SIMULTANEOUS (SUCCESSIVE) OVER RELAXATION

TIMOTHY JONES

ABSTRACT. Numerical methods may be employed to model the potentials created by complex configurations. In this section we consider one such method, an efficient variation of the Jacobi method called simultaneous over relaxation. The derivation follows the primary strains of the proper mathematical derivation but is in no way rigorous.

1. HARMONIC EQUATIONS

In an electrical vacuum, the electrical potential obeys Laplace's equation:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \tag{1.1}$$

A real-valued function is considered harmonic in a domain D if all of its second-order partial derivatives are continuous in D, and if at each point in D the function satisfies Laplace's equation [1]. Such functions come from the real and imaginary parts of complex analytical functions.

Consider the function:

$$f(z) = u(x,y) + iv(x,y) \tag{1.2}$$

We assume the derivative of this function exists at $\mathrm{z_o = x_o + iy_o}$. We let $\Delta z = \Delta x + i\Delta y \to 0$ along the x and y axis independently, which is to say,

$$\frac{df(z_o)}{dz} = \lim_{\Delta z = \Delta x \to 0}\left(\frac{u(x_o+\Delta x, y_0) + iv(x_0+\Delta x, y_0)}{\Delta x} - \frac{u(x_0,y_0)+iv(x_0,y_0)}{\Delta x}\right)$$

Likewise for $\Delta \mathrm{z} = \mathrm{i}\Delta \mathrm{y}$, such that we have the set of equations,

$$\frac{df(z_0)}{z} = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} = -i\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y} \tag{1.3}$$

These are better known as the Cauchy-Riemann equations,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y} \tag{1.4}$$

The demand that

$$\frac{\partial}{\partial y}\frac{\partial u}{\partial x} = \frac{\partial}{\partial x}\frac{\partial u}{\partial y}$$

gives

$$\frac{\partial^2 v}{\partial y^2} = -\frac{\partial^2 v}{\partial x^2}$$

Which is simply Laplace's equation, establishing our claim that potential functions are components of complex functions. With this fact comes the many useful tools of complex analysis. Specifically, Cauchy's Integral Formula reads,

$$f(z_0) = \frac{1}{2\pi}\oint \frac{f(z)}{z - z_0}dz \tag{1.5}$$

for some contour. If we parameterize that contour with $z = z_0 + Re^{it},\ 0 \leq t \leq 2\pi$, and

$$f(z)dz = f(z)\frac{dz}{dt}dt$$

This gives

$$f(z_0) = \frac{1}{2\pi i}\int_0^{2\pi} \frac{f(z_0 + Re^{it})}{Re^{it}} iRe^{it}dt$$

This is better known as the Mean-Value property,

$$f(z_0) = \frac{1}{2\pi}\int_0^{2\pi} f(z_0 + Re^{it})dt \tag{1.6}$$

A harmonic function evaluated at some point is equal to the average value of that function around some circle (or sphere) centered at that point. The **Jacobi** method follows directly from this.

2. Jacobi, Gauss-Seidel Method

The Jacobi method proposes to impose a grid over our physical environment. This grid is the standard square grid type, whereby the mean value method can only sample four points:

This gives

$$V(i,j) = \frac{1}{4}\left(V(i+1,j) + V(i-1,j) + V(i,j+1) + V(i,j-1)\right) \tag{2.1}$$

If we now imagine taking this average over a i by j grid under the boundary constraints in an iterative process, we might see that after many iterations the boundary conditions will have 'flowed' by averaging throughout the system, a process called 'relaxing'. Once the iteration process gives negligible change in our grid, we have found a numerical solution to our potential problem. Under the right conditions, convergence is guaranteed (more on this later).

Whereas the Jacobi method sums the potential as the average of its neighbors in their pre-determined state (i.e. from the previous set of iterations), the Gauss-Seidel (GS) method proposes the use of the latest numbers as they become available.

In Simultaneous Over Relaxation (SOR) we push these methods even further, for if we define $\Delta V = V_{New}(i,j) - V_{Old}(i,j)$, then SOR proposes to force that change by some factor $\alpha > 1$:

$$V_{New}(i,j) = V_{Old}(i,j) + \alpha\Delta V$$

. Finding the right α value without wrecking the process is the point of this report.

3. A More Fundamental Analysis of The Simultaneous Over Relaxation Method

Whereas our principal mathematical justification for an iterative averaging comes from the mean value property, the behavior of the iteration in terms of convergence must ultimately come from the properties of the numerical methods. In this section we will look closer at the mathematical details of our iterative processes with the ultimate goal of finding the optimal SOR coefficient.

We can generalize the Jacobi method by differentiating for boundary conditions,

$$\begin{aligned}
(3.1) \quad V(i,j) &= \frac{1}{4}\left(V(i+1,j)+V(i-1,j)+V(i,j+1)+V(i,j-1)\right) \\
(3.2) \quad x^n_{i,j} &= \frac{1}{4}\left(x^{n-1}_{i+1,j}-x^{n-1}_{i-1,j}-x^{n-1}_{i,j+1}-x^{n-1}_{i,j-1}\right) \\
(3.3) \quad &= \frac{1}{4}\left(\sum_{a=i}^{i+1}\sum_{a=j}^{j+1} x^{n-1}_{a,b}\delta_{a,b}+\sum_{a=i}^{i+1}\sum_{a=j}^{j+1} x^{n-1}_{a,b}\delta'_{a,b}\right)
\end{aligned}$$

Here δ selects those values which are non-boundary values, and δ' indicates those which are boundary conditions.

The above equation can be generalized by writing, for the Jacobi case,

$$Dx^{n+1} = Bx^n + b \tag{3.4}$$

For the GS case, we use the new values as they become available whereby,

$$D(x[k+1]) = Lx[k+1] + Ux[k] + b$$

The reader can convince themselves that L is a lower triangular matrix and U an upper triangular, where the Jacobi form could be simplified as, for example,

$$\begin{pmatrix} D & U & U & U \\ L & D & U & U \\ L & L & D & U \\ L & L & L & D \end{pmatrix}\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} \tag{3.5}$$

That L is a triangular matrix and D is uniformly nonzero along the diagonal means that the determinant of D-L will not be zero, whereby D-L is invertible. Thus we have,

$$\begin{aligned}
x[k+1] &= (D-L)^{-1}Ux[k] + (D-L)^{-1}b, \quad \text{or} \\
(3.6) \quad x[k+1] &= D^{-1}\left(Lx[k+1] + Ux[k] + b\right)
\end{aligned}$$

SOR has us calculate the difference between the new value of the potential we are calculating as recommended by GS, and the old value pre-calculation. Calling this difference Q, SOR then adds wQ to the old coefficients, where w is the SOR coefficient to be determined for optimal progress ($w = 1$ is simply GS, $w > 1$ is SOR). Thus in matrix form, the SOR method can be written

$$\begin{aligned}
(3.7) \quad x[k+1] &= x[k] + w\left(D^{-1}\left(Lx[k+1]+Ux[k]+b\right) - x[k]\right) \\
(D-wL)x[k+1] &= (1-w)Dx[k] + wUx[k] + wb \\
x[k+1] &= (D-wL)^{-1}\left((1-w)D+wU\right)x[k] + (D-wL)^{-1}wb
\end{aligned}$$

And obvious substitution is

$$H = (D-wL)^{-1}\left((1-w)D+wU\right),$$

from which we get

$$x[k+1] = Hx[k] + (D-wL)^{-1}wb \tag{3.8}$$

The reader is encouraged to view reference [2],[3] for a more detailed exposition, as we take more than a few liberties of omitting detail here.

3.1. **Conditions for the the convergence of SOR.** The conditions for the convergence of the simultaneous over-relaxation method are found via Kahan's Theorem (Equaton 3.10). Note that

$$\begin{aligned} det(H) &= det((D-wL)^{-1})det((1-w)D+wU) \\ &= det(D^{-1})det((1-w)D+wU)) \\ &= det((1-w)I+wD^{-1}U) = det((1-w)I) = (1-w)^n \qquad (3.9) \end{aligned}$$

Since $det(H) = \prod_i \lambda_i$, ($\lambda_i$ are eigenvalues of H), and the spectral radius is defined by $\rho(H) = max|\lambda_i|$,

$$\prod_i \lambda_i = det(H) = (1-w)^n \leq max|\lambda_i|^n \;\Rightarrow \rho(H) \geq |w-1| \qquad (3.10)$$

Now the Fundamental Theorem of Linear Iterative Methods (see final section) tells us we have convergence if $\rho(H) < 1$, and since $\rho(H) \geq |w-1|$,

$$1 > |w-1| \Rightarrow 0 < w < 2$$

Obviously $w = 1$ is Jacobi-GS, and $w < 1$ is foolish, so we consider $1 < w < 2$. It is now our task to find which value in this region is most efficient.

3.2. **A relaxed derivation of the SOR coefficient.** We begin our discussion by considering a simpler iteration system,

$$x^{k+1} = Gx^k$$

Let C be the set of all sequences $\{x^k\}$ created by this iteration such that the sequence converges to $\{x^*\}$, i.e. we say

$$\lim_{k\to\infty} \{x^k\} = \{x^*\}$$

Define

$$\alpha = \lim_{k\to\infty} sup|x^k - x^*|^{1/k} \rightarrow 0 \leq \beta \leq 1 \qquad (3.11)$$

We shall take liberties and call this α the asymptotic convergence factor of the general iteration (see the references for a more sophisticated definition). For a given iteration, the sequence will converge at least as rapidly as the sequence α^k goes to zero.

From our previous definitions,

$$x[k+1] - x^* = H(x[k] - x^*) = H(H(x[k-1] - x^*)$$

$$|x^k - x^*| \leq |H^k||x^0 - x^*| \leq |H|^k|x^0 - x^*| \ni \lim_{k\to\infty} sup|x^k - x^*|^{1/k} \leq \rho(H) \qquad (3.12)$$

Here we have used that $|H| \leq \rho(H)$. We have loosely demonstrated that

$$\alpha \leq \rho(H)$$

The spectral radius establishes the upper bound on the rate of convergence. We thus seek to maximize the spectral radius as a to minimize the convergence time. To begin to do so, we wish to roughly demonstrate a rather elegant lemma that says that eigenvalues of the Jacobi iteration matrix occur in pairs (except if there is a zero eigenvalue), which we shall denote $\pm\mu_i$

Suppose we take some random value λ and want to find $det(I\lambda - C)$, where C has eigenvalues a_i. C can be diagonalized with its eigenvalues via $C = TDT^{-1}$, whereby

$$det(\lambda I - C) = \det(T\lambda IT^{-1} - T(Diag)T^{-1}) = det(T)det(\lambda I - Diag)det(T^{-1})$$

Thus, supposing the existance of p zero valued eigenvalues (p may be zero itself):

$$det(\lambda I - C) = det(\lambda I - Diag) = \lambda^p \prod_{i=1}^{r} (\lambda - a_i)$$

For a more thorough discussion of what is to follow, the reader is referred to the references, especially on 2-Cyclic matrices (such as the one we are concerned with). Now $det(\lambda I - C)$ may be written as

$$det(\lambda I - C) = \begin{vmatrix} \lambda I_1 & -C_1 \\ -C_2 & \lambda I_2 \end{vmatrix}$$

And we note that

$$(-1)^n = \left| \begin{pmatrix} -I_1 & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} I_1 & 0 \\ 0 & -I_2 \end{pmatrix} \right|$$

We can thus write that

$$\begin{aligned} det(\lambda I - C) &= (-1)^n \begin{vmatrix} -I_1 & 0 \\ 0 & I_2 \end{vmatrix} \begin{vmatrix} \lambda I_1 & -C_1 \\ -C_2 & \lambda I_2 \end{vmatrix} \begin{vmatrix} I_1 & 0 \\ 0 & -I_2 \end{vmatrix} \\ &= (-1)^n \begin{vmatrix} -\lambda I_1 & -C_1 \\ -C_2 & -\lambda I_2 \end{vmatrix} \\ &= det(\lambda I + C) \qquad (3.13) \end{aligned}$$

$$(3.14) \quad det(\lambda I - C) = det(\lambda I + C) = \lambda^p \prod_{i=1}^{r} (\lambda + a_i) \Rightarrow \forall a_i > 0 \, \exists \, a_s < 0 \, \ni a_s = -a_i$$

Thus (those with a strongly developed mathematical sense will be unhappy with my use of thus here) we have Romanovsky's Lemma:

$$det(\lambda I - C) = \lambda^p \prod_{i=1}^{r/2} (\lambda^2 - \mu_i^2)$$

We can use this lemma to demonstrate another important detail about form of matrices we are considering (formally called regular splittings).

We wish to show that the eigenvalues of $\alpha D^{-1} L + \beta D^{-1} U$ are equal to those of $\alpha^{1/2} \beta^{1/2} D^{-1}(L + U)$. Let

$$\delta = \frac{\alpha^{1/2}}{\beta^{1/2}}$$

We need to demonstrate that

$$\begin{aligned} det(\lambda I - (\alpha D^1 L + \beta D^{-1} U)) &= det(\lambda I - \alpha^{1/2} \beta^{1/2} (\delta D^{-1} L + \delta^{-1} D^{-1} U)) \\ &= det(\lambda I - \alpha^{1/2} \beta^{1/2} (D^{-1} L + D^{-1} U) \qquad (3.15) \end{aligned}$$

That is to say, that the matrix $\eta = L + U$ is consistently ordered. Define

$$\eta(\delta) = \begin{pmatrix} 0 & \delta^{-1} D_1^{-1} C_1 \\ \delta D_2^{-1} C_2 & 0 \end{pmatrix}$$

And

$$S_\delta = \begin{pmatrix} I_1 & 0 \\ 0 & \delta I_2 \end{pmatrix}, \quad S_\delta^{-1} = \begin{pmatrix} I_1 & 0 \\ 0 & \delta^{-1} I_2 \end{pmatrix}$$

It is clear that

$$\eta(\delta) = S_\delta \eta(1) S_\delta^{-1}$$

Whereby in the spirit of previous arguments we have thus shown Equation 3.15 to be true. We are now ready to attack the problem head on. Recall that we defined

$$(3.16) \qquad H = (D - wL)^{-1} ((1 - w)D + wU)$$

from this discussion we can reformulate this as follows:

$$\begin{aligned}
det(\lambda I - H) &= det((D-wL)^{-1})det(D-wL)det(\lambda I - (D-wL)^{-1}((1-w)D+wU)) \\
&= det((D-wL)^{-1})det((D-wL)\lambda I - (1-w)D + wU) \\
&= det(D^{-1})det((D-wL)\lambda I - (1-w)D + wU) \\
&= det((I - wD^{-1}L)\lambda I - (1-w)I - wD^{-1}U) \\
&= det((\lambda + w - 1)I - \lambda w D^{-1}L - wD^{-1}U) \\
&= det((\lambda + w - 1)I - \lambda^{1/2} w D^{-1}C) \quad \text{with } \alpha = w\lambda,\ \beta = w \text{ from 3.15} \\
(3.17) \quad &= (\lambda + w - 1)^p \prod_{i=1}^{r/2} \left((\lambda + w - 1)^2 I - \lambda w^2 \mu_i^2 \right) \text{ Romanovsky's Lemma}
\end{aligned}$$

The eigenvalues of H may be found from

$$(\lambda + w - 1)^2 = \lambda w^2 \mu_i^2$$

From this we get

$$(\sqrt{\lambda})^2 \mp \sqrt{\lambda} w \mu_i + (w-1) = 0 \ \rightarrow \sqrt{\lambda} = \frac{w\mu_i \pm \sqrt{w^2\mu_i^2 - 4(w-1)}}{2}$$

Since μ comes in $\pm$ pairs, we can minimize our convergence time by choosing w such that

$$w^2\mu_i^2 - 4(w-1) = 0$$

We are bound by the maximal μ, the spectral radius, so that

$$w^2 \rho(D^{-1}C)^2 - 4w + 4 = 0$$

Defining $D^{-1}C = J$ and solving for w, we find

$$w = \frac{2 - 2\sqrt{1-\rho(J)^2}}{\rho(J)^2} = \frac{2(1 - 1 + \rho(J)^2)}{\rho(J)^2(1 + \sqrt{1-\rho(J)}} = \frac{2}{1 + \sqrt{1 + \rho(J)^2}} \tag{3.18}$$

To find the spectral radius of J, we need find the eigenvectors/values of J. The Jacobi equation is given, again, by

$$4x_{ij} - (x_{i+1,j} + x_{i-1,j} + x_{i,j+1} + x_{i,j-1}) = b_{i,j}$$

J is formed by stripping off the diagonal and multiplying by $I/4$ Thus the eigenvalue equation gives

$$J\zeta = \nu\zeta \ \rightarrow \frac{1}{4}(\alpha_{i+1,j} + \alpha_{i-1,j} + \alpha_{i,j+1} + \alpha_{i,j-1}) = \nu\alpha_{i,j} \tag{3.19}$$

Introduce the ansatz that

$$\alpha_{i,j}^{k,l} = \sin(\frac{k\pi i}{N})\sin(\frac{l\pi j}{N})$$

where N is the dimensions of the grid (matrix, i.e. the number of rows). Plugging this into the equation above yields the equation

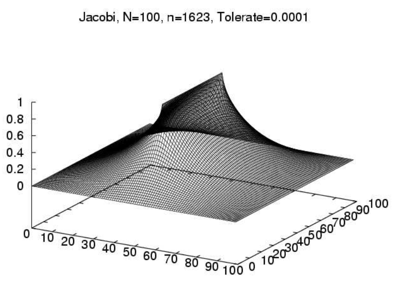
$$\nu_{k,l} = \frac{1}{2}\left(\cos(\frac{\pi k}{N}) + \cos(\frac{\pi l}{N})\right)$$

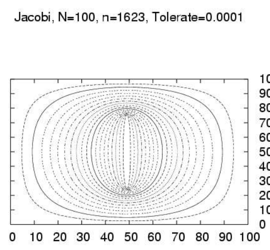
Obviously the maximum possible value will be $\cos(\pi/N)$, this thus defining the spectral radius of concern, where by,

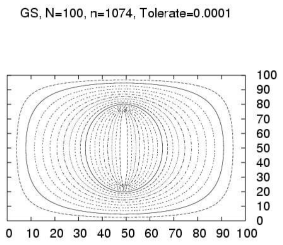
$$w = \frac{2}{1 + \sqrt{1 - \cos^2(\pi/N)}} = \frac{2}{1 + \sin(\pi/N)} \approx \frac{2}{1 + \frac{\pi}{N}} \tag{3.20}$$

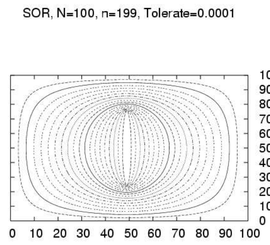
4. A race: Jacobi verses GS verses SOR

To demonstrate the difference between the normal naive Jacobi and the SOR, we test the methods with a basic voltage configuration. Consider a grounded box with a rod half the length centered in the box. When we performed the relaxation, it took Jacobi's method 1623 iterations, GS 1074, and SOR only 199 iterations; this with a tolerance (maximum difference between previous values and new values) of 0.0001. Obviously one can see the advantage of the SOR method. Our results are presented below, and our program used ends this report.









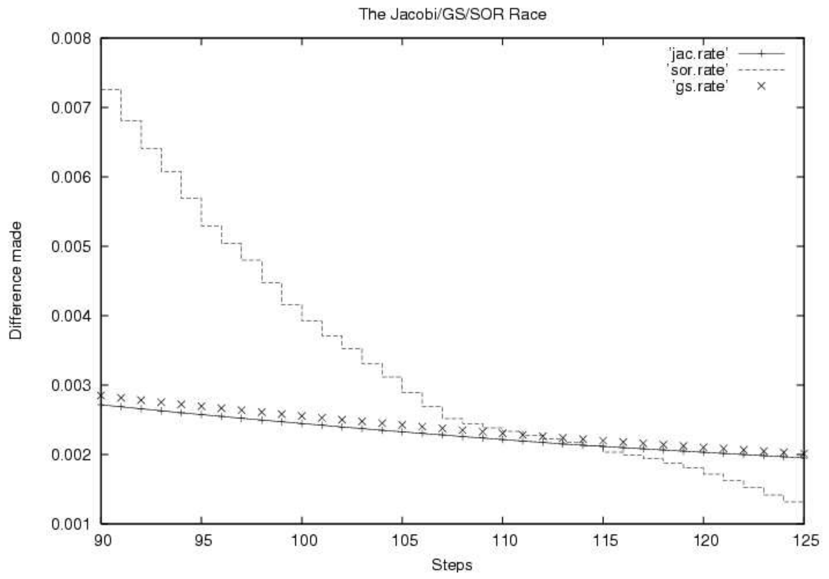


Figure 4.1. Same results, different rates. Jacobi requires 1623 iterations, GS 1074, and SOR only 199 for a simple potential problem and a 100 by 100 grid.

5. An application: The rf Paul Ion Trap

The rf Paul ion trap, a configuration of hyperbolic electrodes, and similar applications, have been extensively studied (see the report on the rf Paul ion trap for more details). For a much more professional relaxation calculation for such traps made using hyperbolic electrodes, the reader is encouraged to study Gaerald Gabrielse's paper [5].

We set our algorithm to create a pseudo cross section of an rf-Paul trap. The upper electrodes are held at -1 Volt, and the middle ring is held at 1 Volt. We proceed with an SOR iteration with a tolerance of 0.0001 (defining the maximal difference between iterations of any point on the grid) of a 1000 by 1000 grid. Our results are presented below. As expected, an unstable equilibrium is found at the center. It took 1318 iterations, and though we lacked the patience to give the Jacobi method a chance to run this scenario, we suspect it would have taken much much longer.

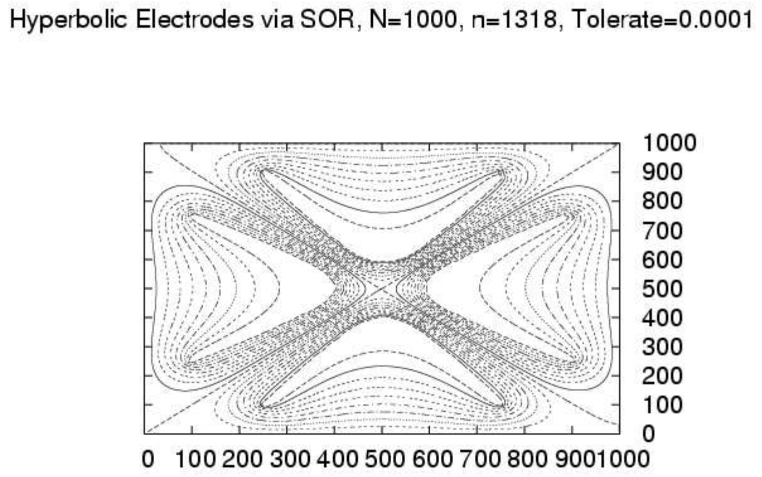


Figure 5.1. The pseudo rf-Paul trap, as modeled using SOR

6. The programs used

For the SOR method (which can be modified to GS and Jacobi):

```
#include<stdio.h>
#include<math.h>

int main(){

  //Set up the array
 int i,j;
 int count=1;
 float ok=0.0001;
 float diff=0;
 float max=0.002;
 double v[100][100];
 double u[100][100];
 int flag=0;
 for(i=0; i<100; i++){
   for(j=0; j<100; j++){v[i][j]=0.0;u[i][j]=0.0;}}

 float w=1.93908;
 float alpha=0;

 //The SOR Method: Avoid boundaries
 //A grounded box surrounds a central
 //plate held at potential located at i=49
 //from j=25 to 75
 for(j=25;j<=75;j++){v[49][j]=1.0;u[49][j]=1.0;}


 //A
 while(max>ok){
   printf("%f\n",max);
   //(B-C)
   max=ok;
   count+=1;
   for(i=1; i<99; i++){  //B
     for(j=1; j<99; j++){  //C
       if(i==49)if(j>=25)if(j<=75){flag=100;}  \\check for boundry
       if(i==49)if(j<25){flag=0;}
       if(i==49)if(j>75){flag=0;}
       if(flag<1){alpha=0.25*(v[i+1][j]+u[i-1][j]+v[i][j+1]+u[i][j-1]);}
       if(flag<1){u[i][j]=v[i][j]+w*(alpha-v[i][j]);}
     diff=fabs(v[i][j]-u[i][j]);
     if(diff>max){max=diff;}

     }}  //~B ~C
   //~(B-C)
     for(i=1; i<99; i++){for(j=1; j<99; j++){v[i][j]=u[i][j];}}
  }//~A

 //Print to file
 FILE *fp;
 fp=fopen("so.txt","w");


 for(i=0; i<100; i++){for(j=0; j<100; j++){
   fprintf(fp,"%i %i %f \n", i,j,v[i][j]);}fprintf(fp,"\n");}


 printf("SOR took %i iterations.\n",count);

}
```

The above code plots a simple bar at potential V. We can modify it for our hyperbolic electordes as,

```
#include<stdio.h>
#include<math.h>

int main(){

  //Set up the array
 int i,j;
 int count=1;
 float ok=0.0001;
 float diff=0;
 float max=0.002;
 double v[1000][1000];
 double u[1000][1000];
 int flag=0;
 for(i=0; i<1000; i++){
   for(j=0; j<1000; j++){v[i][j]=0.0;u[i][j]=0.0;}}

 float w=1.93908;
 float alpha=0;

 //Set hyperbolic conditions:

 // Upper
 for(j=250;j<=750;j++){
        i=10*(int)floor(50+sqrt(100+2.4*(j/10-50)*(j/10-50)));
        v[i][j]=1.0; v[i+1][j]=1.0; v[i-1][j]=1.0; v[i][j+1]=1.0;
                                                    \ v[i][j-1]=1.0;
        u[i][j]=1.0; u[i+1][j]=1.0; u[i-1][j]=1.0; u[i][j+1]=1.0;
                                                   \ u[i][j-1]=1.0;}

 //Lower
 for(j=250;j<=750;j++){
        i=10*(int)floor(50-sqrt(100+2.4*(j/10-50)*(j/10-50)));
        v[i][j]=1.0; v[i+1][j]=1.0; v[i-1][j]=1.0; v[i][j+1]=1.0;
                                                     \ v[i][j-1]=1.0;
        u[i][j]=1.0; u[i+1][j]=1.0; u[i-1][j]=1.0; u[i][j+1]=1.0;
                                                    \ u[i][j-1]=1.0;}



// Left
 for(i=250;i<=750;i++){
        j=10*(int)floor(50+sqrt(100+2.4*(i/10-50)*(i/10-50)));
        v[i][j]=-1.0; v[i+1][j]=-1.0; v[i-1][j]=-1.0; v[i][j+1]=-1.0;
                                                    \ v[i][j-1]=-1.0;
        u[i][j]=-1.0; u[i+1][j]=-1.0; u[i-1][j]=-1.0; u[i][j+1]=-1.0;
                                                    \  u[i][j-1]=-1.0;}

 //Right
 for(i=250;i<=750;i++){
        j=10*(int)floor(50-sqrt(100+2.4*(i/10-50)*(i/10-50)));
        v[i][j]=-1.0; v[i+1][j]=-1.0; v[i-1][j]=-1.0; v[i][j+1]=-1.0;
                                                    \ v[i][j-1]=-1.0;
        u[i][j]=-1.0; u[i+1][j]=-1.0; u[i-1][j]=-1.0; u[i][j+1]=-1.0;
                                                    \ u[i][j-1]=-1.0;}



 //A
 while(max>ok){
   printf("%f\n",max);
   //(B-C)
   max=ok;
   count+=1;
   for(i=1; i<999; i++){  //B
     for(j=1; j<999; j++){  //C
       // if(v[i][j]>1){v[i][j]=1.0;}  //Strays
       // if(v[i][j]<-1){v[i][j]=-1.0;}
       //if(v[i][j]<=-1.0){flag=100;}
       if(v[i][j]>=1){flag=100;}
       if(v[i][j]<=-1){flag=100;}
       if(v[i][j]<1 && v[i][j]>-1){flag=0;}
       //Problem:  what if more
       if(flag<1){alpha=0.25*(v[i+1][j]+u[i-1][j]+v[i][j+1]+u[i][j-1]);}
       if(flag<1){
              u[i][j]=v[i][j]+w*(alpha-v[i][j]);
              if(u[i][j]>1){u[i][j]=1;}
              if(u[i][j]<-1){u[i][j]=-1;}}
     diff=fabs(v[i][j]-u[i][j]);
     if(diff>max){max=diff;}

     }}  //~B ~C
   //~(B-C)
     for(i=1; i<999; i++){for(j=1; j<999; j++){v[i][j]=u[i][j];}}
  }//~A

 //Print to file
 FILE *fp;
 fp=fopen("so2.txt","w");

 int num=0; int numy=0;
 for(i=0; i<1000; i++){for(j=0; j<1000; j++){
  fprintf(fp,"%i %i %f \n", i,j,v[i][j]);} fprintf(fp,"\n");
   numy++;
 }


 printf("SOR took %i iterations.\n",count);

}
```

We also used the following gnuplot script, where our previous program's output put a blank line following each j cycle to conform with gnuplot's contour method:

```
set data style lines
set terminal postscript
#set nokey
set cntrparam levels 20
set output 'hyper1.ps'
set title "Hyperbolic Electrodes via SOR, n=1000, Tolerate=0.0001"
splot 'so.txt'
set contour
set nosurface
set view 0,0
set output 'hyper2.ps'
splot 'so.txt'
```

References

[1] E.B Saff, A.D. Snider, *Fundamentals of Complex Analysis for Mathematics, Science, and Engineering*, (Prentice Hall, Upper Saddle River, New Jersey)

[2] James M. Ortega, *Numerical Analysis: A Second Course*, (Academic Press, 1972)

[3] David M. Young, *Iterative Solution of Large Linear Systems*, (Academic Press, New York and London 1971)

[4] Richard S. Varga, *Matrix Iterative Analysis*, (Prentice Hall, 1962)

[5] Gerald Gabrielse, *Relaxation calculation of the electrostatic properties of compensated Penning traps with hyperbolic electrodes*, (Physical Review A, 27, 2277)

[6] Press et al., *Numerical Recipes in C, 2nd Ed.*, (Cambridge University Press, 1992)