Schwarz–Christoffel Toolbox User's Guide

Version 2.3

Tobin A. Driscoll*

The Schwarz–Christoffel Toolbox (SC Toolbox) is a collection of M-files for the interactive computation and visualization of Schwarz–Christoffel conformal maps in MATLAB[1] version 6.0 or later. (Earlier versions of the toolbox are avaiable for earlier versions of MATLAB.) The toolbox is a descendant of SCPACK, a Fortran package developed by L. N. Trefethen in the early 1980's [Tre80, Tre89]. However, the SC Toolbox is interactive and graphical, requires no programming by the user, and has many capabilities not in SCPACK.

1 Executive summary

You can do everything graphically by starting `scgui` or by using command line functions. Here is an outline of the typical mapping process.

1. Create a polygon by
   - drawing it in the Polygon Editor (`polyedit`), or
   - calling `polygon` with a vector of vertices (and, for an unbounded polygon, a vector of angles).
2. Create a map by solving numerically for the necessary parameters (functions `diskmap`, `hplmap`, `extermap`, `stripmap`, and `rectmap`). Map types differ primarily by choice of the image region, except that `extermap` maps to the exterior rather than the interior of the given polygon.
3. You may now
   - Look at the SC parameters and an accuracy assessment (e.g., type the name of the map without a semicolon).
   - Visualize the map's action (`plot`).
   - Evaluate the map in both directions (parentheses, `eval`, or `evalinv`).
   - Extract internal map data for your own use (`parameters` and others).

*Department of Mathematical Sciences, Ewing Hall, University of Delaware, Newark, DE 19716; driscoll@math.udel.edu.

[1] MATLAB is a registered trademark of The MathWorks, Inc.

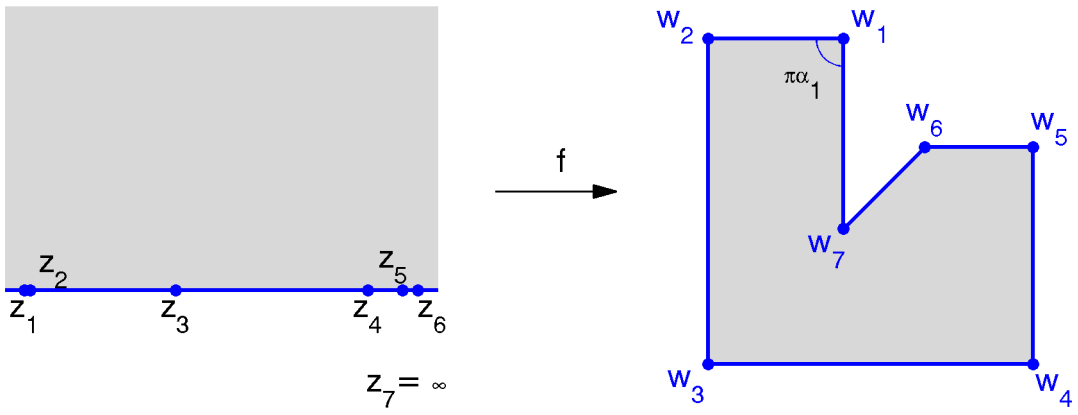


Figure 1: Notational conventions for the Schwarz–Christoffel transformation. In this case z_1 and z_2 are mathematically distinct but graphically difficult to distinguish.

2 Introduction

2.1 Schwarz–Christoffel mapping

The basic Schwarz–Christoffel formula is a recipe for a conformal map f from the complex upper half-plane (the **canonical domain**) to the interior of a polygon (the **physical domain**). The "polygon" may have cracks or vertices at infinity. Its vertices are denoted $w_1, \ldots, w_n$, and the numbers $\alpha_1\pi, \ldots, \alpha_n\pi$ are the interior angles at the vertices.[2] The pre-images of the vertices, or **prevertices**, are real and denoted by $z_1, \ldots, z_n$. They satisfy

$$z_1 < z_2 < \cdots < z_n = \infty.$$

Figure 1 illustrates these definitions.

If vertex w_j is finite, $0 < \alpha_j \leq 2$. If w_j is infinite, $-2 \leq \alpha_j \leq 0$.[3] A necessary constraint is that

$$\sum_{j=1}^{n} \alpha_j = n - 2.$$

Essentially, this means that the total turn is 2π.

The **Schwarz–Christoffel formula** for the map f is

$$f(z) = f(z_0) + c \int_{z_0}^{z} \prod_{j=1}^{n-1} (\zeta - z_j)^{\alpha_j - 1} \, d\zeta. \tag{1}$$

The main practical difficulty with this formula is that except in special cases, the prevertices z_j cannot be computed analytically. Because Möbius transformations have three degrees of

[2] Earlier versions of the toolbox, and much of the literature, instead use $\beta_1, \ldots, \beta_n$, where $\beta_j = \alpha_j - 1$.

[3] This is consistent at the point at infinity with the notion of "interior angle" as the signed angle swept from the outgoing edge, through the interior, to the incoming edge.

freedom, three of the prevertices, including the already fixed z_n, may be be chosen arbitrarily. The remaining $n-3$ prevertices are then determined uniquely and can be obtained by solving a system of nonlinear equations. This is known as the **Schwarz–Christoffel parameter problem**, and its solution is the first step in any Schwarz–Christoffel map. Once the parameter problem is solved, the multiplicative constant c can be found, and f and its inverse can be computed numerically.

Many modifications of the basic Schwarz–Christoffel formula are possible. For example, if the fundamental domain is the unit disk rather than the upper half-plane, the prevertices z_j lie counterclockwise on the unit circle, and the resulting formula is identical except that the product has n terms rather than $n-1$. Other variations of the formula map from the strip $0 \le \operatorname{Im} z \le 1$ or from a rectangle. These two variations are particularly important when the target region is highly elongated in one direction.

Still another variant is the exterior map, in which the fundamental domain is the unit disk and the target region is the exterior of a polygon. In this case the integrand has an additional singularity in the interior of the disk. Assuming this singularity is fixed at the origin, only one prevertex may be chosen arbitrarily.

2.2 Toolbox features

- Solution of the parameter problem for half-plane, disk, strip, rectangle, and exterior mapping
- Cross-ratio formulation of the parameter problem for multiply elongated regions (see section 4.3)
- Graphical input of polygons (see section 3.2)
- Computation of forward and inverse maps
- Adaptive plotting of images of orthogonal meshes
- Graphical and object-oriented user interfaces

2.3 Requirements

The most recent edition of the SC Toolbox requires MATLAB version 6.0.[4] As far as I know it works well under Release 13 (MATLAB 6.5); however, some changes to the language in that release may cause problems still undetected. The toolbox is as platform-independent as MATLAB is—that is, there should be few problems. One notable exception is that graphical interfaces may not look exactly as intended on all platforms.

The graphical interfaces should be accessible to MATLAB novices and experts alike. Command-line use of the toolbox demands some understanding of MATLAB fundamentals such as vectors, matrices, functions, and graphics. An understanding of the use of classes and objects is helpful, but probably not necessary.

[4] See the website below for obtaining toolbox versions for earlier versions of MATLAB.

2.4 Obtaining and installing the SC Toolbox

The latest version of the SC Toolbox is available over the Web at

`http://www.math.udel.edu/~driscoll/SC`

If you cannot reach this page, please send mail to the author at `driscoll@na-net.ornl.gov`. The toolbox is distributed as a zip archive.

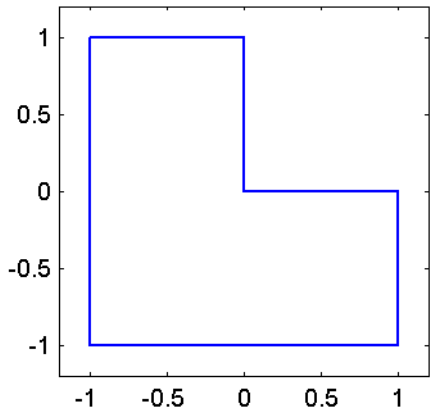
By default, the toolbox is installed into a directory tree rooted at the name `sc`. When you want to use the SC Toolbox, you are responsible for making sure that this directory is on the MATLAB path. See the help text for `path`, `addpath`, and `editpath` in MATLAB.

The SC Toolbox uses the package NESOLVE from Richard Behrens' NONLINPK. The necessary files are automatically unpacked into the SC Toolbox's `private` subdirectory.

2.5 A simple example

Here we create the SC map from the unit disk to a MATLAB favorite, an L-shaped region. First, we create the polygon and plot it:

```
>> p=polygon([i -1+i -1-i 1-i 1 0])

p =

   0.0000 + 1.0000i
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
   1.0000 - 1.0000i
   1.0000
   0.0000

>> plot(p)
```



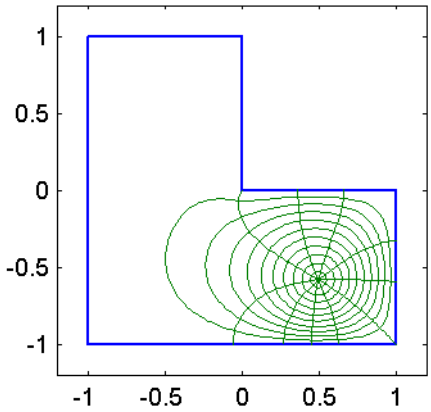
Next, we construct the disk map.

```
>> f = diskmap(p);

Number of iterations: 13
Number of function evaluations: 18
Final norm(F(x)): 3.36023e-010
Number of restarts for secant methods: 0
>> f

  diskmap object:

      vertex              alpha         prevertex              arg/pi
 ------------------------------------------------------------------------
  0.00000 + 1.00000i     0.50000    0.98974 + 0.14286i    0.045628948204
 -1.00000 + 1.00000i     0.50000    0.98811 + 0.15378i    0.049144854230
 -1.00000 - 1.00000i     0.50000    0.95325 + 0.30217i    0.097710854029
  1.00000 - 1.00000i     0.50000   -1.00000 + 0.00000i    1.000000000000
  1.00000 + 0.00000i     0.50000    0.00000 - 1.00000i    1.500000000000
  0.00000 + 0.00000i     1.50000    1.00000 + 0.00000i    2.000000000000

  Conformal center at 0.4955 - 0.5829i
  c = -0.48783135 + 0.29499692i           Apparent accuracy = 6.31e-008
```

After a brief computation, the parameters are found and some statistics about the iteration used to find them are printed out. By entering the name of the map on a line without a semicolon, we get a summary about the map. This summary lists the vertices of the target polygon, their angle parameters, their preimages under the map, and (since they are on the unit circle here) the arguments of those prevertices. Also reported are the image of the origin (known as the **conformal center**), the multiplicitive constant in the map, and an experimentally determined accuracy estimate. The default is to find maps accurate to within 10^{-8} in the image domain, but that is not a guaranteed bound.

We now create a graphical representation of the map.

```
>> plot(f)
```

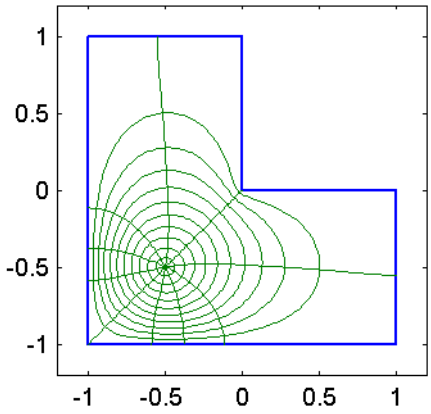


What is seen here are the images of ten evenly spaced circles centered at the origin and ten evenly spaced radii in the unit disk. Notice how greatly distorted some sectors of the resulting

grid become. Also, the intersections are (of course) all orthogonal.

It would be more visually pleasing if the conformal center were along the line of symmetry in the image region. At construction time, you have no control over where the conformal center will be. However, it is a simple (and fast) matter to change the center after the fact.

```
>> f = center(f,-0.5-0.5i)

  diskmap object:

      vertex              alpha         prevertex             arg/pi
 ------------------------------------------------------------------------
  0.00000 + 1.00000i     0.50000    0.83684 + 0.54744i    0.184399349354
 -1.00000 + 1.00000i     0.50000    0.78821 + 0.61540i    0.211005533366
 -1.00000 - 1.00000i     0.50000   -1.00000 + 0.00000i    0.999999999908
  1.00000 - 1.00000i     0.50000    0.78821 - 0.61540i    1.788994466509
  1.00000 + 0.00000i     0.50000    0.83684 - 0.54744i    1.815600650489
  0.00000 + 0.00000i     1.50000    1.00000 + 0.00000i    2.000000000000

  Conformal center at -0.5000 - 0.5000i
  c = 0.46215045 + 0.46215045i           Apparent accuracy = 3.07e-008

>> plot(f)
```



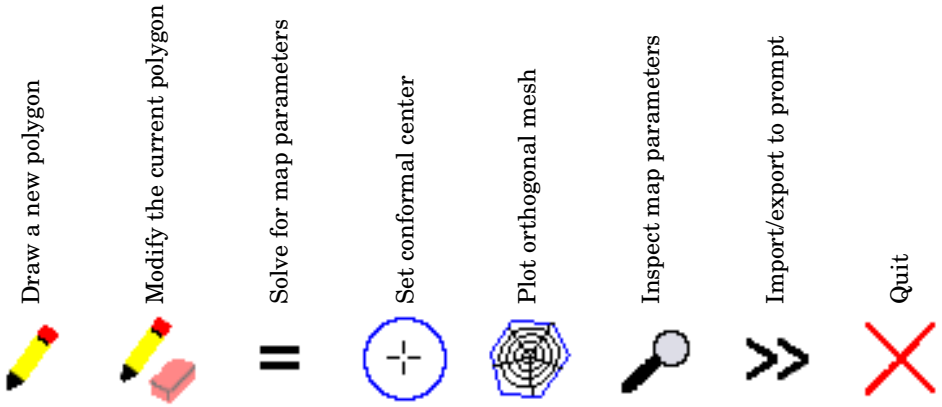
In the following sections we give more details on both the graphical and command-line interfaces. In both types of interfaces the usual progression remains polygon $\rightarrow$ map parameters $\rightarrow$ applications of the map.

3 Graphical interfaces

In the toolbox interfaces, buttons along the top of a window perform actions while controls along the right side affect properties that influence the actions. To get a short explanation of any button, let the mouse pointer hover over it. At certain times buttons or controls may be "grayed out," indicating that they are unavailable in the interface's current state.

3.1 Toolbox interface

To start the toolbox interface, enter `scgui`. A new figure window will be created, looking like Figure 2. (There may be differences in appearance across platforms.) The buttons along the top edge take the following actions:



The import/export function allows data to be transferred between the GUI and the MATLAB workspace. Here "import" means from MATLAB to the GUI, and "export" is the opposite direction.

The side controls are as follows:

Tolerance desired	Requested accuracy for the next parameter solution.
Canonical domain	Type of canonical domain (map type).
View	View the physical domain (polygon), the canonical one, or both simultaneously. In the canonical domain, the prevertices are marked by black dots for clarity.
Mesh lines	These determine the values of `x` and `y` or `r` and `θ` that are used in the canonical domain for mesh plots. After a plot, these boxes are updated to show the used values. If these values are edited and the plot is redrawn, the plot will use the new values.

Once a map's parameters have been found, you can map individual points back and forth between the domains. Click inside (outside for exterior maps) the polygon or in the canonical domain, depending on the current view. The forward or inverse map is computed and a point is drawn in each domain. You may click as many times as you like. Right-clicking (or the equivalent on your system) on any point brings up a dialog to let you inspect or change the coordinates of the points.

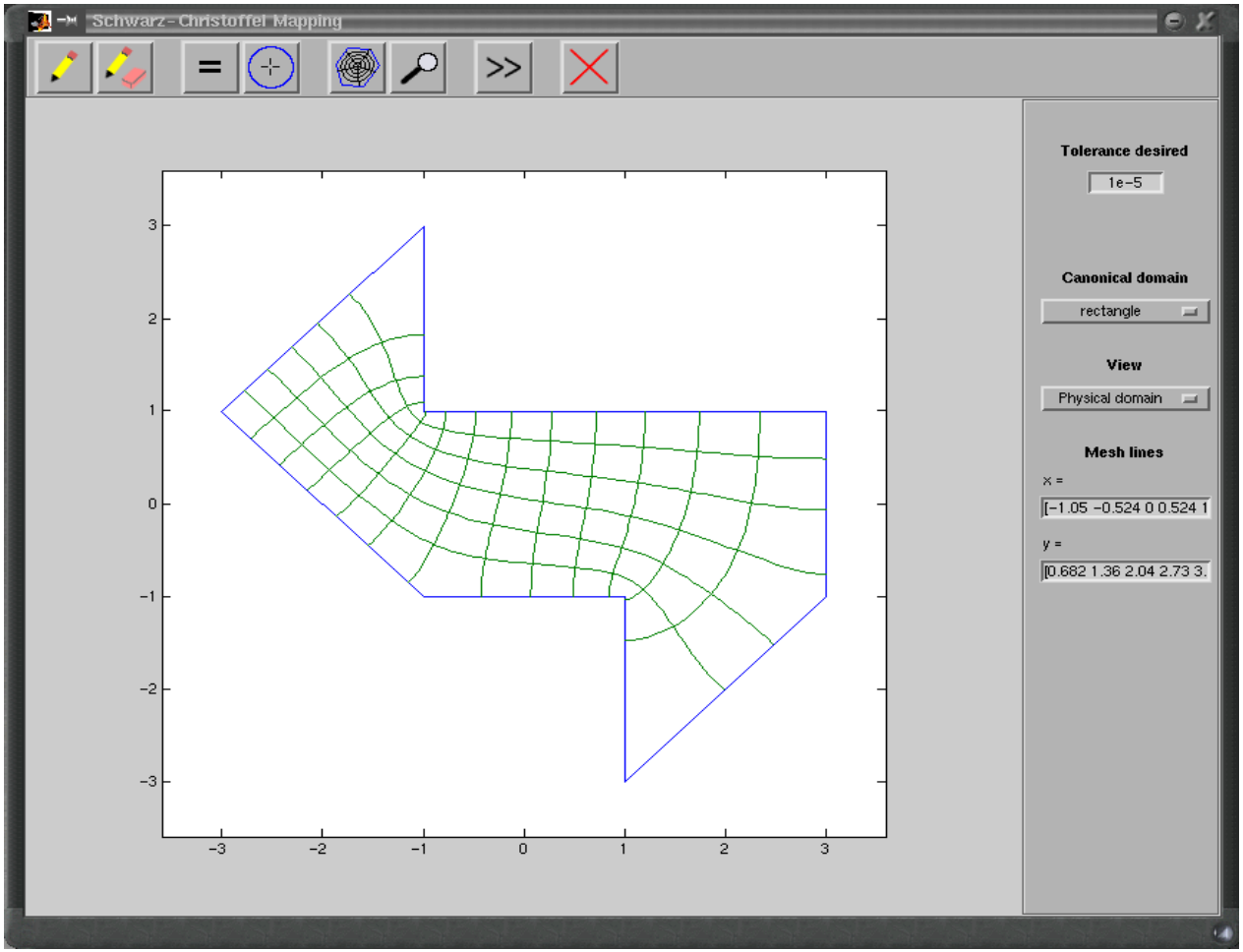
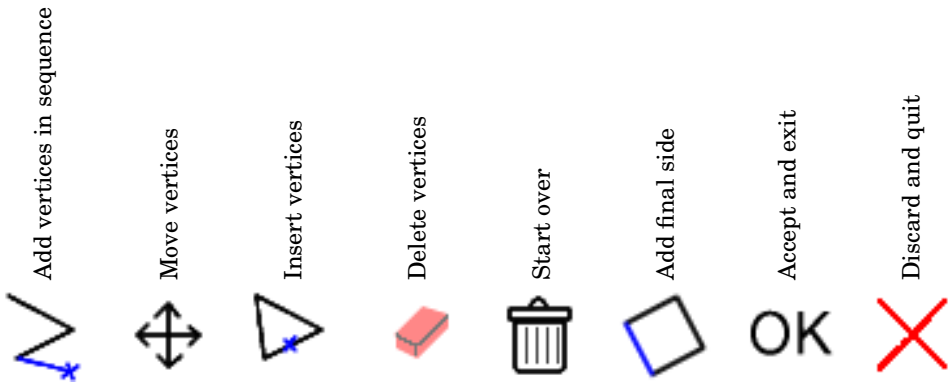


Figure 2: The interface window created by `scgui` (Linux version shown).

3.2 Polygon Editor

The "draw" and "modify" functions of the toolbox GUI bring up a new window for the Polygon Editor, which should look like Figure 3. If you are modifying a polygon, it is displayed in the axes. Otherwise the axes are empty. The buttons along the top refer to these actions:



The first four of these buttons are mutually exclusive and define the current mode of operation:

Add	Vertices are placed one at a time in order by mouse clicks. Hold the mouse button down at a location to preview the next edge of the polygon. The most recently drawn vertex is shown in red.
Move	Click and drag a vertex to a new location. You may move only finite vertices, and they must stay inside the axes box.
Insert	Click on an existing polygon side to create a new vertex on it. This vertex may then be moved to introduce a new corner. Vertices may be inserted on finite and infinite edges.
Delete	Click on a vertex to remove it. Only finite vertices may be deleted.

During the **Add** mode, a click outside the axes box begins defining an infinite vertex. You must immediately click again outside the box in order to start the "return" edge. The following vertex must be finite, and you cannot place it in a location that would cause the return edge to intersect the previous edge at a finite point. The preview edge will reflect this restriction and limit you to valid points.

To finish or "close" the polygon, you can either click on the first vertex while in **Add** mode or use the "final side" button. At this point the last edge is drawn and **Add** mode becomes permanently disabled (even if the last vertex is subsequently deleted). The other editing modes remain available for making changes.

Moving a neighbor of an infinite vertex causes the infinite edge to move as well so as to keep the angle at infinity constant. Deleting a neighbor of infinity changes the angle at infinity so that only the infinite edge moves. You also cannot delete a finite vertex that is doubly adjacent to infinity. (The other toolbox routines cannot deal with adjacent infinite vertices anyway.)

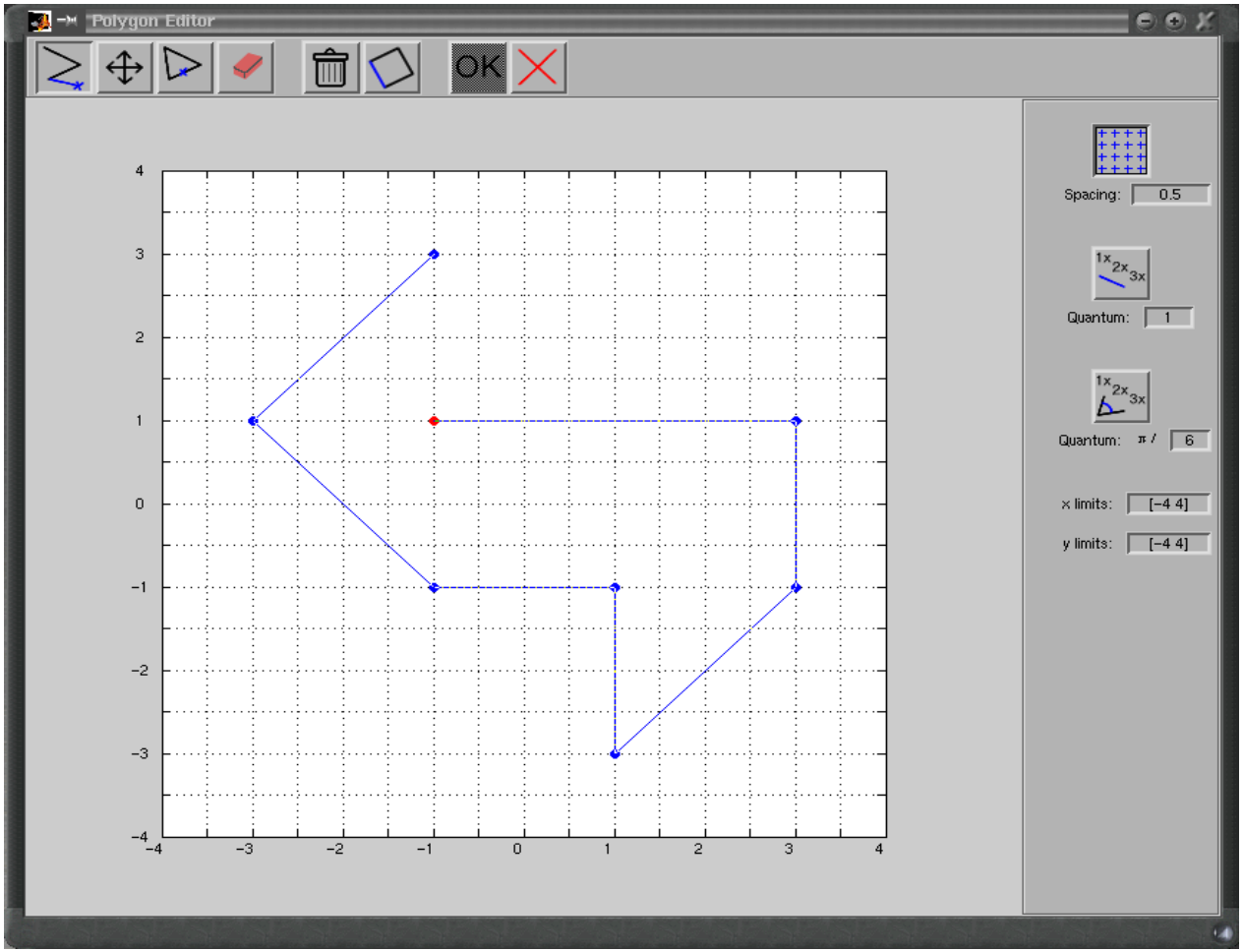


Figure 3: The Polygon Editor, after some vertices have been created (Linux version shown).

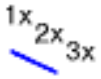
It is not difficult in adding, moving, and deleting vertices to create something which is not a valid polygon. One notable example is when a formerly infinite vertex implicitly becomes finite as defined by the intersection of edges. The toolbox will not be able to make sense of this situation, and the Polygon Editor does little to stop you from creating it.

When you are satisfied with the polygon, click on **OK** to return to the main GUI, where your polygon will replace any other data that might exist. You can click on **Start over** at any time to erase all the vertices and reenter the **Add** mode. Clicking on **Quit** will return to the main GUI without changing any data already there.

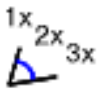
Restricting vertex placement

The three top controls on the right of the Polygon Editor restrict the placement of vertices in order to create regularity:

Snap all vertices onto a discrete grid. The spacing of the grid can be controlled by editing the field below the button. Snapping also affects clicks outside the axes box (infinite vertices). It is deactivated when the other restriction modes below are in use.



Discretize the side lengths of the polygon. The quantum of the discretization may be edited. This feature is deactivated when snapping is activated or while adding an infinite vertex, and it is disabled when in **Move** mode.



Discretize the angles of the polygon. The quantum of the discretization may be edited in terms of fractions of 180°. This feature is deactivated when snapping is activated, and it is disabled when in **Move** mode.

In all cases the preview line (displayed when the mouse button is held down in **Add** mode) reflects any restrictions.

4 Command-line interface

Each polygon or SC map is an **object** belonging to a particular **class**. A class has a **constructor**, used to create objects of that class, and **methods**, which define all the operations known for objects of the class. You can use the `methods` command to get a listing of methods; e.g., `methods polygon`.

The classes of significance to a user are shown in Figure 4. The `scmap` class is a "parent" class for all the specific SC map classes. You will not normally create objects of this class directly.

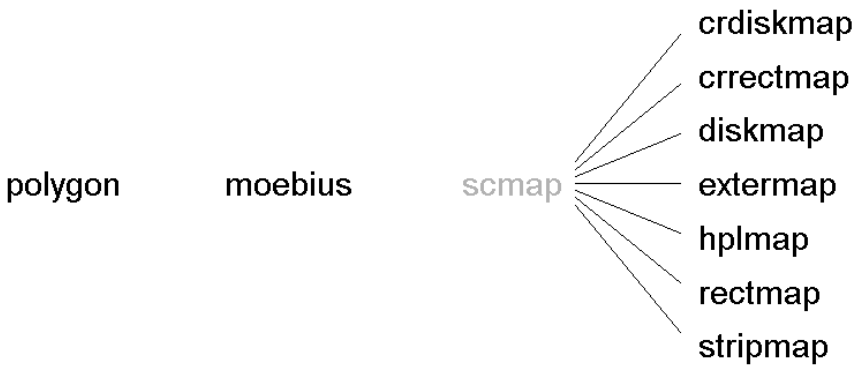


Figure 4: Classes in the SC Toolbox.

Below we describe the capabilities and typical usage of the classes. The most general syntax reference can be found in the online help.

4.1 Polygons

A polygon object holds vertex and angle data. If the polygon is finite (i.e., has no infinite vertices), the angle data is computed automatically by the constructor. Vertices may be specified in either clockwise or counterclockwise order. For unbounded polygons, there is no way to determine the angles at an infinite vertex and its neighbors, so at least this much must be provided to the constructor. (See the "infinite vertices" demo from `scdemo` to get a feel for how to specify such angles.) Polygons may instead be drawn using the Polygon Editor by invoking `p=polyedit`. See section 3.2.

Once a polygon has been created, you may operate on it using the methods listed in Table 1. Suppose `p` is a constructed polygon. The syntax for most methods is the usual functional notation, e.g. `plot(p)`. The arithmetic operators are used in the usual infix notation to translate and scale by complex scalars, e.g. `i*p + 2`. Subscript referencing also obeys the usual MATLAB conventions, e.g. `p(1:4)` for the first four vertices.

Table 1: Methods for polygons

`polyedit`	Graphically modify the polygon.
`display`	Show the vertices as a two-column matrix.
`plot`	Plot the polygon.
`cdt`	Compute constrained Delaunay triangulation.
`length`	Return the number of vertices.
`vertex`	Return the vertices as a complex vector.
`angle`	Return the interior angles, divided by π.
`+`, `-`, `*`	Translate/scale the vertices.
`()`	Retreive or assign selected vertices.
`isinpoly`	Detect points in interior (bounded polygons only).

Table 2: Schwarz–Christoffel map constructors

Name	Canonical domain	Physical domain
`diskmap`	Unit disk	Polygon interior
`hplmap`	Upper half-plane	Polygon interior
`stripmap`	Bi-infinite strip	Polygon interior
`rectmap`	Rectangle	Generalized quadrilateral
`extermap`	Disk	Polygon exterior
`crdiskmap`	Disk (cross-ratio representation)	Bounded polygon interior
`crrectmap`	Axes-aligned polygon	Bounded polygon interior

4.2 SC maps

To create a map, you specify a polygon that determines the target region and invoke a map constructor. The constructor's name determines the canonical region and the type of map. Valid constructor names are given in Table 2. The constructor's main task is to find the correct values of the unknown parameters in the SC map. This may take anywhere from a few seconds to several minutes; the time required scales roughly as the cube of the number of vertices. A progress indicator is displayed by default.

The constructors for the SC maps have a number of possible calling sequences. For example:

- `diskmap(p)`
  Here `p` is a polygon. The map is created to the appropriate region defined by `p` (its interior, except for `extermap`), using default settings.
- `diskmap(p,options)`

Override the default options, such as desired accuracy, in finding map parameters. See the online help for `scmapopt` on how to create the `options` argument.

- `diskmap(f,p)`
  If `f` is an existing diskmap, and `p` is a polygon, find the map to `p` using the parameters of `f` as a starting point. This continuation approach is useful for rare difficult regions for which the nonlinear equations solver failed to find a solution directly.
- `diskmap(p,z)` or `diskmap(p,z,c)`
  Create a map using specified prevertices. No parameter problem is solved, and the prevertices are used even if they are not compatible with the given polygon. (In that case, a completely inaccurate map is created.) If the constant `c` is not specified, it is found by integrating along one side of the polygon.
- `diskmap(z,alpha)` or `diskmap(z,alpha,c)`
  Create a map using the specified prevertices `z` and the angle parameters `alpha`. No parameter problem is solved. The image polygon is constructed by evaluation of SC integrals. If the constant `c` is not given, it is taken to be 1.

Some notable exceptions to the patterns above are:

- In `stripmap`, you may supply an additional argument in the form of a length-two vector giving the indices of the vertices that map to the ends of the strip. If you do not, you will be prompted to select these vertices graphically.
- In `rectmap`, you may supply an additional argument in the form of a length-four vector containing the indices of the vertices that map to rectangle corners. These must be given in counterclockwise order, *with the first two describing a long rectangle edge*. If this argument is not given, you will be prompted to select the corners graphically.
- The `crdiskmap` and `crrectmap` variants do not use prevertices directly, so those calling sequences that supply them are not understood. See section 4.3.

In some cases it is trivial to change the canonical domain of an SC map using a Möbius transformation. Where applicable, the map constructors can automatically convert an existing map in this fashion, foregoing the parameter problem solution. For example, if `f` is an hplmap, `diskmap(f)` creates an equivalent diskmap to the polygon of `f`. The conversions `hplmap` to `diskmap` and `stripmap` to `diskmap` are also possible.

Once a map `f` is created, you can get an estimate of its accuracy by entering `accuracy(f)`. This integrates between neighboring prevertices (excluding infinities) and compares to the actual sides of the polygon, reporting the maximum discrepancy.

Once a map has been constructed, you can use generic function names and syntaxes to do common tasks. For example, evaluation of a map called `f` at a point `z` can be done with parenthetical notation, as `f(z)`; a plot of orthogonal grid lines can be generated by `plot(f)`. The available commands are given in detail below.

Evaluation

Suppose that `f` is an SC map and that `zp` and `wp` are complex vectors.

- `f(zp)`or `eval(f,zp)`
  Evaluates the map at the point(s) in `zp`. Input points are checked for being interior to the source domain. `Inf` may be a valid input depending on type of map.
- `eval(f,zp,tol)`
  Attempts to find values within `tol`. (You can relax, but not improve on, the tolerance suggested by `accuracy`.)
- `evalinv(f,wp)` or `eval(inv(f),wp)`
  Evaluate the inverse mapping (from polygon to canonical). Much slower than forward mapping, as it first solves an ODE and then applies a Newton iteration to the forward map. There is *no* checking of input points for validity.
- `evalinv(f,wp,tol)`
  Inverse mapping with a tolerance request.
- `evaldiff(f,zp)` or `eval(diff(f),wp)`
  Evaluate the derivative of the SC map. Very fast and accurate, since the map itself is an integral.

For inverses and derivatives it is possible to create a "dummy" object. For example, one could use the idiom `g=inv(f); g(wp)`.

It is *much* more efficient to call any of these methods on a vector of points all at once, rather than one at a time in a loop.

Plotting

- `[h,val1,val2] = plot(f)`
  Plots the image of a "natural" cartesian grid (rectilinear or polar) under the map. Ten curves per direction are chosen automatically. The return arguments are handles to the drawn lines, and the abscissae/ordinates or radii/angles of the source grid.
- `[h,val1,val2] = plot(f,num1,num2)`
  Draw `num1` and `num2` curves in the orthogonal directions.
- `h = plot(f,val1,val2)`
  Specify the source curve locations.

Points are chosen adaptively in the canonical domain in order to attempt to get smooth curves in the image. Occasionally the refinement times out and an ugly straight line segment is drawn; this is sometimes associated with a ray that terminates at the preimage of a "distant" vertex. Also, adaptive refinement is done only inside the axes box at the time plotting begins. Normally the axes box is selected automatically based on the size of the region, but if the axes are fixed and held at the time of the call then those limits are used. In short, zooming to a particular region should be done *prior* to the `plot` call.

4.3 Cross-ratio (CR) formulations

Two classes, `crdiskmap` and `crrectmap`, use a rather different internal representation and solution for the unknown map parameters. A detailed discussion of the underlying representation and method is given in [DV98].

Crowding

One of the greatest challenges in numerical Schwarz–Christoffel mapping is the **crowding phenomenon**. Crowding is a problem when one deals numerically with the map to an elongated region. Elongated polygons have prevertices which are spaced exponentially close in the half-plane or disk, becoming indistinguishable in double precision when the local aspect ratio exceeds about 20. Even for lesser aspect ratios, the parameter problem can become exceedingly difficult to solve numerically. [5]

The traditional response to crowding is to choose a different canonical domain. Thus, the strip and rectangle maps are useful when the target region has a single principal direction of elongation. These variations work well in such situations.

For multiply elongated regions, it is possible to choose other canonical domains, such as slit strips [DV98, How94]. But this technique has several severe drawbacks, and the CR formulation was created as a universal alternative. If a `diskmap` construction gives multiple warnings about "severe crowding," or the convergence seems to bog down for a long time, try `crdiskmap` instead.

Usage

Externally `crdiskmap` is just like `diskmap`, and the resulting map can be used to map points or draw a plot in the same way. The display of a `crdiskmap` is different, reflecting the different internal representation.[6] You can use `diskmap(f)` to convert a `crdiskmap` into a `diskmap`, but because of crowding, the resulting map may not be accurately computable everywhere.

You should also be aware that as a preprocessing step, `crdiskmap` may add trivial vertices along the edges of the original polygon. There is no way to suppress this behavior, since it is crucial to the success of the algorithm. For regions with many elongations, the number of additional vertices may be considerable, adding to the computational time.

Rectified maps

Because of ill-conditioning, the disk is not an ideal fundamental domain for an elongated region, even if one can compute the map accurately. For instance, if one wants to generate an orthogonal grid using the disk, one must include points exponentially close to the boundary of the disk in order to get a reasonable distribution of grid points in the image.

As an alternative, we can use the prevertices determined in a disk map and change the angles (i.e., exponents) in the SC formula (1). The result will be a new polygon that is conformally equivalent to the original, being related to it by one inverse and one forward SC map. If we use only the exponents $\alpha = 0.5$, 1, 1.5, and 2, the new polygon will have all right angles and, after rotation, all sides parallel to the cartesian axes. We then refer to the composite map (axes-parallel polygon to original target) as a **rectified** map. Axes-parallel polygons are convenient for grid generation or finite differences.

[5] The crowding problem can occur for exterior maps as well. The protoypical example is the map to the exterior of a long, thin, U-shaped region.

[6] You are not given prevertex positions, but instead $n - 3$ cross-ratios of 4-tuples of them. These 4-tuples form overlapping quadrilaterals among the vertices.

Table 3: Methods for Möbius maps

`display`	Print the transformation in fractional-linear form.
`double`	Convert to a length-4 vector of constants.
`()` or `eval`	Evaluate at point(s).
`inv`	Return the inverse of the map.
`+`, `-`, `*`, `/`	Translate/scale image by a complex constant.

Note that, once the angles are specified, there is no control over the side lengths, save for a global scaling. These lengths are determined automatically; see [DV98] for details.[7] It may happen that the rectified polygon is self-intersecting.

The `crectmap` class creates rectified maps from the CR formulation. The constructor may be called with an existing `crdiskmap` or a polygon. In the latter case, the CR parameter problem is solved first. There is no unique, or even obviously "natural," choice of the angles in a rectified polygon. You may supply them as a vector argument in the constructor call. Otherwise, a side-by-side figure of target and rectilinear domains will be created, and you can assign angles graphically. There is a constraint: that $\sum \alpha_j = n - 2$. When this constraint is satisfied, you can press the **Recompute** button to determine and display the rectilinear polygon. This is much faster than solving the parameter problem, so you are free to experiment with different sets of angles.

4.4 Möbius transformations

A `moebius` class is defined for the determination and application of Möbius transformations. There are two ways to create such maps:

1. Specify two triplets of corresponding source/image points (optionally including `Inf`), or
2. Give the constants in the transformation explicitly.

The methods for `moebius` maps are shown in Table 3. The notation `f(z)`, if `f` is a `moebius` map and `z` is a scalar or vector, evaluates `f` at `z`.

4.5 Composite maps

In some applications it is useful to apply more than one SC map. For example, the map between two polygons can be found as one forward and one inverse SC map. In other cases one may want to use Möbius transformations or other transformations as additional processing steps.

As a convenience for evaluating such composite maps, the SC Toolbox allows you to define a `composite` map object. The syntax is

```
f = composite(f1,f2,...,fn);
```

[7] In the important case where four of the α's are 0.5 and the others are 1, the rectilinear polygon is a rectangle and the side lengths determine the the rectangle's aspect ratio, or conformal modulus.

Table 4: Methods for examining map data.

`center`	Return/set the conformal center (for maps from the disk).
`polygon`	Return the polygon that defines the target domain.
`parameters`	Return a structure holding the SC parameters.
`scmapopt`	Return the options used for the solution of the parameter problem.
`get`	Alternative to the above.

where any number of member maps are allowed. Each member map must be of one of these types:

- any SC map (`diskmap`, `crdiskmap`, etc.);
- the inverse of an SC map (created using `inv`);
- a `moebius` map (see section 4.4); or
- an `inline` function of one variable.

Once the composite `f` is created, you can evaluate it at `z` with the natural syntax `f(z)`. Of course, you are responsible for making sure each member map can handle the output of the preceding one. For composites that do not involve any `inline` functions, the inverse composition may be created using `inv`.

4.6 Low-level access

The object-oriented interface, in which commands like `plot` and `eval` do the right thing for the type of map passed to them, is suitable for the most common uses of SC maps. However, there are applications in which, say, the prevertices are of interest or particular SC integrals must be evaluated directly. Such details are hidden from the user by default, but they are all available.

Once created, a map object encapsulates all the information needed to compute with it. To retrieve this information, use the methods listed in Table 4. In addition to the syntax `polygon(f)`, `parameters(f)`, etc., the toolbox supports the familiar `get(f,'polygon')` and `get(f,'prevert')`. The methods operating on a map access this information and call low-level routines that do the actual work. These functions are contained in the top-level `sc` directory. Each map type has its own set of low-level functions beginning with a one- or two-letter prefix, e.g. `deplot` to plot disk-exterior maps. The low-level functions themselves are documented, so you should be able to call them directly. Of particular interest are the `xxquad` functions that numerically evaluate SC integrals. These functions need to be used with some care; look at the `xxparam` and `xxplot` functions for tips and caveats.

5 Applications

5.1 Laplace's equation

SC maps can be used to solve Laplace's equation on polygonal regions, subject to piecewise-constant Dirichlet and homogeneous Neumann boundary conditions. Computationally the effort is not much greater than finding an SC map to the region. A function `lapsolve` is provided for this purpose.

`phi = lapsolve(p,bdata)`

Here `p` is a polygon or an `hplmap` to a polygon taken to be the problem domain. (The latter saves time when several different BVPs are to be solved on the same region.) The vector `bdata` specifies the boundary condition on each side; a numerical value represents a Dirichlet condition and `NaN` represents a zero Neumann condition.

`phi = lapsolve(p)`

In this version a graphical interface is created in order to let you assign the boundary values interactively.

In either case the resulting `phi` is a `composite` (see section 4.5) object consisting of three steps: an inverted SC map from the problem domain to the disk; a second SC map to a "rectified" region whose sides align with the coordinate axes; and extraction of the real part of the result. (For mathematical details see Chapter 5 of [DT02].) In practice one can use the syntax `phi(z)` to compute `phi` at the points in `z`.

To aid the creation of plots of the solution, a triangulation routine for the interior of a polygon is offered in the `polygon` class. A simple calling sequence would be

```
phi = lapsolve(p);
[tri,x,y] = triangulate(p);
trisurf(tri,x,y,phi(x+i*y))
```

There are some important caveats about these functions:

- No allowance is made for crowding; hence `lapsolve` will likely fail on elongated regions.
- `triangulate` is based on the built-in `delaunay` function, which does not really support nonconvex regions. In practice `triangulate` should work fine unless the polygon has a slit. In that case the solution computed by `lapsolve` is valid, but visualization as suggested above will have problems.
- Very close to corners with small interior angle (say, $\pi/6$ or less), the inverse mapping step in `phi` may fail to give an accurate result. (The issue is a small basin of attraction for Newton's method due to the nearby singularity in the map.) A fix for this problem is still being investigated.

5.2 Faber polynomials

```
F = faber(f,k)
```

This function uses the Schwarz–Christoffel exterior map to compute the coefficients of **Faber polynomials** for a polygon. A Faber polynomial is the analytic part of the Laurent series of a power of the map. These polynomials have a number of interesting properties and uses [Ell86, Mar65, SV93].

The parameter `k` is the highest degree of the polynomials being sought. The output `F` is a cell array of length `k+1`. Each element of the array is a vector of polynomial coefficients. Thus, `F(j)` is a vector of length `j` representing a polynomial of degree `j-1`. The coefficients are in decreasing order of power (see `polyval`), and the first coefficient is always real.

You can also call `faber` on a polygon, in which case an `extermap` is computed first.

6 Additional notes

6.1 Slow convergence

The nonlinear systems encountered in the solution of the parameter problem can bog down or even halt the numerical solution. There is no quick way to choose a good initial guess. You may notice very slow progress, or even failure to reach the requested tolerance. It is often possible to speed matters via continuation from a less difficult target region, or to use `crdiskmap` if the difficulty may be caused by elongations in the polygon.

6.2 Further reading

The book [DT02] is meant as a reference for and introduction to theoretical and practical issues regarding SC maps. For introductory material about the transformation, see [CB90, Hen74, SS93]. The theory and implementation of many variations of the transformation can be found in [Däp88, Dav79, Flo86, FZ87, Hoe86, How90, How94, HT90, Pea91, Rep79, SD85, Woo61]. Two surveys of variations and applications are [Tre93, TD98].

The SC Toolbox implements and extends the methods described by Trefethen in his appendix to [SS93]. For more on the design of SCPACK, which is very similar, see [Hen74, Tre80, Tre89]. A paper specifically on the toolbox appeared in [Dri96]. The cross-ratio formulation is introduced in detail in [DV98].

A FORTRAN implementation of SC mapping for doubly-connected regions bounded by polygons called DSCPACK is due to Hu [Hu95].

For more on numerical conformal mapping in general, see [Hen86, Tre86]. An excellent FORTRAN package called CONFPACK [Hou90] is available from `netlib.org` for maps to regions with piecewise smooth boundary.

6.3 Suggestions and bug reports

I have tried to make the SC Toolbox robust, but I do not explicitly or implicitly warrant its accuracy or reliability. Also, The MathWorks, Inc. is not in any way responsible for the SC Toolbox's design or maintenance.

I welcome complaints, suggestions, inquiries, and bug reports related to any aspect of the SC Toolbox. I can be reached at `driscoll@math.udel.edu`. Feel free to report bugs, make suggestions, or describe an interesting application.

I reserve copyright on the M-files and this guide. You may change or add to the toolbox's code for your own use. However, I request you redistribute only unmodified versions of the toolbox. I am always glad to receive corrections and additions that I can incorporate into the package.

6.4 Acknowledgments

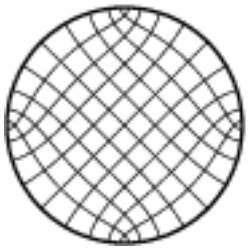
This material is based upon work supported at various times by a National Science Foundation Graduate Research Fellowship, by NSF Grant DMS-9116110, by DOE grant DE-FG02-9YER25199, and by an NSF Mathematical Sciences Postdoctoral Research Fellowship. In particular this work would not have been possible without the NSF.[8]

I would like to thank Nick Trefethen for leading me to this project, guiding and encouraging me throughout its development, and serving as Chief Beta-Tester-For-Life. I also thank Steve Vavasis for his insights leading to the cross-ratio formulations.

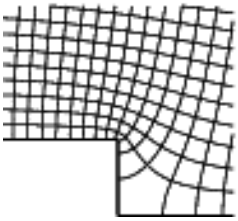
[8] Any opinions, findings, conclusions or recommendations expressed in this publication by the author do not necessarily reflect the views of the National Science Foundation.

7 More examples

```
p = polygon([1+i,-1+i,-1-i,1-i]);
f = diskmap(p);
f = center(f,0);
plot(exp(i*linspace(0,2*pi,180)));
hold on, axis equal
[X,Y] = meshgrid((-4:4)/5,(-100:100)/100);
plot(evalinv(f,X+i*Y),'k')
[X,Y] = meshgrid((-100:100)/100,(-4:4)/5);
plot(evalinv(f,X'+i*Y'),'k')
```

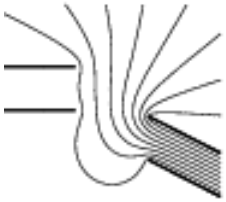


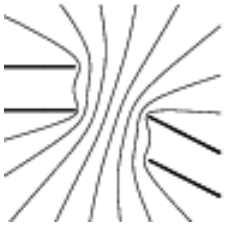
```
p = polygon([i,-i,Inf],[3/2,1/2,-1]);
f = hplmap(p);
axis([-3 3 -1.5 4.5]), hold on
plot(f,0.7*(-10:6),0.7*(1:12))
```



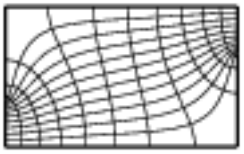
```
p = polygon([-4-i,4-i,4+i,-4+i]);
f = diskmap(p);
f = center(f,0);
plot(f,0.2*(1:4),angle(prevertex(f)))
```

```
w = [Inf,-1-i,-2.5-i,Inf,2.4-3.3i,...
    Inf,2.4-1.3i,Inf,-1+i,-2.5+i];
alf = [0,2,1,-.85,2,0,2,-1.15,2,1];
p = polygon(w,alf);
f1 = stripmap(p,[6 8]);
f2 = stripmap(p,[4 8]);
axis([-4.3 5.7 -6.15 3.85]), hold on
plot(f1,0,8)
pause, cla
plot(f2,0,8)
```

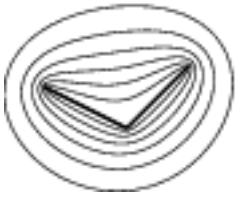




```
p = polygon([-5-i,-5-3i,5-3i,5+i,5+3i,-5+3i]);
f = rectmap(p,[1 2 4 5]);
plot(f)
```

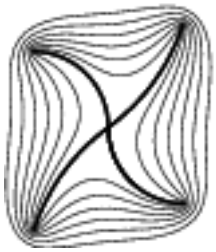


```
p = i*polygon([-0.5,1-1.5i,-0.5,0.5+2i]);
f = extermap(p);
axis([-3.05 2.8 -2.3 2.5]), hold on
plot(f,(4:9)/10,0)
```

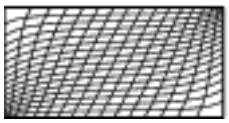


```
f = rectmap(drawpoly);
r = rectangle(f);
M = pi/max(imag(r));
a = exp(min(real(r))*M);
b = exp(max(real(r))*M);
rad = log(linspace(a,b,8))/M;
plot(f,rad(2:end-1),6);
H = copyobj(get(gca,'child'),gca);
for n=1:length(H),
  set(H(n),'xdata',-get(H(n),'xdata'))
end
set(H,'linesty','--'), axis auto
```

```
p = polygon([1+i,1+2i,Inf,-.705+.971i,...
    Inf,-1-i,Inf,.705-.971i,Inf],...
    [2,1,-.3,2,-.7,2,-.3,2,-.7]);
f = center(diskmap(p),0);
axis(5.5*[-1 1 -1 1]), hold on
h = plot(f,0.7:0.05:0.95,0);
for n=1:length(h)
  w = (get(h(n),'xd') + i*get(h(n),'yd'));
  set(h(n),'xd',real(1./w),'yd',imag(1./w))
end
h = findobj(gca,'color',[0 0 1]);
for n=1:length(h)
  w = get(h(n),'xd') + i*get(h(n),'yd');
  if abs(w(1)) > abs(w(2)), w=w([2 1]); end
  u1 = w(1) + linspace(0,2*diff(w),100);
  u2 = w(1) + linspace(2*diff(w),100*diff(w),100);
  plot(1./[u1 u2]);
end
delete(h)
axis auto
```



```
p = polygon([i 0 2 2+i]);
theta = [1/2 1/6 1/2 1/6];
f1 = hplmap(p);
thetas = theta([end 1:end-1]);
alf = -mod(theta-thetas,1)+1;
f2 = hplmap(prevertex(f1),alf,i);
Q = polygon(f2); w = vertex(Q);
m = min(imag(w)); M = max(imag(w));
a = max(real(w)); t = [.01 1:99 99.99]/100;
plot(p), hold on
[X,Y] = meshgrid((1:15)*a/16,t*m);
plot(f1(evalinv(f2,X+i*(Y+M*X/a))),'k')
[X,Y] = meshgrid(t*a,(1:15)*m/16);
plot(f1(evalinv(f2,X'+i*(Y'+M*X'/a))),'k')
```



```
f = extermap(drawpoly);
phi = faber(f,8);
[x,y] = meshgrid(linspace(-4,4,80));
w = polyval(phi{9},x+i*y);
contour(x,y,abs(w),[1 1]);
hold on
plot(polygon(f))
```

References

[CB90] R. V. Churchill and J. W. Brown. *Complex Variables and Applications*. McGraw-Hill, 5th edition, 1990.

[Däp88] H. Däppen. *Die Schwarz–Christoffel-Abbildung für zweifach zusammenhängende Gebiete mit Anwendungen*. PhD thesis, ETH Zürich, 1988.

[Dav79] R. T. Davis. Numerical methods for coordinate generation based on Schwarz–Christoffel transformations. In *4th AIAA Comput. Fluid Dynamics Conf.*, pages 1–15, Williamsburg, VA, 1979.

[Dri96] T. A. Driscoll. A MATLAB toolbox for Schwarz–Christoffel mapping. *ACM Trans. Math. Soft.*, 22:168–186, 1996.

[DT02] T. A. Driscoll and L. N. Trefethen. *Schwarz–Christoffel Mapping*. Cambridge University Press, Cambridge, UK, 2002.

[DV98] T. A. Driscoll and S. A. Vavasis. Numerical conformal mapping using cross-ratios and Delaunay triangulation. *SIAM J. Sci. Comput.*, 19:1783–1803, 1998.

[Ell86] S. W. Ellacott. A survey of Faber methods in numerical approximation. *Comp. & Maths. with Appls.*, 12B:1103–1107, 1986.

[Flo86] J. M. Floryan. Conformal-mapping-based coordinate generation method for flows in periodic configurations. *J. Comput. Phys.*, 62:221–247, 1986.

[FZ87] J. M. Floryan and C. Zemach. Schwarz–Christoffel mappings: A general approach. *J. Comput. Phys.*, 72:347–371, 1987.

[Hen74] P. Henrici. *Applied and Computational Complex Analysis: Power Series, Integration, Conformal Mapping, Location of Zeros*, volume 1. Wiley, 1974.

[Hen86] P. Henrici. *Applied and Computational Complex Analysis: Discrete Fourier Analysis, Cauchy Integrals, Construction of Conformal Maps, Univalent Functions*, volume 3. Wiley, 1986.

[Hoe86] M. Hoekstra. Coordinate generation in symmetrical interior, exterior, or annular 2D domains, using a generalized Schwarz–Christoffel transformation. In J. Hauser and C. Taylor, editors, *Numerical Grid Generation in Computational Fluid Mechanics*. Pineridge Press, 1986.

[Hou90] D. M. Hough. User's guide to CONFPACK. ETH Zürich IPS Research Report 90-11, 1990.

[How90] L. H. Howell. *Computation of Conformal Maps by Modified Schwarz–Christoffel Transformations*. PhD thesis, MIT, 1990.

[How94] L. H. Howell. Schwarz–Christoffel methods for multiply-elongated regions. In *Proc. of the 14th IMACS World Congress on Computation and Applied Mathematics*, 1994.

[HT90] L. H. Howell and L. N. Trefethen. A modified Schwarz–Christoffel transformation for elongated regions. *SIAM J. Sci. Stat. Comput.*, 11:928–949, 1990.

[Hu95] C. Hu. User's guide to DSCPACK. Nat. Inst. Aviation Res. 95-1, Wichita State Univ., 1995.

[Mar65] A. I. Markushevich. *Theory of Functions of a Complex Variable*. Prentice–Hall, Englewood Cliffs, NJ, 1965. Second edition issued in 1985 by Chelsea, New York.

[Pea91] K. Pearce. A constructive method for numerically computing conformal mappings for gearlike domains. *SIAM J. Sci. Stat. Comput.*, 12:231–246, 1991.

[Rep79] K. Reppe. Berechnung von Magnetfeldern mit Hilfe der konformen Abbildung durch numerische Integration der Abbildungsfunktion von Schwarz–Christoffel. *Siemens Forsch. u. Entwickl. Ber.*, 8:190–195, 1979.

[SD85] K. P. Sridhar and R. T. Davis. A Schwarz–Christoffel method for generating two-dimensional flow grids. *J. Fluids Eng.*, 107:330–337, 1985.

[SS93] E. B. Saff and A. D. Snider. *Fundamentals of Complex Analysis*. Prentice Hall, 2nd edition, 1993.

[SV93] G. Starke and R. S. Varga. A hybrid Arnoldi–Faber iterative method for nonsymmetric systems of linear equations. *Numer. Math.*, 64:213–240, 1993.

[TD98] L. N. Trefethen and T. A. Driscoll. Schwarz-Christoffel mapping in the computer era. In *Proceedings of the International Congress of Mathematicians, Vol. III (Berlin, 1998)*, volume 1998, pages 533–542 (electronic), 1998.

[Tre80] L. N. Trefethen. Numerical computation of the Schwarz–Christoffel transformation. *SIAM J. Sci. Stat. Comput.*, 1:82–102, 1980.

[Tre86] L. N. Trefethen, editor. *Numerical Conformal Mapping*. North-Holland, Amsterdam, 1986. Reprint of J. Comput. Appl. Math. **14** (1986), no. 1–2.

[Tre89] L. N. Trefethen. SCPACK user's guide. MIT Numerical Analysis Report 89-2, 1989.

[Tre93] L. N. Trefethen. Schwarz–Christoffel mapping in the 1980's. Cornell University Computer Science Department Technical Report TR 93-1381, 1993.

[Woo61] L. C. Woods. *The Theory of Subsonic Plane Flow*. Cambridge Univ. Press, 1961.