Conformal Mapping Using the Schwarz-Christoffel Transform 2007-2008

Evan Warner

November 5, 2007

Abstract

The Schwarz-Christoffel transform is a conformal mapping from the upper half of the complex plane to a polygonal domain. It allows many physical problems posed on two-dimensional, polygonal regions, such as heat flow, fluid flow, and electrostatics, to be solved numerically. This type of problem cannot generally be solved in closed form; the Schwarz-Christoffel transform provides an exceptionally accurate method of solution. This project will produce a working software unit that efficiently and accurately calculates Schwarz-Christoffel transforms and inverses. The program will incorporate graphical, easy-to-use interfaces and contain resources to aid in solving physical problems. In addition, research into mathematical extensions to the Schwarz-Christoffel transform, such as the inclusion of simple curves, will be conducted.

Keywords: Schwarz-Christoffel transform, conformal mapping, numerical analysis, Laplace's equation, fluid flow, heat flow

1 Introduction

Many physical problems are expressed as differential or boundary value problems over a surface. Often, these surfaces are or can be approximated by two-dimensional polygons. In this specific case, one method of determining accurate solutions is by taking the polygonal domain to exist in the complex plane and determining a conformal map, which preserves the structure of Laplace's equation, that restates the problem in a simpler domain, most often the upper half-plane. The new problem, now easy to solve analytically or in closed form, is then mapped back to the original domain. For such polygonal domains, a method of determining the specific transform needed is provided by the following formula, known as the Schwarz-Christoffel transform:

$$f(z) = A \int_0^z \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta + B. \tag{1}$$

In this formula, ζ is an independent complex variable in the upper half-plane, the θ_j are the exterior angles of the polygon, the x_j are 'preverticies' of the mapping (given along the real axis), n is the number of verticies of the polygon, and A and B are complex constants that specify the location of the image polygon in the complex plane. The θ_j must satisfy

$$\sum_{j=1}^{n} \theta_j = 2\pi, \tag{2}$$

which ensures the completeness of the image polygon [2]. Unfortunately, the Schwarz-Christoffel formula is not easy to evaluate, and requires both effective integration algorithms and efficient, convergent nonlinear equation solvers. Implementation of such numerical routines is not a trivial problem, and is the subject of this paper.

The initial project may be divided into three separate problems. First, a method to effectively evaluate integrals of the form found in the Schwarz-Christoffel formula is required. Second, a numerical algorithm to solve the so-called Schwarz-Christoffel parameter problem, a system of nonlinear equations for the preverticies, must be developed. Third, a user interface is needed, which should be robust and accessible to allow nonspecialists to systematically solve various physical problems. The three components may be coded simultaneously or in series, as they are by nature almost entirely separable problems.

Once the above steps have been coded and refined, research will be done into improvements and optimizations to the numerical algorithms for various subproblems and extensions. These include the problem of mapping polygons with large aspect ratios, which are generally highly ill-conditioned, and the extension of the Schwarz-Christoffel formula to simple curves. The goal of the project is thus twofold: to produce a piece of software that will be useful in the solution of real, physical problems, and to improve upon the current algorithms for producing the Schwarz-Christoffel transform.

2 Background

The Schwarz-Christoffel transform was first discovered independently in the late 1860s by Elwin Christoffel and Hermann Schwarz. Schwarz used some of the ideas of the transform to provide a more rigorous proof of the Riemann Mapping Theorem, which he had previously shown to be incomplete, but the majority of this work was on a purely theoretical level [5]. The usefulness of the transform was mitigated by the formula's unwieldiness, as the mappings for all but the simplest domains could not be calculated in closed form. Numerical estimates, especially for nonsymmetric polygons with four or more verticies, could not be effectively calculated by hand. Application to physical problems, therefore, was limited at best until the advent of the computer. A computer algorithm to compute the Schwarz-Christoffel transform was first written in the 1960s, and others have been written and modified since then [3].

The first problem in calculating the Schwarz-Christoffel mapping is the evaluation of the integral given by Eq. (1). The integrand contains singularities at each of the endpoints of the image polygon, which tend to render ordinary numerical integration routines either useless or hopelessly slow. In addition, the presence of negative powers in f means that domains of applicability for each of the subfunctions $(\zeta - x_j)^{-\theta_j/\pi}$ must be chosen so that the entire domain in and immediately around the image polygon is meromorphic. Although several quadrature routines have been used for this problem, the method of choice today is Gauss-Jacobi quadrature, which uses a specially-tailored weighting function to choose points of evaluation and weights for the points that maximize efficiency. In practice, the Schwarz-Christoffel formula is altered so that the prevertex x_n is chosen to be both $-\infty$ and $+\infty$ (the values are equivalent for a conformal map, which acts on the Riemann sphere). This can always be done due to the extra degrees of freedom contained in Eq. (1). The integrals that must be evaluated in practice in the course of the Schwarz-Christoffel transform are of the form

$$\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi} d\zeta. \tag{3}$$

These integrals can always be written as required for Gauss-Jacobi quadrature; that is, in the form

$$\int_a^b (z-a)^\alpha (z-b)^\beta \psi(z) dz, \tag{4}$$

where α and β are real numbers greater than -1.

The points and weights of a Gauss-Jacobi quadrature are calculated here using a routine from Numerical Recipes [4] which efficiently estimates and solves for the roots of the Jacobi polynomials, which form the sample points just as the roots of the Chebyshev polynomials form the sample points for standard Gaussian quadrature. These points, however, are uniformly calculated in the range $[-1, 1]$, and the integrals must be adjusted slightly to conform to this range. During the calculation of the preverticies, discussed below, the z in Eq. (4) will be restricted to the real axis; however, in direct calculations once the preverticies have been found, the z will generally be fully complex, which must be dealt with by the program.

The second problem is the Schwarz-Christoffel parameter problem, where the x_j in Eq. (1) are calculated. As described in [2], a series of nonlinear, constrained equations can be formed from the requirement that the image polygon and the desired polygon be similar (the constants A and B in Eq. (1) then ensure congruency). Written out, there are $n-3$ linear equations in $n-3$ unknowns, once the extra degrees of freedom have been taken care of by arbitrarily giving three of the x_j precise values. Here, as in the literature, we take $x_1 = -1$ and $x_2 = 0$ in addition to the already-defined $x_n = \pm\infty$. The equations to be solved are then

$$\frac{|\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|}{|\int_{x_1}^{x_2} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|} - \frac{|w_j - w_{j-1}|}{|w_2 - w_1|} = 0, \tag{5}$$

where $i = 3, 4, ..., n-1$. However, there is an additional complication, as the order of the preverticies on the real axis matters. The extra constraint can be expressed as

$$0 < x_3 < x_4 < \ldots < x_{n-1} < \infty. \tag{6}$$

The nonconstrained problem is relatively easy to solve; however, the constraint prevents a naive application of a Newton's Method variant to this problem. To get around this, Trefethen in [3] suggests a simple change of variables that ensures the inequalities of Eq. (6). Take a new series of variables, χ_j, and let

$$\chi_j = \ln x_j - x_{j-1}. \tag{7}$$

The resulting χ_j will automatically obey Eq. (6), and the original x_j are found by the simple inverse formula

$$x_j = x_{j-1} + e^{\chi_j}. \tag{8}$$

This new set of equations in the χ_j is readily solved by a variant of Newton's Method that does not require the calculation of the Jacobian matrix (which would be hopelessly complex), but rather uses progressive estimates.

3 Development

So far, the software has been written entirely in Java, although certain routines may be later written in C to increase speed if there is a bottleneck at any point in the process. The entire development of the program is designed to be achieved in stages by attacking the subproblems individually. The following is a list of classes, with short descriptions, written up to this point:

- class Complex - this class stores and performs arithmetic on complex numbers, which are not directly supported by Java. Several of the methods, including the multiplication and division algorithms, are designed to run as quickly as possible while avoiding intermediate overflow and floating-point error propagation. The multiplication method, for instance, requires only three real multiplications rather than four. (*see Appendix*)

- class ComplexFunction - this class stores and evaluates functions of one complex variable. Its constructor accepts a String argument and stores the function, if syntactically acceptable, as a String in postfix form. The function may then be evaluated at any point on the complex plane. (*see Appendix*)

- class GaussJacobiWeights - this class calculates and stores the sample points and weights for a given Gauss-Jacobi quadrature over the interval $[-1, 1]$. This routine uses Newton's Method to find the roots of the Jacobi polynomials, which are the sample points for the integral, and was taken and translated from [4].

- class GaussQuad - this class accepts as input ψ, a, b, α, and β from Eq. (4). For an arbitrary integral in that form, shifting and scaling the bounds produces the equivalent integral

$$c^{\alpha+\beta+1} \int_{-1}^{1} (\zeta - 1)^{\alpha} (\zeta + 1)^{\beta} \psi(c\zeta + m) d\zeta, \tag{9}$$

  where $b = \frac{a+b}{2}$ and $c = b - m = m - a$. This integral is then evaluated using the sample points and weights given by the GaussJacobiWeights class and returned. For any GaussQuad object, varying numbers of sample points (and thus varying accuracy) are accepted by its integrate() method.

- class SchwarzChristoffel - this class runs the graphical user interface and calls GaussQuad when necessary. The graph itself has the ability to show axes and manually adjust window parameters.

Once the basic transform is operational, a new set of routines will be implemented to calculate continuous Schwarz-Christoffel problems. Immediately following from Eq. (3) above, we have

$$f'(z) = A \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi}. \tag{10}$$

To change this into a continuous problem, we can rewrite this as

$$f'(z) = A e^{\frac{1}{\pi} \sum_{j=1}^{n-1} -\theta_j \ln{(z - x_j)}}. \tag{11}$$

Then, defining the natural logarithm function as single-valued in the upper half-plane, except where $x_i = z$, f' becomes an analytic function in the required domain. To formulate the continuous-boundary problem, we simply replce the sum in Eq. (11) with an integral, and integrate the entire function to find $f(z)$:

$$f(z) = A \int_{0}^{z} e^{\frac{1}{\pi} \int_{-\infty}^{\infty} -\theta(x) \ln{(\zeta - x_j)} dx} d\zeta + B, \tag{12}$$

where $\theta(x)$ represents the amount of turning per unit length on the real axis, such that

$$\int_{-\infty}^{\infty} \theta(x)dx = 2\pi. \tag{13}$$

The continuous problem therefore has an extra subproblem to solve, namely, the solution of the integral equation, Eq. (12), to find $\theta(x)$ at every x. This will be dealt with in future iterations of the project.

The majority of testing of the program is specific to a single numerical routine; that is, each of the algorithmic components are tested individually. To calculate the GaussQuad routines, for instance, randomly generated sample problems are solved by MATLAB to provide an approximate check on the accuracy of solutions, then precision is achieved by manipulating the number of sample points used for the quadrature.

Once the entire basic routine is complete, a program specifically designed to track approximate error propagation and runtimes of each of the components will be developed. For the majority of the routines used in the program, strict error bounds can be calculated, and for the remaining algorithms error can be accurately estimated. For all subproblems, as well as the entire routine, plots of runtime versus precision will be generated to examine the efficacy of each routine. For the final program, random polygonal generation will be used to dynamically test the program for a range of inputs.

4 Results and Discussion

The purpose of this project was to calculate and display Schwarz-Christoffel transforms, which conformally map the upper half-plane to an arbitrary polygon, efficiently and accurately. In addition, additional research into the Schwarz-Christoffel transform itself, including its extension to curved target domains, was investigated. The evaluation of the Schwarz-Christoffel formula involves several parts, including the efficient calculation of a certain class of integrals as well as a solver of nonlinear systems of equations. Solving the continuous-parameter problem will require numerical solutions to a certain class of integral equations.

As yet, because the programs have not been written, no specific data on accuracy and timing have been produced beyond what is necessary to ensure that the answers are correct to within the modest relative error tolerance of about 10^{-4}. Nevertheless, far greater accuracy is expected once the equipment is in place to test the routines.

For a first-quarter version of the project, the software is moderately successful. The first problem, that of numerical integration, has been solved and refined, and a basic user interface has been designed. Once the equation solver has been implemented, the program itself will be fully functional on a basic level. The completed program will be useful on several levels: as a teaching aid, and as a tool for researchers solving certain equations on polygonal regions. Once the basic Schwarz-Christoffel problem is numerically solved, the program can form an easy basis for testing research in numerical analysis and mathematics that deals with improving or expanding the Schwarz-Christoffel transform.

Appendix

Code for **class Complex**

```
public class Complex
{
  private double x,y;
  public Complex(double u, double v)
  {
    x=u;
    y=v;
  }
  public double real()
  {
    return x;
  }
  public double imag()
  {
    return y;
  }
  public String toString()
  {
    return "{"+x+","+y+"}";
  }
  public Complex add(Complex z)
  {
    return new Complex(x+z.real(),y+z.imag());
  }
  public Complex subtract(Complex z)
  {
    return new Complex(x-z.real(),y-z.imag());
  }
  public Complex multiply(Complex z)
  {
    double temp1=x*z.real();
    double temp2=y*z.imag();
    return new Complex(temp1-temp2,(x+y)*(z.real()+z.imag())-temp1-temp2);
```

```
  }
  public Complex multiply(double a)
  {
    return new Complex(x*a,y*a);
  }
  public Complex divide(Complex z)
  {
    double temp1=z.real()/z.imag();
    double temp2=z.imag()/z.real();
    if(Math.abs(z.real())>=Math.abs(z.imag()))
    {
      double denominator=z.real()+z.imag()*temp2;
      return new Complex((x+y*temp2)/denominator,(y-x*temp2)/denominator);
    }
    else
    {
      double denominator=z.real()*temp1+z.imag();
      return new Complex((x*temp1+y)/denominator,(y*temp1-x)/denominator);
    }
  }
  public Complex divide(double a)
  {
    return new Complex(x/a,y/a);
}
  public double modulus()
  {
    if(x==0&&y==0)
      return 0.0;
    else if(y==0)
      return Math.abs(x);
    else if(x==0)
      return Math.abs(y);
    if(Math.abs(y)>=Math.abs(x))
      return Math.abs(x)*Math.sqrt(1.0+(y*y)/(x*x));
    else
      return Math.abs(y)*Math.sqrt(1.0+(x*x)/(y*y));
  }
  public double argument()
  {
    return Math.atan2(y,x);
  }
  public Complex sqrt()
  {
    double w=0;
    if(x==y&&y==0)
      return new Complex(0.0,0.0);
    else if(Math.abs(x)>=Math.abs(y))
      w=Math.sqrt(Math.abs(x))*Math.sqrt((1.0+Math.sqrt(1.0+(y*y)/(x*x)))/2.0);
    else
      w=Math.sqrt(Math.abs(y))*Math.sqrt((Math.abs(x/y)+Math.sqrt(1.0+(x*x)/(y*y)))/2.0);
    if(x>=0)
      return new Complex(w,y/(2*w));
    else if(y>=0)
      return new Complex(Math.abs(y)/(2*w),w);
    else
      return new Complex(Math.abs(y)/(2*w),-w);
  }
```

```
  public Complex power(double a)
  {
    double theta=this.argument();
    double r=this.modulus();
    Complex temp = new Complex(Math.cos(theta*a),Math.sin(theta*a)).multiply(Math.pow(r,a));
    return temp;
  }
  public Complex ln()
  {
    double theta=this.argument();
    double r=this.modulus();
    return new Complex(Math.log(r),theta);
  }
  public Complex exp()
  {
    double etothex=Math.exp(x);
    return new Complex(etothex*Math.cos(y),etothex*Math.sin(y));
  }
}
```

Code for **class GaussQuad**

```
public class GaussQuad
{
  private static ComplexFunction f;
  private static double alpha, beta;
  private static Complex a, b;
  public GaussQuad(ComplexFunction f1, double alpha1, double beta1, Complex a1, Complex b1)
  {
    f=f1;
    alpha=alpha1;
    beta=beta1;
    a=a1;
    b=b1;
  }
  public Complex integrate(int N)
  {
    GaussJacobiWeights gjw = new GaussJacobiWeights(alpha,beta,N);
    double[] points = gjw.getPoints();
    double[] weights = gjw.getWeights();
    Complex m= b.add(a).divide(2);
    Complex c= b.subtract(m);
    Complex sum = new Complex(0,0);
    for(int i=0;i<weights.length;i++)
      sum=sum.add(f.value((c.multiply(points[i])).add(m)).multiply(weights[i]));
    sum=sum.multiply(c.power(alpha+beta+1));
    return sum;
  }
}
```

References

[1] Howell, L. H. (1990). *Computation of conformal maps by modified Schwarz-Christoffel transformations.* Retrieved September 28, 2007,

from http://citeseer.ist.psu.edu/howell90computation.html.

[2] Saff, E. B., & Snider, A. D. (n.d.). *Funamentals of complex analysis with applications to engineering, science, and mathematics*. Prentice-Hall Engineering/Science/Mathematics.

[3] Trefethen, L. (1979). *Numerical computation of the Schwarz-Christoffel transformation*. Retrieved September 28, 2007, from ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/79/710/CS-TR-79-710.pdf.

[4] Press W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in C, Second Edition*. Cambridge University Press.

[5] O'Connor, J.J., & Robertson, E.F. (2001). *Hermann Amandus Schwarz*. Retrieved November 3, 2007, from http://www-history.mcs.st-andrews.ac.uk/Biographies/Schwarz.html.