Conformal Mapping Using the Schwarz-Christoffel Transform 2007-2008

Evan Warner

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 paper describes the implementation of a working software unit that efficiently and accurately calculates Schwarz-Christoffel transforms and inverses. The program incorporates graphical, easy-to-use interfaces and will contain resources to aid in solving physical problems. Future 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. When this occurs, one method of determining accurate solutions is by assuming the polygonal domain exists 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. Here, the

upper half-plane is used. A solution to the problem, now easy to solve analytically or numerically, 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 prevertices 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, size, and orientation 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 an efficient, convergent method to solve a specific nonlinear equation. Implementation of such numerical routines is not a trivial problem, and is the subject of this paper.

The initial project may be divided into four 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 prevertices, must be developed. Third, methods to evaluate the forward and backward transforms based on given prevertices must be coded. Fourth, a user interface is needed, which should be robust and accessible to allow nonspecialists to systematically solve various physical problems. The four components may be coded simultaneously or in series, as they are by nature almost entirely separable problems. In this project, each part was coded in series.

Subsequent research will be conducted 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 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 last 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 prevertices, discussed below, the z in Eq. (4) are restricted to the real axis; however, in direct calculations once the prevertices have been found, the z will generally be fully complex.

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, ensuring that the target polygon and the image of the Schwarz-Christoffel map are similar, 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 the w are the target vertices, $i = 3, 4, ..., n-1$, and all other variables are as defined above. However, there is an additional complication, as preserving the order of the prevertices on the real axis is important. The extra

constraint can be expressed as

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

The unconstrained 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 employs a forward-difference approximation to the Jacobian matrix.

3 Development

The software has been written entirely in Java, with certain accuracy testing conducted in MATLAB. The entire development of the program is designed to be achieved in stages by attacking the subproblems individually. Several important classes are described below:

- 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 for examples)

- 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 SchwarzFunction - this class evaluates the integrand of a given real-valued Schwarz-Christoffel integral, serving as a storage class for data of this kind.

- 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. (*see* Appendix for example code)

- class RealNewtonRaphson - this class accepts an array of vertices and calculates the necessary prevertices as well as the constants A and B from Eq. (1). The method employs a standard Newton-Raphson method to solve the Eq. (5). At each step, an approximate Jacobian matrix for the function is calculated using a forward-difference method in each dimension; the step vector is then solved for using an LU factorization on the equation.

$$\mathbf{J}\delta\vec{x}=\vec{f}, \tag{10}$$

  where $\mathbf{J}$ represents the Jacobian, $\delta\vec{x}$ the step vector, and $\vec{f}$ the current function vector. Note that by employing a forward-difference method to find the Jacobian, the number of function evaluations can be cut in half, as the current function vector can be reused in the Jacobian calculation. (*see* Appendix for example code)

- class ForwardGaussQuad - this class, using already-calculated values for the prevertices, evaluates the Schwarz-Christoffel integral at a given point. To minimize error caused by the presence of singularities near the path of the integral (the singularities at the endpoints are handled by the Gauss-Jacobi quadrature), the path of integration is divided recursively such that no segment is closer to a singularity than one-half its length, a technique employed in [3]. Such recursive subdivision is known as compound Gauss-Jacobi quadrature.

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

In future iterations of the project, a new set of routines will be implemented to calculate continuous Schwarz-Christoffel problems. Immediately following from Eq. (3) above, we have

$$f'(z) = A \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi}. \tag{11}$$

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

$$f'(z) = Ae^{\frac{1}{\pi} \sum_{j=1}^{n-1} -\theta_j \ln(z - x_j)}. \tag{12}$$

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. (12) with an integral, and integrate the entire function to find $f(z)$:

$$f(z) = A \int_0^z e^{\frac{1}{\pi} \int_{-\infty}^{\infty} -\theta(x) \ln(\zeta - x_j) dx} d\zeta + B, \tag{13}$$

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

$$\int_{-\infty}^{\infty} \theta(x) dx = 2\pi. \tag{14}$$

The continuous problem therefore has an extra subproblem to solve, namely, the solution of the integral equation, Eq. (13), to find $\theta(x)$ at every x.

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.

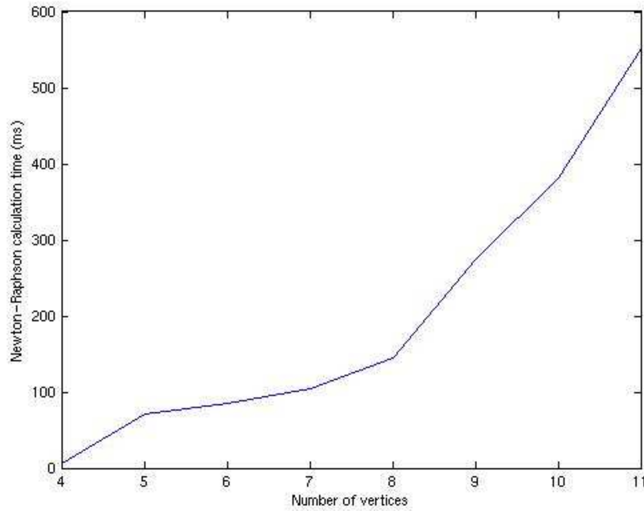

Figure 1: Computation time for prevertex calculation on a typical computer.

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. 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; each of these parts has been fully implemented in an object-oriented context. The GUI is designed to demonstrate conceptually the action of the transform by displaying the w- and z-planes side by side and mapping a given grid into the polygon (*see* Fig. [2]).

Results on polygons eleven and fewer vertices suggest that the program has both a reasonable error tolerance and reasonable computation time requirements (*see* Fig. [1]). The calculation time for prevertex calculation is comparable with routines described in the literature. Efficiency has been gained by implementing a two-step integration in ForwardGaussQuad which deals with the real and complex parts of the path integral separately and minimizes the average depth of recursion.

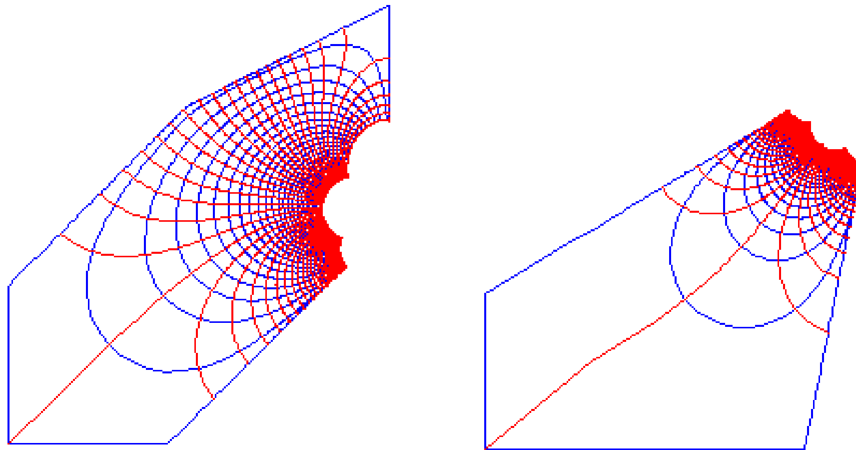

Figure 2: Example outputs.

Appendix

Code for **class Complex**

```
public class Complex
{
...
  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 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 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 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 ForwardGaussQuad**

```
public class ForwardGaussQuad
{
...
  public Complex recurse(int N, Complex a, Complex b, double al, double be)
  {
    if(a.imag()-b.imag()==0&&a.real()-b.real()==0)
      return new Complex(0,0);
```

```
    Complex half = a.add(b).divide(2.0);
    if(closeToSingularity(a,b,prevertex))
      return recurse(N, half, b, 0.0, be).add(recurse(N, a, half, al, 0.0));
    return integrateSubinterval(N, a, b, al, be);
  }
...
  public Complex integrateSubinterval(int N, Complex a, Complex b, double alpha, double beta)
  {
    GaussJacobiWeights gjw = new GaussJacobiWeights(0.0,beta,N);
    double[] points = gjw.getPoints();
    double[] weights = gjw.getWeights();
    double m = (b.real()+a.real())/2;
    double c = a.real()-m;
    double bimag = b.imag();
    Complex sum = new Complex(0.0,0.0);
    for(int i=0;i<weights.length;i++)
      sum=sum.add(value(new Complex(c*points[i]+m,bimag),-1,-1).multiply(weights[i]));
    sum=sum.multiply((new Complex(c,0)).power(beta+1.0));
    gjw = new GaussJacobiWeights(alpha,0.0,N);
    points = gjw.getPoints();
    weights  = gjw.getWeights();
    m = (b.imag()+a.imag())/2;
    c = a.imag()-m;
    double areal = a.real();
    Complex sum2 = new Complex(0.0,0.0);
    for(int i=0;i<weights.length;i++)
      if(alpha!=0.0)
        sum2=sum2.add(value(new Complex(areal,c*points[i]+m),1,-1).multiply(weights[i]));
      else
        sum2=sum2.add(value(new Complex(areal,c*points[i]+m),-1,-1).multiply(weights[i]));
    sum2=sum2.multiply(new Complex(0,c).power(alpha+1.0));
    if(a.imag()==0)
      sum2.imag(-sum2.imag());
    return sum.add(sum2);
  }
...
}
```

Code for **class GaussQuad**

```
public class GaussQuad
{
...
  public double integrate(int N)
  {
    GaussJacobiWeights gjw = new GaussJacobiWeights(alpha,beta,N);
    double[] points = gjw.getPoints();
    double[] weights = gjw.getWeights();
    double m= (b+a)/2;
    double c= b-m;
    double sum = 0;
    for(int i=0;i<weights.length;i++)
      sum+=f.value(c*points[i]+m)*weights[i];
    sum=sum*Math.pow(c,alpha+beta+1);
    return sum;
  }
}
```

Code for **class RealNewtonRaphson**

```
public class RealNewtonRaphson
{
...
  private double[] solvematrix(double[][] A, double[] b)
  {
    int l=n-3;
    double[][] L = new double[l][l];
    double[] q = new double[l];
    double[] x = new double[l];
    for(int p=0;p<l;p++)
      L[p][p]=1;
    for(int p=0;p<l;p++)
      for(int r=p+1;r<l;r++)
      {
        for(int c=p+1;c<l;c++)
        {
          A[r][p]=0;
          A[r][c]=A[r][c]-L[r][p]*A[p][c];
        }
      }
    for(int r=0;r<l;r++)
    {
      double sum=0;
      for(int c=0;c<r;c++)
        sum+=L[r][c]*q[c];
      q[r]=(b[r]-sum);
    }
    for(int r=0;r<l;r++)
    {
      double sum=0;
      for(int c=l-r;c<l;c++)
        sum+=A[l-r-1][c]*x[c];
      x[l-r-1]=(q[l-r-1]-sum)/A[l-r-1][l-r-1];
    }
    return x;
  }
  private double[][] jacobian(double[] funcvalues, double[] x)
  {
    double[][] jac=new double[n][n];
    for(int j=0;j<(n-3);j++)
    {
      double temp = x[j+2];
      double h = TOL*Math.abs(temp);
      if(h==0)
        h=TOL;
      x[j+2]=temp+h;
      h=x[j+2]-temp;
      double[] funcvalues2=function(x);
      x[j+2]=temp;
      for(int i=0;i<(n-3);i++)
        jac[i][j]=(funcvalues2[i]-funcvalues[i])/h;
    }
    return jac;
  }
...
```

```
  private double[] function(double[] x)
  {
    x=xhattox(x);
    double[] xwithstuff=new double[n-1];
    xwithstuff[0]=-1;
    xwithstuff[1]=0;
    for(int i=2;i<(n-1);i++)
      xwithstuff[i]=x[i-2];
    double[] f = new double[n-3];
    SchwarzFunction denom = new SchwarzFunction(x,angle,0);
    GaussQuad gq = new GaussQuad(denom, denom.alpha(), denom.beta(), denom.a(), denom.b());
    double den = Math.abs(gq.integrate(GQN));
    double den1 = vertex[1].subtract(vertex[0]).modulus();
    for(int i=0;i<(n-3);i++)
    {
      SchwarzFunction numer = new SchwarzFunction(x,angle,i+1);
      gq = new GaussQuad(numer, numer.alpha(), numer.beta(), numer.a(), numer.b());
      double num = Math.abs(gq.integrate(GQN));
      double num1 = vertex[i+2].subtract(vertex[i+1]).modulus();
      f[i]=num/den-num1/den1;
    }
    return f;
  }
...
}
```

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.