DAFTAR PUSTAKA

Andriani, F.D., Suwandi, E.A., 2016. *Pemodelan Kedepan Geolistrik Resistivitas Menggunakan Metode Beda Hingga (Kasus 2D: Model Lapisan Yang Homogen)*. Prosiding Seminar Nasional Fisika (Ejournal). 5 : 74-77.

Asih, T.S.N., Waluya, B.St., Supriyono, 2018. *Perbandingan Finite Diference Method dan Finite Element Method Dalam Mencari Solusi Persamaan Diferensial Parsial*. Prisma. 1 :885-888.

Febriana, R.KN., Minarto, E. dan Tryono, F.X.Y 2017. *Identifikasi Sebaran Aliran Air Bawah Tanah (Groundwater) dengan Metode Vertical Electrical Sounding (VES) Konfigurasi Schlumberger di Wilayah Cepu, Blora Jawa Tengah*. Jurnal Sains dan Seni ITS, 6(2), 29-33.

Grandis, H. 2009. *Pengantar Pemodelan Inversi Geofisika*. Bandung: Himpunan Ahli Geofisika Indonesia (HAGI).

Hendrajaya, L. 1990. *Geolistrik Tahanan Jenis. Laboratorium Fisika Bumi*. Bandung: Institut Teknologi Bandung

Irfan AKCA, 2016. *Elris2D: A Matlab Package for The 2D of DC Resistivity/Ip Data*. Acta Geophysica. 64(2) : 443-462.

Lines, L.R. and Treitel, S. 1984. *A Review of Least-Squares Inversion and Its Application to Geophysical Problems*. Geophysical Prospecting, 32, 159-186.

Loke, M., H. 2018. *Tutorial 2-D and 3-D Electrical Imaging Surveys*.

Menke, W.1984. *Geophysical Data Analysis: Understanding Invers Problem Theory and Practice*. Tulsa Oklahoma.Society of Exploration Geophysics Course Notes Series, No. 6 Ist Edn., SEG Publisher.

Munir, R.2010. *Metode Numerik*. Bandung: Penerbit Informatika.

Roy, I.G. 1999. *An Efficient Non-Linier Least-Squares 1D Inversion Scheme for Resisitivity and IP Sounding data*. Geophys. Prospect.

Telford,W.M., Geldart, L.P., Sherif, R.E. 1990. *Applied Geophysics Second Edition*. Cambridge, United Kingdom : Cambridge University Press.

Vebrianto, S. 2015. *Eksplorasi Metode Geolistrik: Resistivitas Polarisasi Terinduksi dan Potensial Diri*. Malang : Universitas Brawijaya Press (UB Press).

Lampiran 1: Tampilan Script Program *Elris2D*

```
elseif ischar(fid)
    data=[];
    pathname=pwd;
    return
end

data.prfadi=fgetl(fid);
data.ela=fscanf(fid,'%f',1);
data.eldiz=fscanf(fid,'%d',1);%1: Wenner, 2:Pole-pole 3: Dipole-dipole 6: Pole-dipole 7: Wenner-Schlumberger
if data.eldiz==11
    data.subeldiz=fscanf(fid,'%d',1);
    fgetl(fid);
    fgetl(fid);
    fgetl(fid);
end
data.nd=fscanf(fid,'%d',1);
data.mp=fscanf(fid,'%d',1);%MIDPOINT
data.ip=fscanf(fid,'%d',1);%IP

data=oku(data,fid); %Call data reader function
topog=fscanf(fid,'%d',1);
data.topog=topog;
if topog
    topsay=fscanf(fid,'%d',1);
    data.topo = fscanf(fid, '%g %g', [2 topsay])';    % Read topography data
else
    fscanf(fid,'%d',5) ;
    fscanf(fid,'%s',2);
    data.sd=fscanf(fid, '%g ', [1 data.nd])';
end
```

forward.m

```
function [J,ro]=forward(yky,t,es,sig,so,nel,akel,opt,tev,k1,x,V1,data,prho,npar,par,p)
% Forward calculation routine
% This function calculates and returns the response of a 2D resistivity
% model and the Jacobian matrix J.
tt=zeros(nel,nel,npar);
xx2=zeros(nel,nel);
[delta,b1,c1,b2,c2,b3,c3]=pdetrgm(p,t);
pl_1=sig./(4.*delta);
el=1:es;
%% Finite Elements
for nky=1:length(yky)

    a=sig.*(1/6)*yky(nky).^2.*delta;
    b=a/2;
    k11(el)=pl_1.*((b1(el).^2+c1(el).^2))+a;
    k11(el)=pl_1.*((b1(el).^2+c1(el).^2))+a;
    k12(el)=pl_1.*((b1(el).*b2(el)+c1(el).*c2(el)))+b;
    k13(el)=pl_1.*((b1(el).*b3(el)+c1(el).*c3(el)))+b;
    k21(el)=k12(el);
    k22(el)=pl_1.*((b2(el).^2+c2(el).^2))+a;
    k23(el)=pl_1.*(b2(el).*b3(el)+c2(el).*c3(el))+b;
    k31(el)=k13(el);
    k32(el)=k23(el);
    k33(el)=pl_1.*(b3(el).^2+c3(el).^2)+a;
    k11=sparse([k11,k12,k13,k21,k22,k23,k31,k32,k33]);
    K1=accumarray(x,k11);
    fi1=K1\so;
    for ael=1:nel
        yuzpot1=fi1(akel,ael);
        xx1(:,ael)=full(yuzpot1');
    end
end
V0=xx1*yky(1)/pi;
```

\

k1f.m

```
function [V1,k1,x]=k1f(t,es,delta,b1,b2,b3,c1,c2,c3,so,nel,akel,yky)
x1=t(1,:)';x2=t(2,:)';x3=t(3,:)';
x=[x1,x1;x1,x2;x1,x3;x2,x1;x2,x2;x2,x3;x3,x1;x3,x2;x3,x3];
p1_1=1./(4.*delta);
e1=1:es;
for nky=1:length(yky)
    % p2_1=;
    a=(1/6)*yky(nky).^2.*delta;%p2_1;
    b=a/2;
    k11(e1)=p1_1.*((b1(e1).^2+c1(e1).^2))+a;
    k11(e1)=p1_1.*((b1(e1).^2+c1(e1).^2))+a;
    k12(e1)=p1_1.*((b1(e1).*b2(e1)+c1(e1).*c2(e1)))+b;
    k13(e1)=p1_1.*((b1(e1).*b3(e1)+c1(e1).*c3(e1)))+b;
    k21(e1)=k12(e1);%p1*(b(1)*b(2)+c(1)*c(2))+(1/12)*p2;
    k22(e1)=p1_1.*((b2(e1).^2+c2(e1).^2))+a;
    k23(e1)=p1_1.*(b2(e1).*b3(e1)+c2(e1).*c3(e1))+b;
    k31(e1)=k13(e1);%p1*(b(1)*b(3)+c(1)*c(3))+(1/12)*p2;
    k32(e1)=k23(e1);%p1*(b(2)*b(3)+c(2)*c(3))+(1/12)*p2;
    k33(e1)=p1_1.*(b3(e1).^2+c3(e1).^2)+a;
    k1=sparse([k11,k12,k13,k21,k22,k23,k31,k32,k33]);
    K1=accumarray(x,k1);
    fi1=K1\so;
    for ae1=1:nel
        yuzpot1=fi1(akel,ae1);
        xx1(:,ae1)=full(yuzpot1');
    end

end

V1=xx1*yky(1)/pi/1.5;
```

forward.m

```
        end
    end
    V0=xx1*yky(1)/pi;
    %% Calculate apparent resistivities
    [ro]=go2d(V0,V1,data.pat,data.eldiz);
    %% Calculate Jacobian Matrix if it is requested
    if opt==1
        for pno=1:npar
            ccd=sparse(length(fil),length(fil));
            uc=par(pno).ucg;
            for m=1:length(uc)
                CCD=sparse(length(fil),length(fil));
                d=uc(m):es:length(t)*9;
                cevap=full(k1(d));
                for id=1:9
                    CCD(x(d(id),1),x(d(id),2))=cevap(id);
                end
                ccd=ccd+CCD;
            end
            uc=[];
            s1=-ccd*fil;
            fitur=K1\s1;
            tt(:,:,pno)=(fitur(akel,:))';
        end
    end

    if opt==1
        VVT=tt.*yky(1)/pi;
        J=jacob(VVT,V1,data,ro,prho,npar);
    else
        J=0;

        assignin('base','ro',ro)
        assignin('base','J',J)
    end
```

meshgen.m

```
function [p,t,nlay,tev,par,npar,zi]=meshgen(data)

% Mesh generator. 'normal' mode is selected. Model space is constructed by
% dividing rectangular blocks into two triangles. Outer part of the mesh is
% unstructured. Unstructured mesh is produced by Triangle program
dz(1)=data.dz1/1.5;

for i=2:1000
    if sum(dz)<=data.zmax*1.1
        zk=dz(i-1)*1.1;
        dz(i)=zk;
        nlay=i;
    end
end

z=cumsum(dz);
oran=max(z)/data.zmax;
z=z/oran;
zi=-[0 z];


ek=10;

x1=data.xelek(1)-ek*data.ela;
x2=data.xelek(1)*ones(size(zi));
x3=data.xelek(end)*ones(size(zi));
x4=data.xelek(end)+ek*data.ela;
zdoi=zi(end);

zcerceve=[0  zi ones(1,data.nel-2)*zi(end) fliplr(zi)  0 -6*data.zmax -6*data.zmax]';
xcerceve=[x1 x2 data.xelek(2:end-1)' x3 x4 x4 x1];
[x,y]=meshgrid(data.xelek(2:end-1),zi(1:end-1));
```

meshgen.m

```
%preparing input file for Triangle

pfix=data.filename(1:end-4);
fidp=fopen([pfix,'.poly'],'w');
np=length(xcerceve);
fprintf(fidp,'%d 2 1 0\n',np);
for k=1:np
    fprintf(fidp,'%d %8.4f %8.4f 1\n',k,xcerceve(k),zcerceve(k));
end
fprintf(fidp,'%d 0\n',np);

for k=1:np-1
    fprintf(fidp,'%d %8.4f %8.4f 1\n',k,k,k+1);
end
fprintf(fidp,'%d %8.4f %8.4f 1\n1\n',np,np,1);
fprintf(fidp,'1 %8.4f %8.4f',-data.zmax/2,data.xelek(fix((2*data.nel-1)/2)));
fclose(fidp);
eval(['!triangle -Q -q ',pfix,'.poly'])
% p=load([pfix,'.1.node'])
[ node_num, marker ] = node_header_read ( [pfix,'.1.node']);
[ node_xy, node_marker ] = node_data_read ( [pfix,'.1.node'], node_num );
[ element_order, element_num ] = element_header_read ( [pfix,'.1.ele'] );
element_node = element_data_read ( [pfix,'.1.ele'], element_order, ...
    element_num );
t_dis=element_node;
p_dis=node_xy;

delete('*.poly');
delete('*.ele');
delete('*.node');

nz=length(zi);
p_ic=[x(:)';y(:)'];

boy=2*nz+data.nel-2;
liste_dis=2:boy+1;
ic=reshape(length(p_dis)+1:length(p_dis)+numel(x),nz-1,data.nel-2);
alt=nz+2:liste_dis(end-nz);
T=[[2:nz+1]' [ic;alt] liste_dis(end:-1:end-nz+1)'] ;
tri=[];
for k=1:size(T,2)-1
    for m=1:size(T,1)-1
        t1=[T(m,k+1) T(m,k) T(m+1,k)];
```

Command Window

go2d.m

```
function [ro]=go2d(V0,V1,IND,eldiz)

% pause
ro=zeros(1,length(IND.ind1));
switch eldiz
    case 1
        dv0=V0(IND.ind1)-V0(IND.ind2)-V0(IND.ind3)+V0(IND.ind4);
        dv1=V1(IND.ind1)-V1(IND.ind2)-V1(IND.ind3)+V1(IND.ind4);
        ro=dv0./dv1;
    case 2
        dv0=V0(IND.ind1);
        dv1=V1(IND.ind1);
        ro=dv0./dv1;

    case 3
        dv0=V0(IND.ind1)-V0(IND.ind2)-V0(IND.ind3)+V0(IND.ind4);
        dv1=V1(IND.ind1)-V1(IND.ind2)-V1(IND.ind3)+V1(IND.ind4);
        ro=dv0./dv1;
    case 6
        dv0=V0(IND.ind1)-V0(IND.ind2);
        dv1=V1(IND.ind1)-V1(IND.ind2);
        ro=dv0./dv1;
    case 7
        dv0=V0(IND.ind1)-V0(IND.ind2)-V0(IND.ind3)+V0(IND.ind4);
        dv1=V1(IND.ind1)-V1(IND.ind2)-V1(IND.ind3)+V1(IND.ind4);
        ro=dv0./dv1;
    case 11
         dv0=V0(IND.ind1)-V0(IND.ind2);
        dv1=V1(IND.ind1)-V1(IND.ind2);
        ro=dv0./dv1;

end
```

Editor - D:\elris_v121\initial.m

initial.m

```
function [sig,es,ds,akel,V1,k1,prho,so,x,pma,nu]=initial(t,p,data,yky,npar)
% Initialize model and calculate the mesh response for 1ohm-m homogenous space
ds=length(p);
es=length(t);
I=2*pi;

[delta,b1,c1,b2,c2,b3,c3]=pdetrgm(p,t);
try
    [akel]=    knnsearch(p',[data.xelek(:) data.zelek(:)]);
catch
    [tmp1,akel,tmp2]=    intersect(p',[data.xelek(:) data.zelek(:)],'rows');
end

so=spalloc(ds,data.nel,data.nel);
for ael=1:data.nel
    so(akel(ael),ael)=I;
end

if data.ip
    homma=sum(abs(data.ma))/data.nd;
    pma(1:npar)=homma;
    nu(1:es)=homma;

else
    homma=[];
    pma=[];
    nu=[];
end
sig(1:es)=(1./data.homro);


prho(1:npar)=data.homro;
[V1,k1,x]=k1f(t,es,delta,b1,b2,b3,c1,c2,c3,so,data.nel,akel,yky);
```

jacob.m

```
function J=jacob(VVT,V1,data,ro,prho,npar)
for i=1:npar
    M=VVT(:,:,i);
    J(:,i)=go2d(M,V1,data.pat,data.eldiz);
end

for ii=1:length(ro)
    for jj=1:npar
        J(ii,jj)=J(ii,jj)/prho(jj)/(ro(ii));
    end
end
```

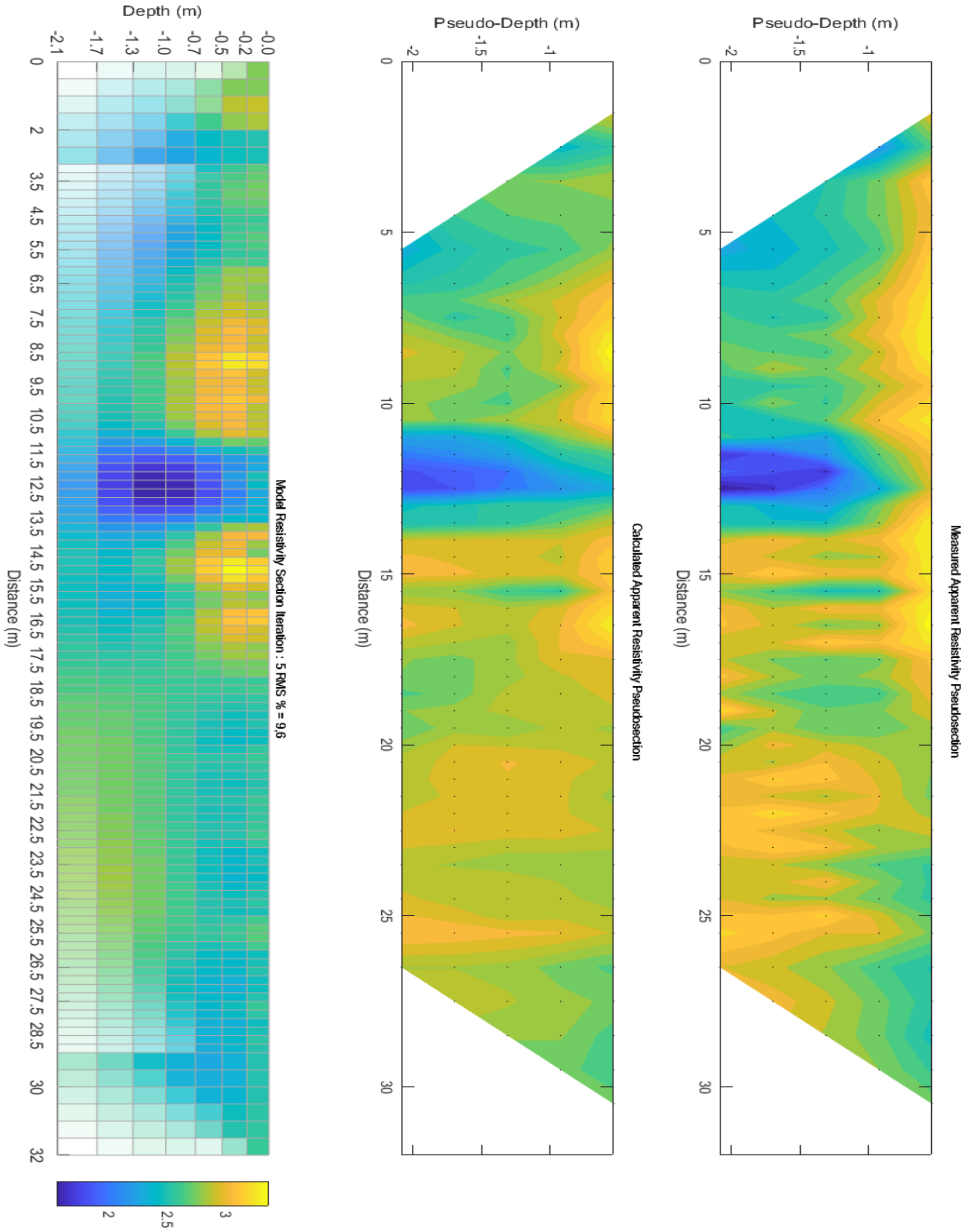
pdetrgm.m*

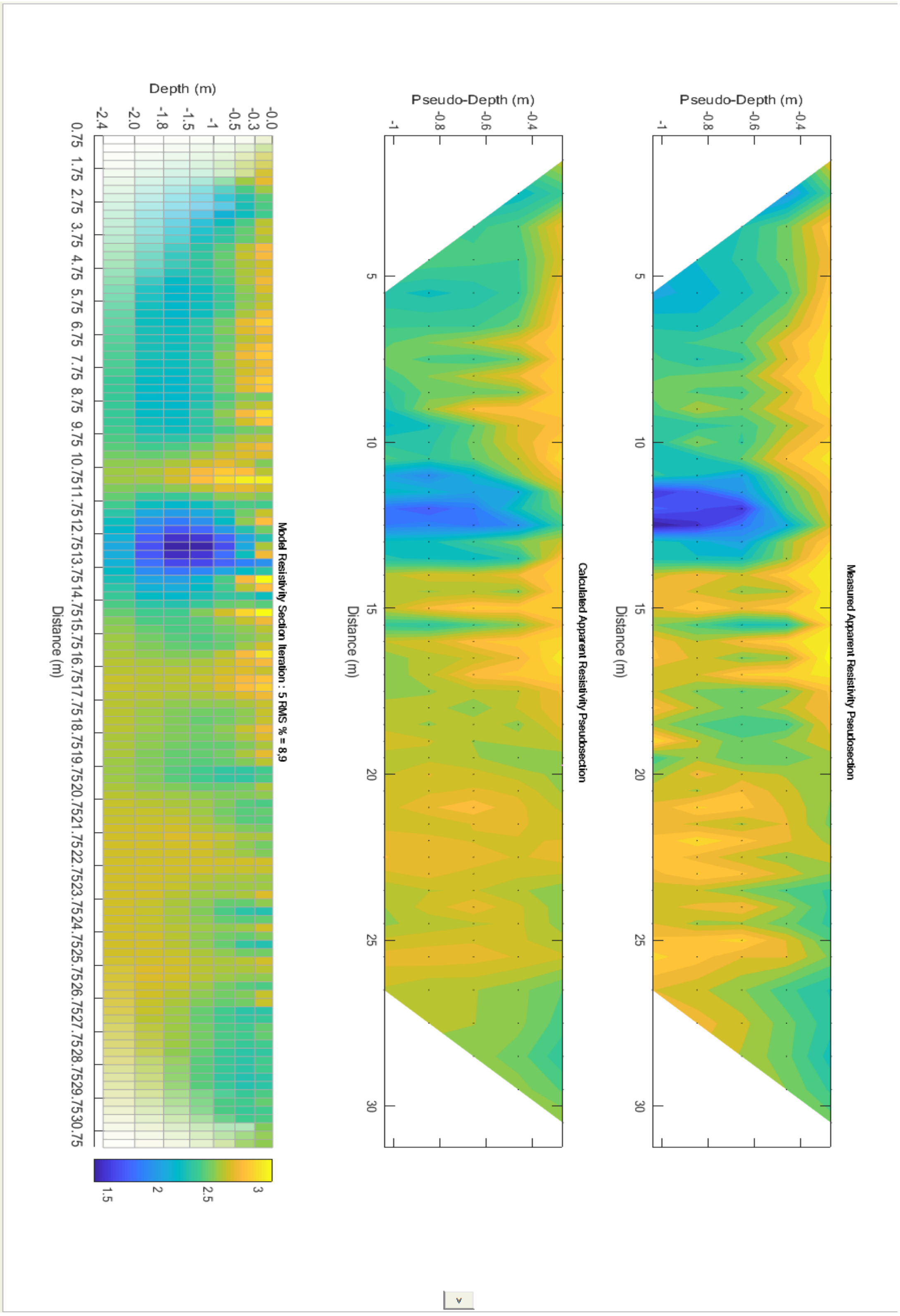
```matlab
function [ar,g1x,g1y,g2x,g2y,g3x,g3y]=pdetrg(p,t)

% Corner point indices
a1=t(1,:);
a2=t(2,:);
a3=t(3,:);

% Triangle sides
r23x=p(1,a3)-p(1,a2);
r23y=p(2,a3)-p(2,a2);
r31x=p(1,a1)-p(1,a3);
r31y=p(2,a1)-p(2,a3);
r12x=p(1,a2)-p(1,a1);
r12y=p(2,a2)-p(2,a1);

% Area
ar=abs(r31x.*r23y-r31y.*r23x)/2;

if nargout==4,
  a1=(r12x.*r31x+r12y.*r31y);
  a2=(r23x.*r12x+r23y.*r12y);
  a3=(r31x.*r23x+r31y.*r23y);
  g1x=0.25*a1./ar;
  g1y=0.25*a2./ar;
  g2x=0.25*a3./ar;
else
  g1x=-r23y;
  g1y=r23x;
  g2x=-r31y;
  g2y=r31x;
  g3x=-r12y;
  g3y=r12x;
%   g1x=-0.5*r23y./ar;
%   g1y=0.5*r23x./ar;
%   g2x=-0.5*r31y./ar;
%   g2y=0.5*r31x./ar;
%   g3x=-0.5*r12y./ar;
%   g3y=0.5*r12x./ar;
end
```

pupd.m

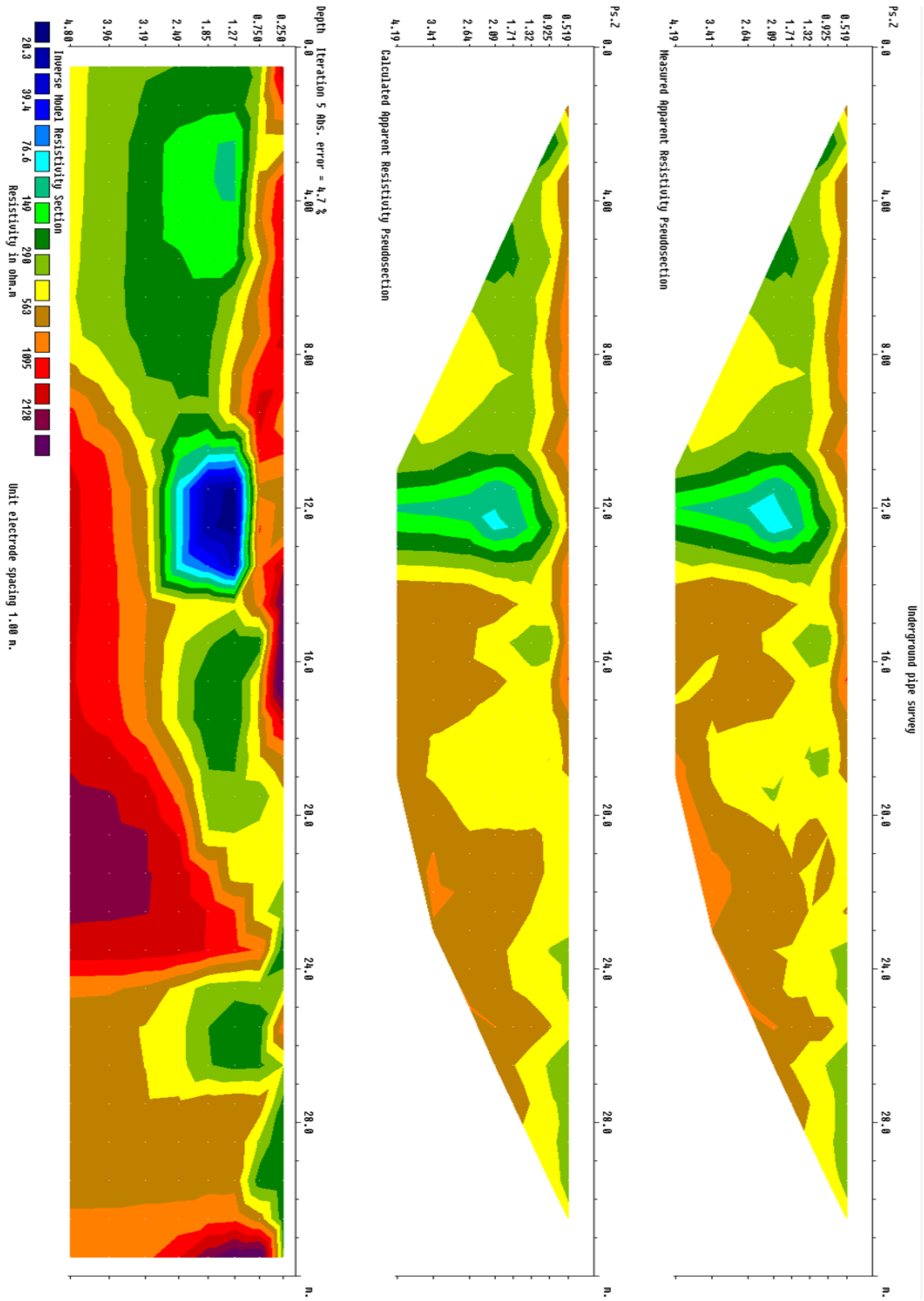
```
function [misfit,sig,prho,ro]=pupd(data,J,par,yky,tri,es,akel,tev,kl,indx,V1,prho,npar,dd,so,p,C,lambda,Rd)
%Updating model parameters

%Damping factor
while lambda<0.01
    lambda=0.01;
end
%smoothness constrained least squares
%smoothness constrain is a second order laplacian
b=(J'*Rd'*Rd*dd-lambda*C*((1./prho(:))));
A=(J'*Rd'*Rd*J+lambda*C);
dp=A\b;
parg=1./((1./prho(:)).*exp(dp));
rhoort=exp(sum(log(parg)./length(parg)));
sigtmp(1:es)=1./(rhoort);
for s=1:npar
    sigtmp(par(s).ucg)=1./parg(s);
end
% Test the updated model
[J,rog]=forward(yky,tri,es,sigtmp,so,data.nel,akel,0,tev,kl,indx,V1,data,prho,npar,par,p);
misfitg=sqrt((Rd*dd)'*(Rd*dd)/data.nd)*100;

misfit=misfitg;
ro=rog;
sig=sigtmp;
prho=parg;
```

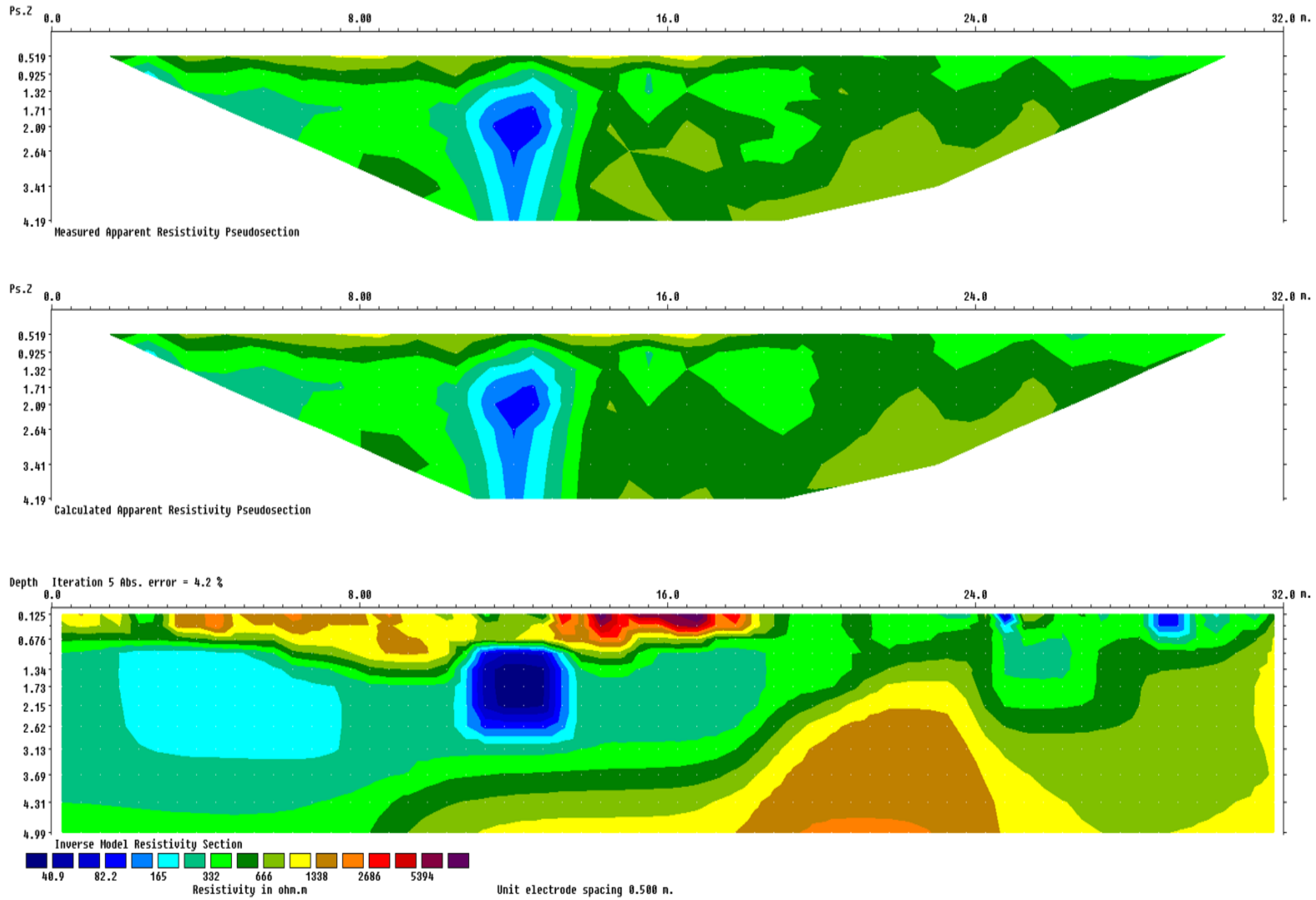
Lampiran 2: Model *Elris2D*




Measured Apparent Resistivity Pseudosection
Pseudo-Depth (m)
Distance (m)
Calculated Apparent Resistivity Pseudosection
Pseudo-Depth (m)
Distance (m)
Model Resistivity Section Iteration: 5 RMS % = 8.9
Depth (m)
Distance (m)

Lampiran 3: Model *Res2Dinv*




Ps.2
0.0
8.00
16.0
24.0
32.0 m.
0.519
0.925
1.32
1.71
2.09
2.64
3.41
4.19
Measured Apparent Resistivity Pseudosection
Ps.2
0.0
8.00
16.0
24.0
32.0 m.
0.519
0.925
1.32
1.71
2.09
2.64
3.41
4.19
Calculated Apparent Resistivity Pseudosection
Depth
Iteration 5 Abs. error = 4.2 %
0.0
8.00
16.0
24.0
32.0 m.
0.125
0.676
1.34
1.73
2.15
2.62
3.13
3.69
4.31
4.99
Inverse Model Resistivity Section
40.9
82.2
165
332
666
1338
2686
5394
Resistivity in ohm.m
Unit electrode spacing 0.500 m.