Schwarz–Christoffel Toolbox User's Guide

Version 2.1

Tobin A. Driscoll[*]

1 Introduction

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 5.2 or later. (Versions of the toolbox prior to 2.0 run in version 4 of MATLAB.) The toolbox is a descendant of SCPACK, a Fortran package developed by L. N. Trefethen in the early 1980's [23, 25]. However, the SC Toolbox is interactive and graphical, requires no programming by the user, and has many capabilities not in SCPACK.

1.1 Definitions

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 $w_1, \dots, w_n$, and the numbers $\alpha_1\pi, \dots, \alpha_n\pi$ are the interior angles at the vertices.[2] The pre-images of the vertices, or **prevertices**, are real and denoted by $z_1, \dots, z_n$. They satisfy

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

Figure 1 illustrates these definitions.

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

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

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

[*]Department of Applied Mathematics, University of Colorado, Boulder, CO 80309-0526 (`driscoll@na-net.ornl.gov`).

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

[2]Earlier versions of the toolbox used $\beta_1, \dots, \beta_n$, 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. See Figure 1.

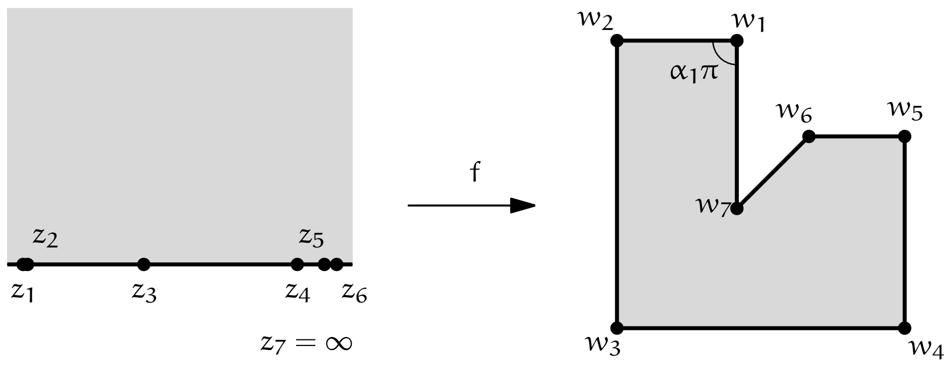


Figure 1: Notational conventions for the Schwarz–Christoffel transformation. In this case z_1 and z_2 are mathematically distinct but graphically difficult to distinguish.

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

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

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 freedom, three of the prevertices, including the already fixed z_n, may be be chosen arbitrarily. The remaining $n-3$ 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 numerical 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 bi-infinite strip $0 \le \Im z \le 1$ or a rectangle, in which cases the integrand involves hyperbolic sines. These two variations are particularly important when the target region is highly elongated in one direction.

Still another instance 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.

1.2 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

- Graphical input of polygons
- Computation of forward and inverse maps
- Adaptive plotting of images of orthogonal meshes
- Graphical and object-oriented user interfaces

1.3 Requirements

The most recent edition of the SC Toolbox will not fully run under a MATLAB version eariler than 5.2.[4] 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 as intended on all platforms. Please let me know if it is unusable on your platform.

Full use of the SC Toolbox requires some understanding of MATLABfundamentals such as vectors, matrices, functions, and graphics. A basic understanding of the use of classes and objects would be helpful, but probably not necessary.

The graphical user interface should be accessible to MATLAB novices and experts alike.

2 Obtaining and installing the SC Toolbox

The latest version of the SC Toolbox is available over the Web from the URL

`http://amath.colorado.edu/appm/faculty/tad/SC-toolbox/`

If you cannot reach this page, please send mail to the author at `driscoll@na-net.ornl.gov`. The toolbox M-files (distributed as a zip archive) and this Guide are available.

By default, the toolbox is installed in a directory tree rooted at the name `sc`. When you want to use the SC Toolbox, you must first use MATLAB's `cd` command to change to the `sc` directory. You may also add the `sc` directory to the search path using `addpath` or `editpath`. The latter gives you the option of automatically including the directory at the beginning of every MATLAB session.

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

[4]On PC-Windows platforms, the shipped version 5.2 has a bug in displaying graphical interfaces. Download the 5.2.1 patch from `www.mathworks.com`, or upgrade.

3 Overview of the Toolbox

The SC Toolbox implements five variations of the Schwarz–Christoffel transformation: mapping a half-plane, disk, strip, or rectangle to the interior of a (possibly unbounded) polygon, and mapping a disk to the exterior of a polygon. In addition, a variant method for the disk case is provided for use with certain ill-conditioned target regions, and this variant has its own extension as well.

To create a map, you specify a polygon that determines the target region and invoke a special function known as a **constructor**. The constructor's name determines the canonical region and the type of map. The valid constructor names are given in Table 1. 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 as the cube of the number of vertices. A progress indicator is displayed by default.

Table 1: 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

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 `map` at a point `z` can be done with parenthetical notation: `map(z)`. Or a plot of orthogonal grid lines can be generated by `plot(map)`. The available commands are given in detail below.

Polygons themselves are created with the constructor `polygon`. For bounded polygons, all that must be supplied is a vector of complex vertices. For unbounded (open) polygons, some angle information must also be provided. Polygons too can be displayed using `plot`.

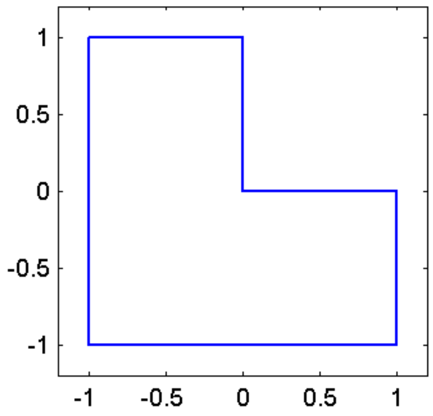
Most of the functionality of the Toolbox is accessible without typing commands at all. Enter `scgui` to start a graphical user interface (GUI) that allows you to draw polygons, construct maps, make plots, and map points. Once you understand the progression 1) polygon, 2) map parameters, and 3) applications of the map, the interface should be fairly intuitive.

3.1 Example: Command line interface

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

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

p =

     0     1
```

```
    -1     1
    -1    -1
     1    -1
     1     0
     0     0

>> plot(p)
```



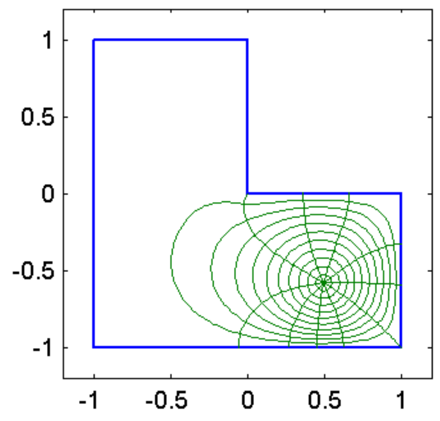
Next, we construct the disk map.

```
>> 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.

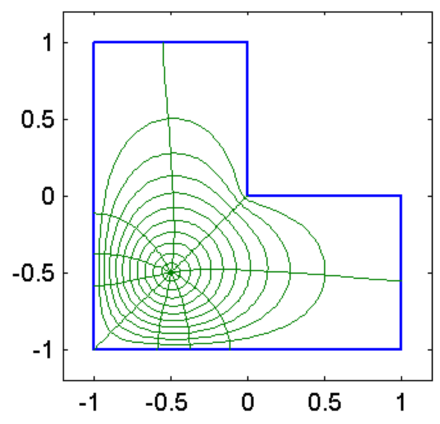
```
>> plot(f)
```



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, by Möbius transformation of the disk, it is a simple 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)
```



Notice how the prevertices and constant of the map have changed.

More examples and tutorials are available by running `scdemo`.

4 User's reference

Each polygon or SC map is an **object** belonging to a particular **class**. A class has a **constructor**, used to create objects, 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 2. The `scmap` class is a "parent"

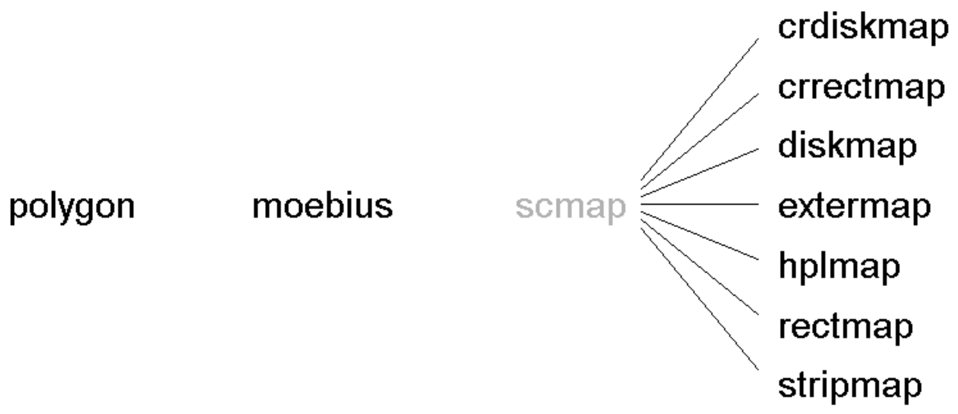


Figure 2: Classes in the SC Toolbox.

class for all the specific SC map classes. You will not normally create objects of this class directly.

Below we detail the capabilities and usage of the classes. We do not provide general syntax reference here—for that, use the online `help`.

4.1 Polygons

A polygon 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.[5] 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.)

Once a polygon has been created, you may operate on it using the methods listed in Table 2. Suppose `p` is a created polygon. The syntax for most of these is the usual functional notation, e.g. `plot(p)`. The arithmetic operators are used in the usual infix notation to add 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.

Polygons can be drawn using the mouse rather than typed in. See section 4.6.

[5]On its own, a polygon does not have an "interior" or "exterior," so the ordering has no significance.

Table 2: Methods for polygons

`modify`	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).

4.2 Möbius transformations

As a convenience, a `moebius` class is defined for application of Möbius transformations. There are two ways to create such maps:

1. Specifying two triplets of corresponding points, or
2. Giving the constants in the transformation explicitly.

In case 1, `Inf` is a valid input as a source or image point.

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`.

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.

4.3 SC maps

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, for all except `extermap` types), using default settings.

- `diskmap(p,options)`
  Override the default options, such as desired accuracy, in finding map parameters. See `scmapopt` to create the `options` argument.
- `diskmap(f,p)`
  If `f` is an existing map of the same type, and `p` is a polygon, find the map to `p` using the parameters of `f` as a starting point. This is useful for 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 taken to be 1.

Some notable exceptions to the pattern above:

- In `stripmap`, an additional argument is a two-vector giving the indices of the vertices that map from the ends of the strip. If this is missing, the user must select them graphically.
- In `rectmap`, an additional argument is a length-four vector containing the indices of the vertices that map to rectangle corners. These must be in counterclockwise order, with the first two describing a long rectangle edge. If this is missing, the user must select them graphically.
- The `crdiskmap` formulation does not use prevertices directly, so those calling sequences are undefined. The `crrectmap` is also different from other types. See section 4.4.

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 convert an existing map in this fashion, sparing the parameter problem solution. For example, `diskmap(f)` creates a diskmap to the polygon of the hplmap `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, returning the maximum difference.

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`. (May relax, 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,valc1,valc2] = plot(f)`
  Plots the image of a "natural" cartesian grid (rectilinear or polar) under the map. Ten lines 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,valc1,valc2] = plot(f,numc1,numc2)`
  Draw `numc1` and `numc2` curves in the orthogonal directions.
- `h = plot(f,valc1,valc2)`
  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 reaches a maximum 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 for smooth curves.

4.4 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 [5].

Crowding

One of the greatest challenges in numerical Schwarz–Christoffel mapping is the **crowding phenomenon**. Crowding is a problem anytime one deals numerically with the map from the canonical domain 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. [6]

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 appropriate situations.

For multiply elongated regions, it is possible to choose other canonical domains, such as slit strips [5, 14]. 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

For superficial purposes, use of `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.[7] You can use `diskmap(f)` to convert a `crdiskmap` into a `diskmap`, but because of crowding, the map may not be computable everywhere with accuracy.

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.

The ability of a `crdiskmap` to compute accurately at all points despite crowding is difficult to exploit directly from the current interface. As a demonstration, the function `crlapl` by S. Vavasis is provided. This solves the Laplace's equation in the polygon with piecewise constant Dirichlet boundary conditions, at a point within the polygon. If all the boundary data has the same sign, the solution will have high relative accuracy, even if it is many orders of magnitude smaller than the boundary data.

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.

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

[7]You are not given prevertex positions, but instead the cross-ratios of $n-3$ 4-tuples of them. These 4-tuples in turn are formed from overlapping quadrilaterals found in a triangulation of the vertices.

As an alternative, suppose an embedding of the prevertices is known to map to a target polygon. If we leave the prevertices fixed and change the α's that determine the angles of the Schwarz–Christoffel image, a new polygon will result. This new polygon will be conformally equivalent to the target, being related by one inverse and one forward SC map.

A convenient choice is to make all the interior angles of the new polygon multiples of right angles; that is, to take $\alpha = 0.5$, 1, 1.5, or 2 in each case. The resulting polygon (after rotation) will have all sides parallel to one of the cartesian axes, and we refer to the composite map (axes-parallel polygon to original target) as a **rectified** map. Axes-parallel polygons are convenient to work with, and they can be useful for grid generation or finite differences on a transplanted problem.

Note that, once the angles are specified, there is no control over the side lengths, save for a global scaling. (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.) These lengths are determined automatically; see [5] for details. It may happen that the rectified polygon is self-intersecting; that is, parts of the plane are covered more than once.

The `crrectmap` class is for working with rectified polygons. 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 choice of the angles in the rectified polygon, and their selection is not automated. 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 with the mouse. There is a constraint, namely, that $\sum \alpha_j = n - 2$, or that a total turn of 2π is made. 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.

The `crrectmap` class can deal with multiply sheeted (self-intersecting) rectilinear polygons.[8]

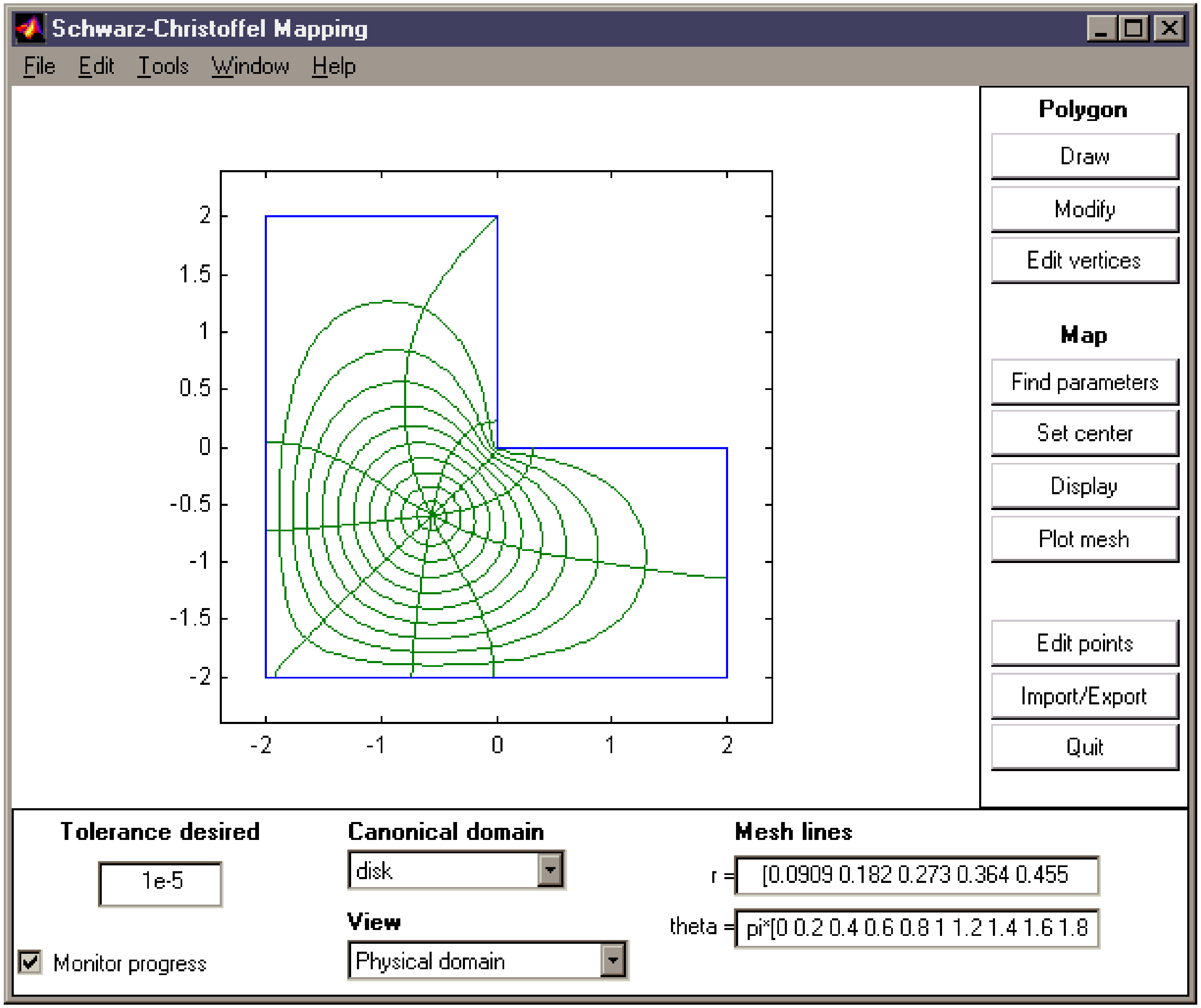
4.5 Graphical user interface (GUI)

To start the GUI, enter `scgui`. A new figure window will be created, looking like Figure 3. (There may be differences in appearance across platforms.)

The buttons, on the right edge of the window, are used to perform actions. The boxes and selections along the bottom of the window are used to control properties of the numerical solutions and plots.

In brief, the buttons take the following actions:

[8]It may occur, however, that the triangles imposed by the triangulation of the target polygon may not be inside the rectilinear polygon, and this can cause trouble when mapping. Work is ongoing to solve this difficulty.



Tolerance desired: 1e-5

[✓] Monitor progress

Canonical domain: disk

View: Physical domain

Mesh lines

r = [0.0909 0.182 0.273 0.364 0.455

theta = pi*[0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8

Figure 3: GUI configuration (Windows 98 version shown with reduced coloring).

Draw	Draw a new polygon.
Modify	Modify existing polygon.
Edit vertices	Edit the vertices in a dialog box.
Find parameters	Solve the parameter problem.
Set center	Click to select a new conformal center.
Display	Show map info in command window.
Plot	Invoke `plot` method.
Edit points	Review/edit individually mapped points (see below).
Import/Export	Transfer quantities to/from workspace variables.
Quit	Leave the GUI.

The controls along the bottom allow you to select the desired tolerance, canonical domain, and whether you want to view the progress of the parameter problem solution. The boxes labeled **Mesh lines** correspond to the arguments accepted by `plot`. After a mesh has been drawn, the values of these boxes are updated to reflect the actual curves drawn. These can be edited and used to redraw.

The **View** popup menu allows you to toggle between views of the physical domain (polygon) and the canonical one. In the canonical domain, the prevertices are marked by black dots for clarity.

Once the solution has been found, you can map individual points back and forth between the domains. Simply 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 magenta circle is drawn in each domain. You may click as many times as you like. To view the (pre)images, use the **View** popup. Pressing the **Edit points** button brings up a text box with a vector of the points in the currently viewed domain. This allows you to map precise locations, or clear the current set of points.

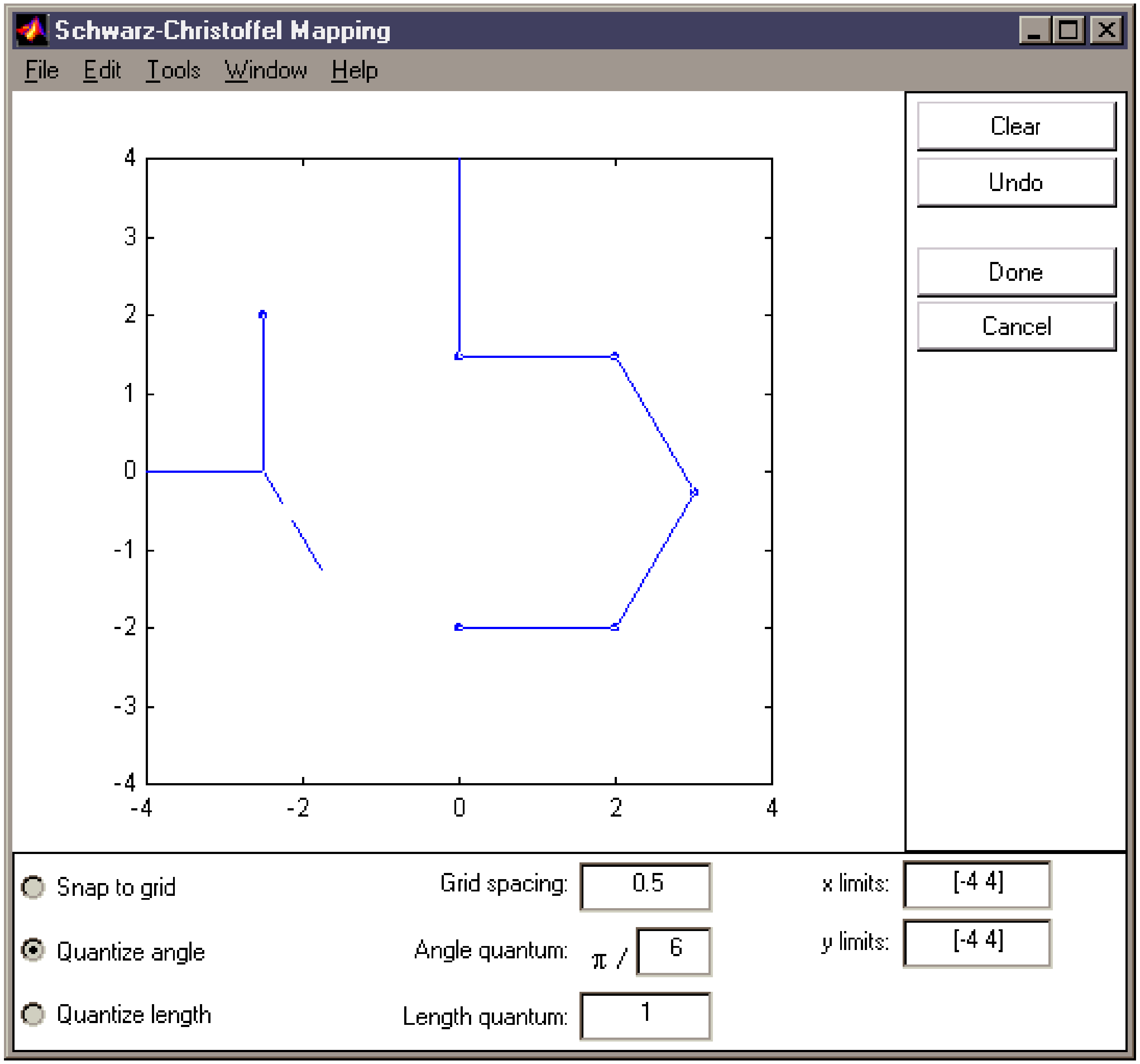
The **Import/Export** button brings up a dialog box in which you can give the polygon, map, or mapped-point vectors a name in the base workspace (accesible from the command prompt). If you click **Export**, variables in the base workspace with the given names will have the corresponding values. The **Import** button brings workspace values into the GUI.

4.6 Drawing polygons

The **Draw** button in the GUI invokes a polygon-drawing function named `drawpoly`. You can also call this function directly from the command line. The function begins by creating controls and buttons in the current figure, then waits for you to begin drawing. See Figure 4. By default the axes limits are reset to predetermined values, but if the current axes are being held and the limits have been fixed, they are not changed.

Use the mouse to move the pointer into the figure window and the pointer will become a crosshair. Move to the location of the first vertex and click the left mouse button. If the cursor was inside the axes box, the vertex will be drawn, and you can move to place the next vertex. (Clicks outside the box are for infinite vertices, as explained below.) As you move, a dashed line will follow to preview the next polygon edge. Click the left button again, and the vertex and a solid polygon edge will be drawn.[9]

[9]Movement and clicks can be slow to register with MATLAB, so proceed with some deliberation.



Snap to grid
[✓ Quantize angle]
Quantize length

Grid spacing: 0.5
Angle quantum: π / 6
Length quantum: 1

x limits: [-4 4]
y limits: [-4 4]

Figure 4: Snapshot of drawing a polygon using `drawpoly` (Windows 98 version shown with reduced coloring).

The **Undo** and **Clear** buttons remove the last and all of the vertices, respectively. To finish the polygon, you have several options:

- Place the last vertex normally, and click the **Close** button; or
- Place the last vertex, then click at the first vertex again; or
- Place the last vertex using the middle or right mouse button.

If you wish to exit drawing without the polygon being returned by the `drawpoly` function, click **Cancel** instead.

Snapping to a grid

You can place some or all of your vertices exactly on a discrete grid. This feature is controlled by the checkbox[10] labeled **Snap to grid**. When this box is activated, a grid of dotted lines is drawn to indicate the allowable vertex sites. By default, the grid is spaced by $1/16$ of the axes dimensions in each direction. You can vary the spacing by changing the value in the adjacent text box. As you move the pointer, the preview line will now end only on a grid point. You may continue to place vertices as before. When the checkbox is deactivated, the grid lines will be erased and the restriction removed.

Quantizing lengths and angles

You can also restrict side lengths and turning angles of the polygon to be multiples of a fixed amount. This can be useful for drawing parts of regular polygons. Two checkboxes in the figure window control these modes.

When side lengths are quantized, the preview line will end only at points which give the resulting side a length that is an integer multiple of a length quantum. The initial value of that quantum is $1/8$ of the axes size, and it can be changed in the adjacent box. When angles are quantized, the angle between the preview line and the most recent side is restricted to integer multiples of π/m, where m is initially 6. The angle quantization feature is particularly handy for drawing slits (cracks).

The quantization modes can be used independently or simultaneously. The quantization and grid modes are mutually exclusive, and activating one automatically deactivates the other(s).

Infinite vertices

To put a vertex at infinity, click the mouse outside the axes box at the desired exit angle. The mouse pointer will change to a cross. Now click again outside the axes box where you want the edge from infinity to return. (No preview line will follow between these clicks, since no edge actually exists.) The pointer will then revert to a crosshair.

(The **Snap to grid** feature still has effect on clicks outside the axes box, even though the grid is not drawn outside the box. Also, the length-quantization mode will be temporarily disabled, but angle quantization will work as expected.)

[10]A checkbox toggles between two states each time you click the mouse on it.

Now move to place the next (necessarily finite) vertex. There is a restriction—you may not cause the resulting edge to intersect the previous edge at a *finite* point. The preview edge will reflect this restriction and limit you to valid points.

4.7 Graphically modifying polygons

The **Modify** GUI button calls a graphical editing method called `modify`. You can also call `modify` directly, as in `q=modify(p)`. The polygon is drawn with large dots at the vertices. To move a vertex, position the cursor over it and press and hold the left mouse button. The dot changes color, and adjacent sides become dashed. Drag the vertex to its new location and release the mouse.

The button marked **Add** allows you to add a single vertex. Click on this button and the cursor changes to a crosshair. If you then click on a side of the polygon, a new vertex will be added to that side. On a finite side, the new vertex appears at the midpoint; on an infinite one, the vertex appears at a "reasonable" distance from its finite neighbor. Once the vertex has been added, the cursor reverts to a circle and the default mode is resumed. To cancel the addition before clicking on a side, press **Add** again.

The **Delete** button works similarly. The cursor changes to a fleur, and clicking on a vertex deletes it. The default mode then resumes. You cannot delete infinite vertices.

As you adjust a neighbor of an infinite vertex, the point at which the infinite side leaves the axes box moves as well, so as to keep the angle at infinity constant. To change the angle at infinity, first add a vertex to the infinite edge, move it, and then delete it. You cannot move or delete an infinite vertex, and finite vertices cannot be moved to infinity. You also cannot delete a finite vertex that is doubly adjacent to infinity. (The other toolbox routines cannot deal with adjacent infinite vertices anyway.)

When you are finished making changes, click on **Done**. Keep in mind that `modify` makes no attempt to ensure that the changes you make result in a true polygon. (One way to cause trouble is to make the sides of an infinite vertex meet at a finite point outside the axes box. Even though such a polygon could be legal if the intersection point were used as the vertex, the toolbox routines will expect an infinite vertex and will be hopelessly confused.) In the context of the GUI, the new polygon replaces the old one and the current GUI map is deleted. You can click the **Cancel** button in `modify` to reject changes and preserve the current polygon and map.

From the command line, there is an additional optional output `idx` that helps you keep track of the changes that were made. It has the same length as the new polygon, and each numeric entry of `idx` is the index that the corresponding entry of the new polygon had in the original polygon. Entries of `idx` corresponding to added vertices have the value `NaN` (not a number).

5 Additional notes

5.1 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 basic ordinary use of the SC maps. However, more may be needed in applications where, e.g., the prevertices are of interest or particular

Table 4: Methods for examining map data.

`center`	Return (or set) the conformal center (for `diskmap` and `crdiskmap` only).
`polygon`	Return the polygon that defines the target domain.
`parameters`	Return a structure holding the SC parameters (e.g., prevertices, multiplicative constant).
`scmapopt`	Return an options structure used in the solution of the parameter problem.
`get`	Alternative to the above.

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.[11] 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 on caveats.

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.

5.2 Application: 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 proposed uses[6, 17, 22].

[11]It might seem preferable to have these functions stored in the proper class directories. However, these routines comprised the toolbox before the object-oriented interface became available, so they are offered for compatibility. Besides, they would not properly be methods on their own and so would become inaccessible for direct use.

The parameter `k` is the highest degree of polynomial 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.

5.3 Further reading

For more on the Schwarz–Christoffel transformation, see [1, 9, 20]. The theory and implementation of many variations of the transformation can be found in [2, 3, 7, 8, 11, 13, 14, 15, 18, 19, 21, 28]. Two surveys of variations and applications are [26, 27]. A forthcoming book on the subject by Driscoll and Trefethen should become available sometime in 2000–2001.

The SC Toolbox implements and extends the methods described by Trefethen in his appendix to [20]. For more on the design of SCPACK, which is very similar, see [9, 23, 25]. A paper specifically on the toolbox appeared in [4]. The cross-ratio formulation is introduced in detail in [5].

An FORTRAN implementation of SC mapping for doubly-connected regions bounded by polygons called DSCPACK is due to Hu [16].

For more on numerical conformal mapping in general, see [10, 24]. An excellent FORTRAN package called CONFPACK [12] is available from `netlib.org` for maps to regions with piecewise smooth boundary.

5.4 Suggestions and bug reports

Remember: Anything free comes with no guarantee! 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 on the Internet as `driscoll@na-net.ornl.gov`. 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.

5.5 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.[12]

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.

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

References

[1] R. V. CHURCHILL AND J. W. BROWN, *Complex Variables and Applications*, McGraw-Hill, 5th ed., 1990.

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

[3] R. T. DAVIS, *Numerical methods for coordinate generation based on Schwarz–Christoffel transformations*, in 4th AIAA Comput. Fluid Dynamics Conf., Williamsburg, VA, 1979, pp. 1–15.

[4] T. A. DRISCOLL, *A* MATLAB *toolbox for Schwarz–Christoffel mapping*, ACM Trans. Math. Soft., 22 (1996), pp. 168–186.

[5] T. A. DRISCOLL AND S. A. VAVASIS, *Numerical conformal mapping using cross-ratios and Delaunay triangulation*, SIAM J. Sci. Comput., 19 (1998), pp. 1783–1803.

[6] S. W. ELLACOTT, *A survey of Faber methods in numerical approximation*, Comp. & Maths. with Appls., 12B (1986), pp. 1103–1107.

[7] J. M. FLORYAN, *Conformal-mapping-based coordinate generation method for flows in periodic configurations*, J. Comput. Phys., 62 (1986), pp. 221–247.

[8] J. M. FLORYAN AND C. ZEMACH, *Schwarz–Christoffel mappings: A general approach*, J. Comput. Phys., 72 (1987), pp. 347–371.

[9] P. HENRICI, *Applied and Computational Complex Analysis: Power Series, Integration, Conformal Mapping, Location of Zeros*, vol. 1, Wiley, 1974.

[10] ——, *Applied and Computational Complex Analysis: Discrete Fourier Analysis, Cauchy Integrals, Construction of Conformal Maps, Univalent Functions*, vol. 3, Wiley, 1986.

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

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

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

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

[15] L. H. HOWELL AND L. N. TREFETHEN, *A modified Schwarz–Christoffel transformation for elongated regions*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 928–949.

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

[17] 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.

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

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

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

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

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

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

[24] ——, ed., *Numerical Conformal Mapping*, North-Holland, Amsterdam, 1986. Reprint of J. Comput. Appl. Math. **14** (1986), no. 1–2.

[25] ——, *SCPACK user's guide*. MIT Numerical Analysis Report 89-2, 1989.

[26] ——, *Schwarz–Christoffel mapping in the 1980's*. Cornell University Computer Science Department Technical Report TR 93-1381, 1993.

[27] 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), vol. 1998, 1998, pp. 533–542 (electronic).

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