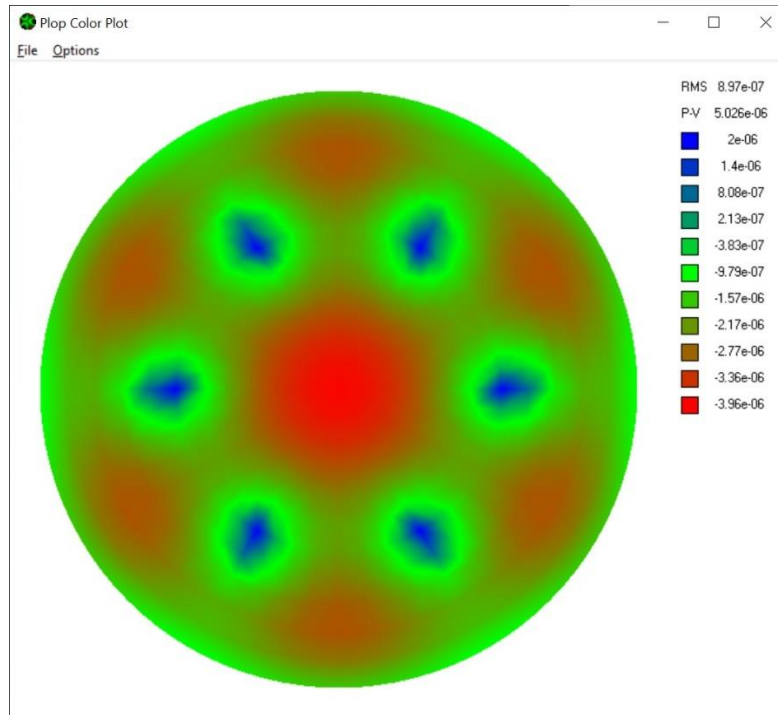
Note that Plop does not include any surface error that is obscured by the secondary mirror, and won't try to make the surface error smaller in this region. That's why the mirror cell design includes the secondary mirror size as one of its parameters.

Graphic Plots

Let's take a view of some of the plots we can see. Under the Graphic Plot menu item, there are lots of entries.

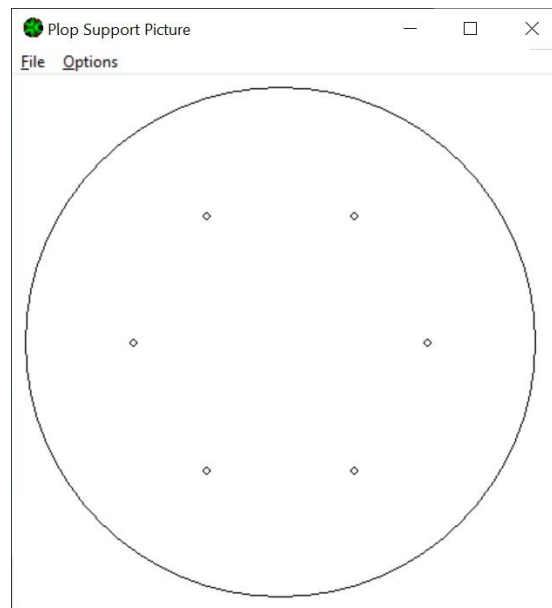


Click Color Plot to see a view of the surface deformation and a scale.

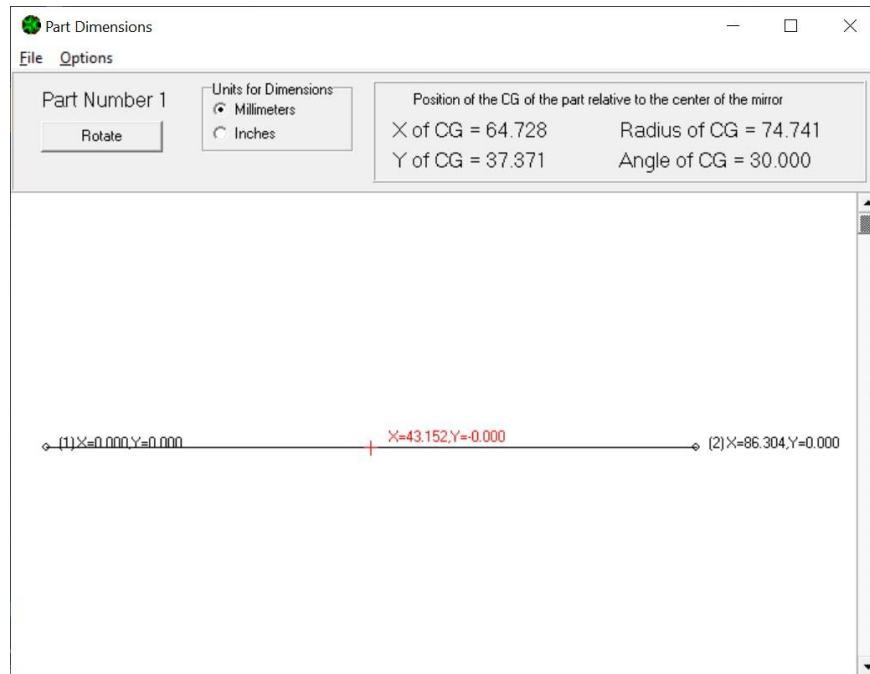


You can see how the mirror is high above the 6 support points, and low elsewhere. It also has a big depression in the center, since Plop doesn't attempt to reduce surface error where it doesn't matter.

You can see a diagram showing where the supports are located by using the Supports plot.



This is nice to look at, and helps confirm that Plop is actually modeling what you think you are trying to build, but doesn't have the detailed dimensional information you need to build the cell. You need to use the Cell Parts and Part Dimensions plots. These will show a diagram of where the parts are, with labels on them, so you can refer between the two diagrams.



For our 6 point cell, there are 3 identical parts, which are bars. The distance between the points is shown as 86.304mm, and the center support is at 43.152. The plot also tells you where the center of the support should be located, both in polar and Cartesian coordinates. Note that there is a scrollbar on the part dimension form. You can use this to scroll between the various cell parts if there is more than one.


Using Z88

Using Z88 to analyze the mirror is optional. To do this you need to check the Generate X88 Input After Running Plop on the run form. This must be done before you run Plop. We could have done that originally, but we didn't, so check the box, then run Start Plop. It won't take as long because Plop has already optimized the cell. Now click the Run Z88 box.

Plop Run Controls

Graphic Plots


Plop P-V Error	5.02596e-06	Num Trials	35	Z88 P-V Error	1.82002e-06
Plop RMS Error	8.97036e-07	Step Size	0.000275462	Z88 RMS Error	4.77935e-07
Refocus change in focal length	0				

Matrix Solution Progress 

Run Plop | Zernike Polynomials | Monte Carlo | Trace |

- ☒ Update pictures as optimization progresses
- ☒ Refocus Error Calculation
- ☐ Refocus Includes Tilt in Mirror
- ☐ Use P-V error for optimization (almost always a bad idea)
- ☒ Use Basis (automatic if none specified in edit sheet)
- ☒ Generate Z88 input after running Plop

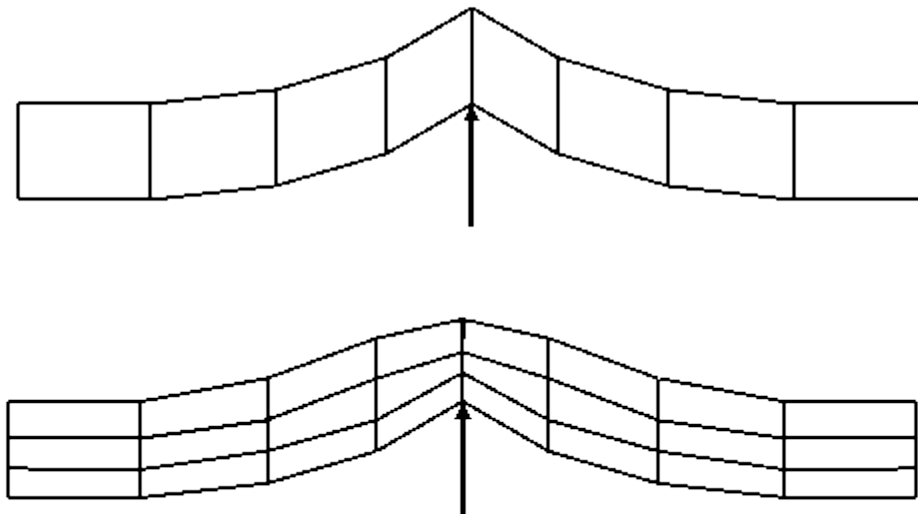
Z88 Run Controls

 Run Z88

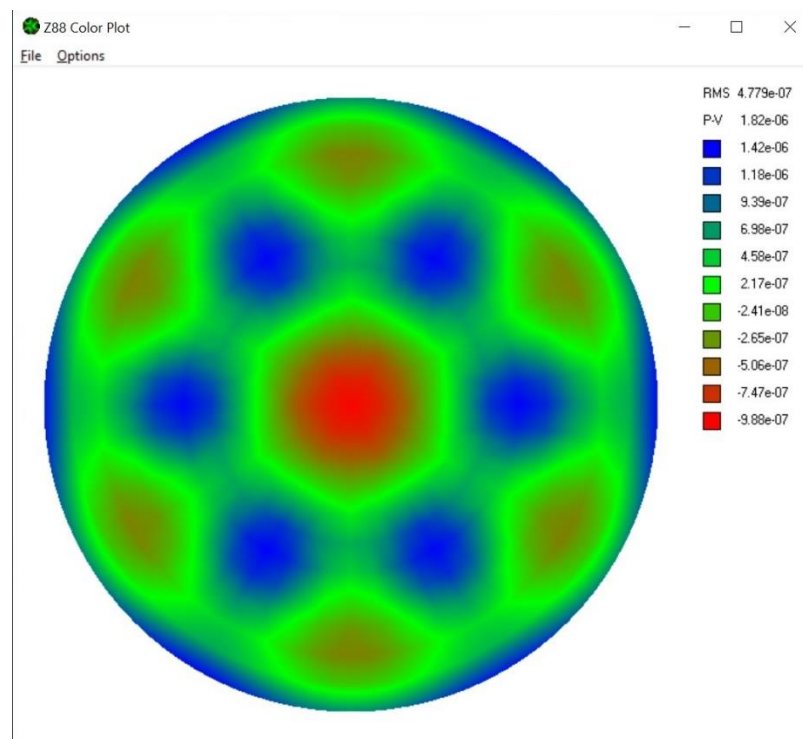
Run Plop Controls

Start Plop Pause Resume Abort

Once this is done the error boxes for Z88 will also be loaded, and you can generate a graphic plot for the Z88 results as well. For smaller cells, Z88 will usually show a smaller surface deformation than Plate, but this is also more accurate. Plate's model of the mirror does not consider any compression of the thickness. Z88 models the mirror as a set of 3D regions, and models the forces in each of the 3 dimensions. We can visualize the difference by comparing the two diagrams below, showing how a single point support deforms the mirror. The Plate model treats the mirror as having no change in the thickness, while the Z88 model softens the peak above the support point. Thus, the Z88 analysis will show a smaller deformation, and is more accurate than Plate.



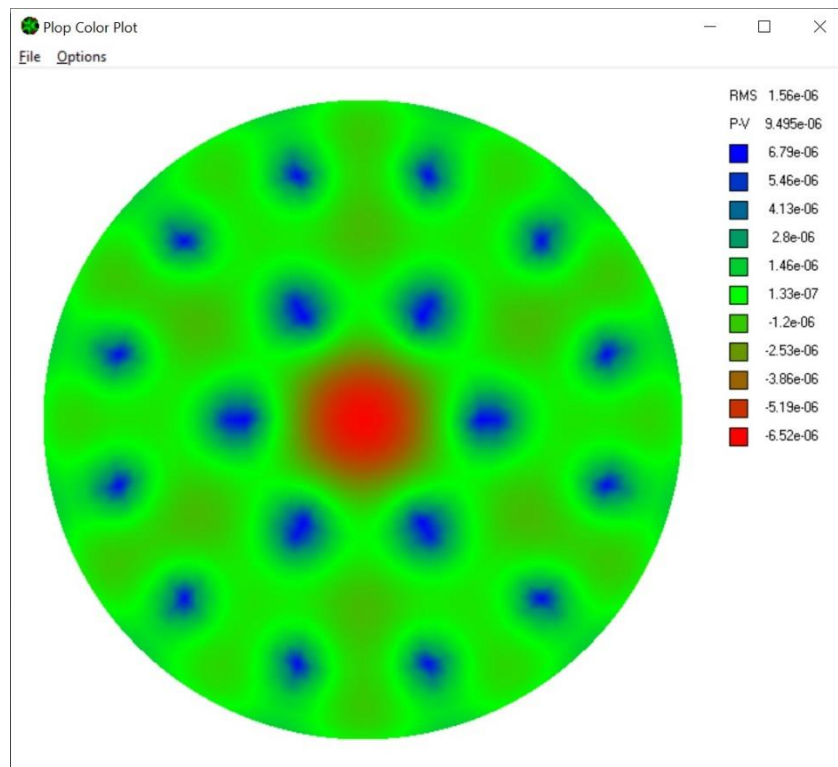
Now we take a look at the Z88 Color Plot and can visualize the softer peaks on the mirror surface.



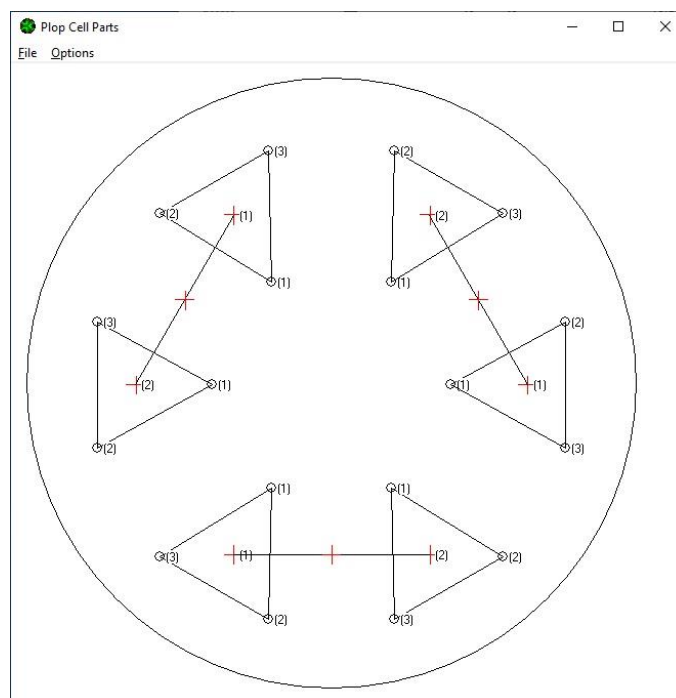
The vertical scale is different from the previous picture (you can constrain the Z scale under the Options menu), but you can see less pronounced peakiness compared to the previous color plot.

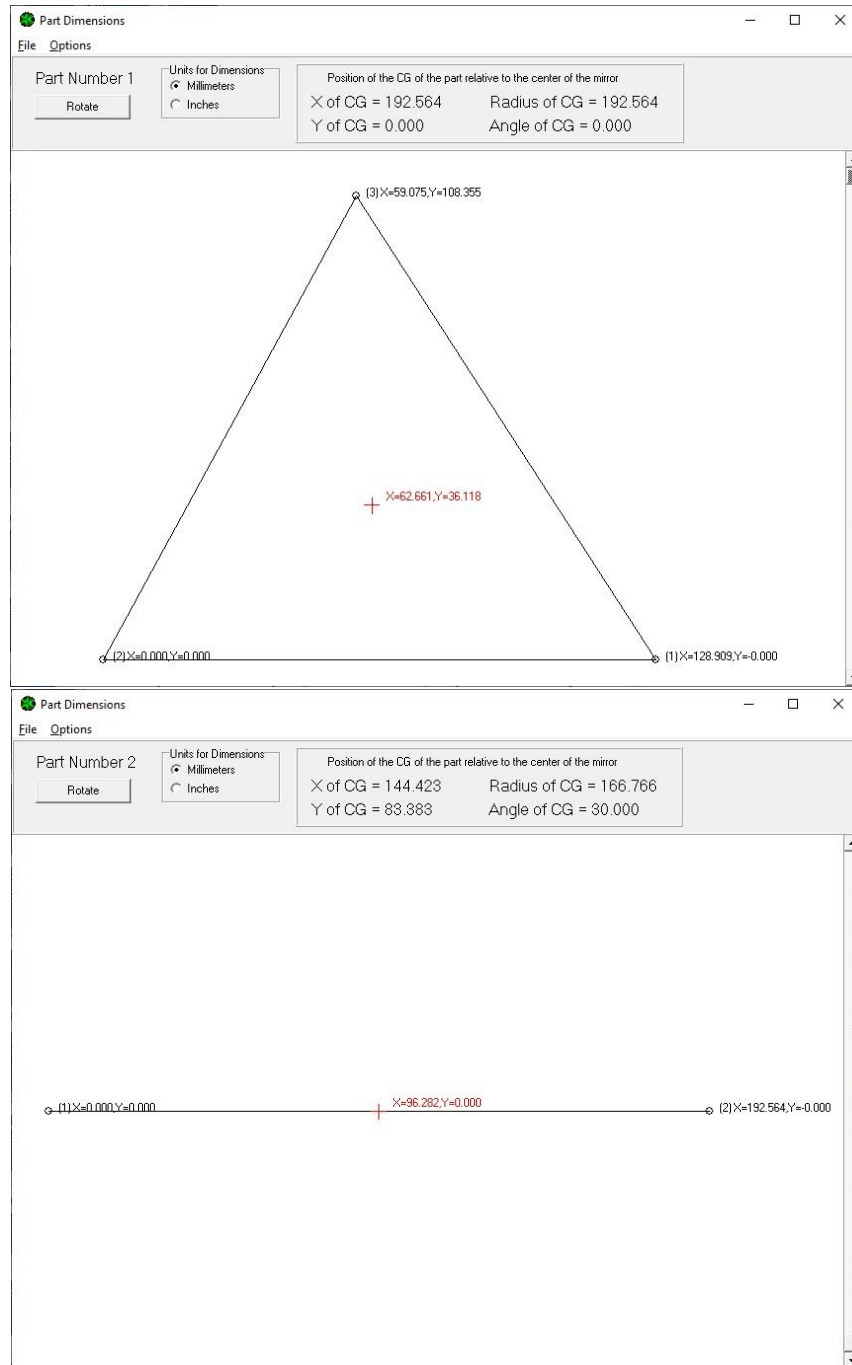
A More Complex Example

Use Automatic Cell Designer to create a 600mm diameter, 50mm thickness, 3000mm F/L mirror with 120mm secondary and 18 points. Run Plop and view the color picture.



The parts diagram plot will now show the set of 3 bars and 6 triangles. Using the part dimension plot, you can scroll to see the exact dimensions and locations of the parts. Note the labels on each of the locations to help you orient the part with the overall design shown in the part diagram plot.

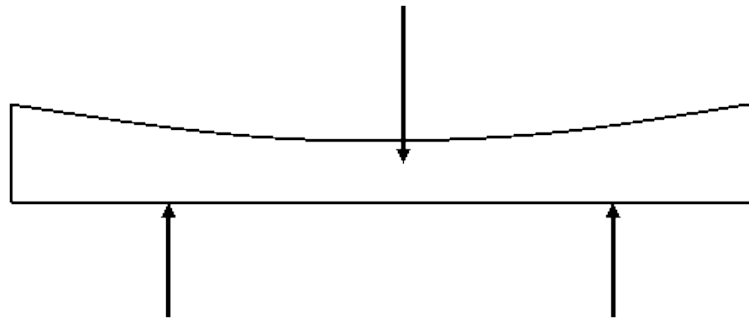




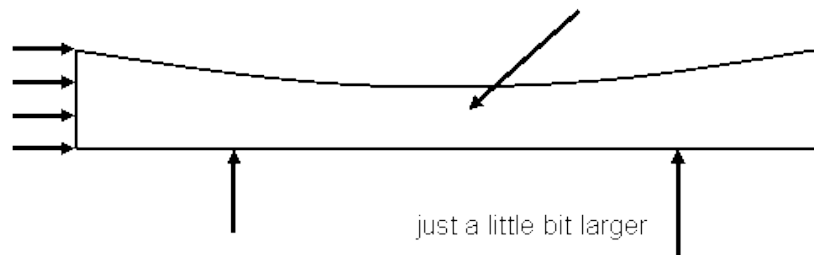
This should be enough for the vast majority of cell designers.

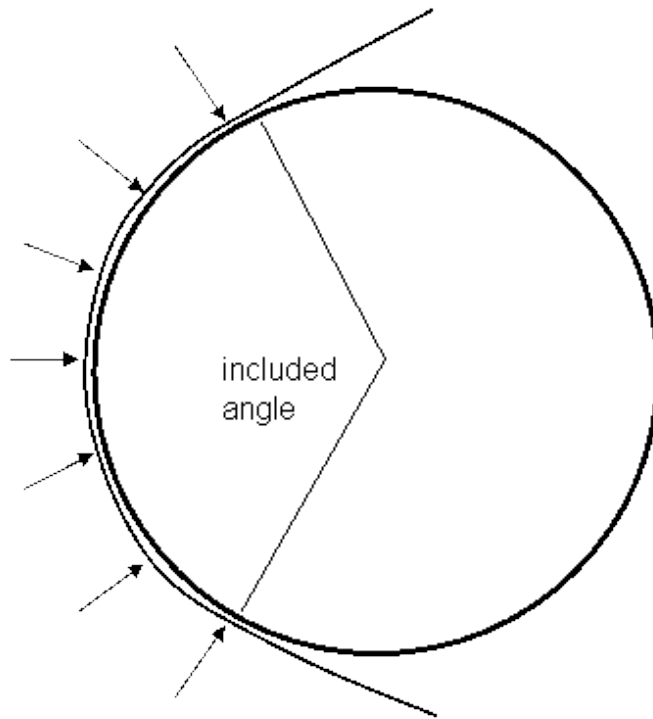
Z88 and Tilt

The next feature likely to be useful for more advanced analysis is the use of Z88 to analyze the distortion of a mirror cell when it is tilted off the vertical axis; or simply off-axis. To begin, let's look at why we would want to do the 3D analysis. The on-axis model assumes that gravitational forces run vertically through the cell.

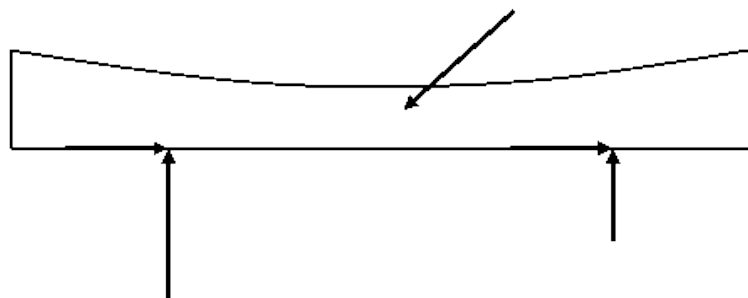


When the telescope is tilted, the forces also include some that are orthogonal to the mirror cell, and create non-uniform forces on the cell supports, and create 3D stresses in the mirror. There are two commonly used ways of supporting the mirror against transverse forces, namely gluing and a sling, and each has different effects on the mirror, shown approximately below. First, take a look at the use of a sling. It provides transverse forces across the entire thickness of the mirror, and around a large part of its circumference; up to 180 degrees.

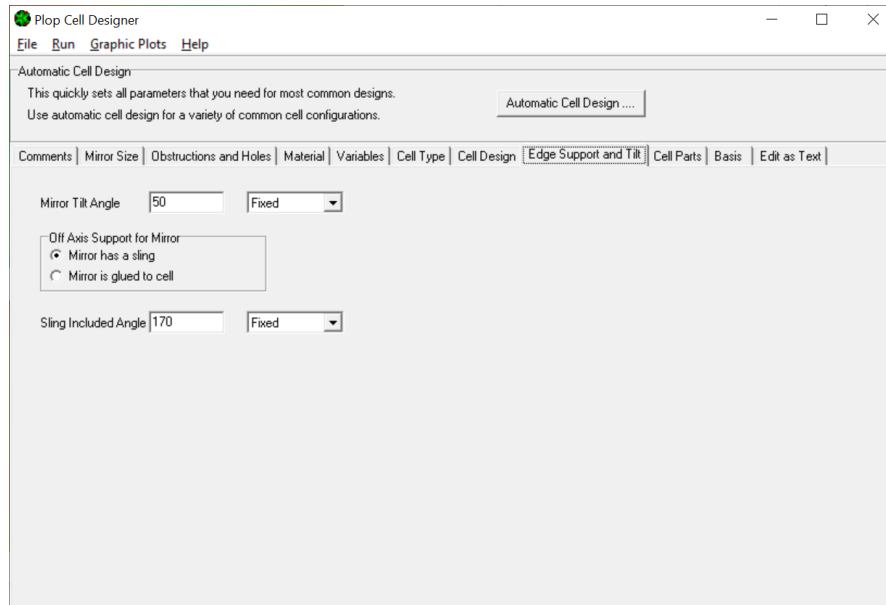




The other technique is to glue the mirror to the cell, so the supports on the cell provide the lateral forces to compensate. Note the horizontal forces at the base of the mirror.

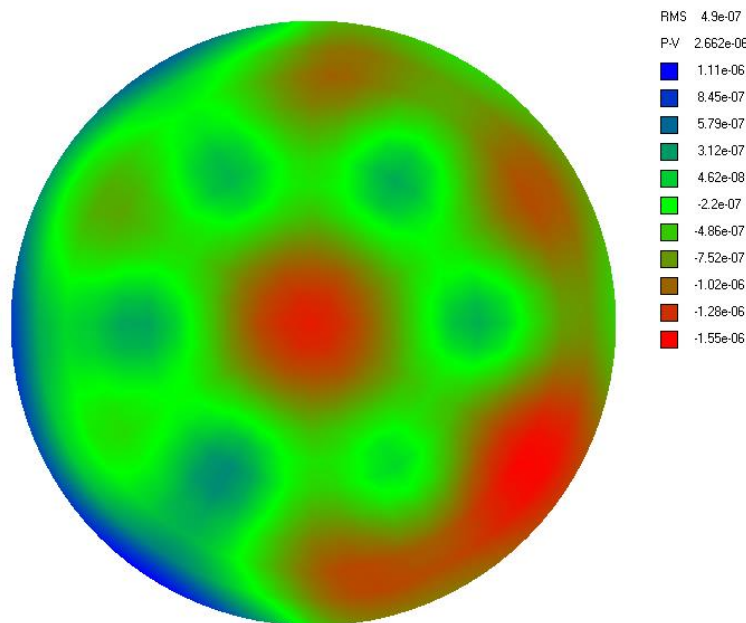


Plop can model both of these using Z88. Let's use the first example of the 300mm mirror to explore. After entering all the parameters, on the cell designer form, select the Edge Support and Tilt tab. We'll use a tilt of 50 degrees from vertical, select a sling, with a 170 degree included angle.

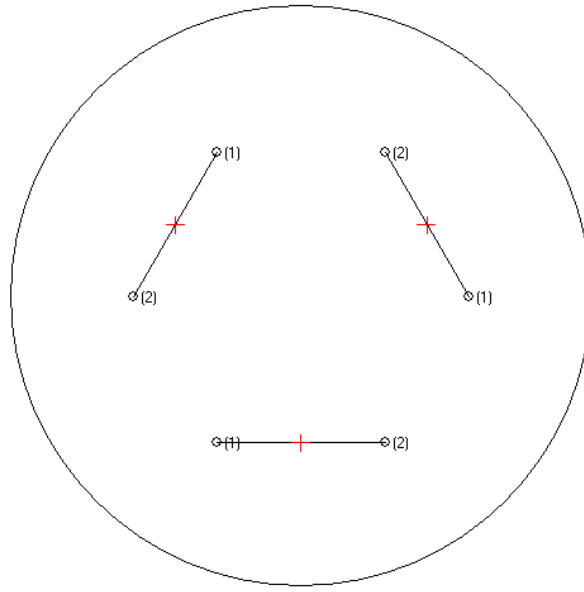


Now we can go back to the run menu. Important!!!! You must run Plop to generate the Z88 input before running Z88. This is a bad design interface, but there you go.

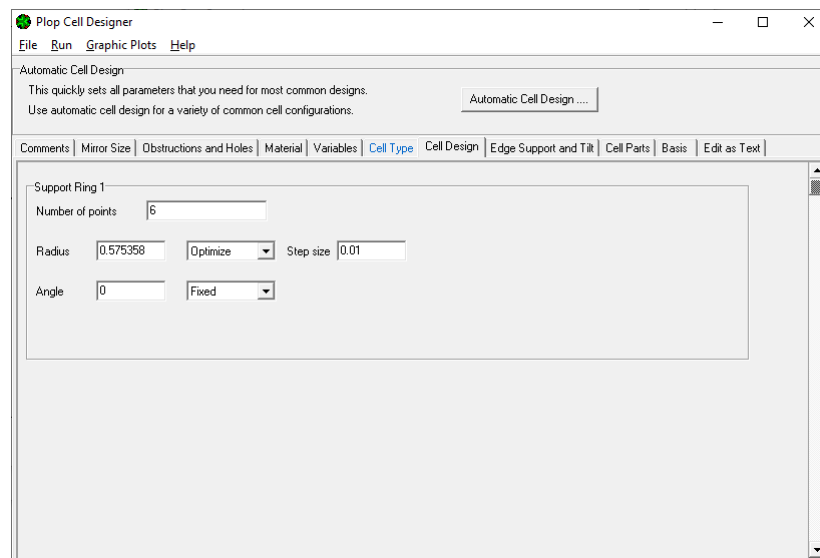
So click Run->Run Plop, and then run plop, then run Z88. Now view the Z88 color plot.



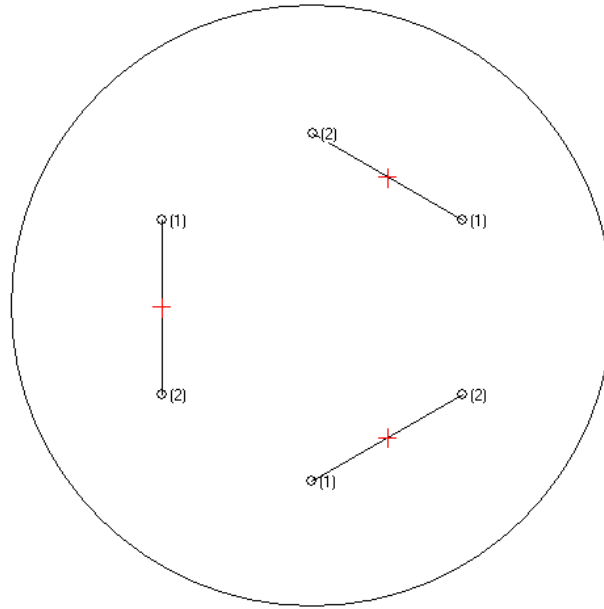
Well that's certainly different. It looks pretty incomprehensible without a little explanation. First of all, tilt in the mirror is not in what one would typically expect. Tilt is across the horizontal line in the picture, so left is low and right is high. For whatever reason, that's the way I decided to do it. Now take a look at the cell structure again.



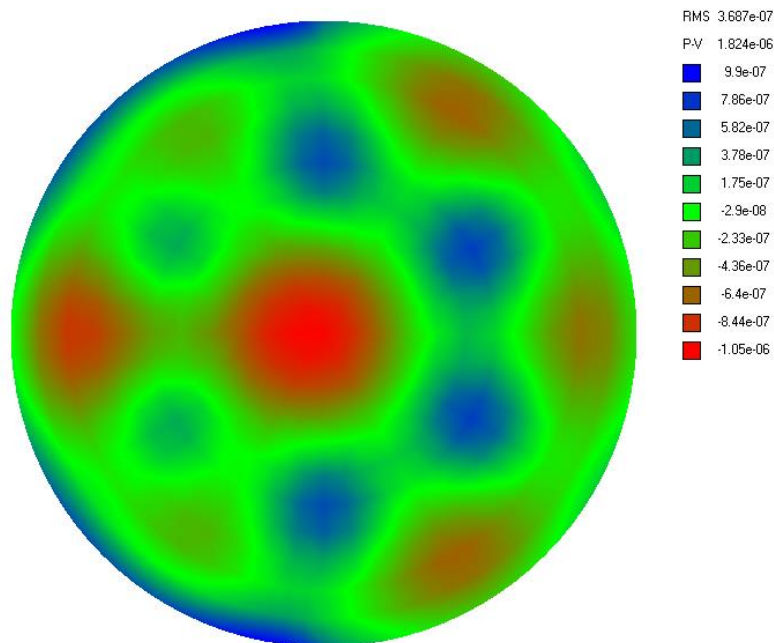
Now it is not so surprising that a left to right tilt has some strange effect on the mirror when it is tilted. We really should make the cell symmetrical about the horizontal axis to try and reduce this skewiness. To do so, we need to go into the details of the cell designer. Select the cell design page and see this.



Notice that the angle is 0. This means that the first cell support is located at angle 0. Now try changing that to 30, so the supports are located symmetrically across the horizontal axis, and run Plop and look at the cell parts again.

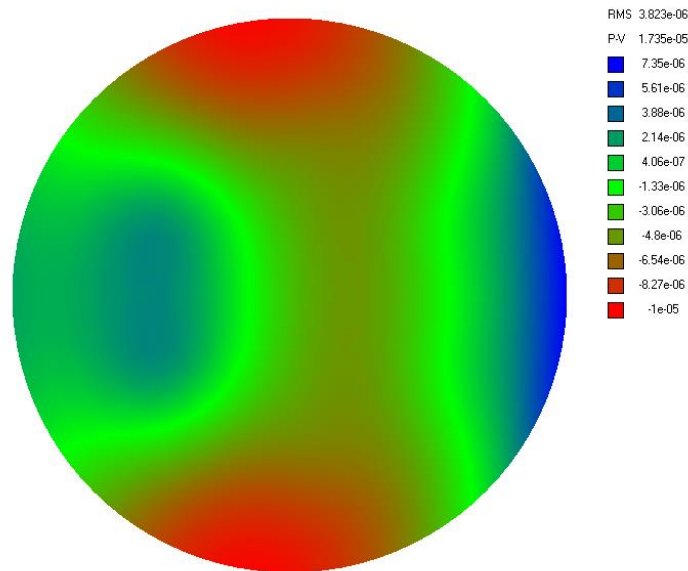


Now a left to right tilt will not have such weird effects, so run Z88 and look at the color plot.



This looks much more sensible, as well as having a much smaller error.

Now go back to the main form and Edge Support and Tilt tab, select Mirror is glued to cell, go to run form, select Refocus includes tilt on the run page, run Plop, then run Z88, and see what happens. The reason to include Refocus Includes Tilt is that Z88 can model 3D effects, and these can cause some slope deformation of the mirror when it is tilted. The tilt will cause an image shift, but since this is of no consequence to the viewer, we should remove it from the overall deformation.



Well that's interesting. Gluing the mirror to the cell has about 10X larger surface deformation.

Detailed Manual for GuiPlop

Here we will give a hopefully complete manual for Gui-Plop. Before doing so a few words of explanation about GuiPlop vs. Plop should be given. Plop was originally developed as a command line program for optimizing and performing experiments on a range of mirror cells. Experiments include being able to vary some of the parameters across a range, and seeing what happens. For example, we might want to consider a 3 point cell, and see what happens to surface error on all mirrors from 100mm to 400mm in diameter, looking at steps of 10mm. Or, we might be interested in looking at mirrors of thickness 25, 35, and 50mm. Plop lets you do these things with its specification language.

GuiPlop was developed to make the features of Plop available in a GUI based interface, with more focus on enabling an average ATM to design a cell. In this context, it doesn't really make any sense to do things like vary the mirror diameter or thickness and plot all the results. Nevertheless, GuiPlop is designed to be able to generate all of the files that Plop can understand, so there are many controls that are not very useful in the GUI context. Furthermore, Plop has a general method for varying or optimizing parameters, and to keep the interface consistent, GuiPlop uses the same interface for any parameter that can be varied in a continuous way.

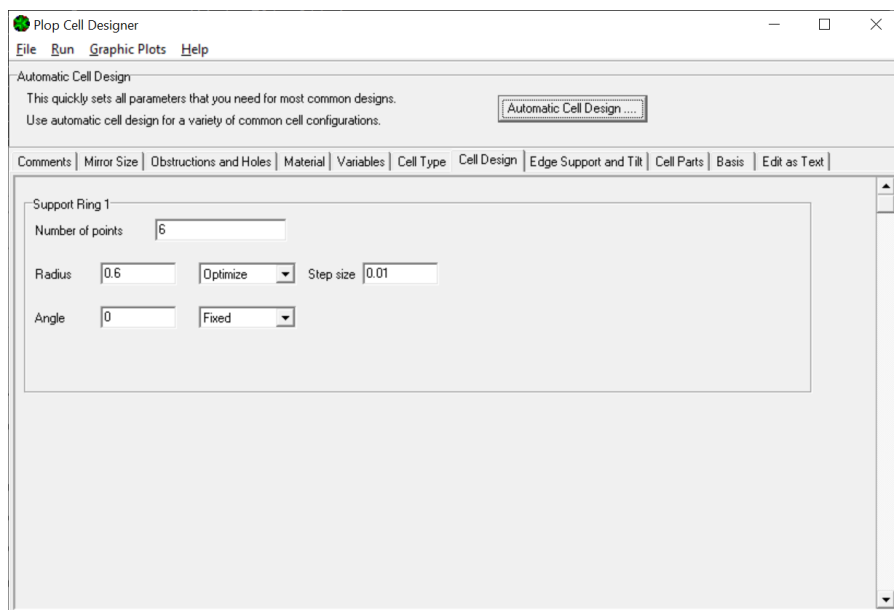
To describe the relationship between GuiPlop and Plop in a bit more detail, GuiPlop is mostly a GUI interface that generates the type of configuration data that Plop understands. You can see this at any time by clicking on the Edit As Text tab in the Cell Designer. When you run Plop, GuiPlop simply generates this file and hands it off to Plop to do the analysis or optimizations or sweeps. This is what is saved in the .gr file.

Since all continuous parameters are handled in the same way by GuiPlop, let's first be precise about what is a continuous parameter vs discrete. A continuous parameter is anything that can be varied to an arbitrarily small precision, and represented by a decimal number. So you could have a mirror that is 300.2432mm in diameter, 63.2423mm thick, and the cell supports could be at radius 76.345mm. All

continuous parameters are handled in the same way, so we will describe the GUI interface for these below.

Discrete parameters are necessarily integers, such as the number of points in your cell. You can have 6 points or even 7 if you are peculiar enough, but there's no way you can have 6.234 points. All discrete parameters must have a specific numerical value.

So let's take a look at how continuous parameters are specified in GuiPlop. This will use the basic 6 point cell we described first. Taking a look at the Cell Design tab, we will see the following.



Here there are 2 parameters describing the cell. Every continuous parameter will have a parameter mode menu box to the right, listing Fixed, Optimize, Scan Range, Scan Set, and Monte Carlo. Depending on which of these is selected, other edit boxes may appear.

If the parameter mode is Fixed, that value is the only one that will be used.

If the parameter mode is Optimize, Plop will adjust the value of it to minimize the surface error. A Step Size edit box will also be present. Here you should enter a reasonable size of variation for the Plop optimizer to attempt to adjust to optimize the mirror. Plop will start by adjusting the parameter by this amount, and increase or decrease it as it proceeds, so just try and get something you think is sensible to have some modest effect.

In this example, the Radius parameter is set to be optimized, since we want to find the best radius for the support points, but the Angle is set to fixed, because it is meaningless to rotate the support points in the cell.

For examples of scans, we can look at [examples/example5/example5.gr](#). This is an experiment to vary the diameter of the mirror, and try a set of different thicknesses. Choosing Scan range on a parameter gives you three more boxes to specify the starting value, end value, and number of steps to try. Plop knows that you would forget about the endpoint, so the number of steps actually means the fraction of distance

between start and end. So for from=100,end=400,steps=20, the step size will actually be $(400-100)/20=15$, and Plop will sample diameters 100,115,130,...,400 which is 21 different values.

For the Scan set mode, you can give a list of specific values separated by spaces. Plop will run each of these specific values.

When there are multiple parameters with Scan set or Scan value, Plop will perform the analysis or optimization for all combinations. So for the example below, we have $21 * 6 = 126$ different scenarios that will be run.

Plop Cell Designer

File Run Graphic Plots Help

Automatic Cell Design

This quickly sets all parameters that you need for most common designs.
Use automatic cell design for a variety of common cell configurations.

Automatic Cell Design ...

Comments Mirror Size Obstructions and Holes Material Variables Cell Type Cell Design Edge Support and Tilt Cell Parts Basis Edit as Text

Diameter and Thickness

Diameter 317.5 Scan range From 100 To 400 Num steps 20

Thickness 54 Scan set Values 22.22 25.4 31.75 38.1 44.45 54

Focal Length

Form of focal length description

☐ Focal Length

☒ F-ratio

☐ Sagitta

☐ Relative Sagitta (sag/thick)

Focal 5 Fixed

Lastly, a parameter can have Monte Carlo mode selected. In this case you can enter the variation, and Plop will perform the analysis randomly varying the parameter by up to this amount (rectangular PDF with this value of max absolute variation.)

With these preliminaries over, the next sections will explain the various pages of the cell designer; although they are in order of how you would probably like to understand them, this isn't exactly the order that you would need to use them in the case of a complicated cell design.

Cell Designer Comments

Well, this is pretty obvious. Say something that describes your cell.

Cell Designer Mirror Size

These are a set of fairly obvious parameters describing the diameter, thickness, and focal length of the mirror. As noted above, since these are all continuous parameters, you can describe them using any of the options, but are most likely to want to keep them fixed.

The mirror focal length can be measured using any one of focal length, f-ratio, sagitta, or relative sagitta, which is the sagitta divided by the thickness of the mirror. The last two of these are not likely to be useful for designing a specific cell, more so for studies.

The screenshot shows the 'Plop Cell Designer' application window. The title bar includes the application name and standard window controls. The menu bar contains 'File', 'Run', 'Graphic Plots', and 'Help'. The main interface is divided into several sections. At the top, there's a section for 'Automatic Cell Design' with a description and an 'Automatic Cell Design' button. Below this is a tabbed interface with tabs for 'Comments', 'Mirror Size', 'Obstructions and Holes', 'Material', 'Variables', 'Cell Type', 'Cell Design', 'Edge Support and Tilt', 'Cell Parts', 'Basis', and 'Edit as Text'. The 'Cell Design' tab is currently active. It contains two main sections: 'Diameter and Thickness' and 'Focal Length'. In the 'Diameter and Thickness' section, 'Diameter' is set to 300 and 'Thickness' is set to 50, both with 'Fixed' dropdown menus. The 'Focal Length' section has a 'Form of focal length description' group box with four radio buttons: 'Focal Length' (selected), 'F-ratio', 'Sagitta', and 'Relative Sagitta (sag/thick)'. Below this, the 'Focal' value is set to 1500 with a 'Fixed' dropdown menu.

Cell Designer Obstructions and Holes

The obstruction part of this tab describes a visual obstruction such as a secondary mirror where we don't care about the mirror surface error because it is blocked out of the visual field. This can be described in any of the parameters radius of the obstruction, diameter of the obstruction, or relative radius.

In contrast, the Central hole describes an actual cut out in the center of the mirror, and affects the structural characteristics of the mirror, so it might have a different surface error. Taking the original example with a 60mm cutout we see this.

Plop Cell Designer

File Run Graphic Plots Help

Automatic Cell Design

This quickly sets all parameters that you need for most common designs.
Use automatic cell design for a variety of common cell configurations.

Automatic Cell Design

Comments | Mirror Size | Obstructions and Holes | Material | Variables | Cell Type | Cell Design | Edge Support and Tilt | Cell Parts | Basis | Edit as Text

Central Obstruction due to Secondary

Obstruction Description

☐ Obstruction Radius

☒ Obstruction Diameter

☐ Relative Obstruction Radius

Obstruction 60 Fixed

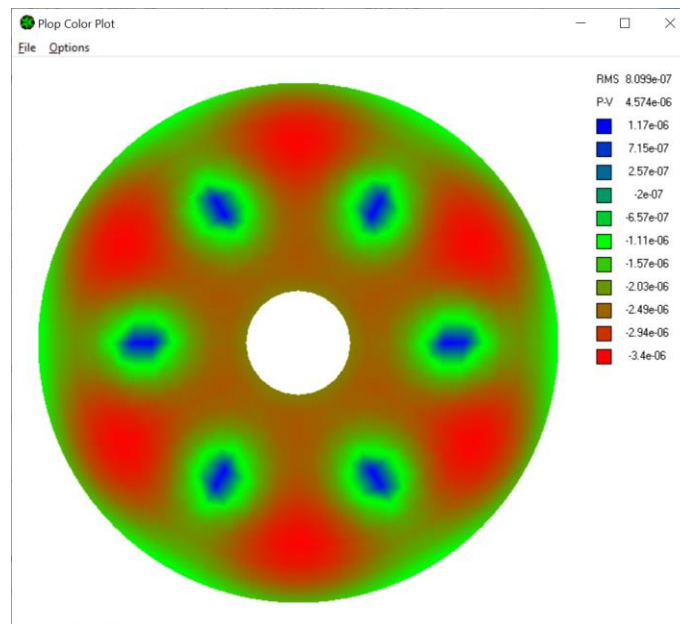
Central Hole in Mirror

Hole Description

☒ Hole Diameter

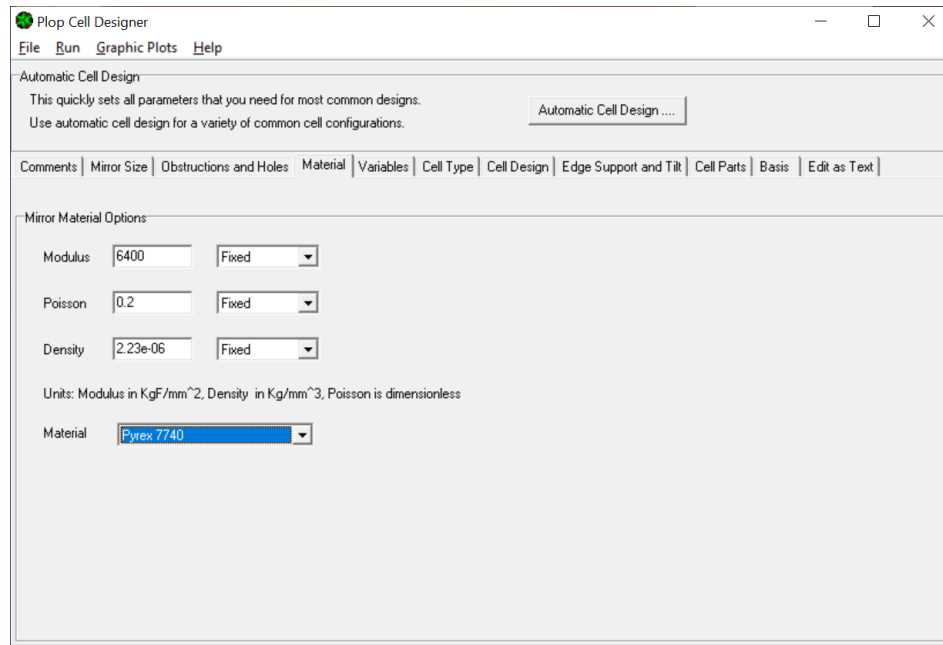
☐ Relative Hole Diameter

Hole 60 Fixed



Cell Designer Material

Here you select the mechanical properties of the mirror material. As for any parameter, you can vary or optimize them, which might be an interesting experiment, but few people have the luxury of optimizing the exact properties of the mirror material. This offers a list of common glasses, and the numerical values will be blank until you select something from the drop down Material list.



Cell Designer Variables (but not just yet)

The next tab is Variables, which should be completed before you go onto the next tabs. However, this is somewhat complicated and most users won't need to touch this. So we will deal with the rest of the tabs before coming back to this one.

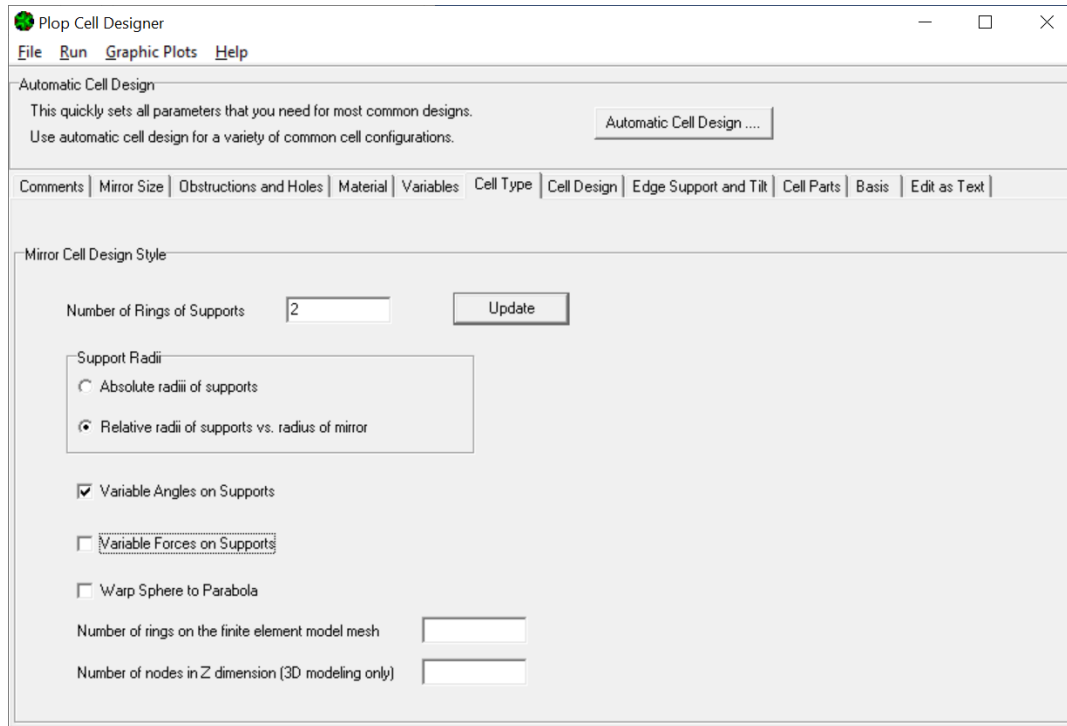
Cell Designer Cell Type

The next few tabs deal with the design of the set of cell supports. A few words of overview will be provided.

Plop views a mirror cell as a set of supports providing some force to the mirror. The basic unit of the mirror supports is a ring. A ring is a collection of supports that are all at the same radius, and distributed at equal angles around the mirror. For example, a ring could contain 3 supports at a radius of 60mm, in which case they would be at 120 degrees from each other. On this tab, you must enter the number of distinct rings that the cell contains. However, each ring can have a different angle at which the points start. For simple designs this will just be the number of distinct radii at which the points are located. For more advanced designs, it might be necessary to have more rings than actual radii, because the angles might be different between some of the points at the same radius. We will return to this later.

Let's do an 18 point cell like earlier, with one ring of 6 points and one ring of 12 points. The Number of Rings of Supports needs to be 2. You need to click Update when after changing this.

A ring of supports must contain at least 3 supports. They will be numbered from 1 to N for N supports. This is important to remember, because when describing the parts, it will be necessary to refer to a particular support point in a ring.



Next, the radii can be specified either as absolute values or a fraction of the mirror radius.

The next few check boxes let you specify whether you want to be able to specify the angle of the first point. The first^h point will be located at the specified angle on the next tab, and each point is located 360/N degrees counterclockwise from the previous. The angle will be 0 if you don't specify it. If there is more than 1 ring, you probably want to check this so you can have a different angle between the various rings.

Next is Variable Forces on Supports. Checking this allows you to provide a different force on each support ring. Most designs have the same force on each support point, but if you want to make the forces different, check this.

Warp Sphere to Parabola tells Plop to assume that the shape of the mirror is actually spherical, and it should adjust the support forces to try and warp the mirror into a parabolical surface. This is not an entirely useful option.

The Number of Rings controls the size of pieces that the FEM software breaks the mirror into. A rule of thumb is diameter/thickness*2. For our 18 point cell we will select 24. More will have better precision but there is no real point increasing beyond this. The Number of Nodes in Z dimension controls how many layers the mirror is sliced into for Z88, and 4 or 5 is generally adequate, but this can be left blank if you are not using Z88.

Cell Design Tab

This specifies all of the information about all of the supports rings. You can scroll down between the various rings if there are more than 2.

Each ring needs to specify how many points on the ring, and what radius they are at, as well as the angle of point 0. Take a reasonable guess of radius at .5 and .8, since we specified Relative Radius on the Cell Type tab. For the simple 18 point cell, the inner ring is at angle 0, and we want a pair of points on the outer ring evenly distributed on either side at equal angles. Since there are 12 points, they are 30 degrees apart, so want +/- 15 degrees. Set the angle to 15 degrees. We also want to optimize both so set to optimize with a step size of 0.01.

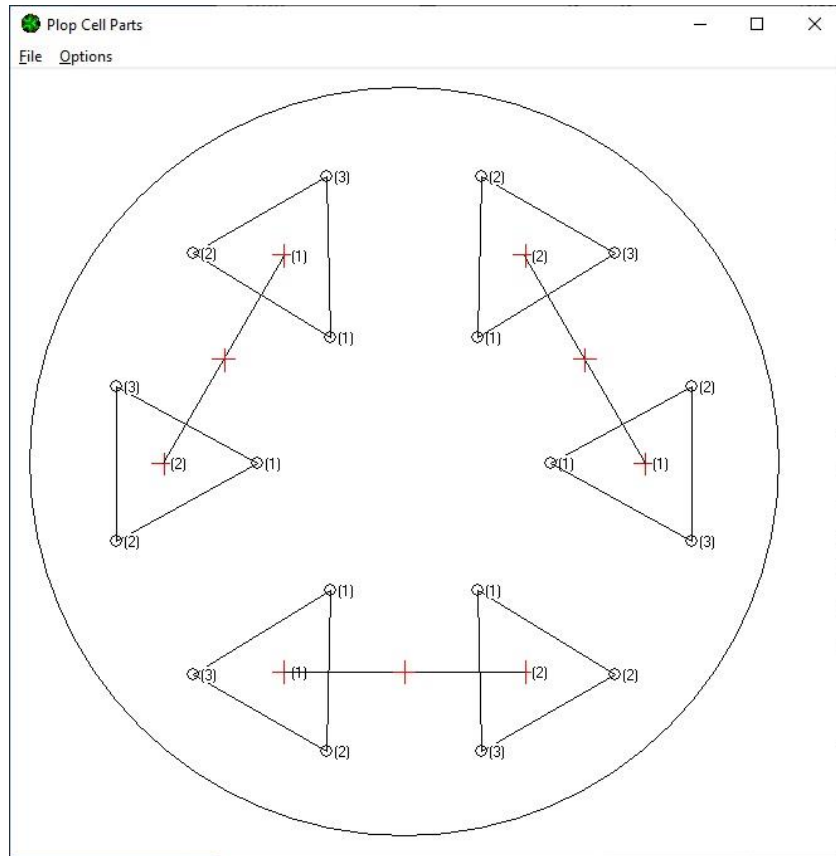
This completes the cell design. It is only necessary to specify the locations of the support points and their forces in order to completely determine how the mirror will be supported. While Plop can compute the cell part dimensions, this is purely an aid to the designer, since the mirror deformation is determined only by the forces, not how the cell is designed to provide them. So we can now proceed to run Plop as before. On the Cell Design form, you can click the Run menu item and run Plop.

Edge Support and Tilt

Here you can specify if the mirror is tilted off axis by setting the Mirror Tilt angle. See the description in the example above.

Cell Parts

This tab lets you define the geometry of parts. First, enter the number of parts. Like support rings, parts will be described in terms of a number of identical components around a circle, so the number of parts is only the number with a unique geometry. For the 18 point cell, there are two types: (1) triangles that support one point on the inner ring and two on the outer ring, and (2) bars that support a pair of triangles. So enter 2 in the number of parts. The diagram below shows the plan for the parts.



Plop understands only two types of components: triangles and bars. On each component, you can select which type it is in the drop down box.

Starting with the triangles, there are 6 of them, so this should be entered in the Quantity edit box. Plop will understand that this many identical copies are created around a circle. Once triangle is selected, there will be 3 lines to specify what the corners of the triangle are supporting. Any support can support either a point on the mirror, as entered on the Cell Design sheet, or another part. The triangles are supporting mirror points, so set the support to Support Ring for all 3. The edit boxes now let you select which support ring and which point on that ring is supported. Taking the first triangle, it uses point 1 on ring 1, and point 1 on ring 2, and point 12 on ring 2. Since there are 6 of them, and 6 points on the ring 1, but 12 on the ring 2, Plop knows to create the 6 copies of the part using a spacing of 1 point per triangle on the ring 1, but 2 points on ring 2.

Proceeding to the next level of the supports, scroll down and set the second part to a bar. Here, there are 3 bars, and looking at the first one, it supports a Part Center which is the triangle (part number 1) and at position 1, as well as another Part Center, triangle (part number 1) at position 2.

This allows a cell to be described as a stack of parts, each of which can support a point on the mirror, or the center of gravity of another part.

Basis Tab

You probably do not want to touch this.

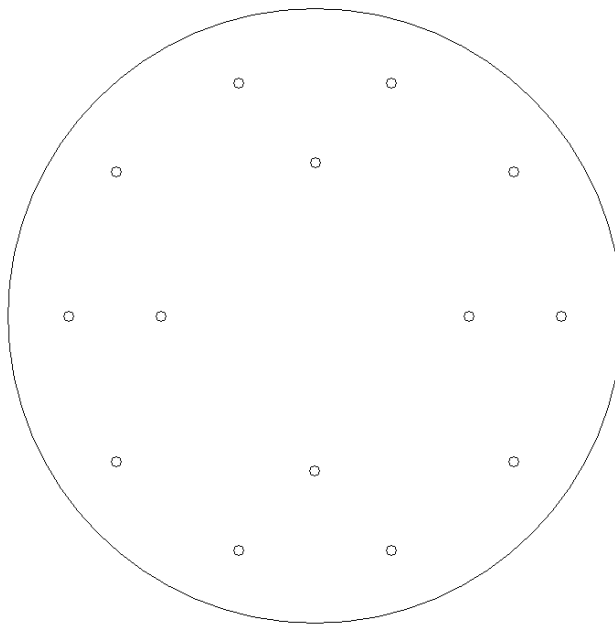
Plop speeds up calculations by breaking the mirror cell supports into a set of simpler rings, and evaluating the mirror surface error for each possible radius on the FEM mesh. This set of simple rings is called the Basis. Then after computing this, it can find the surface error for any combination of radii by summing up a linear combination of the basis rings at various rotations.

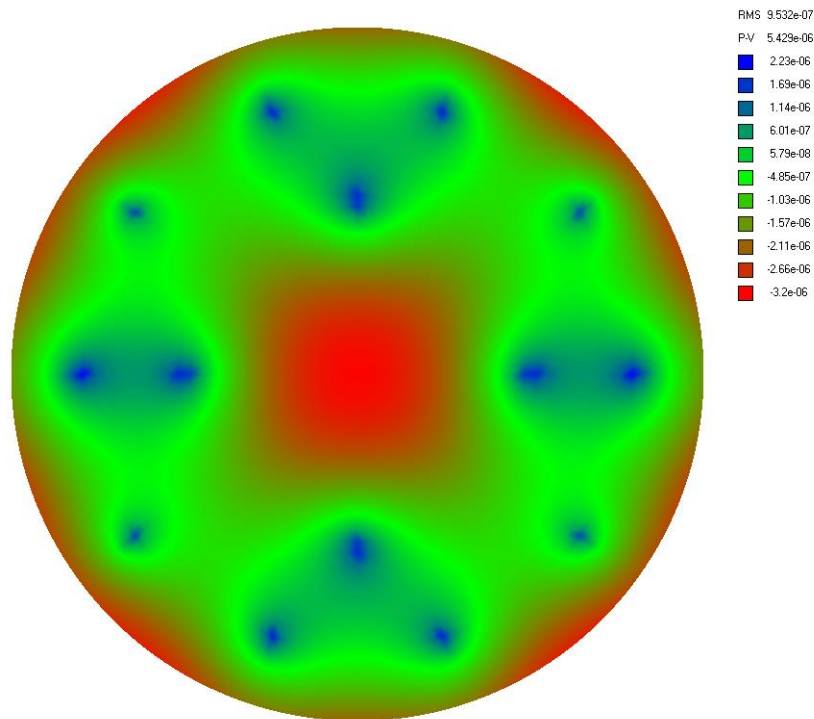
For our 18 point example, the simplest way to do this is to consider a set of 3 supports on a ring. 2 copies of this rotated by 0 and 60 degrees can model a 6 point ring, and 4 copies at 0, 30, 60, and 90 degrees can model the 12 point ring.

Plop will do this for you automatically, but only in a very simple way where all the support rings have a common divisor.

Lets take a look at a weird cell with 4 points on the ring 1, and 10 points on ring 2. We see that there is no common divisor at least size 3 (which is needed to support the mirror in a stable way.) So it is necessary to manually create the basis using suitable divisors of 4 and 10, for which 4 and 5 are suitable. Therefore the Number of Basis Rings needs to be set to 2, Update, and enter 4 and 5 in the Number of Points. I can't remember what the First Mesh Ring is for, probably to do with holes in the center.

Plop will now run properly and we can see the fairly peculiar results.





Using Variables

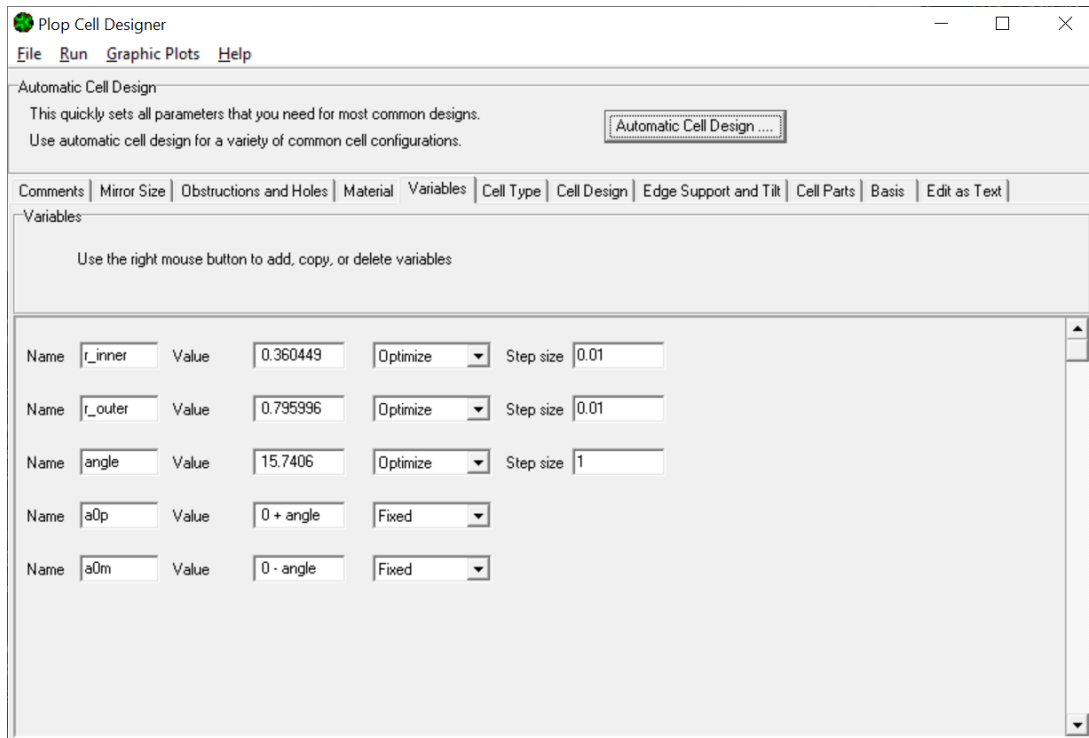
To motivate variables, let's consider a variation of the 18 point cell. In the basic design, the outer ring of 12 supports are uniformly located at multiples of 30 degrees, starting at 15 degrees, so they are symmetrically located around the inner ring of points. Perhaps this is not optimal, and the triangles should be wider or narrower. There is no way to do this with a uniform spacing of 30 degrees. However, supposed we considered the outer ring of supports as two sets of 6 points, which would each have a spacing of 60 degrees. If we can constrain both of the rings to have the same radius, but one starts at some angle a and the other at the angle $-a$ then they would still be equilateral triangles with two points on the outer ring and one on the inner.

To do this we use variables. The easiest way to understand this is to use Automatic Cell Design, select an 18 point cell, and check Allow Angles to Vary, which will select an 18 point cell as described above. Now go to the Variables tab.

The variables tab allows you to add, insert, duplicate, delete, copy, or paste variables using the right mouse button.

A variable is a parameter like any other, so it can be a fixed or optimize value, or any of the scans. Since the variables are the underlying parameters that control the cell design, these should have the desired attributes for the analysis or optimization.

The 18 point example has 3 fundamental parameters which are r_{inner} , specifying the radius of the inner 6 supports, r_{outer} specifying the radius of the outer 12, and angle which specifies the starting angle in + and - for the pairs of points on the outer ring.



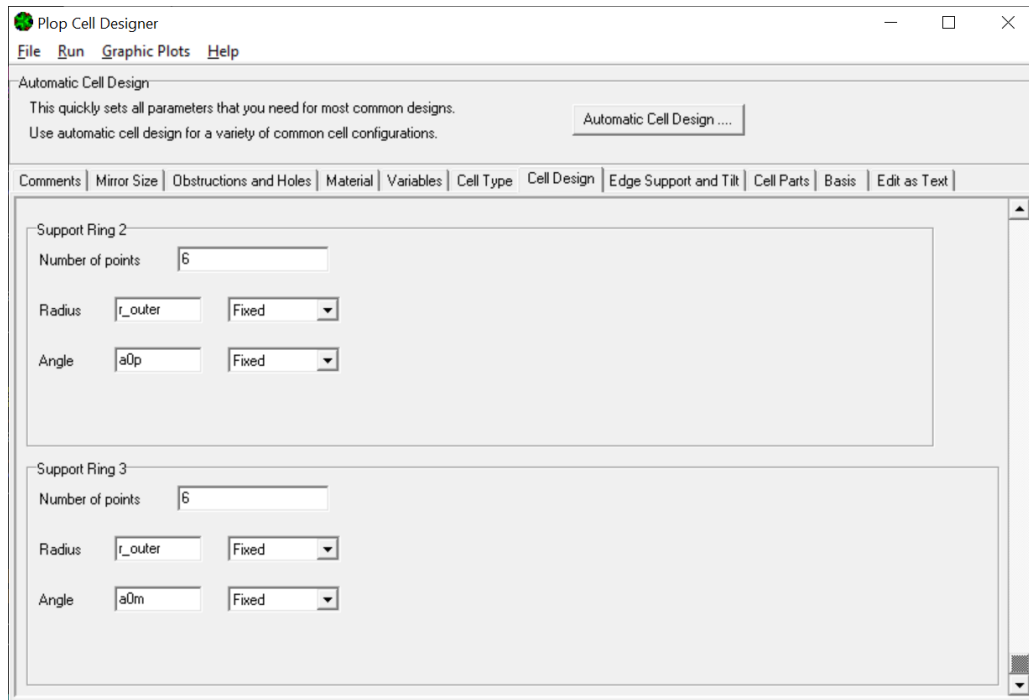
Next, we need to define variables for any mathematical expression that is needed to define the rest of the cell design. It is possible to use variables instead of actual numerical values for any of the Value entries in any edit box, but mathematical formulas can only appear in the definition of variables. In this case we will be very verbose and define a0p and a0m as +angle and −angle respectively.

Mathematical expressions in GuiPlop are very simple and can either be a number, a variable, or a simple mathematical expression $op1\ op\ op2$, where $op1$ and $op2$ are numbers or variables, and op is one of +, -, *, /. Building a complicated expression will therefore take several intermediate variables to express the desired formula. The operands and operator must be separated by blank characters. This really is a primitive language, but adequate for getting things done.

Also note that variables that are expressions must be Fixed, since they are dependent on the expression, and not independent values.

Turning to the Cell Design tab, the cell can now be expressed using these variables. The inner ring is now defined to have r_inner as its radius, and there are two outer rings with 6 supports starting at a0p and a0m respectively. Since the cell design is using variables, these must be specified as Fixed since they are not independent variables.

Anything that is defined as a function of variables must be Fixed. Independent variables can either be fixed or any of the choices that are available.



Note that the Cell Parts also need to be defined differently from the basic 18 point, since there are now three rings and the triangles include one support from each of the three.

Edit As Text

As mentioned earlier, GuiPlop simply creates a text file that Plop can read. The Edit as Text tab can show this.

Plop and GuiPlop Code

Plop and GuiPlop are two separate executables. Code for Plop is in the plop folder, and for GuiPlop in the gui_plop folder below that. Plop is designed to be compiled as a standalone and has a makefile. GuiPlop requires C++ Builder and includes project files for C++ Builder version 5. This is about 20 years old. Porting to the newest version is not difficult except for the massive pain of dealing with UTF strings.

Plop User's Manual

This is horribly out of date but included for completeness.

Plop is a program written by David Lewis and Toshimi Taki for the design and optimization of mirror cells. Plop is an abbreviation of Plate Optimizer. Plate is a tool written by Toshimi Taki that uses the finite element method (FEM) to analyze the deformation of plates under loads. Toshimi published his results in Sky and Telescope in April, 1996. Plate was written in about 800 lines of Fortran, and required that the user manually design the mesh to approximate the mirror as a set of small triangular pieces. Plate's output

is also a table of numerical values giving the deformation of each point. This numerical input and output makes it difficult to prepare the input, and requires further analysis of the output of the program.

David Lewis began with Toshimi's Plate code, and expanded around it into Plop. Plop adds several features to extend the range of functionality and increase the ease of use, so that a designer can quickly test or optimize a cell design without knowing the details of FEM. In particular, Plop includes:

Automatic generation of the mesh from a few parameters, such as diameter, thickness, focal length, and the location of the mirror supports.

Accurate error calculation of the deformation of the mirror, including the ability of the user to refocus the telescope to its best focus.

Faster numerical methods, in particular the use of sparse matrix methods, and generation of the mesh in a way that reduces matrix fillins.

Graphical output of the mirror deformation as either a contour or color plot. This includes .gif output thanks to Thomas Boutell (see acknowledgment at bottom.)

Scanning a set of parameters across a range of values.

Automatic optimization of a set of parameters to minimize deformation.

Advanced optimization using a basis set of deformations to determine the error of a given configuration as a linear sum of the basis. This feature was inspired by reading Luc Arnold's paper. Luc actually uses analytical models, but his paper inspired the idea of optimization, and decomposing the problem into a linearly weighted sum of a basis set.

Description of the design variables using mathematical formulae, for example to express symmetrical designs.

Monte Carlo analysis to test for sensitivity of design to construction tolerances.

The rest of this document describes how to run Plop, and the kinds of information that you can give Plop to control its operation.

Plop requires about 8MB to run moderate size problems, and 32MB for larger problems. Very large problems have been run using up to 100MB. Plop dynamically allocates most memory, but has an upper bound on the number of points in the mesh. The default is 2000 points, and can be adjusted using the **-a** flag (see below.)

In addition to Plop, which is an integrated package, there are three separate programs that, called **grid_gen**, **plate**, and **plot**, that can perform the various functions independently. Grid_gen will generate a mesh. Plate is a modification of Toshimi Taki's code, and performs the finite element method, but uses sparse matrix methods to speed up the computation. Plot will produce a graphical output of the results. These are not documented here.

New in 1.3

Version 1.3 adds variables, Monte Carlo analysis, a 5 times speedup for basis analysis, and support for mirrors with cored centers.

Basics of FEM

In order to understand what Plop does, and in particular how to use some of the more advanced features, it is useful to have a basic idea of how FEM is used to analyze a mirror cell. The deformation of the mirror can be computed by numerically solving a partial differential equation. The differential equation describes the deformation of each point on the mirror's surface as a function of the thickness, stiffness, mass, and supports. To solve this, it is necessary to break up the mirror into a set of small segments, each of which is a triangle. This set of triangles is referred to as the mesh. Each triangle is defined by the location of the three points at its corners. We also need to compute the weight of each segment. The mirror cell supports provide a set of forces at some of the triangle corner points. In order to precisely model the configuration, it is necessary to make sure that each support coincides exactly with one of the points associated with some triangles. This task, called mesh generation, is the first part of Plop's job, and can be performed by a separate program called `grid_gen`.

Here is an example mesh generated by Plop for a typical support. This is for a 9-point cell. Each of the support points is circled. Note how Plop tries to keep the size of each of the mesh triangles approximately the same throughout the entire mesh.

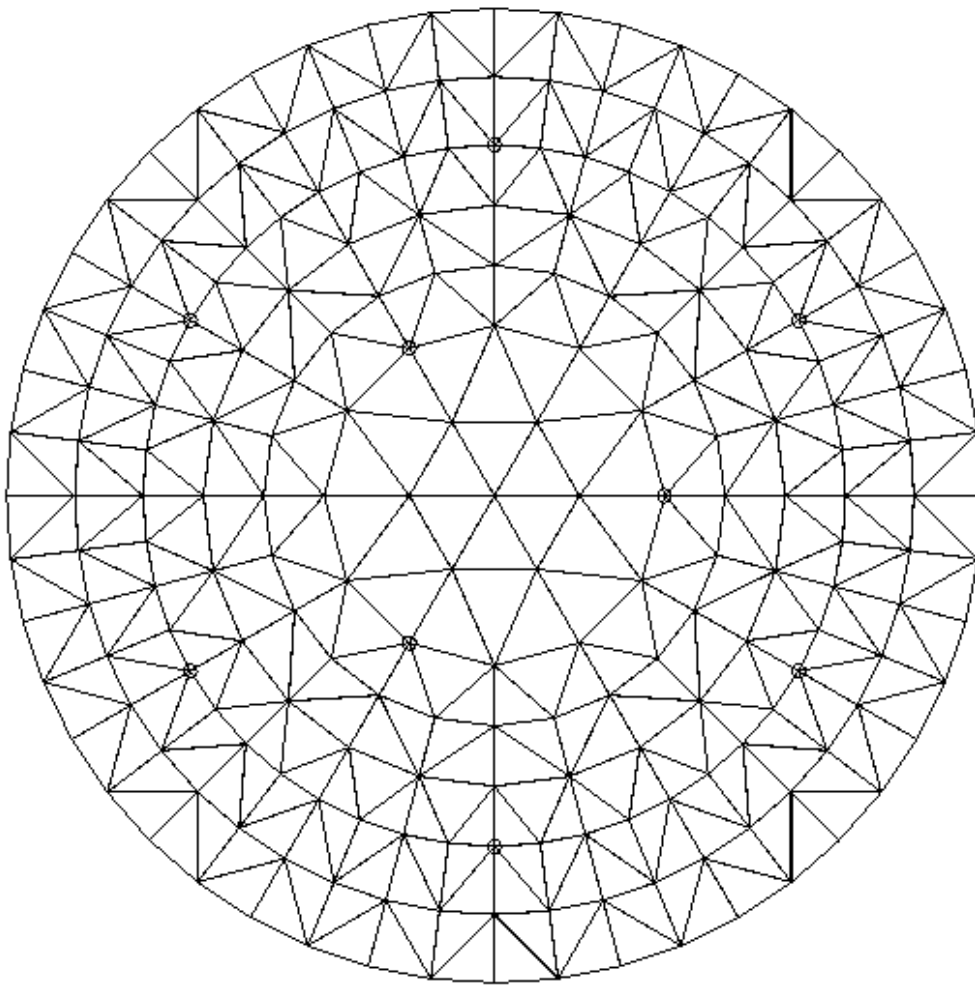


Plate is then used to perform FEM, which solves the differential equation and produces a number for each of the triangle corner points indicating how much it is deflected by the forces on the mirror.

Plop uses these results, and calculates the surface error resulting from the deformations. It can either compute RMS error, which is more common, or peak to valley (P-V) error. Plop can use either the deformations directly as they result from the FEM, or it can find the best parabola that minimizes RMS error, and subtract that from the error. This is equivalent to the user refocusing the telescope to find the best focus point.

All units used in Plop are millimeters (mm.)

Fine Point: Error calculation requires a careful analysis. The most obvious method is simply to compute the error at each of the mesh points, and to be careful, weight each of these displacements of these by the size of the triangle. This is not very good, because there are typically few mesh points, and the deformation is actually a continuous function across this surface. This means that the error that is measured can take huge jumps for relatively small changes in the geometry, and makes the optimizer's task difficult. It also typically understates the error, so it is necessary to use many mesh points, which takes a large CPU time, to obtain an accurate estimate.

Plop avoids this by using a more detailed estimate. It breaks each of the mesh triangles into 9 smaller error triangles, and measures the displacement of each of these. It uses linear interpolation across the mesh triangles to estimate the displacement at the corners of the error triangles. This eliminates problems that used to occur in early versions of Plop with the optimizer not converging.

Running Plop

Plop is presently a console mode program running under Windows-95. It evolved from a Unix environment, where it was convenient to use it in this manner because of the more powerful shells. Under Windows, it is somewhat clumsier to use it, so it may evolve into a GUI program. To run it, you can use either the MS-DOS command window, or type a command into the Run window. The command should be of the form:

```
plop.exe [options] input_file
```

You will need to specify appropriate path names, and possibly want to redirect the output to a file.

You can run Plop without providing any arguments, in which case it will prompt you for them.

The specification of the analysis to be performed is described in a file that contains a description of the mirror, the cell, and the types and ranges for the analyses to be performed. The optional options control various options such as refocusing, the format of the output, debugging, and producing a picture of the result.

The convention is that the analysis file has the extension ".gr", which is short for grid (Plop was built in several pieces, the first of which was grid_gen, to automatically generate the mesh for Plate. This led to the .gr extension.)

You can also run Plop by double-clicking it. This will prompt you for the options and the file name.

Format of the .gr file

The .gr file contains a sequence of lines, each of which specifies a parameter or command, or possibly a comment. Blank lines are allowed, and a line that starts with a ";" is considered a comment and is ignored. The easiest way to create a file is with any ASCII text editor such as notepad or wordpad, which can edit text files. The examples supplied on the disk are formatted using Unix conventions, (LF only at end of a line) and notepad will produce a single line containing the whole file. It is better to use wordpad on these files.

A parameter specification has a keyword specifying the parameter to be set, and one or more values associated with it. For example, one of the parameters is the diameter of the mirror. This is specified with the **diameter** keyword as follows:

```
diameter 300
```

Use standard scientific notation for floating point numbers, i.e. diameter 3e2 or diameter 3.0e2 would be acceptable. A parameter can also have a variable name listed as the value. Variables must be defined before they are used. See section 0 for a description of variables.

In the case that there are multiple numerical values associated with a parameter, they are listed consecutively, and separated by one or more blanks.

The following set of keywords are used by plop:

diameter, thickness, density, modulus, poisson, focal-length, f-ratio, sagitta, rel-sagitta, n-mesh-rings, support-radii, rel-support-radii, mesh-radii, rel-force, num-support, support-mesh-ring, support-angle, points-on-ring, obstruction-radius, rel-obs-radius, basis-ring-size, basis-ring-min, optimize, scan-set, scan-var, monte, var, hole-diameter, rel-hole-diameter

Not all keywords are required. Sensible defaults are provided for some of the parameters.

Many of the parameters describe linear dimensions. The units of all linear dimensions are mm. In case you forget, one inch is exactly 25.4 mm.

Physical Properties of the Mirror Material

The **density** keyword specifies the density in kg/mm³. The **modulus** keyword gives the Young's modulus. The **poisson** keyword gives the Poisson ratio of the material. These default to Pyrex 7740, with values 2.23e-6, 6400, and 0.2.

Geometry of the Mirror

You must specify the diameter, thickness, and focal length of the mirror.

Use the **diameter** keyword to specify the diameter.

Use the **thickness** keyword to specify the thickness at the edge of the mirror.

Use **hole-diameter** or **rel-hole-diameter** to specify the size of a hole in the center of the mirror. The latter specifies the diameter of the hole relative to the diameter of the mirror.

Plate, the FEM solver in Plop, is based on plate theory. This means that the mirror should be thin compared to its diameter. Specifically, the thickness should be no more than 20% of the diameter.

Use one of the keywords **focal-length**, **f-ratio**, **sagitta**, and **rel-sagitta** to specify the focal length. The **focal-length** keyword gives the absolute dimension of the focal length. The **f-ratio** keyword specifies the focal length as an f-ratio. The **sagitta** keyword specifies the focal length as an absolute measurement of the sagitta of the mirror. The **rel-sagitta** keyword gives the sagitta of the mirror as a ratio of the absolute sagitta divided by the edge thickness of the mirror.

This apparent excess of ways to specify a single parameter is more useful than it seems at this point. The reason is that one may want to analyze a set of mirrors, and hold some parameter constant across the range of analyses. For instance, you might want to analyze a set of f/6 mirrors for various diameters, or to analyze a set of mirrors in which the sagitta is 5% of the edge thickness. Choosing the appropriate keyword from the above list will let you hold the corresponding geometrical ratio constant.

Secondary Mirror Obstruction

The secondary mirror obstructs the primary mirror, and the obstructed part of the primary mirror should be ignored from the error calculations. Arnold showed that this can affect the best choice of supports for the mirror. You can tell Plop the size of the secondary mirror, or its size relative to the primary mirror, by using either the **obstruction-radius** or **rel-obs-radius** keywords. The **obstruction-radius** keyword specifies the radius of the secondary. The **rel-obs-radius** keyword specifies the ratio of the radius of the secondary to the radius of the primary. Note that the secondary is specified in terms of radius, not diameter. For the ratio, you can choose to think of the ratio of either the radii or the diameters, since they are equal. For example, a 300 mm diameter mirror with a 45 mm diameter secondary could be described with either of the following:

```
obstruction-radius 22.5
```

```
rel-obs-radius .15
```

Geometry of the Mirror Cell

This section begins with a description of the kinds of mirror cells that Plop can model.

A mirror cell provides a set of supports for the mirror, each one of which exerts some force at what is essentially a single point on the mirror.

The support points must be described in a certain framework that corresponds to common mirror cells. The supports are arranged in a set of rings, called support rings. Each support ring contains some number of supports at a given radius. The supports are spaced around the ring at equal angles, so the number of supports defines the angle between the supports. The angle at which the first support occurs is the only other piece of information that must be defined in order to complete the specification.

Consider a typical 9-point support for a 300 mm mirror. It might have 3 supports at radius 45mm, and 6 supports at radius 120 mm. The **num-support** keyword is used to define the number of supports in each mesh ring. The inner ring has supports at angles 0,120, and 240 degrees, and the outer one has supports at angles 30,90,150,210,270, and 330 degrees. This corresponds to the picture above. The keywords

support-radii and **support-angle** are used to describe the radii and the angle of the first support in each ring. Each must contain one number for each ring of supports in the cell. The description is:

num-support 3 6

support-radii 45 120

support-angle 0 30

It is also common to want to describe a cell in terms of the ratio of the support ring radius compared to the radius of the mirror. This is useful in examining the properties of a set of cells with the same relative configuration. The **rel-support-radii** keyword describes the radius of the supports divided by the radius of the mirror. The following statement could also be used to describe the example cell above.

rel-support-radii .3 .8

There are some important considerations for generating the mesh from the support configuration. Plop insists that each support point be located at the corner of a triangle. It also makes the triangles a uniform size around each ring. Most inconveniently, it requires that the first point be at angle 0. This means that it is important that the angle of the first support be divisible into 360 degrees. For example, if the first support is at 30 degrees, we can use as few as 12 triangles around the ring at that point. On the other hand, if the first support is at 1 degree, it will be necessary to have 360 triangles around the ring at that point in the mesh.

This can be avoided by using the basis generation method described later, but this is best left until you have a good understanding of Plop's operation.

Forces on Mirror Cell Supports

By default, each of the supports in the mirror cell has the same supporting force. Plop allows you to specify forces that are different for each support using the **rel-force** keyword. This has the following format:

rel-force num1 num2 num3 ...

Each of the numbers corresponds with one support ring. There must be exactly as many numbers as there are support rings. The numbers are arbitrary units. The force associated with a given support is computed according to the number given for that ring, divided by the total of all the numbers. For example, if a 9 point support should have 1.5 times as much force in each of the outer ring of 6 supports as the each of the 3 supports in the inner ring, the following would be used:

rel-force 1 1.5

The statement

rel-force 2 3

is also equivalent, since the ratio of the numbers is identical. In both cases, the supports on the inner ring would have $1/(3*1+6*1.5) = 0.08333$ of the total force, and the supports on the outer ring would have $1.5(3*1+6*1.5) = 0.125$ of the total force.

Mesh Generation Parameters

You must tell Plop how fine a mesh to generate. Plop constructs the mesh by constructing a set of rings of various radii, and filling each ring with triangles. The number of triangles is arranged such that support points line up with a triangle point, and to ensure reasonably size triangles. In the absence of other constraints, Plop will use 6×2^n points around a mesh ring by default, such that the i th ring contains at least $6 \times i$ points. Typically, this means the number of points is 1, 6, 12, 12, 24, 24, etc. For configurations that cannot fit into this, Plop will vary the number of mesh points to align with the support points. All that the user needs to specify is number of mesh rings. This is done with the **n-mesh-rings** keyword. This describes the number of rings of points on the mesh. Since each triangle is between two mesh rings, the number of rings of triangles will be one fewer. In the example above, n-mesh-rings is set to 8, and there are 7 rings of triangles. This is done with the following line:

```
n-mesh-rings 8
```

For most common analyses, specifying the number of mesh rings will be sufficient, but optimization requires that careful attention be paid to the details of associating mesh rings with supports.

As a guideline, the following number of mesh rings are recommended as minimums for various types of mirror cells.

Cell Type	Minimum number of Mesh Rings Required
3 point	8
6 point	11
9 point	11
18 point	17

Scanning Parameters

Plop contains facilities to allow you to scan one or more parameters or variables across a range. This is useful for conducting studies across a range of values without having to edit a separate .gr file for each analysis. For example, you may want to determine the deformation for 10 different mirror diameters, and for each one of those, study 10 different focal lengths. Considering the combinations of diameters and focal length, there are 100 separate analyses that need to be performed. Using two Plop commands can perform all of these analyses from a single .gr file.

There are two different methods for scanning a range of parameters. The first, **scan-var**, performs the analysis of a parameter for a set of equally spaced values. The format is as follows:

```
scan-var parm-name [index] start-value end-value step-ratio
```

The *[index]* indicates an optional value of *index*. The *parm-name* is the keyword or variable that describes the parameter that you wish to vary. If the parameter is a vector, that can contain more than one value, such as **support-radii**, then the *index* must be present, and indicates which one of the elements of the vector is to be scanned. The index must be present even if the vector only contains a single value. Any

parameter that can meaningfully be a floating point value may be scanned. Also, if the scanned parameter is a vector, then all of the vector must be initialized with the corresponding keyword. If, on the other hand, the scanned parameter is a scalar, there is no need to initialize it using its keyword. This is to because the scan will necessarily set a scalar value, but might not set all of the values in a vector parameter.

Parameters such as **num-supports** and other values that must be integers cannot be scanned. The starting and end values are given by *start-value* and *end-value* respectively. The step size is determined by dividing the distance between the end and starting points by the integer specified in *step-ratio*. Note that this means that the number of steps is always one greater than *step-ratio*. For example, if you wanted to perform an analysis of all mirrors from 100 to 200 mm in diameter, with at 20 mm step size, you could use the following:

```
scan-var diameter 100 200 5
```

This performs 6 analyses at 100,120,140,160,180, and 200 mm diameters. Similarly, to scan the radius of the inner support ring on a 9 point support from 30mm to 60 mm using a 5mm step, the following would be used:

```
scan-var support-radius 0 30 60 6
```

Note the 0 index to indicate that it is support-radius 0 to be scanned, while the diameter scan does not require an index.

Plop allows non-uniform steps, using the **scan-set** keyword. The format is:

```
scan-set parm-name value1 value2 value3 ...
```

In this command, you explicitly specify the values of the parameter for each analysis. For example, if you want to examine f/4, f/4.5, f/5 and f/6 mirrors, you could use the following:

```
scan-set f-ratio 4 4.5 5 6
```

Optimization

One of the most powerful features of Plop is its ability to automatically optimize the design of a mirror cell to minimize the surface error. The **optimize** keyword instructs Plop to vary a parameter until the smallest error is found. The syntax is as follows:

```
optimize parm-name [index] step_max
```

As the case in the scan keywords, *parm-name* and the optional *index* specify the parameter that should be optimized. The *step-size* is the maximum change of the parameter that Plop should make in a single optimization step. This should be chosen to be small compared to the expected value of the parameter that you are optimizing. For example, we usually use **rel-support-radii** to express the supports' radii relative to the radius of the mirror. We expect that 0.02 of the radius is sufficiently small compared to the relative radius of the supports.

Plop will iterate until the change from one analysis to the next is about 1e-4 times the value of the *step_max* parameter, or 0.01%, or until the change in error between iterations is 1e-9 times the error.

Variables

Most mirror cells have some symmetrical geometry, but it may not be possible to describe this using the parameters listed above. For example, consider a 9 point cell, where the outer ring of 6 supports is composed of pairs of points arranged symmetrically about each of the 3 inner points, and it is desired to determine both the best angle and radius of the supports. This can be constructed as 3 rings of 3 supports, but there is no means to enforce symmetry. Unfortunately, the behaviour of numerical optimization is such that some asymmetry is likely to occur, and the user is left to puzzle out what to do.

Plop includes variables to describe formulae that can be used as parameters. In the above example, we would like a set of rings with 3 supports each. The outer 2 rings would be located at angles $60 + da$ and $60 - da$, assuming the inner ring supported this. Both outer rings would have the same radius. Plop can describe a structure such as this as follows:

```
var da
```

```
var aplus 60 + da
```

```
var aminus 60 - da
```

```
var r
```

```
num-support 3 3 3
```

```
rel-support-radius .3 r r
```

```
support-angle 60 aplus aminus
```

This describes the cell as 3 rings of 3 supports, but constrains the outer 6 supports to have the same radius, and be symmetrical about the inner 3 supports. The parameters to be optimized are **rel-support-radii 0**, **r**, and **da**.

A variable declaration can be either **var name**, or **var opnd1 op opnd2**. In the former declaration, the value of the variable name must be set by a subsequent scan or optimization. In the latter form, the **opnd1** and **opnd2** can be numeric values, or the name of other variables. The **op** can be one of +, -, *, /. Variables must be defined before they are used.

Monte Carlo Analysis

For problems with a small number of independent parameters, we can use the **scan-set** keyword to test every possible combination of cells that result from tolerances in fabricating the cell. The number of such tests increases exponentially with the number of tolerances that must be examined. Plop provides a Monte Carlo analysis for easily performing a number of tests. This uses the **monte** keyword, with the name of the variable or parameter, together with the maximum deviation from the nominal value. Plop will choose random values from the range from the negative of the deviation to the positive value of the deviation, compared to the nominal value. This is especially handy in conjunction with the -w command, so we can just modify the optimal cell description by including a few **monte** keywords with the appropriate deviations. For example:

```
var r1
```

```
monte r1 .01
```

```
rel-support-radii 0.3 r1 r1
```

Be careful to note implicit correlations caused by use of variables, rather than physical parameters. For example, expressing a symmetrical design using variables, and randomly varying the variable would still always result in a symmetrical design. The **-wp** flag allows you to save the physical parameters of the design, not the one based on variables. This is useful for performing a Monte Carlo with uncorrelated fabrication errors.

Interaction and Ordering of Scanning and Optimization

Optimization and scanning can both be performed within a single .gr file. This means that some of the parameters are scanned, and for each such value, the optimization is performed to determine the best value of the optimization parameters. It is easy to write a single .gr file that, for every diameter and every focal length, finds the best location of the supports. Here is a fragment of the .gr file to do so:

```
scan-var diameter 100 200 10
```

```
scan-set f-ratio 4 4.5 5 6
```

```
optimize support-radii 0 .02
```

Use of multiple **scan-set** and **scan-var** performs analyses for all combinations of the parameters. The **scan-var** is performed for each of the **scan-set** analyses. For each type of scan, an analysis of all combinations of parameters that occur later in the .gr file is performed for each given scan. I.e., parameters listed later are varied more rapidly.

Advanced Mesh Generation Parameters

You probably do not want to read this section, which is mainly used for debugging purposes.

Plop will usually be able to generate the mesh given only the radii of the supports and the number of mesh rings. However, the algorithm for mapping the supports to various rings on the mesh rings may fail if you give it a peculiar mirror cell configuration. In particular, if the supports are at very close radii, it may not be able to find a mesh ring for every support. In this case, you need to tell Plop the radii of the mesh rings, and which mesh ring is to be used for each of the support rings. The mesh-radii keyword is used together with a vector of the radii at which the mesh rings are located. The support-mesh-ring keyword specifies which of the mesh rings is to be used for the supports. For example:

```
mesh-radii 0 20 40 60 80 100
```

```
support-mesh-ring 2
```

would be used if you wanted 6 mesh rings and a single ring of supports at radius 40.

This is not terribly likely to be useful, but you were warned.

Control Options for Plop

Operation of Plop is controlled by a set of flags supplied at the command line, or at the prompt if you start Plop without any arguments. The options available are listed below, and the detailed description of each in subsequent sections. The arguments start with a '-' if you put them in the command line. Do not use the '-' if you are typing them as commands to a prompt from Plop.

Flag	Meaning
-a <i>n</i>	maximum mesh points
-b	verbose
-c[<i>n num</i>]	contour plots
-d <i>file</i>	generate Plate data file
-e <i>num</i>	number of Monte Carlo tests
-g <i>name</i>	mesh generation strategy
-m <i>file</i>	generate picture of mesh
-n <i>num</i>	number of colours to use in plots
-o	trace progress of optimizer
-p <i>file</i>	plot map of deformation
-q	quiet mode
-r	toggle refocus
-s <i>num</i>	size of picture
-u	reuse best optimizer result
-v	use P-V error measure
-w[p] <i>name</i>	save result of optimizer
-z <i>num</i>	z range for colors in picture
-D <i>name</i>	debugging
x	run Plop (prompt mode only)

Allocation

Plop allows 5000 mesh points by default. There can be up to 3 times as many triangles as there are points. Most of the memory in reasonable size problems is used for storing matrix elements. This is allocated in 500KB chunks, and Plop can allocate a full 32 bit address space of elements. The -a argument can be used to adjust the number of mesh points allowed according to the size of your machine and the demands of the problem. The argument gives the maximum number of mesh points. Plop will advise you if you run out of space.

Verbose Mode

Use the -b flag to obtain some information about the size of the mesh when generating the mesh.

Contour Flag

When plotting the deformation, use the **-c** flag to specify that a contour plot should be generated. The default is a color plot. Use the **-cn** flag with a number to specify the number of contours.

Print Plate Data

When running a single FEM, use the **-d** flag to print the results of Plate in a file. This is in Toshimi's format.

Number of Monte Carlo Tests

The **-e** flag specifies the number of monte carlo tests. The default is 100.

Mesh Generation Strategies

One of the most powerful features of Plop is automatic optimization of mirror cells. Optimization is a intricate procedure that requires some understanding in order to apply it and obtain good results. One of the critical issues is the way that supports are placed at mesh points. When a mesh is generated, Plop will enforce the placement of the supports on mesh rings to coincide with the radii of the mesh rings. It does so by finding a good fit between the number of mesh rings between adjacent supports that tends to equalize the size of the mesh rings. For example, consider a simple cell with one ring of supports, and a mesh that has 5 rings. Ideally, we would like to place the mesh rings at relative radii 0, 0.25, 0.50, 0.75, and 1.0. In the presence of supports that are not located at one of these radii, Plop sets one of the radii to coincide with the supports, and adjusts the spacing of the remaining rings to allocate equal space to each ring. There must always be a mesh ring at 0 and a ring at 1, so there are only 3 mesh rings that can be placed at any given location. For example, if we want the support at 0.4, the nearest uniformly spaced ring would be ring 2, at 0.5. Plop will therefore assign the support to ring 2, and set ring 2 to be located at 0.4. It will divide the remaining space among the rings evenly, and place ring 1 inside the support ring at 0.2, and ring 3 outside the support ring at 0.7. In another circumstance, if the support is to be at 0.35, the closest uniformly spaced ring is ring 1, so Plop will choose to place a ring at 0.35, but to put all other rings outside. Ring 0 will still be at 0, ring 4 at 1.0 and the remaining rings will be at 0.57, and 0.88.

The reason that it is necessary to understand mesh generation when performing optimization is because a key requirement of optimization is that the error function vary smoothly with the parameters that are being optimized. As long as a support is associated with a given mesh ring, the error will vary smoothly with the radius of the support. However, as a support is moved to a location that causes it to be associated with a different ring, then the numerical error of the FEM may jump, even though the actual error will not do so. If the error is plotted as some parameter is varied smoothly, the error curve will be smooth, with the exception of discrete jumps as Plop moves the supports from one mesh ring to another.

This can cause mildly confusing results when plotting scans of parameters, but the error is typically small. The difficulty is that the optimizer has no insight into discrete jumps in an otherwise smooth curve, but simply compares the error at one point to the error at another point. The optimizer can become hopelessly confused as a result of these jumps.

Since the jumps occur as a result of moving the supports in discrete steps from one ring to another, there are basically two approaches to avoiding this problem: (1) place each support ring on some mesh ring at

the beginning of the optimization, and don't move it thereafter, and (2) move the supports smoothly from one ring to another. In (1), the ring number that is used for support will remain fixed across multiple FEMs. In the above example, if the support was initially at 0.4, ring 2 would be used. It would continue to be used for subsequent FEMs, so that if the support had later been moved to 0.35, the remaining rings would be uniformly spaced at 0, 0.175, 0.675, and 1. If the support rings are moved a long distance from their starting position, large numerical errors can occur due to the distorted size of the mesh rings.

In (2) fix the rings are fixed at their uniformly spaced locations, and Plop is used to perform 5 different FEMs, one for placing the support ring at each mesh ring. A subsequent analysis that places a support at a location between these uniformly spaced mesh rings is handled by using a weighted sum of the results. For example, the analysis of the support ring at 0.4 would use .6 of the displacement resulting from the support at 0.5, plus 0.4 of the displacement resulting from the support at 0.25. This method is called the basis method, where a basis set of FEMs are performed, and all subsequent support configurations are derived from the basis set of FEMs. This method is extremely fast because it is possible to perform several FEMs using the same mesh, but different supports, nearly as fast as performing a single FEM. Each additional support analysis can take less than 1% of the time for the first analysis. Essentially, in the time required to perform a single FEM, any number of subsequent analyses can be performed.

The disadvantage is that the supports are now modeled essentially as being spread across two rings, when in fact we would like to model them as point supports. However, because of the speed of the basis method, it is possible to use very large meshes, with 1,000 or more triangles, reducing the spreading out of the force. Of course, the supports are not actually points, so the truth lies somewhere in between.

There is an enormous speed advantage to the basis method. For example, an analysis using 30 mesh rings takes about 200 seconds on a 233Mhz P-II. Once this is done, subsequent basis analyses can be performed in 0.4 seconds. This is 500 times faster. An optimization performed on a large cell performed about 10000 analyses. Using FEM alone, this would take about a month, using the basis method it took about an hour.

Plop provides various strategies that perform both (1) and (2) in various ways. These are controlled by the grid-gen strategy argument to Plop. This has one of six possible values: **never**, **always**, **once**, **twice**, **first**, and **basis**. The default is **always**. The number of these strategies is evidence of the difficulty in finding a good strategy to handle mesh generation. The differences in the strategies are primarily related to the interaction of scanning and optimization. Only the basis strategy uses weighted sums of the basis; the remainder perform one FEM for each error analysis, but the details of the mapping between supports and mesh rings differ. The basis strategy is likely the best, as well as the fastest, but requires more skill to compose the problem.

Use the **always** method if your problem converges; if you have a complex problem with many parameters, or are obtaining poor results, use the **basis** method, which takes more care in setting up the problem, but is much faster.

To specify the generation strategy, use the **-g** flag followed by one or more spaces and the name of the mesh generation strategy. Each of the strategies is outlined below. All of the strategies except basis perform a FEM for every analysis.

-g always

Map supports to rings for every analysis. This is the default.

-g never

Map supports to rings for the first analysis, and reuse this mapping for every subsequent FEM.

-g first

For each optimization that is performed, map the supports to rings for the first FEM of that optimization, and reuse that mapping for every subsequent FEM in that optimization. This will help make sure that the optimizer does not encounter discontinuities, but will adjust the mapping to a better fit between optimizations.

-g twice

For each optimization, map the supports to rings on the first FEM, then reuse this mapping for subsequent FEMs. After finding the optimum, perform another mapping and optimize using more FEMs using the new mapping to find a new optimum. This makes sure that Plop uses a mapping that is near to the one at the optimum, but can take a long time, or go off track if the first mapping leads to a bad solution.

- g once

This is similar to -g twice, in that it does the first optimization twice, but then never changes the mapping for that optimization, or for any subsequent optimizations.

-g basis

This is the most complex to use, and requires using the **basis-ring-size** and **basis-ring-min** keywords in the Plop .gr file. This specifies that Plop should generate a basis for each distinct physical configuration, and then use that basis to find the error of any configuration that can be derived from it.

A configuration can be derived from a basis as long as the physical geometry does not change - for example, the diameter, thickness, focal length, and physical properties of the material. If these change between analyses, Plop will perform a FEM to regenerate the basis.

Plop generates a basis by considering the placement of the supports at each of the mesh rings. Complexity arises from the fact that there may be a different number of supports in each ring of supports. This complicates mesh generation, which normally ensures that the number of points on each mesh ring is an integral multiple of the number of supports that are located on that ring.

For example, consider an 27-point cell with 3 rings of supports, with 6, 9, and 12 supports in the rings. In order to form the basis for all of these, there must be at least 36 points on each mesh ring (36 is the least common multiple of 6, 9, and 12.) This is a large number for the inner rings and is not necessary for accuracy. Plop provides the capability of using fewer points than the number of points on a support ring. Plop will allow a support ring to be modeled as a set of points that divides the number of supports. In this example, Plop could use a basis set that had only 3 support point in each mesh ring. The 6 support ring would be modeled as two of the 3-point rings, one aligned with the 6-support ring, and the second rotated by an extra 60 degrees compared to the first ring. The 9 point ring would be modeled as three contributions rotated by 0, 40, and 80 degrees respectively. The 12 point ring would be modeled as four contributions.

To speed up the computation, it is advantageous to break each support ring into as few components as possible. This can be done if it is known that the rings with a larger number of supports will only be used beyond some minimum ring number, and will occur only at locations where they will not cause an increase in the number of mesh points used for the ring. In this example, it is likely that the 12-point support ring will only occur in the outer few mesh rings, so it is reasonable to use 12 support points in the basis for these rings.

The purpose of the **basis-ring-size** and **basis-ring-min** keywords is to specify the number of support points that will be analysed at each ring beyond a given mesh ring number. The **basis-ring-size** specifies the number of support points that will be used in each support ring used for basis generation. The **basis-ring-min** specifies the minimum mesh ring number to which the corresponding **basis-ring-size** applies. For the above example, we might decide to use basis support rings of 3, 6, and 9 to minimize the computation. We might decide to only use the 9-point mesh at rings 3 and higher, assuming a 8 ring mesh. This would be done with the following:

```
basis-ring-size 6 9 12
```

```
basis-ring-min 0 0 3
```

Using this configuration, the basis would be generated using 8 analyses for the 6-support, 8 analyses for the 9-support, and 5 analysis for the 12-support. Each analysis would be decomposed into a weighted sum of the results of the basis analyses.

It is possible to use fewer points in the basis set than there are supports in the cell. Plop can decompose a support ring into a sum of smaller rings. It can also accommodate rotations that are off the mesh. For example, it would be possible to rotate the 9-point ring by one degree without necessitating the use of 360 mesh points, as is done with the other forms of analysis.

The basis strategy for mesh generation is presently the best for optimization. A side effect of using the basis is that the error varies smoothly as the support positions are changed. However, Plop at present requires that the user input the **basis-ring-size** and **basis-ring-min** parameters as described above, which are somewhat complicated to understand. The simplest way to generate these parameters is to use the number of supports as the basis-ring-size, and to set the basis-ring-min to numbers increase slowly

Generate Picture of Mesh

For a single analysis, use **-m file** to save a picture of the mesh in the file. This is a .gif file.

Number of Colors for Plot

To change the number of colors used for the plot, use the **-n num** flag. The default is 200 for color plots. Decreasing this to 10 or 20 leads to something that looks more like a color contour plot.

Trace Optimizer

Plop will only print the results of the best solution found for an optimization problem. You can watch the optimizer try various configurations and the error result for each by using the **-o** flag.

Plot Map of Deformation

Use the **-p** *file* flag to store a picture of the deformation in the file. This is a .gif file.

Quiet Mode

For each analysis or optimization, Plop will print out a line that begins with "results:" and contains the names of all of the parameters being optimized, followed by the error. This is easy to read, but may be hard to import into another tool to produce a plot or a table. To print a concise version of the output that only contains the numeric fields, use the **-q** option.

Refocus Flag and Error Measures

The error due to deformation of the mirror will often exceed the actual error encountered in use of the mirror. This is because the user will focus the system to obtain the clearest image. To model this, Plop can refocus the deformation by finding the parabola that best fits the deformation. Plop then subtracts out this parabola from the deformation computed by Plop. This can dramatically reduce the error, and change the best way to support the mirror. See the web page.

Plop does refocusing by default. If you do not want to refocus, use the **-r** flag to toggle it.

Size of Plot

The plot is 500 pixels square by default. To change it, use the **-s** *num* flag with a number to specify the size of the picture.

Reuse Optimizer Result

In multiple optimizations, the results of an optimization will often be similar to previous ones, in particular if the appropriate relative dimensions are used instead of absolute ones. To speed up the optimizer, it is helpful to begin the new search at the same place that the previous one ended. Plop will do this by default. To prevent this, use the **-u** option.

Use Peak to Valley Error

Plop calculates the error as RMS error. RMS error is typically about 0.2 or 0.25 of the peak-to-valley error. RMS error is considered to be a better measure as it includes the entire surface, while P-V only reflects the error at two extreme points in the surface.

A reasonable limit for the RMS error due to the mirror cell is about 1/128 wave, which is about 4.2e-9 meter, or 4.2e-6 mm, for 550nm green light. This means 1/64 wave on the wavefront reflected from the mirror, and corresponds to about 1/16 wave peak to valley. This is based on Toshimi Taki's suggestion that about 1/4 of the error budget may be allocated to the mirror cell.

Plop uses RMS error by default. To toggle the error metric, use the **-v** flag.

Save Result of Optimization

Plop will print out the best result found during optimization, but if you want to save it as a .gr file, use the **-w** flag with the name of the file to save it in. The **-w** flag will print the descriptions of all variables and the optimum value of optimized variables. The **-wp** flag will print the actual values of the physical variables, and ignore the variables. This is particularly useful for use in a subsequent Monte Carlo analysis. This is because performing a Monte Carlo on the set of variables will only sample random values of the variables, which can cause correlation of the mirror cell errors that is unlikely to exist in the actual fabrication errors. By using the actual physical parameters, the Monte Carlo can use independent random variations that are more representative of the errors likely to occur in practice.

For more elaborate analysis, use variables to represent the actual fabrication errors that can occur. For example, a set of points might be supported by a triangle, and any error in placing the triangle will move all of these points uniformly. This can be described using variables.

Z Range for Plots

Plop will compute the colors so the range of colors or range of contours spans the entire range of values computed by the analysis. To make it easy to compare two different analyses using the same range of color to represent a given in the plot, you can use the **-z num** flag. The number specifies the range of deformations that is mapped into the range of colors or contours on the plot, as appropriate.

Debugging

Debugging can be enabled by using **-Dname**, where *name* is one of **grid_gen**, **opt**, **basis**, **triangle**, or **interpolate**. If you need to use any of these, you should already have the source code and know why you want to.

Examples and Guidelines

A number of examples are included with the software. These illustrate how to use Plop for various configurations.

To help in determining the tradeoff of CPU time and memory versus accuracy, we ran the following experiment. A 9 point cell was evaluated using various numbers of mesh rings, using the basis method for error calculation. We measured CPU time, memory, and the predicted error. The following table lists all of these. The experiment was run on a 233MHz PII with 96MB of RAM. Note that this is for a FEM, and that using the basis method, each subsequent requires only a small fraction of the time stated.

The table shows several interesting properties, as we presume that analyses with more rings converge to the actual result. First, RMS error before refocusing converges quickly, so even 6 rings gives a result within 5% of the result for 35 rings. However, the error after refocus is the difference of two much larger numbers, and so the relative accuracy is much degraded. Be sure to note the 10X different scale factor between these two columns. The predicted RMS error after refocus for the 6 ring example is 17% smaller than the error of the 35 ring example. However, we only require 15 rings to come within 5% of the 35 ring result. It should also be noted that the error measure is still increasing at 35 rings, and may be slightly higher.

Mesh Rings	CPU Time (seconds)	Memory (MB)	Number of Points in Mesh	Number of Triangles in Mesh	RMS Error before Refocus (10^{-6} mm)	RMS Error after Refocusing (10^{-7} mm)
6	1	1.5	115	180	8.19	8.26
8	1	2	211	372	8.26	8.53
10	3	3.1	355	612	8.48	8.95
15	9	7.8	835	1572	8.54	9.52
20	32	18.7	1603	3012	8.59	9.69
25	98	36.9	2563	4932	8.60	9.81
30	206	60.4	3525	6852	8.60	9.87
35	506	96.8	4867	9348	8.61	9.92

Acknowledgements

The key computational component of Plop is Plate, written by Toshimi Taki. His contribution to Plop is extremely valuable, and core to the software.

Richard Schwartz provided many useful references and suggestions that led to some of the ideas in Plop.

Luc Arnold's paper provided valuable ideas for optimizing mirror cells.

This code includes gd.c, written and distributed by Thomas Boutell, which requires the enclosed copyright notice. This applies only to the files gd.c, gd.h and mtable.c.

Portions copyright 1994, 1995, 1996, 1997, 1998, by Cold Spring Harbor Laboratory. Funded under Grant P41-RR02188 by the National Institutes of Health.

Portions copyright 1996, 1997, 1998, by Boutell.Com, Inc.

GIF decompression code copyright 1990, 1991, 1993, by David Koblas (koblas@netcom.com).

Non-LZW-based GIF compression code copyright 1998, by Hutchison Avenue Software Corporation (<http://www.hasc.com/>, info@hasc.com).

Permission has been granted to copy and distribute gd in any context, including a commercial application, provided that this notice is present in user-accessible supporting documentation.

This does not affect your ownership of the derived work itself, and the intent is to assure proper credit for the authors of gd, not to interfere with your productive use of gd. If you have questions, ask. "Derived works" includes all programs that utilize the library. Credit must be given in user-accessible documentation.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright

notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.