Plop User's Guide

David Lewis

Rev 1.3, Dec 29 1998

1 Introduction

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.

1.1 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.

2 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.

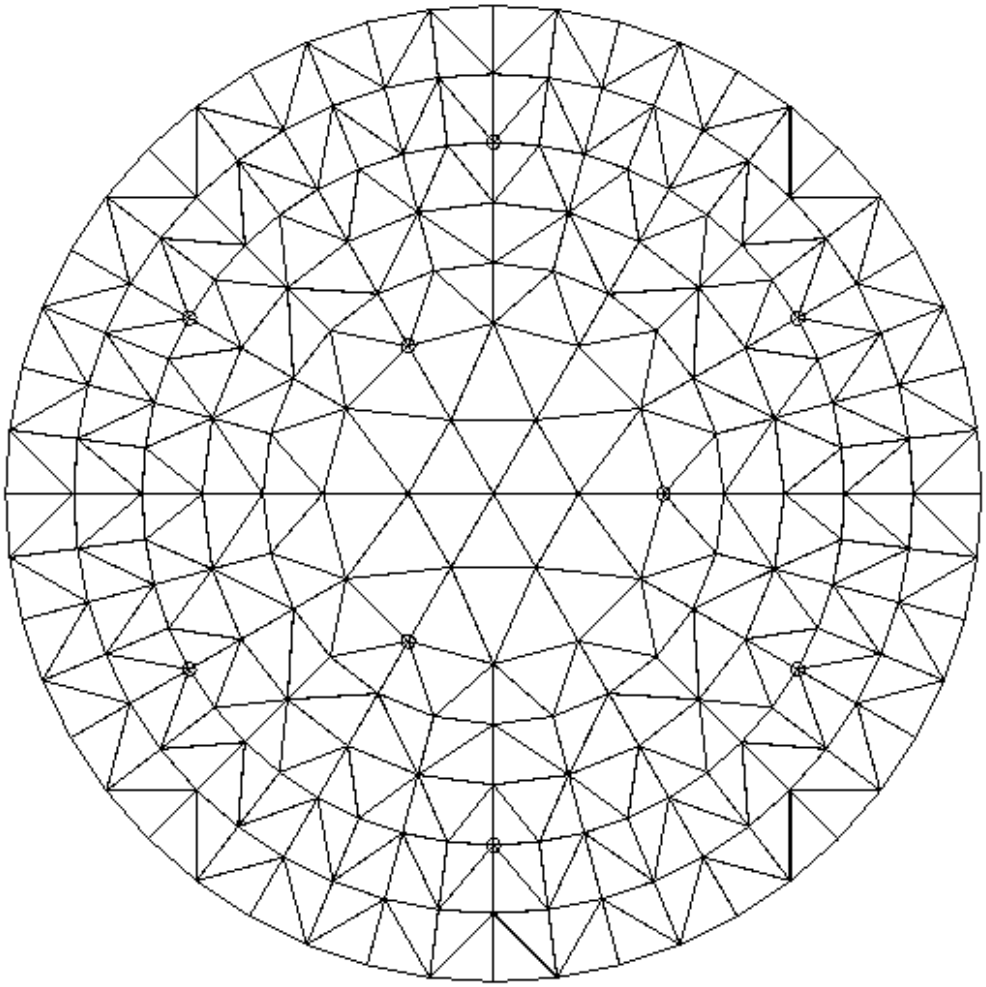

Plate is then used to perform FEM, which solves the differential equation and produces a number for each of the triangle corner points indicating how much it is deflected by the forces on the mirror.

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.

2.1.1 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.

2.2 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 at 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 2.11 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.

2.3 Physical Properties of the Mirror Material

The **density** keyword specifies the density in kg/mm^3. 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.

2.4 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.

2.5 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

2.6 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 fist 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 uses 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.

2.7 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.

2.8 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

2.9 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

2.10 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.

2.11 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 he 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.

2.12 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 devations. 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.

2.13 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.

2.14 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.

3 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 *n*	maximum mesh points
-b	verbose
-c[n *num*]	contour plots
-d *file*	generate Plate data file
-e *num*	number of Monte Carlo tests
-g *name*	mesh generation strategy
-m *file*	generate picture of mesh
-n *num*	number of colours to use in plots
-o	trace progress of optimizer
-p *file*	plot map of deformation
-q	quiet mode

-r	toggle refocus
-s *num*	size of picture
-u	reuse best optimizer result
-v	use P-V error measure
-w[p] *name*	save result of optimizer
-z *num*	z range for colors in picture
-D *name*	debugging
x	run Plop (prompt mode only)

3.1 Allocation

Plop allows 2000 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.

3.2 Verbose Mode

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

3.3 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.

3.4 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.

3.5 Number of Monte Carlo Tests

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

3.6 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 a 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 thant 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. One 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.

3.6.1 -g always

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

3.6.2 -g never

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

3.6.3 -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.

3.6.4 -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.

3.6.5 - 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.

3.6.6 -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

3.7 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.

3.8 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.

3.9 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.

3.10 Plot Map of Deformation

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

3.11 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.

3.12 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.

3.13 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.

3.14 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.

3.15 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.

3.16 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.

3.17 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.

3.18 Debugging

Debugging can be enabled by using **-D***name*, 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.

4 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

5 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.