University of Minnesota: Twin Cities

CE 8351: Analytical Modeling in Civil Engineering

Project 2: Elliptical, Impermeable Element

Part II: Non-Method of Images, Intersecting Well Branch Cut

Christopher Bulkley-Logston

March 12th, 2015

Contents

0 Given Formulas

While the method images is a useful tool for modeling the flow in an aquifer around an impermeable object in the presence of a discharge well, there also exists a method of numerical approximation that does not use images as discussed in *Analytical Groundwater Mechanics* (draft - 2016) by Otto D.L. Strack of the University of Minnesota.

0.1 Taylor Expansion Term

The effect of the impermeable object on the aquifer's complex potential can be expressed as an expanded series:

$$\Omega_m(\chi) = \sum_{m=1}^{M} \alpha_m \chi^{-m} \tag{1}$$

Where:

- χ is the complex location in the χ-plane (transformed from z)
- M is the total number of expansion coefficients
- α_m is the expansion coefficient for each of the series' terms, discussed in section 0.2.

It should be noted that with no discharge well, only 1 expansion term is necessary ($M = 1$). With the case of well flow, at least 20 are recommended ($M > 20$), as accuracy increases with M.

0.2 Cauchy Integral

This numerical method involves using a Cauchy integral to calculate a coefficient for each expanded term in eq. 1 using each of the N nodes chosen along the ellipse boundary, as shown in figure 1. Given a full 2π radians around the ellipse, the angular distance between each of the nodes is set to be the same for each sample taken:

$$\Delta\theta = \frac{2\pi}{N} \tag{2}$$

This integral process sums the streamline value Ψ at each of the boundary's nodes and averages them such that a constant-streamline-boundary condition is maintained. Accounting for each expansion term at each node, a matrix is developed as follows:

$$a_{(n,m)} = \Psi_{\text{other}}(z_n) e^{-im\theta_n} \tag{3}$$

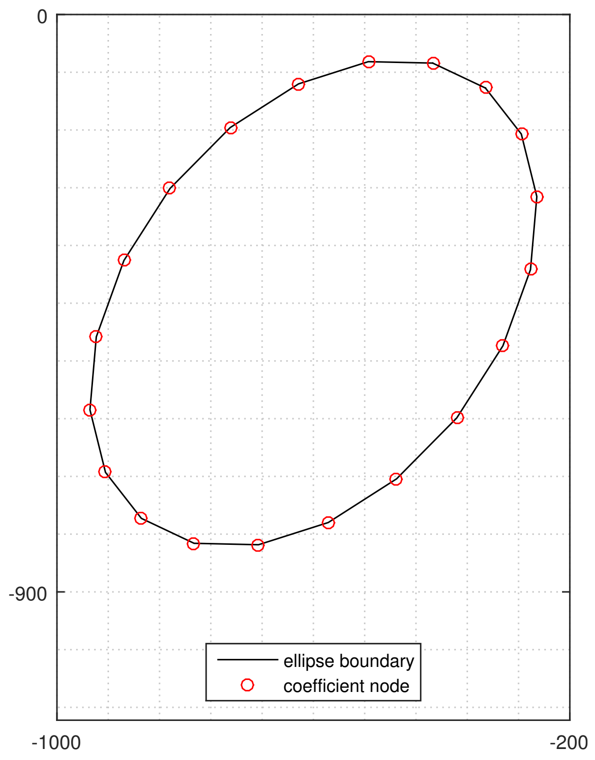

Figure 1: Coefficient nodes along ellipse

Equation 3 makes uses a preliminary Ω_{other}, which is developed to account for the well and the uniform flow's effect on the boundary's streamline.

$$\Omega_{\text{other}} = \Omega_{\text{uf}} + \Omega_{\text{well}} = Q_o e^{-i\beta} z + \frac{Q_w}{2\pi}(z - z_w) \tag{4}$$

Where:

- z = complex location where complex potential is calculated;
- Q_0 = magnitude uniform flow per length;
- β = direction of uniform flow;
- Q_w = well discharge;
- z_w = complex well location;
- The imaginary part of the complex potential is its streamline vale $\Psi_{\text{other}} = \Im(\Omega_{\text{total}})$

With this, the coefficient for each term of the expansion is taken as the conjugate of each node's coefficient, knowing they comprise the average value of the boundary's Ψ.

$$\alpha_m = \frac{2\sum\limits_{n=1}^{N} \overline{a_{(n,m)}}}{N} \tag{5}$$

It should be noted that the numerator is multiplied by 2 in order to be consistent with the definition of a Cauchy integral when applied to holomorphic functions using imaginary numbers.

With these α values known for each term, eq. 1 can be added to the total governing complex potential function as follows:

$$\Omega_{\text{total}}(z) = \Omega_{\text{uf}} + \Omega_{\text{well}} + \Omega_m = Q_o e^{-i\beta} z + \frac{Q_w}{2\pi}(z - z_w) + \sum_{m=1}^{M} \alpha_m \chi(z)^{-m} + C \tag{6}$$

Where:

- Where χ can be calculated as a function of Z using:

$$\chi(Z) = \nu\left(Z + \sqrt{Z+1}\sqrt{Z-1}\right) \tag{7}$$

- where ν can be calculated knowing major/minor ellipse axis ratio ρ using:

$$\nu = \frac{1}{\sqrt{1-\rho^{-2}}} - \sqrt{\frac{1}{1-\rho^{-2}} - 1} \tag{8}$$

- and Z can be calculated as a function of physical z using:

$$Z(z) = \frac{z - \frac{1}{2}(z_1 + z_2)}{\frac{1}{2}(z_2 - z_1)} \tag{9}$$

- where z_1 and z_2 are the ellipse foci complex locations

1 Non-intersecting Branch Cut

Using this numerical method, the flow net shown in figure 2 is developed and placed next to the method of images flownet shown in figure 3. It can be seen that the well's branch cut is the only discernible difference between either figure.

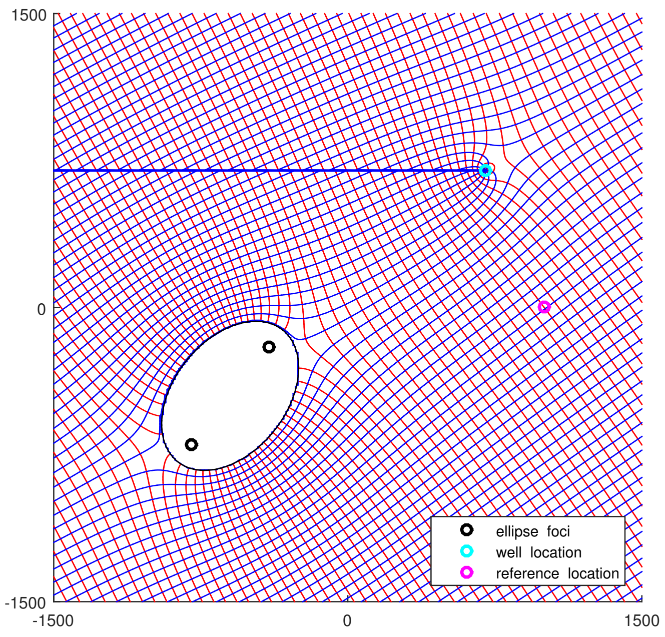

Figure 2: Flownet - numerical method

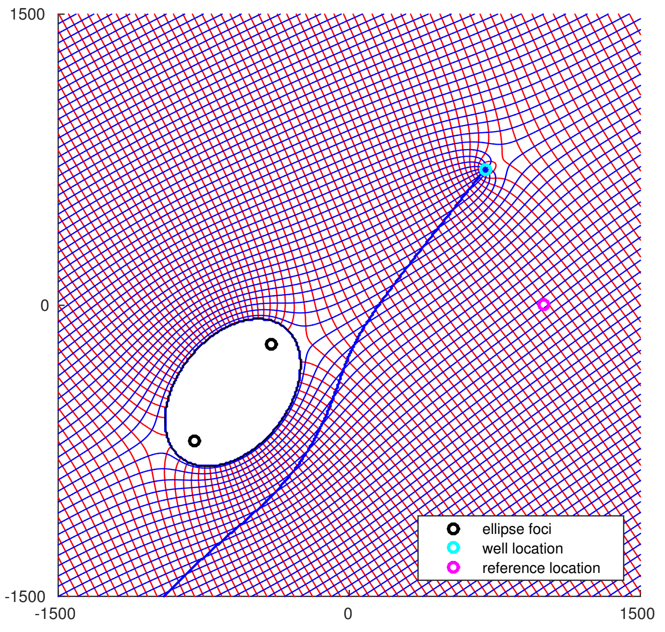

Figure 3: Flownet - method of images

In both cases, the following parameters are set:

- $z_1 = -800 - 700i$
- $z_2 = -400 - 200i$
- $z_w = 700 + 700i$
- $z_0 = 1000 + 0i$
- $\phi_0 = 28m$
- $k = 10\frac{m^3}{day}$
- $Q_w = 200\frac{m^3}{day}$
- $Q_0 = 0\frac{m^3}{day}$
- $\beta = 0°$
- $\rho = 1.5$
- aquifer is unconfined

Also, as discussed before:

- Both method's models are calibrated by adjusting the constant C in each's Ω_{total} function to be consistent with known discharge potential at the reference location:

$$C = \Phi_{\text{ref}} - \Re\left[\Omega_{\text{total}}(C = 0, z = z_{\text{ref}})\right] \tag{10}$$

- After calibration, the total function is tested to make sure it meets this reference boundary condition:

$$\textbf{assert}\ \left[\Omega_{\text{total}}(z_{\text{ref}}) = \Phi_{\text{ref}} = \frac{1}{2}k\phi_{\text{ref}}^2\right] \tag{11}$$

- In both cases, the condition that the ellipse boundary streamline is constant is also tested.

$$\sum_{n=1}^{N-1} |\Psi_{n+1} - \Psi_n| \approx 0 \tag{12}$$

As shown in figure 4, this value is held almost completley constant.

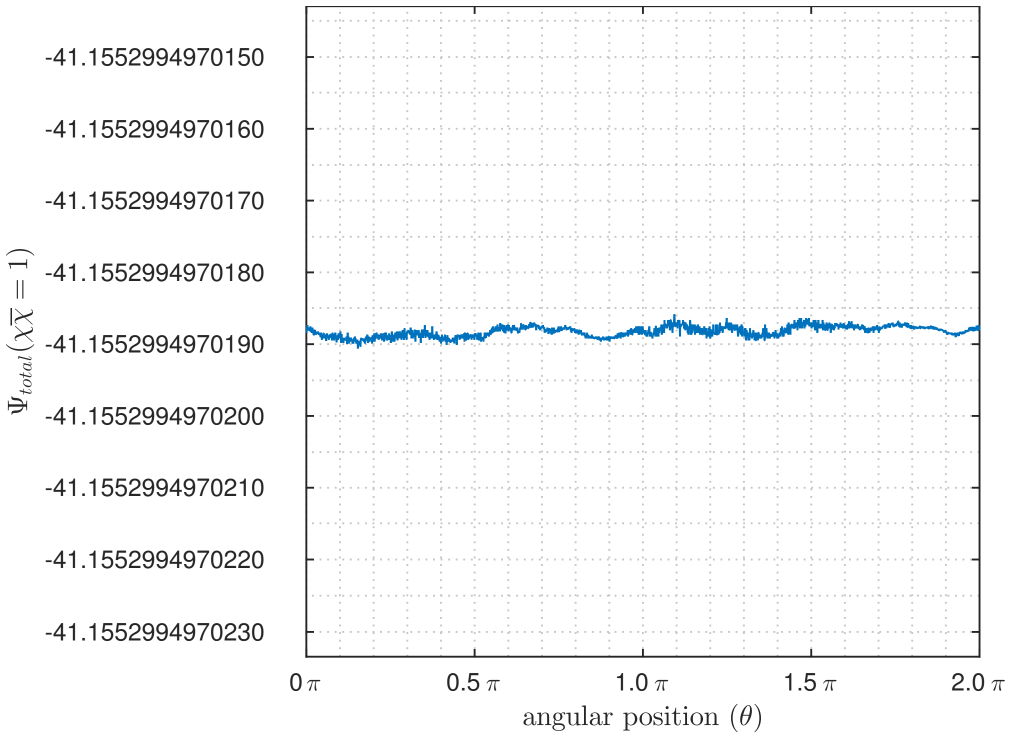

Figure 4: Streamline values around ellipse

2 Intersecting Branch Cut

2.1 Uncorrected Results

Special attention must be paid to the special case in which the branch cut of the discharge feature (well in this case) intersects the impermeable object. As shown in figure 5, when the well is moved to $z_w = 700 - 350i$, its intersecting branch cut renders the flownet incorrect. While this flownet is flawed for several reasons, the most pressing violation is that of the constraint of constant streamline around the boundary. This is confirmed in figure 6, which, when compared to the appropriately constant values in figure 4, confirms this violation.

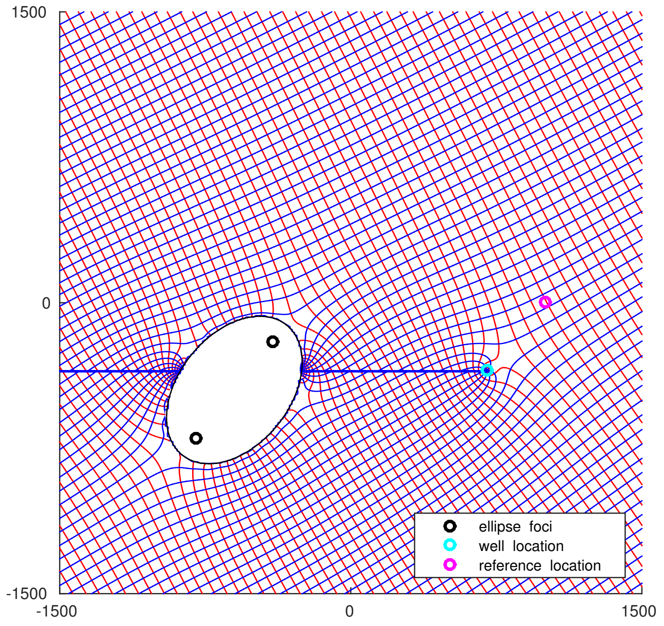

Figure 5: Uncorrected flow net for intersecting branch cut

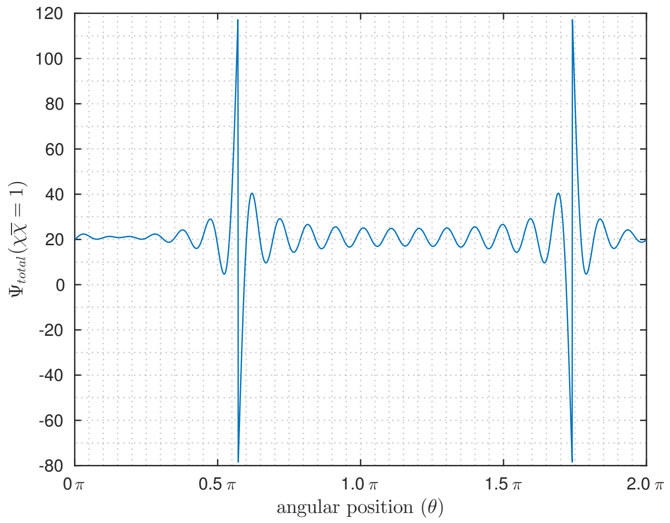

Figure 6: Uncorrected streamline around ellipse with intersecting branch cut

2.2 Correction Method 1 - Re-orientation of Branch Cut

In the case of a single impermeable object with a single well, the issues related with this intersection can be addressed simply by realigning the single branch cut such that no intersection occurs. This can be done by modifying the well term such that the branch cut it rotated by Θ (with Θ in the left, horizontal direction) as follows:

$$\Omega_{\text{well}} = \frac{Q_w}{2\pi} \ln\left[(z - z_w)\, e^{-i\Theta}\right] \tag{13}$$

The re-orientation shown in 7 is indeed an acceptable solution, as confirmed by the nearly constant streamline along the boundary shown in figure 8.

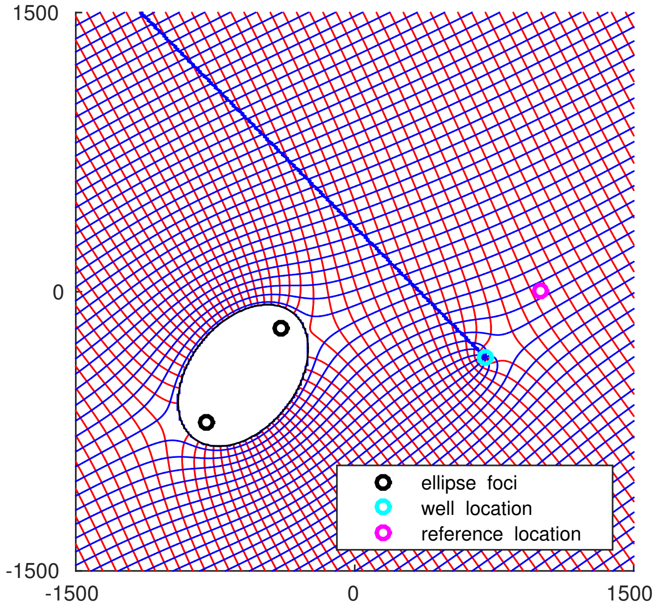

Figure 7: Flow net with re-oriented branch cut $\left(\Theta = \frac{\pi}{4}\right)$

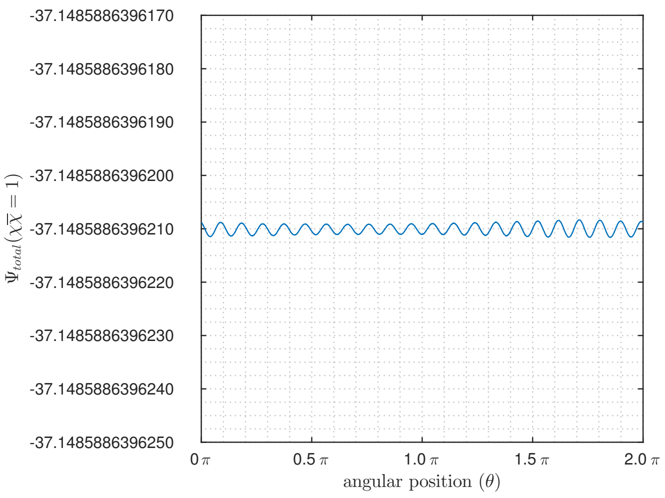

Figure 8: Corrected streamline around ellipse with with re-oriented branch cut

2.3 Correction Method 2 - Ψ_{other} Offset

2.3.1 Jump Offset with 1 Well

While the re-alignment method discussed section 2.2 is preferably straight forward, it becomes less applicable in more complicated cases. That is, in the event that several impermeable objects are modelled within the aquifer, the branch cut has more obstacles to dodge. Such a case would be further complicated by the introduction of multiple discharge features (more wells, line sinks, etc.), as each of their branch cuts would need to be separately aligned away from the impermeable objects.

Therefore, a more general solution is to offset the anticipated jump that each feature's branch cut will have on an impermeable object's Ψ_{other} during the Cauchy integral process. As shown in figure 9, this intersection causes a severe jump in the streamline values fed into the Cauchy integral process. This in turn yields incorrect α coefficients, which, when used in eq. 6, produce the flawed flownet shown in figure 5.

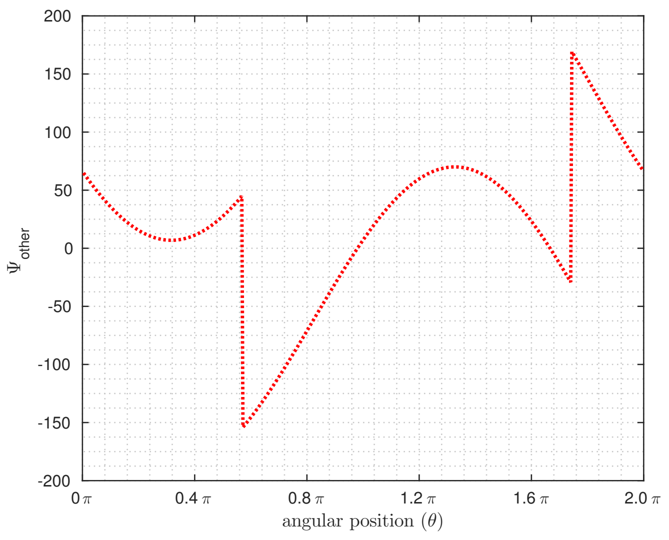

Figure 9: Uncorrected Ψ_{other} used in Cauchy integral with intersecting branch cut

Therefore, this jump can be fixed my modifying the Cauchy integral process such that when a large difference between any two nodes' streamlines values is detected, it is calculated and summed back in order to be removed. For a set of 500 nodes ($N = 500$), a jump level of $\Delta\Psi = 15$ is more than enough to detect this jump. This is done by correcting $\Psi_{\text{other}_{\text{unfixed}}}$ into $\Psi_{\text{other}_{\text{fixed}}}$ with the lines of code shown below:

```
% --- create array detecting jump ----------------------------------------
jump_detection = 15;
pos_neg = 1;
jump_tracker = 100*ones(1,N);
for n = 1:N
   if n ~= 1
     if abs(Psi_other_unfixed(n) - Psi_other_unfixed(n-1)) > jump_detection

           pos_neg = -1*pos_neg;          %change sign for each jump crossed
     end
        jump_tracker(n) = pos_neg;
   end
end

% --- correct Psi values each time detected jump occurs  -----------------
offset = 0;
for n = 1:N
    if n ~= 1
        if jump_tracker(n) ~= jump_tracker(n-1)
            offset = Psi_other_unfixed(n) - Psi_other_unfixed(n-1);
        end
    end
            Psi_other_unfixed(n) = Psi_other_unfixed(n) - offset;
end
Psi_other_fixed = Psi_other_unfixed;
```

This is a passage from a larger Cauchy integral script shown in appendix section A.2. Before these lines are executed, the array Ψ_{other} has been determined. This section accounts for multiple well branch cuts as well as changing positive/negative jump order.

Using this correction process, the Ψ_{other} is corrected as shown in figure 10.

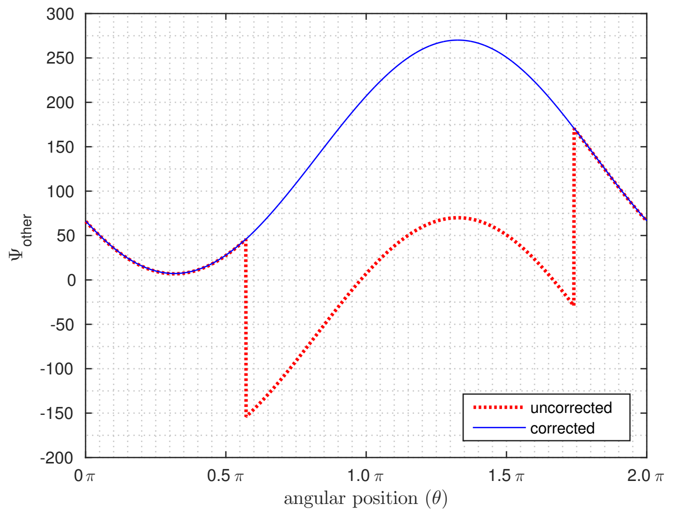

Figure 10: Correction of Ψ_{other} used in Cauchy integral

This correction yields the fixed flownet shown in figure 11.

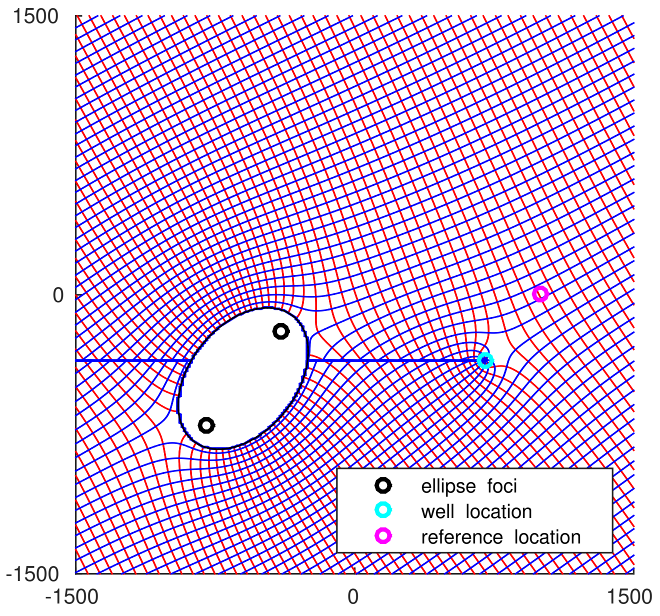

Figure 11: Flownet exhibiting correction using Ψ_{other} offset method

It is noted that this jump-removal process technically breaks the constraint of having a constant streamline value along the ellipse boundary. As shown in figure 12, this process yields two separate values for $\Psi_{\text{other}}(\chi\overline{\chi} = 1)$.

This is expected however, given that the difference between both values shown is the same as the jump removed from Ψ_{other} as shown in figure 10.

$$\Delta\Psi\left(\chi\overline{\chi} = 1\right) = -61.63 - 137.2 = -198.83$$

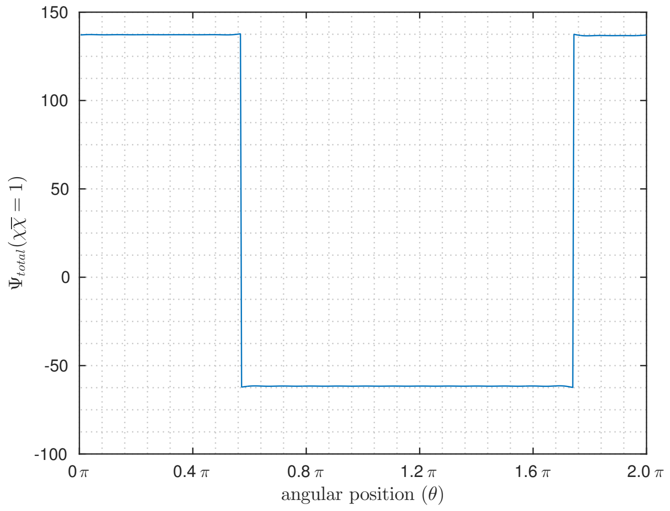

Figure 12: Corrected streamline around ellipse with jump-correction method

2.3.2 Jump Offset Robustness Test - 2 wells

As a test of robustness, this offset method is used with several wells, causing different orders in negative and positive jumps in the Ψ_{other} offset routine. In this first additional test, another well is added such that $z_{w_1} = 700 - 350i$ and $z_{w_2} = -700 - 1000i$. Their branch cuts are oriented to *intentionally* intersect the impermeable ellipse. Both their discharges are kept the same as the single well from before.

It should be noted that, as shown in figure 15, there are in fact two negative jumps in Ψ_{other} to be overcome. This is due to the fact that, when travelling around the ellipse boundary upwards form $\theta = 0$, there are in fact two *exiting* branch cuts to be overcome before any *entering* branch cuts are encountered. Therefore, two compounding jumps are addressed, resulting in three separate Ψ values along the ellipse boundary as shown in figure 14.

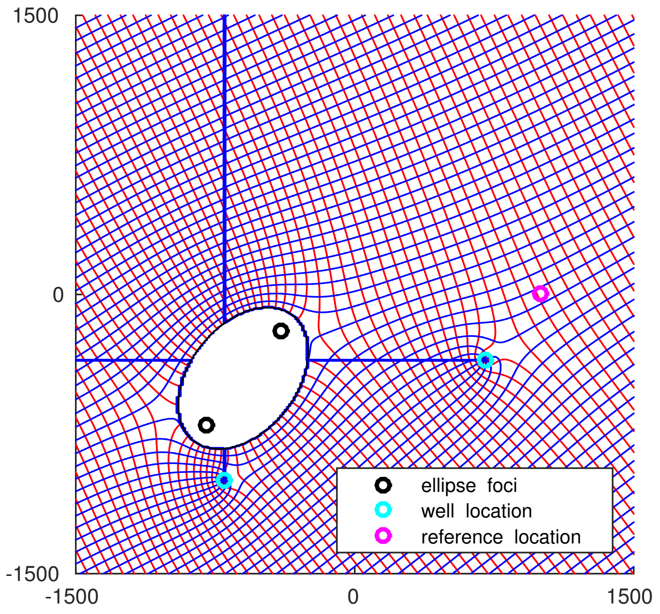

Figure 13: Flownet for jump-offset correction, 2 wells

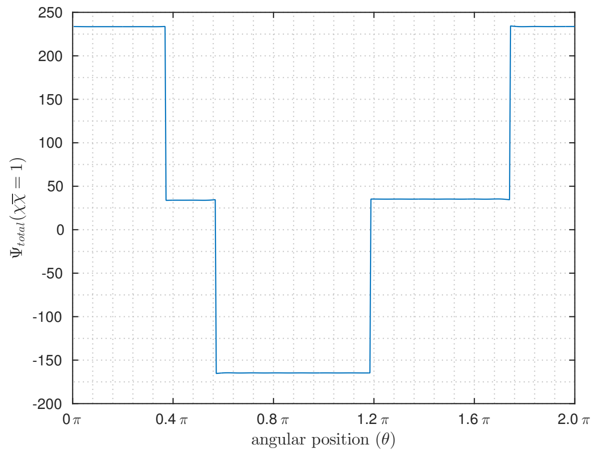

Figure 14: Boundary Ψ after jump correction, 2 wells

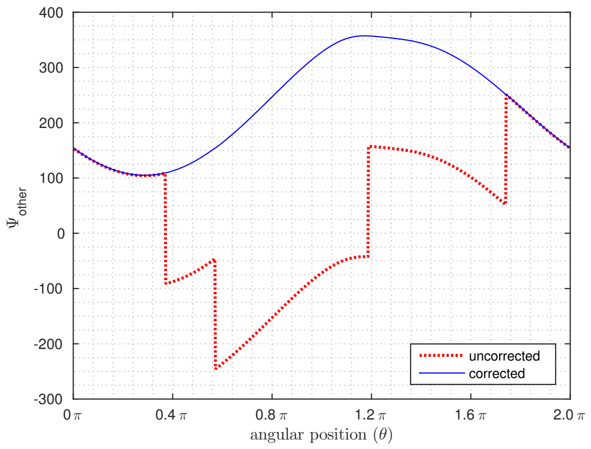

Figure 15: Correction of Ψ_{other}, 2 wells

2.3.3 Jump Offset Robustness Test - 3 wells

This method again holds up with the addition of another well of the same discharge, where $z_{w_1} = 700 - 350i$, $z_{w_2} = -700 - 1000i$ and $z_{w_3} = -1000 + 0i$. In a way, this test is presents a more intuitive correction than that shown for 2 wells. This is because, when walking along the ellipse boundary upwards from $\theta = 0$, any exiting/entering branch cut is offset either directly before or after addressing another of the same discharge but opposite direction. Therefore, as shown in figure 17, there is only one jump value to be fixed throughout $0 \leq \theta \leq 2\pi$, resulting in only two Ψ values for the same domain along the final ellipse boundary.

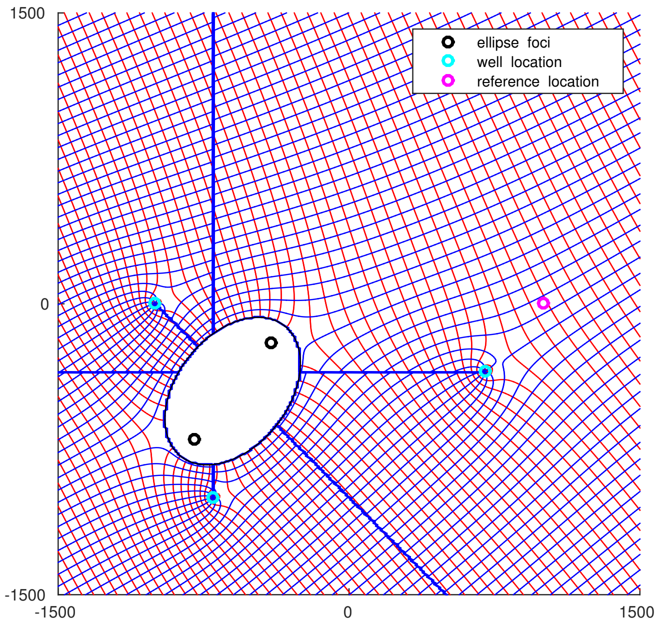

Figure 16: Flownet for jump-offset correction, 3 wells

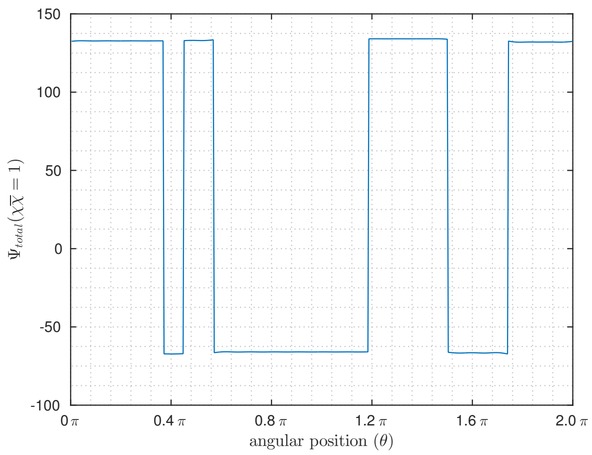

Figure 17: Boundary Ψ after jump correction, 3 wells

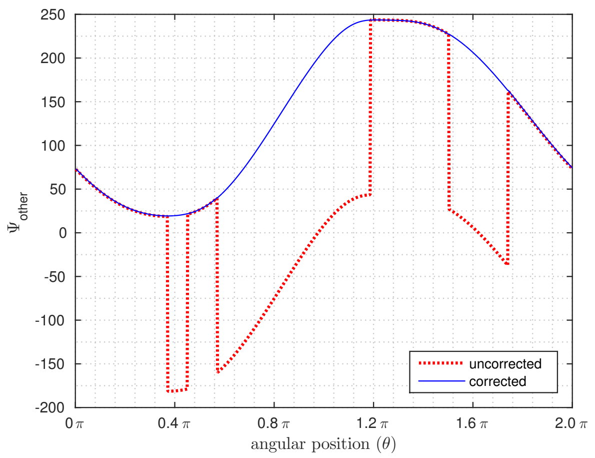

Figure 18: Correction of Ψ_{other}, 3 wells

A Appendix

A.1 Master Script

```
clc
close all
clear all

Qw = 200;                            % well discharge [m^3/day]
Qx0 = -0.4;                           % magnitude of uniform flow [m^2/day]
beta = pi/6;                         % orientation of uniform flow [rad]
roe = 1.5;                           % ratio of ellipse major and semi axis

W0 = Qx0*exp(-1i*beta);

z1 = complex( -800,-700);            % ellipse first focal point
z2 = complex( -400,-200);             % ellipse second focal point
zw = complex( 700, -350);             % well location
z_ref= complex(1000,0);              % location of know head
phi_ref = 28;                        % head at z_ref [m]

base = 0;                            % aquifer datum set to zero [m]
k = 10;                              % hydraulic conductivity [m/day]

nu = func_nu(roe);                   % ellipse geometry parameter
rw = 0.05;                           % well ra%dius [m]

% --- design resolutions ---------------------------------------------------

N = 500;                                                         %lake nodes
if Qw == 0;
m = 1;                  %taylor expansion coefficients, 1 with no well flow
else
m = 20;          %taylor expansion coefficients, at least 20 with well flow
end
N_grid = 100;                                           %flow net resolution

nint = 20;                                        %flow net contour quantity

% --- get alpha(s) from cauchy integral ------------------------------------

alpha = Cauchy_integral(N,m,z1,z2,nu,W0,zw,Qw,rw);

% --- calibrate omega total for knwon reference head and check -------------

Phi_ref = Phi_of_phi(phi_ref,base,k);
C = Phi_ref - real(Omega_total_noncal(z_ref,alpha,m,z1,z2,nu,W0,zw,Qw,rw));

assert(real(Omega_total_cal(z_ref,alpha,m,z1,z2,nu,W0,zw,Qw,rw,C))==...
                             Phi_ref,'reference head condition not met');

% --- check ellipse boundary condition (Psi is constant) -------------------
boundary_check_tolerance = 2e-1;

z_check = Ellipse_point_marker(roe,z1,z2,N,0);
Psi_check = zeros(1,length(z_check));
for ii = 1:length(z_check)
Psi_check(ii) = imag(Omega_total_cal(z_check(ii),...
                                          alpha,m,z1,z2,nu,W0,zw,Qw,rw,C));
if ii~=1
    if abs(Psi_check(ii) - Psi_check(ii-1)) > boundary_check_tolerance
  disp('error - lake boundary streamline not constant where');
disp(['Psi= ' num2str(Psi_check(ii-1)) ' at z= '  ...
                                          num2str(z_check(ii-1)) ' and ']);
disp(['Psi= ' num2str(Psi_check(ii)) ' at z= ' ...
                             num2str(z_check(ii)) ' at node ' num2str(ii)]);
    end
end
end

% --- plot flow net --------------------------------------------------------

wind = 1500;

figure;
ContourMe_flow_net(-wind, 2.2*wind, N_grid, -wind, wind, N_grid,...
               @(z)Omega_total_cal(z,alpha,m,z1,z2,nu,W0,zw,Qw,rw,C),nint);
Ellipse_aoverb(roe,z1,z2,10*N );
lwell = plot(real(zw),imag(zw),'co','linewidth',2);
%plot(-700,-1000,'co','linewidth',2)
%plot(-1000, 0,'co','linewidth',2)

lfoc = plot(real([z1 z2]),imag([z1 z2]),'ko','linewidth',2);
lref = plot(real(z_ref),imag(z_ref),'mo','linewidth',2);
ax = gca;
ax.XTick = [-wind 0 wind];
ax.YTick = [-wind 0 wind];
hold off
legend ([lfoc ,lwell ,lref],{'ellipse  foci','well  location',...
    'reference  location '},'Location','northeast');
axis([-wind wind -wind wind])
%print('140','-depsc2','-r300');
```

A.2 Cauchy Integral

```
function [ alpha ] = Cauchy_integral(N,m,z1,z2,nu,W0,zw,Qw,rw)

if N<2*m
    N=2*m;
end

deltheta=2*pi/N;
cauch = zeros(N,m);
a=zeros(1,m);

%%% sample Psi around element boundary,
  % (noting value includes well branch cut)

Psi_other_unfixed = zeros(1,N);
theta = zeros(1,N);
for n = 1:N
theta(n) = n*deltheta;
chi=exp(1i*theta(n));
z = z_of_chi(chi,nu,z1,z2 );
Psi_other_unfixed(n) = imag(Omega_other(z,W0,zw,Qw,rw));
end

% --- create array detecting jump ------------------------------------------
jump_detection = 15;
pos_neg = 1;
jump_tracker = 100*ones(1,N);
for n = 1:N
   if n ~= 1
     if abs(Psi_other_unfixed(n) - Psi_other_unfixed(n-1)) > jump_detection

           pos_neg = -1*pos_neg;         %change sign for each jump crossed
     end
        jump_tracker(n) = pos_neg;
   end
end

% --- correct Psi values each time detected jump occurs  ------------------
offset = 0;
for n = 1:N
    if n ~= 1
        if jump_tracker(n) ~= jump_tracker(n-1)
            offset = Psi_other_unfixed(n) - Psi_other_unfixed(n-1);
        end
    end
            Psi_other_unfixed(n) = Psi_other_unfixed(n) - offset;
end
Psi_other_fixed = Psi_other_unfixed;

% --- calculate cauchy coefficients ----------------------------------------
for n = 1:N
    for jj = 1:m
        cauch(n,jj) = Psi_other_fixed(n)*exp(-1i*jj*theta(n));
    end
end

% --- calculate a's --------------------------------------------------------
for jj = 1:m
    a(1,jj) = 0;
        for n = 1:N
            a(1,jj) = a(1,jj)+cauch(n,jj);
        end
end

% --- export conugate alpha's ----------------------------------------------
a = 2i*a/N;
alpha = conj(a);

end
```

A.3 Function Scripts

A.3.1 $\mathbf{Z = f(z, z_1, z_2)}$

```
function [ Z] = bigZ_of_z(z,z1,z2)
Z = (z - 0.5*(z1+z2))/(0.5*(z2-z1));
end
```

A.3.2 $\chi = \mathbf{f}(\mathbf{Z}, \nu)$

```
function [ chi ] = chi_of_bigZ(Z,nu)
chi = nu*(Z + sqrt(Z+1)*sqrt(Z-1));
end
```

A.3.3 $\nu = \mathbf{f}(\rho)$

```
function [nu] = func_nu(roe)
A = 1/sqrt(1 - roe^(-2));
nu = A - sqrt(A^2 - 1);
end
```

A.3.4 $\chi = \mathbf{f}(\mathbf{z}, \mathbf{z_1}, \mathbf{z_2}, \nu)$

```
function [ chi ] = chi_of_z(z,z1,z2,nu)
Z = (z - 0.5*(z1+z2))/(0.5*(z2-z1));
chi = nu*(Z + sqrt(Z+1)*sqrt(Z-1));
end
```

A.3.5 $\mathbf{z} = \mathbf{f}(\chi)$

```
function [ z ] = z_of_chi(chi,nu,z1,z2 )
bigZ = 0.5*(chi/nu + nu/chi);
z = 0.5*bigZ*(z2 - z1) + 0.5*(z2+z1);
end
```

A.3.6 Uniform Flow Term

```
function [ Omega_uf ] = Omega_uf( W0,z )
Omega_uf = W0*z;
end
```

A.3.7 Well Term

```
function [ Omega_well ] = Omega_well(z,zw,Qw,rw)
if z ~= zw
Omega_well = Qw/(2*pi)*log((exp(complex(0,0*-pi)))*(z - zw)/rw);
else
Omega_well = NaN;
end
```

A.3.8 $\mathbf{\Omega_{other}}$

```
function [ Omega_other ] = Omega_other(z,W0,zw,Qw,rw)
Omega_other = Omega_uf( W0,z ) +  Omega_well(z,zw,Qw,rw);
end
```

A.3.9 Uncalibrated $\mathbf{\Omega_{total}}$

```
function [ Omega_total ] = Omega_total_noncal(z,alpha,m,z1,z2,nu,W0,zw,Qw,rw)

bigZ = bigZ_of_z(z,z1,z2);
chi_m = chi_of_bigZ(bigZ,nu);

Omega_m = 0;
for jj = 1:m
Omega_m = Omega_m +alpha(jj)*chi_m^(-jj);
end

Omega_total = Omega_m + Omega_well(z,zw,Qw,rw) + Omega_uf( W0,z );

end
```

A.3.10 Calibrated $\mathbf{\Omega_{total}}$

```
function [ Omega_total ] = Omega_total_cal(z,alpha,m,z1,z2,nu,W0,zw,Qw,rw,C)

bigZ = bigZ_of_z(z,z1,z2);
chi_m = chi_of_bigZ(bigZ,nu);

   if chi_m*conj(chi_m) < 1 - 10^(-10)
        Omega_total = NaN;

   else
        Omega_m = 0;
     for jj = 1:m
        Omega_m = Omega_m +alpha(jj)*chi_m^(-jj);
     end

  Omega_total = Omega_m + Omega_well(z,zw,Qw,rw) + Omega_uf( W0,z )+C;


   end
end
```

A.3.11 $\mathbf{\Phi} = \mathbf{f}(\phi)$

```
function [Phi] = Phi_of_phi(phi,base,k)
if phi > base
Phi = 0.5*k*(phi-base)^2;
elseif phi < base
Phi = 0;
end
```

A.3.12 $\phi = \mathbf{f}(\mathbf{\Phi})$

```
%%% function file for dertermining discharge potential from head
function [phi] = Phi_to_phi(Phi,base,k)
phi = base + sqrt(2*Phi/k);
end
```

A.3.13 Ellipse Point Generator

```
function [z_check] = Ellipse_point_marker( roe,z1,z2,npoints,plot_q)
% Ratio the major principal axis dividded by the minor one
% z1 and z2 are the foci
t=0;
arg=(roe-1)/(roe+1);
nu=sqrt(arg);
n=npoints;
delt=2*pi/n;
x=zeros(1,n);
y=zeros(1,n);
for i=1:n
    t=t+delt;
    chi=cos(t)+1i*sin(t);
    Z=0.5*(chi/nu+nu/chi);
    z=0.5*(z2-z1)*Z+0.5*(z1+z2);
    x(i)=real(z);
    y(i)=imag(z);
end
z_check  = complex(x, y);
if plot_q == 1
plot(x,y,'ro');
axis equal;
hold on;
xe=zeros(1,2);
ye=zeros(1,2);
xe(1,1)=real(z1);
xe(1,2)=real(z2);
ye(1,1)=imag(z1);
ye(1,2)=imag(z2);
plot(xe,ye,'ro');
end
end
```

A.4 Plotting Routines

A.4.1 Ellipse Plotter

```
function [] = Ellipse_aoverb(roe,z1,z2,npoints )
% Ratio the major principal axis dividded by the minor one
% z1 and z2 are the foci
t=0;
arg=(roe-1)/(roe+1);
nu=sqrt(arg);
n=npoints;
delt=2*pi/n;
x=zeros(1,n);
y=zeros(1,n);
for i=1:n
    t=t+delt;
    chi=cos(t)+1i*sin(t);
    Z=0.5*(chi/nu+nu/chi);
    z=0.5*(z2-z1)*Z+0.5*(z1+z2);
    x(i)=real(z);
    y(i)=imag(z);
end
plot(x,y,'k');
axis equal;
hold on;
end
```

A.4.2 Flownet Routine

```
function [Grid] = ContourMe_flow_net(xfrom, xto, Nx, yfrom, yto, Ny,func,nint)

Grid = zeros(Ny,Nx);

X = linspace(xfrom, xto, Nx);
Y = linspace(yfrom, yto, Ny);

for row = 1:Ny
    for col = 1:Nx
        Grid(row,col) = func(complex( X(col), Y(row) ) );
    end
end
Bmax=max(imag(Grid));
Bmin=min(imag(Grid));
Cmax=max(Bmax);
Cmin=min(Bmin);
D=Cmax-Cmin;
del=D/nint;
Bmax=max(real(Grid));
Bmin=min(real(Grid));
Cmax=max(Bmax);
Cmin=min(Bmin);
D=Cmax-Cmin;
nintr=round(D/del);

hold on
contour(X, Y,real(Grid),nintr,'r');
contour(X, Y,imag(Grid),nint,'b');

axis square
axis equal
```