Eos491Assignment03

December 4, 2015

EOS 491/526 Assignment #3
Daniel Scanks V00788200
Question #1

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
%matplotlib inline
import numpy.linalg as ag

print('solving overdetermined problem from Assignment 2, Question 1:')

#sensitivity matrix
A = np.matrix([[1.,1.,1.,1.,1.],[1.,1.,1.,1.,0.5],[1.,1.,1.,0.5,0],[1.,1.,0.5,0,0],[1.,0.5,0,0,
A_T = A.transpose()

#true model
m_t = np.matrix([1,2,3,4,5])
m_t = m_t.transpose()



#true data
d_t = np.matrix([15.,12.5,8.,4.5,2.,0.5])
d = d_t.transpose()


#calculating m with SVD
U,s,V_T = ag.svd(A) # SVD decomposition of A into components
one1 = np.dot(V_T.T,ag.inv(np.diag(s)))
U = np.delete(U,np.s_[-1:],1) #Making U NxM
pseudoinv_svd = np.dot(one1,U.T)
m= np.dot(pseudoinv_svd,d)
print('model solution using svd psuedoinverse:')
print m
print('Vs.')
print('least-squared-error solution from assignment 2:')
print(np.matrix([[1],[2],[3],[4],[5]]))
print('The SVD pseudo-inverse method does produce the same result as the least-squared-error me

print()
print('solving under-determined problem from Assignment 2, Question 1:')
#sensitivity matrix
A = np.matrix([[1.,1.,1.],[1.,1.,0]])
```

```python
A_T = A.transpose()

#true model
mt = np.matrix([1.,2.,3.])
m_t = mt.transpose()

#true data
d_t = np.matrix([6.,3.])
d = d_t.transpose()

U,s,V_T = ag.svd(A) # SVD decomposition of A into components (s = lamda)
s = np.matrix([[2.13577921,0,],[0,0.66215345]]) #making s/Lamda into MxM diagonal matrix
invs = ag.inv(s)
invs = np.matrix([[ 0.46821319,0,0],[ 0, 1.51022395,0],[0,0,0]]) #Moore-Penrose
print('Moore-Penrose Inverse Lambda Matrix:')
print invs
one = np.dot(V_T.T,invs)
U = np.matrix([[-0.78820544,-0.61541221,0],[-0.61541221,0.78820544,0]]) #making U work for unde
pseudoinv_svd = np.dot(one,U.T)
m= np.dot(pseudoinv_svd,d)
print('model solution using svd psuedoinverse:')
print m
print('Vs.')
print('smallest model solution from assignment 2:')
print(np.matrix([[1.5],[1.5],[3]]))
print('The SVD pseudo-inverse method does produce the same result as the smallest method for an
```

```
solving overdetermined problem from Assignment 2, Question 1:
model solution using svd psuedoinverse:
[[ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]]
Vs.
least-squared-error solution from assignment 2:
[[1]
 [2]
 [3]
 [4]
 [5]]
The SVD pseudo-inverse method does produce the same result as the least-squared-error method for an over
()
solving under-determined problem from Assignment 2, Question 1:
Moore-Penrose Inverse Lambda Matrix:
[[ 0.46821319  0.          0.        ]
 [ 0.          1.51022395  0.        ]
 [ 0.          0.          0.        ]]
model solution using svd psuedoinverse:
[[ 1.5       ]
 [ 1.5       ]
 [ 2.99999998]]
Vs.
smallest model solution from assignment 2:
```

```
[[ 1.5]
 [ 1.5]
 [ 3. ]]
The SVD pseudo-inverse method does produce the same result as the smallest method for an underdetermine
```

Question 2

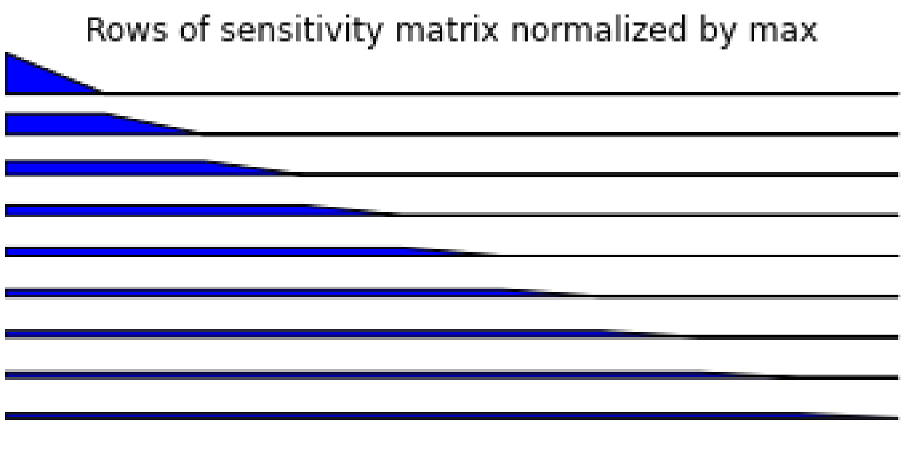
```
In [3]: print('Question 2')


        x = 500. #distance

        # reciever depth values (m)
        z = np.arange(10,110,10)


        #noise (s)
        n= np.random.normal(0,0.0001,10)
        n = np.matrix(n)
        noise = n.T
        #Cd matrix
        Cdinv = (1/(0.0001**2))*np.ones(10)
        Cdinv = np.diag(Cdinv)


        #velocities (m/s)
        v = np.matrix([1500.,1495.,1490.,1487.,1490.,1495.,1500.,1505.,1508.,1510.])
        #slowness vector for source at 0m - rays pass through all layers
        m0 = np.array([1/1500.,1/1495.,1/1490.,1/1487.,1/1490.,1/1495.,1/1500.,1/1505.,1/1508.,1/1510.]
        m0 = np.matrix(m0)
        m0 = m0.T
        #slowness vector for source at 90m- ray passes through 10th layer twice(surface and through)
        m90 = np.array([1/1495.,1/1490.,1/1487.,1/1490.,1/1495.,1/1500.,1/1505.,1/1508.,1/1510.,1/1510.
        m90 = np.matrix(m90)
        m90 = m90.T
        #slowness vector for source at 50m- ray passes through 6th layer twice(surface and through)
        m50 = np.array([1/1495.,1/1490.,1/1487.,1/1490.,1/1495.,1/1500.,1/1500.,1/1505.,1/1508.,1/1510.
        m50 = np.matrix(m50)
        m50 = m50.T

        #a square, even determined
        print('a) even determined')
        #d = Am + n
        #layer thicknesses
        t = 10

        # source at 0m
        sin0 = np.zeros(10)
        for i in range(0,10):
            sin0[i] = abs(z[i])/(np.sqrt(x**2+(z[i])**2))


        #Asquare
        A0 = np.zeros((10,10))
        for i in range(0,10):
```

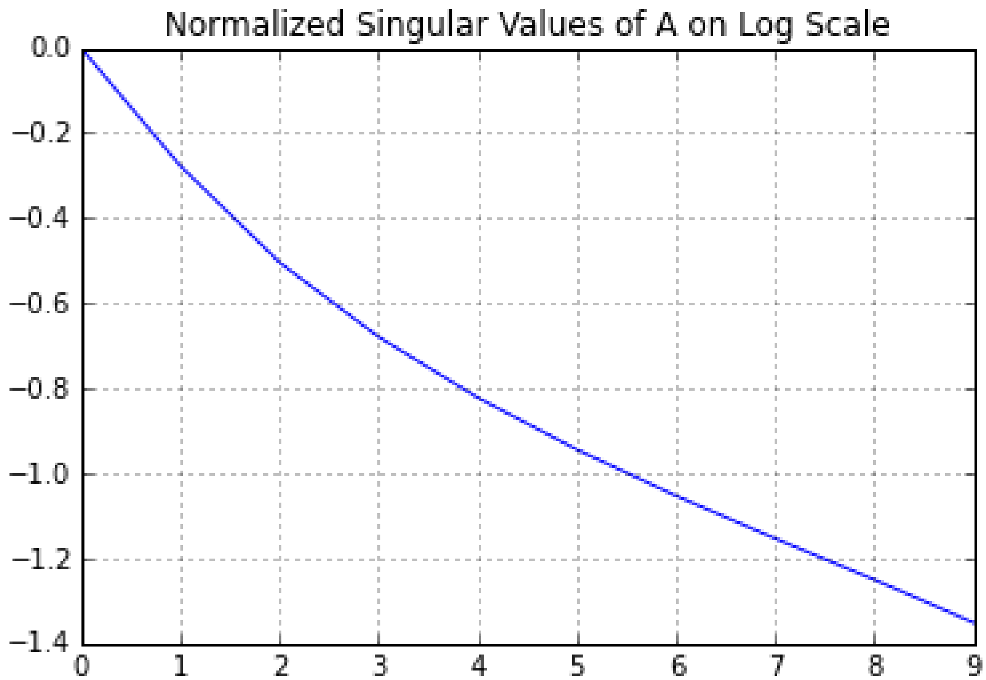
```
    n= 0
    while(n < i+1):
        A0[i,n] = t/sin0[i]
        n= n+1

#data
d0 = np.dot(A0,m0)
print('observed traveltime data for source at 0m')
d0_obs = d0 +noise
print d0_obs
#invert to find predicted model
A0inv = ag.inv(A0)
m0t = np.dot(A0inv,d0_obs)
print(' predicted slowness model for source at 0m')
print m0t


#i)
norm = np.max(abs(A0))
A = A0/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of sensitivity matrix normalized by max')
plt.axis('off')
plt.show()

#ii)
U, s, V = np.linalg.svd(A0, full_matrices=True)
snorm = s/707.18515799
plt.plot(np.log10(snorm))
plt.title('Normalized Singular Values of A on Log Scale')
plt.grid(True)
plt.show()

#iii/iv)
depth = np.array((10,20,30,40,50,60,70,80,90,100))

one = np.dot(A0.T,A0)
two = ag.inv(one)
cm = np.dot(0.0001**2,two)
vals = np.sqrt(np.diag(cm))
vsq = np.array((1500.**2,1495**2.,1490.**2,1487.**2,1490.**2,1495.**2,1500.**2,1505.**2,1508.**
#error values on velocities for each depth(layer)
unvel = np.zeros(10)
for i in range(0,10):
    unvel[i] = vals[i]*vsq[i]
x= np.squeeze(np.asarray(1/m0))
x1 = np.squeeze(np.asarray(1/m0t))
plt.figure(figsize=(6,3))
```

```python
plt.plot(x,depth, label = 'true model',color ='b')
plt.plot(x1,depth, label = 'recovered model',color ='k')
plt.title('Velocity Models as a Function of Depth With Standard Deviations(Green)')
plt.fill_betweenx(depth,x1-unvel,x1+unvel,facecolor ='#7EFF99',edgecolor='#3F7F4C')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Velocity (m/s)')
plt.legend()
plt.show()
plt.figure(figsize=(6,3))
plt.plot(unvel,depth)
plt.title('Standard Deviation as a Function of Depth')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Standard Deviation (m/s)')
plt.show()

#v)
#correltion matrix
Rc = np.zeros((10,10))
for i in range (0,10):
    for j in range(0,10):
        Rc[i,j] = cm[i,j]/(np.sqrt(cm[i,i]*cm[j,j]))


norm = np.max(abs(Rc))
A = Rc/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of Correlation matrix normalized by max')
plt.axis('off')
plt.show()

#vi)
#resolution matrix
Rm = np.zeros((10,10))
for i in range (0,10):
    Rm[i,i] = 1.

norm = np.max(abs(Rm))
A = Rm/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.axis('off')
```

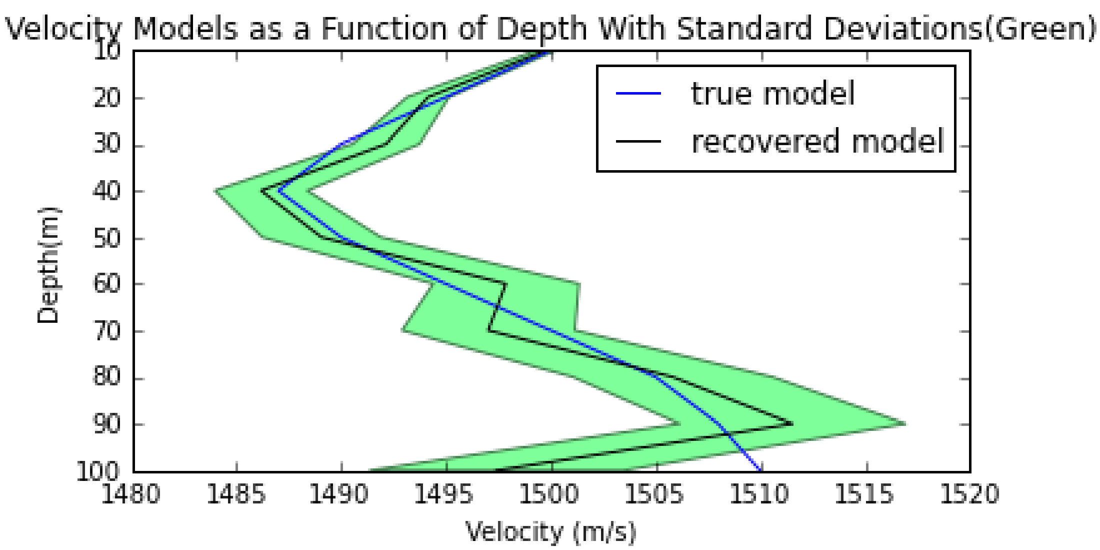
```
plt.title('Rows of Model Resolution Matrix normalized by max value' )
plt.show()
#vii)
print('For a source at 0m, each of the 10 layers will recieve from 1-10 rays through it. This mo
print('for each layer. The first layer recieves all 10 rays, while the tenth recieves only one.
print('deviation increases with depth as seen in the model comparison. b/c of this the recovere
print( 'increasing depth. The resolution matrix is equal to the identity matrix')
```

```
Question 2
a) even determined
observed traveltime data for source at 0m
[[ 0.33344808]
 [ 0.33427644]
 [ 0.33496996]
 [ 0.33595485]
 [ 0.33673544]
 [ 0.33725702]
 [ 0.33799631]
 [ 0.33864947]
 [ 0.33936283]
 [ 0.34060099]]
 predicted slowness model for source at 0m
[[ 0.00066676]
 [ 0.00066927]
 [ 0.00067017]
 [ 0.00067287]
 [ 0.00067156]
 [ 0.00066761]
 [ 0.00066799]
 [ 0.00066409]
 [ 0.00066157]
 [ 0.00066782]]
```

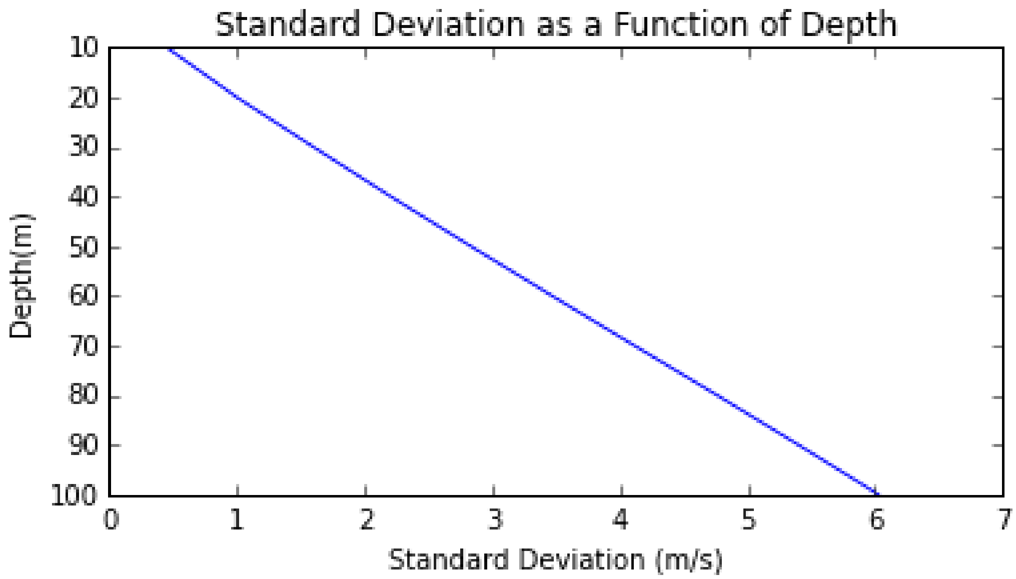


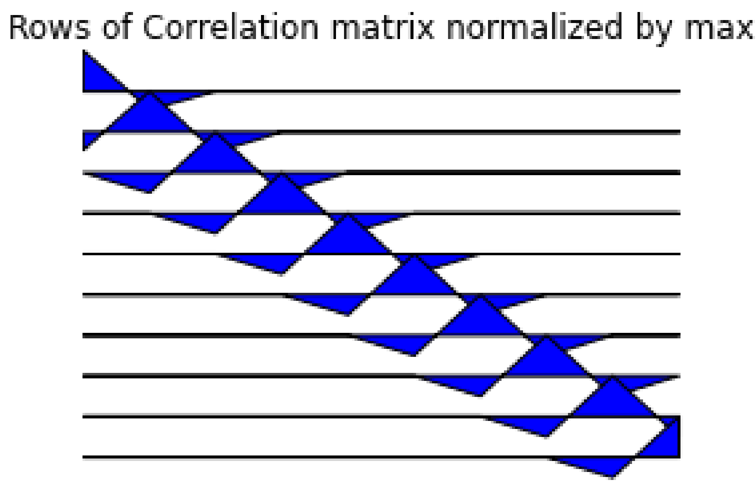


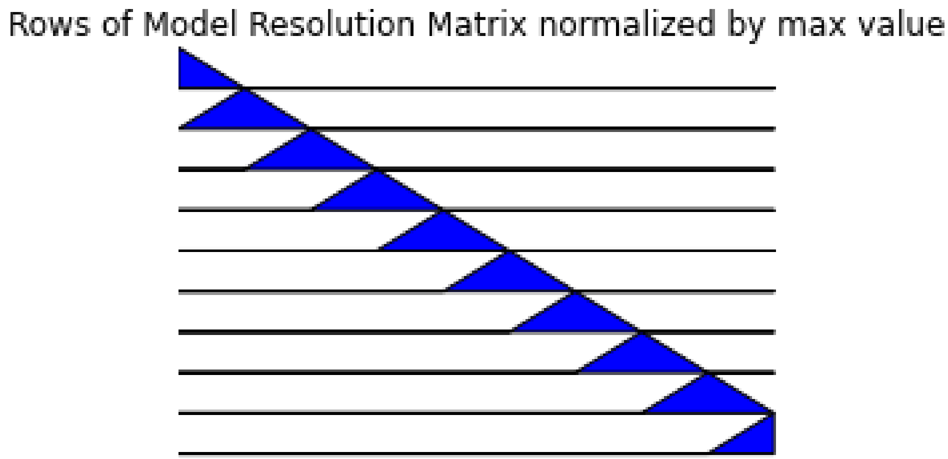
Normalized Singular Values of A on Log Scale
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
0 1 2 3 4 5 6 7 8 9



Velocity Models as a Function of Depth With Standard Deviations(Green)
true model
recovered model
Depth(m)
10 20 30 40 50 60 70 80 90 100
1480 1485 1490 1495 1500 1505 1510 1515 1520
Velocity (m/s)


Standard Deviation as a Function of Depth
Depth(m)
10
20
30
40
50
60
70
80
90
100
0
1
2
3
4
5
6
7
Standard Deviation (m/s)


Rows of Correlation matrix normalized by max



For a source at 0m, each of the 10 layers will recieve from 1-10 rays through it. This means the there w
for each layer. The first layer recieves all 10 rays, while the tenth recieves only one. As a result the
deviation increases with depth as seen in the model comparison. b/c of this the recovered model is less
increasing depth. The resolution matrix is equal to the identity matrix

```
In [4]: #------------------------------------------------------------part b
        z = np.arange(10,110,10)
        x= 500.
        print('b) over-determined')
        # source at 50m
        sin50 = np.zeros(10)
        for i in range(0,10):
            sin50[i] = abs(z[i]-50)/(np.sqrt(x**2+(z[i]-50)**2))

        A50 = np.zeros((10,10))
        for i in range(0,10):
            if i == 4:
                A50[i,5] = x
            elif i < 4:
                n=4
                while(n > i):
                    A50[i,n] = t/sin50[i]
                    n = n-1
            else:
                n= 5
                while(n < i+1):
                    A50[i,n] = t/sin50[i]
                    n = n+1


        # source at 90m
        sin90 = np.zeros(10)
        for i in range(0,10):
```

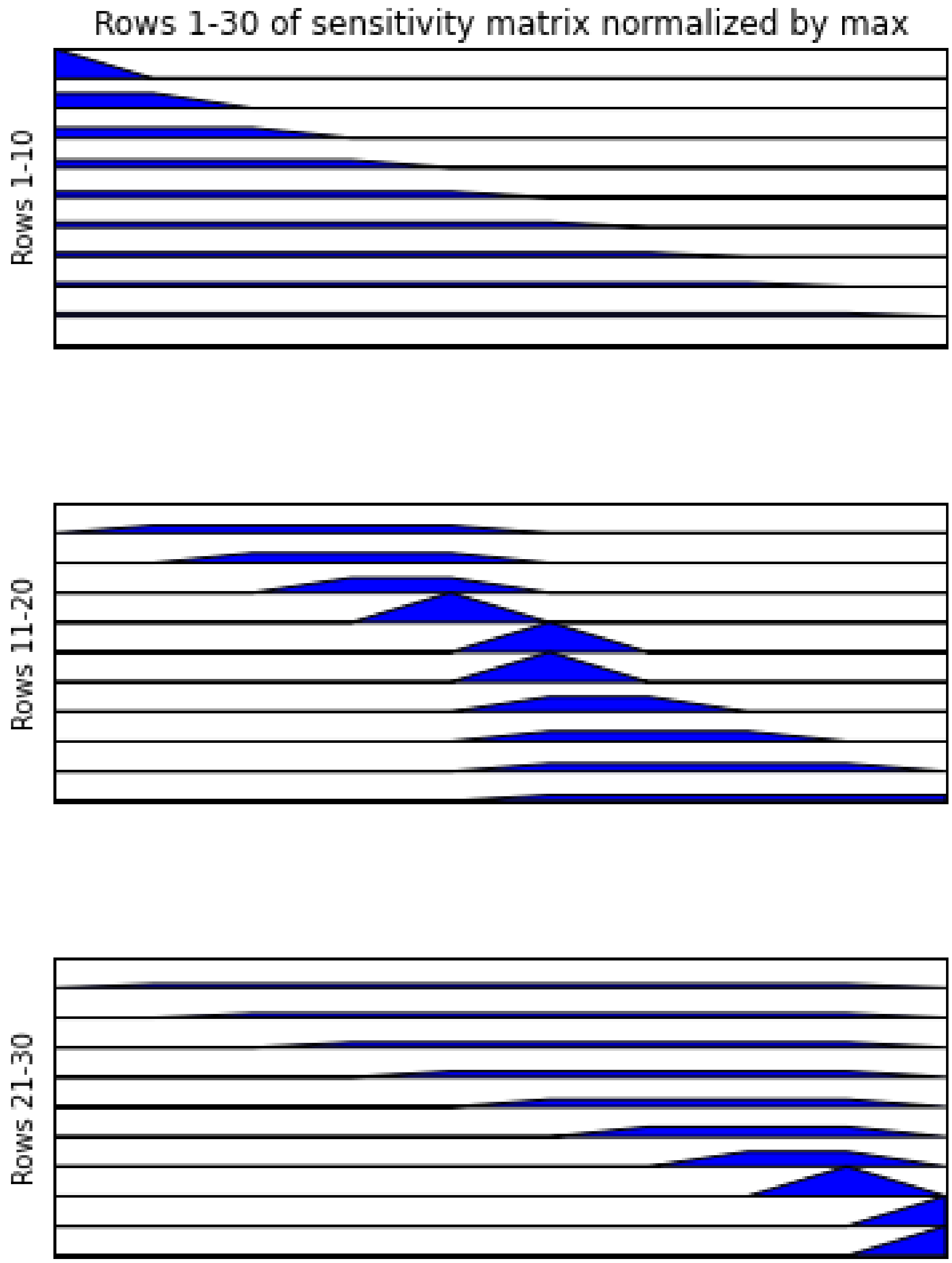
```
    sin90[i] = abs(z[i]-90)/(np.sqrt(x**2+(z[i]-90)**2))

A90 = np.zeros((10,10))
for i in range(0,10):
    if i == 8:
        A90[i,9] = x
    elif i == 9:
        A90[i,9] = np.abs(z[9]-z[8])/sin90[9]
    else:
        n= 8
        while(n > i):
            A90[i,n] = t/sin90[i]
            n = n-1
#noise
n= np.random.normal(0,0.0001,30)
n = np.matrix(n)
noise = n.T

#new A
Ab = np.concatenate((A0, A50, A90), axis=0)
#data
db = np.dot(Ab,m0)
db_obs = db +noise
#invert to find predicted model
one = np.dot(Ab.T,Ab)
invone = ag.inv(one)
two = np.dot(invone,Ab.T)
mb = np.dot(two,db_obs)
print(' predicted slowness model for sources at 0,50,90m')
print mb

#i)
norm = np.max(abs(A0))
A = A0/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(1,A.shape[1]+1)
plt.title('Rows 1-30 of sensitivity matrix normalized by max')
plt.ylabel('Rows 1-10')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
norm = np.max(abs(A50))
A = A50/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
```

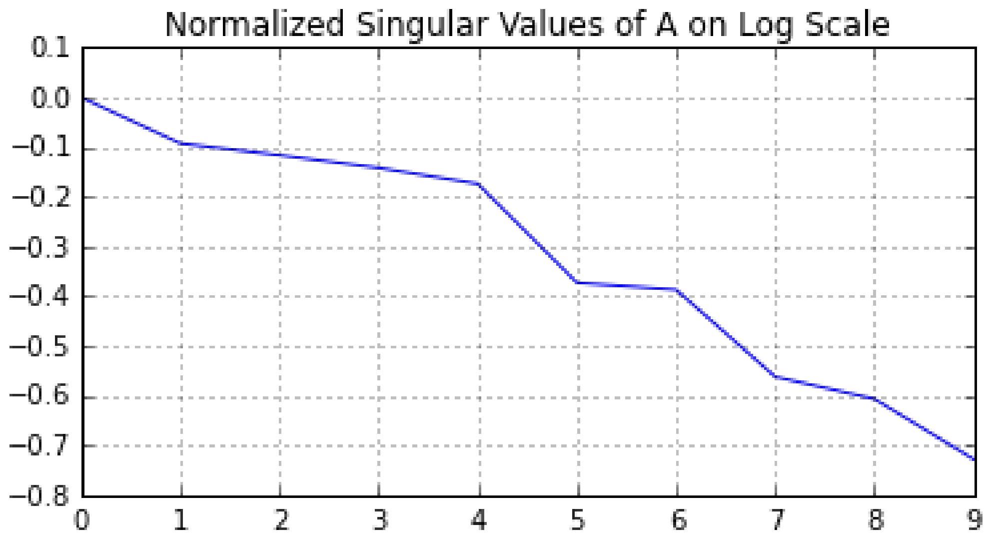
```python
plt.ylim(1,A.shape[1]+1)
plt.ylabel('Rows 11-20')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
norm = np.max(abs(A90))
A = A90/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(1,A.shape[1]+1)
plt.ylabel('Rows 21-30')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
#ii)
U, s, V = np.linalg.svd(Ab, full_matrices=True)
snorm = s/np.max(s)
plt.figure(figsize=(6,3))
plt.plot(np.log10(snorm))
plt.title('Normalized Singular Values of A on Log Scale')
plt.grid(True)
plt.show()
#iii/iv)
depth = np.array((10,20,30,40,50,60,70,80,90,100))
one = np.dot(Ab.T,Ab)
two = ag.inv(one)
cm = np.dot(0.0001**2,two)
vals = np.sqrt(np.diag(cm))
vsq = np.array((1500.**2,1495**2.,1490.**2,1487.**2,1490.**2,1495.**2,1500.**2,1505.**2,1508.**
#error values on velocities for each depth(layer)
unvel = np.zeros(10)
for i in range(0,10):
    unvel[i] = vals[i]*vsq[i]
x= np.squeeze(np.asarray(1/m0))
x1 = np.squeeze(np.asarray(1/mb))
plt.figure(figsize=(6,3))
plt.plot(x,depth, label = 'true model',color ='b')
plt.plot(x1,depth, label = 'recovered model',color ='k')
plt.title('Velocity Models as a Function of Depth With Standard Deviations(Green)')
plt.fill_betweenx(depth,x1-unvel,x1+unvel,facecolor ='#7EFF99',edgecolor='#3F7F4C')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Velocity (m/s)')
plt.legend()
plt.show()
plt.figure(figsize=(6,3))
plt.plot(unvel,depth)
plt.title('Standard Deviation as a Function of Depth')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
```

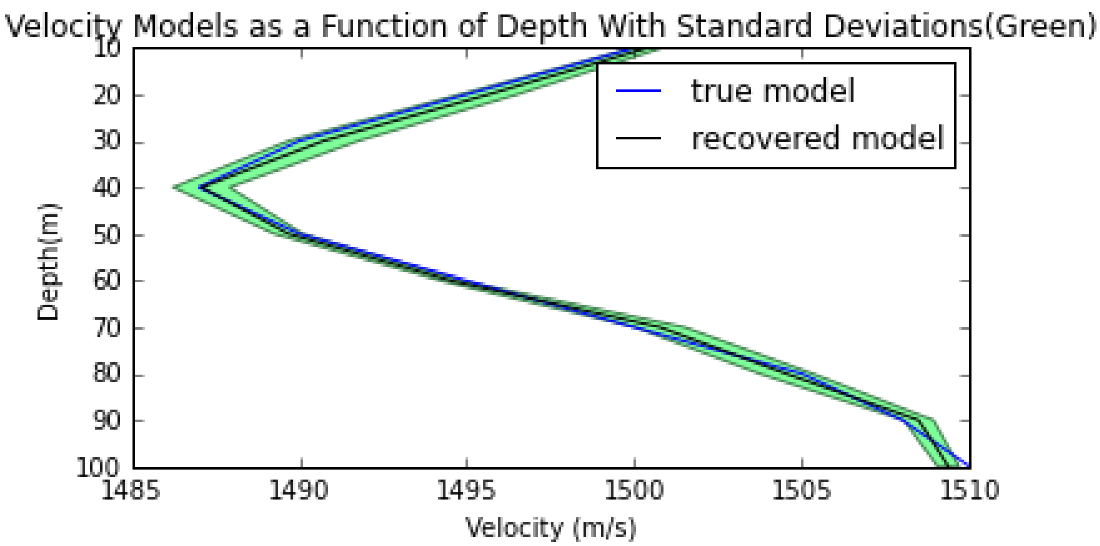
```
plt.xlabel('Standard Deviation (m/s)')
plt.show()
#v)
#correltion matrix
Rc = np.zeros((10,10))
for i in range (0,10):
    for j in range(0,10):
        Rc[i,j] = cm[i,j]/(np.sqrt(cm[i,i]*cm[j,j]))


norm = np.max(abs(Rc))
A = Rc/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of Correlation matrix normalized by max')
plt.axis('off')
plt.show()


#vi)
#resolution matrix
Rm = np.zeros((10,10))
for i in range (0,10):
    Rm[i,i] = 1.

norm = np.max(abs(Rm))
A = Rm/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of Model Resolution matrix normalized by max')
plt.axis('off')
plt.show()
#vii)
print('for this overdetermined case, we see that all layers except the first see multiple rays,
print('standard deviation is much lower as the recovered model is more accurate. The resolution
print('identity matrix for this overdetermined case.')
```

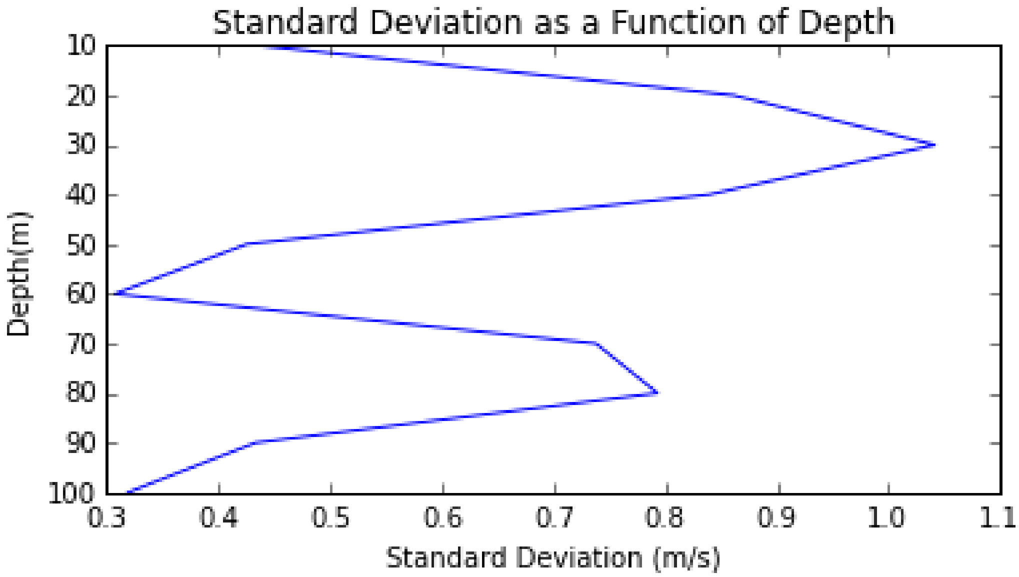
```
b) over-determined
 predicted slowness model for sources at 0,50,90m
[[ 0.00066649]
 [ 0.0006686 ]
 [ 0.00067081]
 [ 0.00067246]
 [ 0.00067129]
```

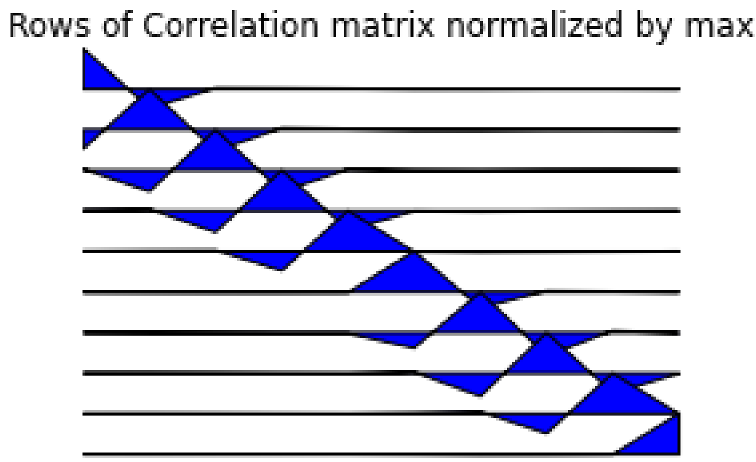
```
[ 0.0006691 ]
[ 0.00066633]
[ 0.00066466]
[ 0.00066292]
[ 0.00066253]]
```

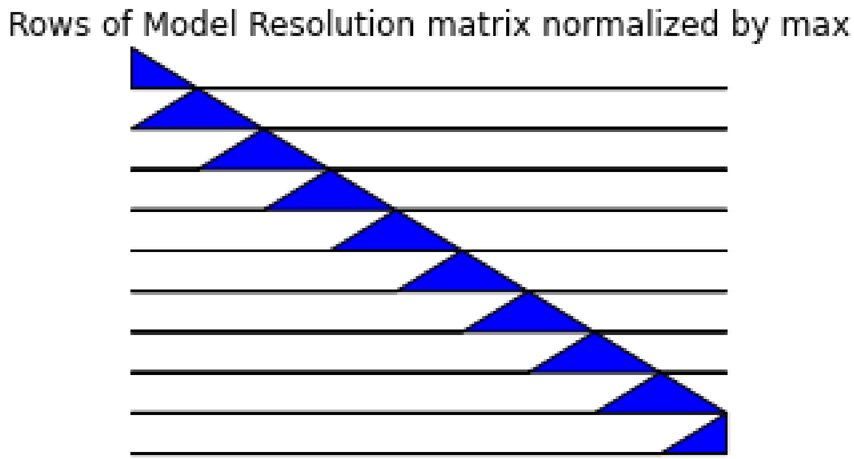



Normalized Singular Values of A on Log Scale
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
0
1
2
3
4
5
6
7
8
9


Velocity Models as a Function of Depth With Standard Deviations(Green)
true model
recovered model
Depth(m)
10
20
30
40
50
60
70
80
90
100
1485
1490
1495
1500
1505
1510
Velocity (m/s)


Standard Deviation as a Function of Depth
Depth(m)
10
20
30
40
50
60
70
80
90
100
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
Standard Deviation (m/s)


Rows of Correlation matrix normalized by max



for this overdetermined case, we see that all layers except the first see multiple rays, and therefore t
standard deviation is much lower as the recovered model is more accurate. The resolution matrix is equal
identity matrix for this overdetermined case.

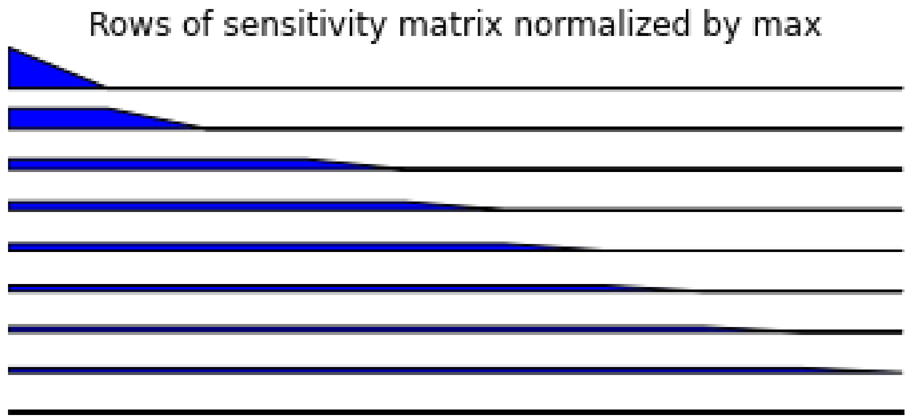
```
In [6]: #------------------------------------------------------------------partc
        print('c)')
        zmiss = np.array((10,20,40,50,60,70,80,90,100))
        x = 500.
        #new matrix (A0 minus row 3)
        Ac = np.delete(A0, (2), axis=0)
        #noise
        n= np.random.normal(0,0.0001,9)
        n = np.matrix(n)
        noise = n.T
        #data
        dc = np.dot(Ac,m0)
        dc_obs = dc +noise
        #invert to find predicted model
        one = np.dot(Ac,Ac.T)
        oneinv = ag.inv(one)
        two = np.dot(Ac.T,oneinv)
        mc = np.dot(two,dc_obs)
        print(' predicted slowness model for source at 0m with no 30m receiver')
        print mc

        #i)
        norm = np.max(abs(Ac))
        A = Ac/(norm)
        T = range(A.shape[1])
        plt.figure(figsize=(6,3))
        for i in range(A.shape[0]):
            base = A[i,]*0.
            base.fill(A.shape[1]-i)
```

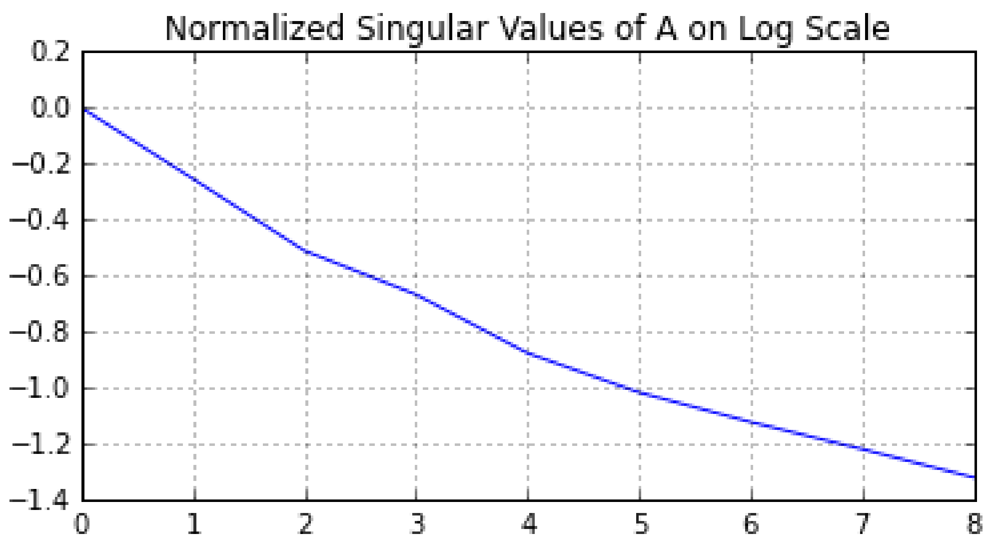
```
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of sensitivity matrix normalized by max')
plt.axis('off')
plt.show()
#ii)
U, s, V = np.linalg.svd(Ac, full_matrices=True)
snorm = s/np.max(s)
plt.figure(figsize=(6,3))
plt.plot(np.log10(snorm))
plt.title('Normalized Singular Values of A on Log Scale')
plt.grid(True)
plt.show()
#iii/iv)
depth = np.array((10,20,30,40,50,60,70,80,90,100))
one = np.dot(Ac,Ac.T)
oneinv = ag.inv(one)
two = np.dot(0.0001**2,Ac.T)
three = np.dot(oneinv,oneinv)
four = np.dot(two,three)
cm = np.dot(four,Ac)
vals = np.sqrt(np.diag(cm))
vsq = np.array((1500.**2,1495**2.,1490.**2,1487.**2,1490.**2,1495.**2,1500.**2,1505.**2,1508.**
#error values on velocities for each depth(layer)

unvel = np.zeros(10)
for i in range(0,10):
    unvel[i] = vals[i]*vsq[i]
x= np.squeeze(np.asarray(1/m0))
x1 = np.squeeze(np.asarray(1/mc))
plt.figure(figsize=(6,3))
plt.plot(x,depth, label = 'true model',color ='b')
plt.plot(x1,depth, label = 'recovered model',color ='k')
plt.title('Velocity Models as a Function of Depth With Standard Deviations(Green)')
plt.fill_betweenx(depth,x1-unvel,x1+unvel,facecolor ='#7EFF99',edgecolor='#3F7F4C')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Velocity (m/s)')
plt.legend()
plt.show()
plt.figure(figsize=(6,3))
plt.plot(unvel,depth)
plt.title('Standard Deviation as a Function of Depth')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Standard Deviation (m/s)')
plt.show()
#v)
#correltion matrix
Rc = np.zeros((10,10))
for i in range (0,10):
    for j in range(0,10):
        Rc[i,j] = cm[i,j]/(np.sqrt(cm[i,i]*cm[j,j]))
```

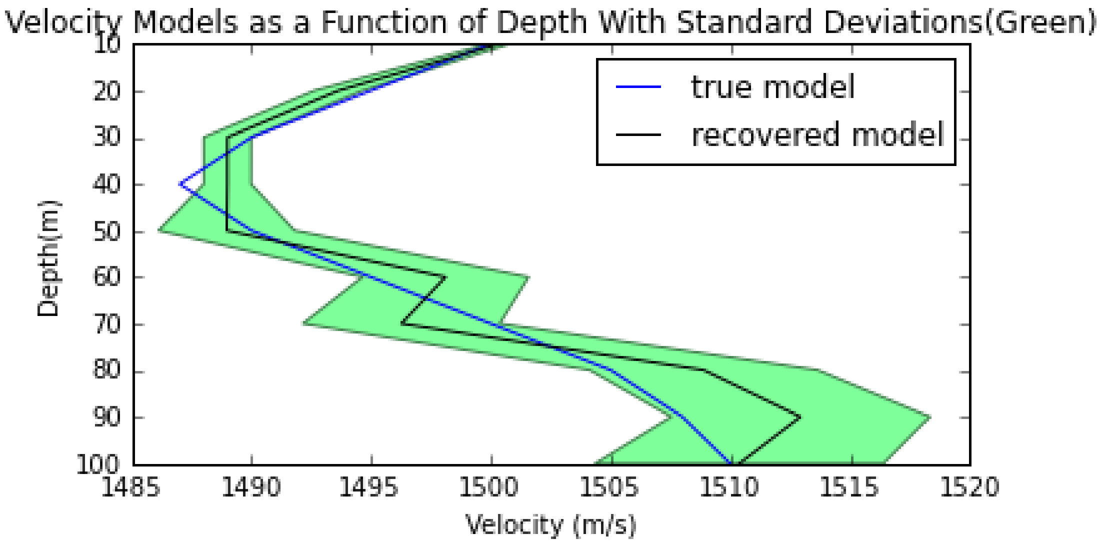
```python
norm = np.max(abs(Rc))
A = Rc/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of Correlation matrix normalized by max')
plt.axis('off')
plt.show()
#vi)
#resolution matrix
one1 = np.dot(Ac,Ac.T)
one1inv = ag.inv(one1)
Ag = np.dot(Ac.T,one1inv)
Rm = np.dot(Ag,Ac)

norm = np.max(abs(Rm))
A = Rm/(norm)
T = range(A.shape[1])
plt.figure(figsize=(4,3))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(0,A.shape[1]+1)
plt.title('Rows of Model Resolution matrix normalized by max')
plt.axis('off')
plt.show()
#vii)
print('The missing reciever at 30m makes the the model resolution and correlation matrices skew
print('The correlation rows above and below the 3rd are somewhat skewed too. The standard devia
print('the standard deviation for a). The deviation is pretty constained and increases with dep
```

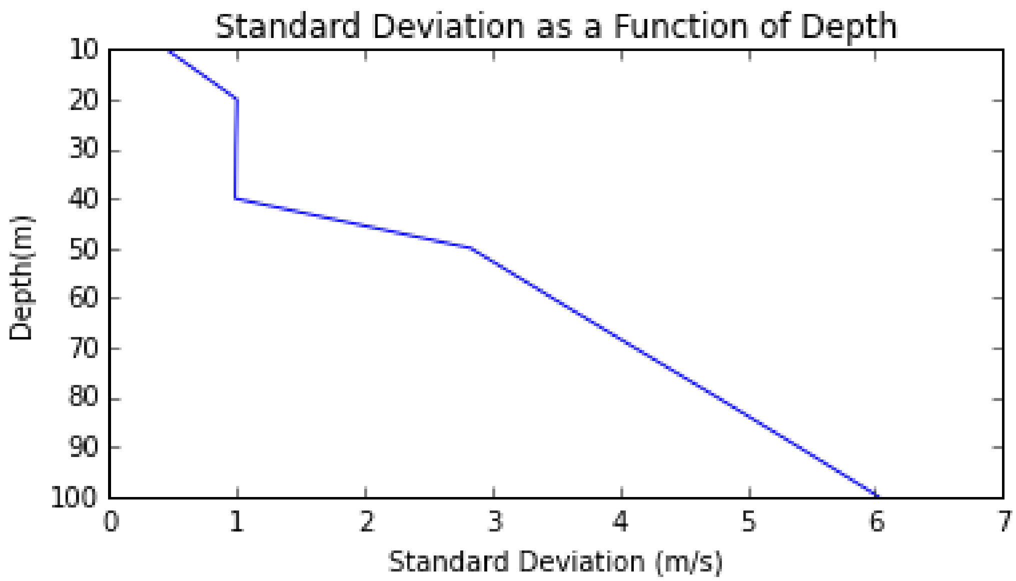
c)

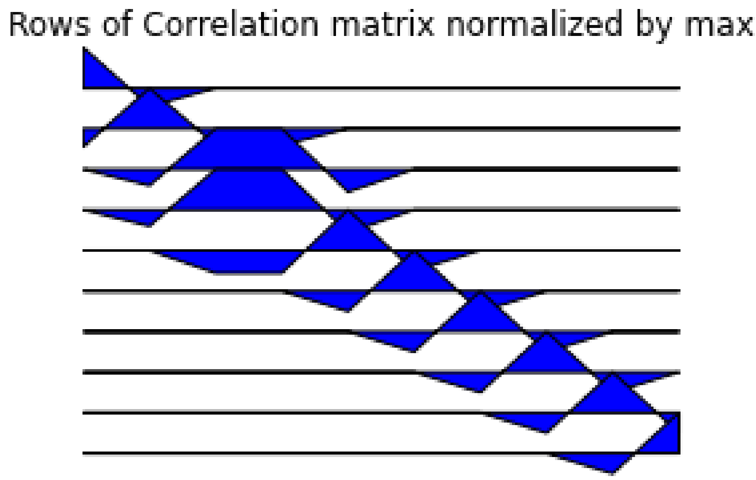
```
 predicted slowness model for source at 0m with no 30m receiver
[[ 0.00066652]
 [ 0.0006695 ]
 [ 0.00067159]
 [ 0.00067159]
 [ 0.00067161]
 [ 0.0006675 ]
 [ 0.00066833]
 [ 0.00066274]
 [ 0.00066097]
 [ 0.00066209]]
```

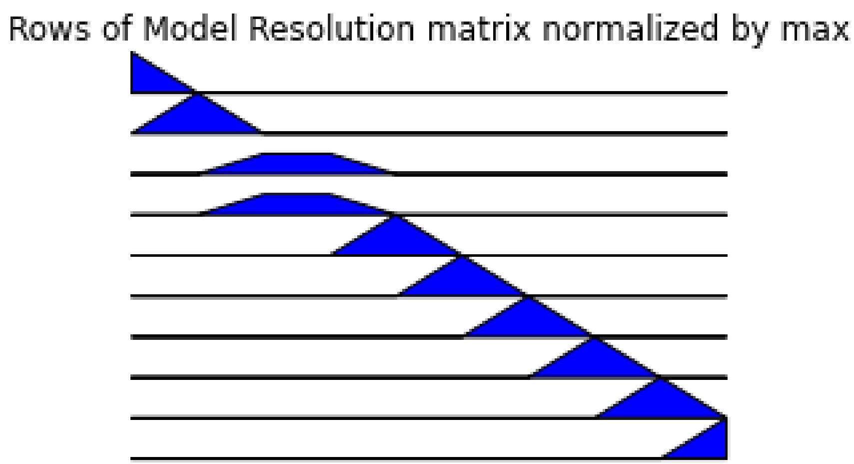

Rows of sensitivity matrix normalized by max


Normalized Singular Values of A on Log Scale
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
0
1
2
3
4
5
6
7
8


Velocity Models as a Function of Depth With Standard Deviations(Green)
true model
recovered model
Depth(m)
Velocity (m/s)
10
20
30
40
50
60
70
80
90
100
1485
1490
1495
1500
1505
1510
1515
1520


Standard Deviation as a Function of Depth
Depth(m)
Standard Deviation (m/s)
10
20
30
40
50
60
70
80
90
100
0
1
2
3
4
5
6
7






```
The missing reciever at 30m makes the the model resolution and correlation matrices skewed on the plots
The correlation rows above and below the 3rd are somewhat skewed too. The standard deviation is similar
the standard deviation for a). The deviation is pretty constained and increases with depth for the most
```

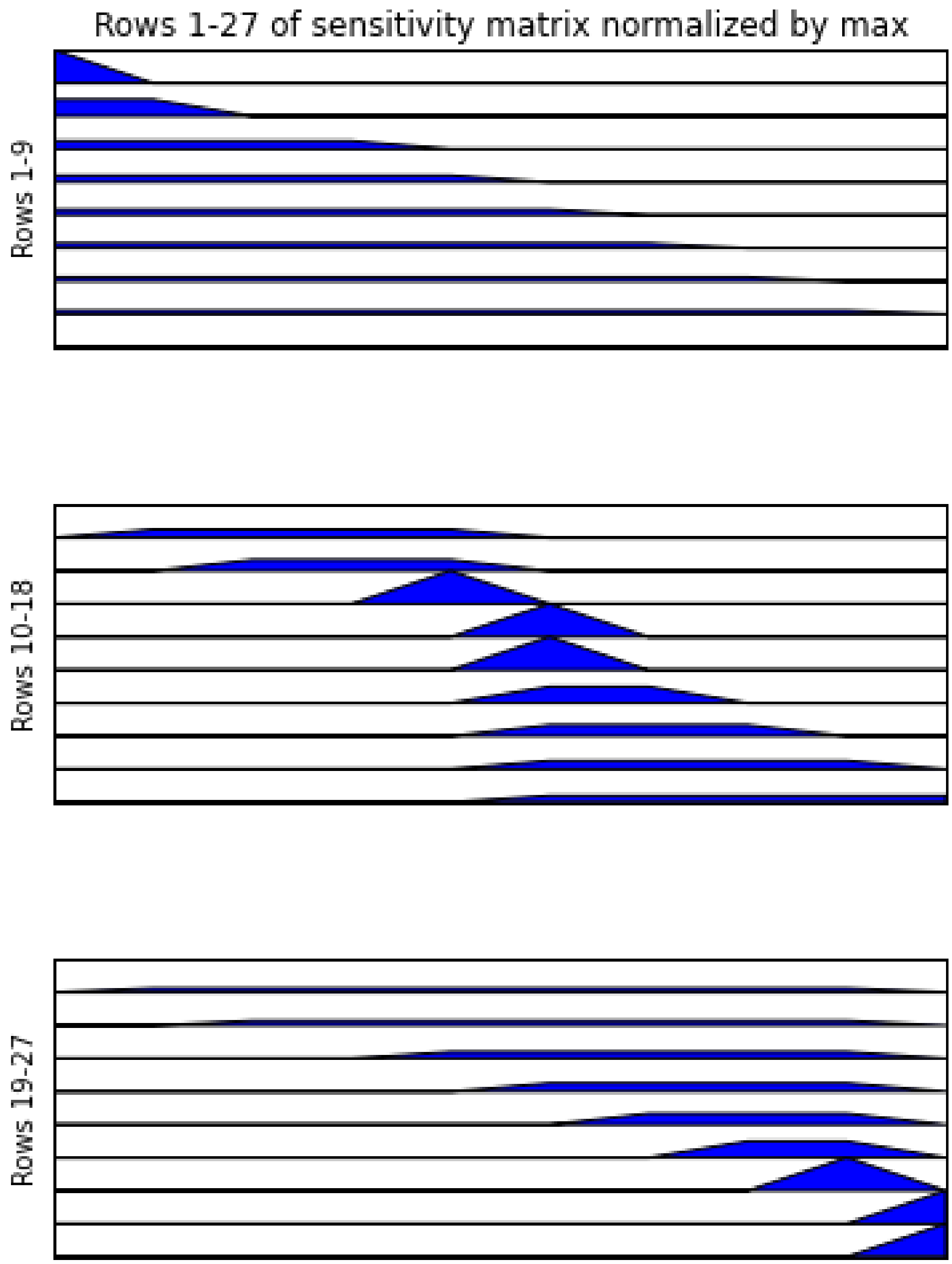
```
In [7]: from scipy import linalg
        print('d) mixed determined')
        A0n = np.delete(A0, (2), axis=0)
        A50n = np.delete(A50, (2), axis=0)
        A90n = np.delete(A90, (2), axis=0)
        #noise
        n= np.random.normal(0,0.0001,27)
        n = np.matrix(n)
```

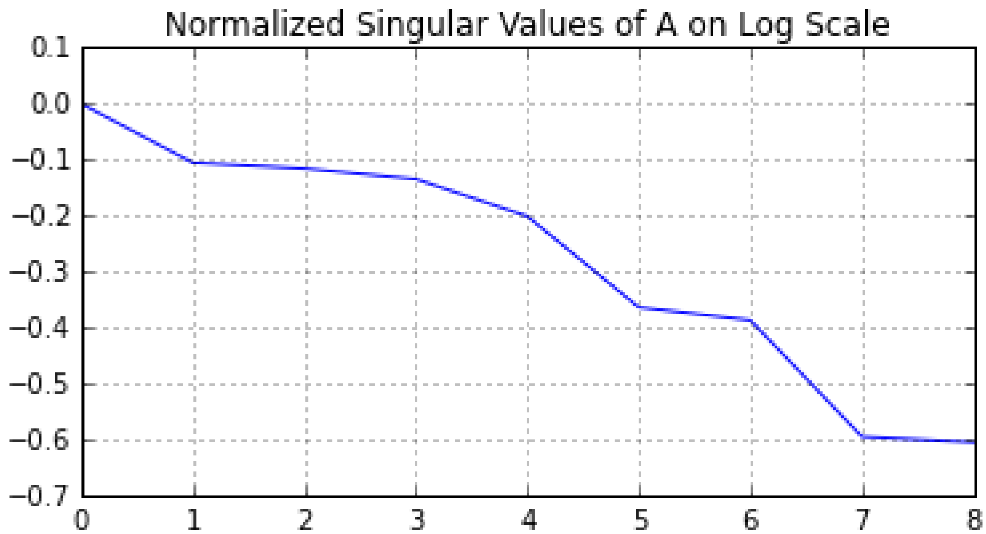
```
noise = n.T
#new A
Ad = np.concatenate((A0n, A50n, A90n), axis=0)
#data
dd = np.dot(Ad,m0)
dd_obs = dd +noise
#invert to find predicted model
N,M = Ad.shape
U,s,V_T = ag.svd(Ad) # SVD decomposition of A into components (n = lamda)
s = linalg.diagsvd(s,M,M)
s = np.delete(s,(9),1)
s = np.delete(s,(9),0)
V= V_T.T
V = np.delete(V,(9),1)
Unew = np.delete(U, np.s_[9:27:], axis = 1)
s = ag.inv(s)
one = np.dot(V,s)
pseudoinv_svd = np.dot(one,Unew.T)
md= np.dot(pseudoinv_svd,dd_obs)
print(' predicted slowness model for sources at 0,50,90m without 30m reciever')
#md= np.array(md)
print md
#i
norm = np.max(abs(A0n))
A = A0n/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(2,A.shape[1]+1)
plt.title('Rows 1-27 of sensitivity matrix normalized by max')
plt.ylabel('Rows 1-9')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
norm = np.max(abs(A50n))
A = A50n/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(2,A.shape[1]+1)
plt.ylabel('Rows 10-18')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
norm = np.max(abs(A90n))
A = A90n/(norm)
T = range(A.shape[1])
plt.figure(figsize=(6,2))
```

```
for i in range(A.shape[0]):
    base = A[i,]*0.
    base.fill(A.shape[1]-i)
    plt.fill_between(T, base, A[i,]+base)
plt.ylim(2,A.shape[1]+1)
plt.ylabel('Rows 19-27')
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.show()
#ii)
s1 = ag.inv(s)
s1 = np.diag(s1)
snorm = s1/np.max(s1)
plt.figure(figsize=(6,3))
plt.plot(np.log10(snorm))
plt.title('Normalized Singular Values of A on Log Scale')
plt.grid(True)
plt.show()
#iii/iv)
depth = np.array((10,20,30,40,50,60,70,80,90,100))
one = np.dot(s,s)
two = np.dot(V,one)
cm = np.dot(two,V.T)
vals = np.diag(cm)
vsq = np.array((1500.**2,1495**2.,1490.**2,1487.**2,1490.**2,1495.**2,1500.**2,1505.**2,1508.**
#error values on velocities for each depth(layer)
unvel = np.zeros(10)
for i in range(0,10):
    unvel[i] = vals[i]*vsq[i]
print unvel
x= np.squeeze(np.asarray(1/m0))
x1 = np.squeeze(np.asarray(1/md))
plt.figure(figsize=(6,3))
plt.plot(x,depth, label = 'true model',color ='b')
plt.plot(x1,depth, label = 'recovered model',color ='k')
plt.title('Velocity Models as a Function of Depth With Standard Deviations(Green)')
plt.fill_betweenx(depth,x1-unvel,x1+unvel,facecolor ='#7EFF99',edgecolor='#3F7F4C')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Velocity (m/s)')
plt.legend()
plt.show()
plt.figure(figsize=(6,3))
plt.plot(unvel,depth)
plt.title('Standard Deviation as a Function of Depth')
plt.gca().invert_yaxis()
plt.ylabel('Depth(m)')
plt.xlabel('Standard Deviation (m/s)')
plt.show()

#v)
#correltion matrix
Rc = np.zeros((10,10))
for i in range (0,10):
```

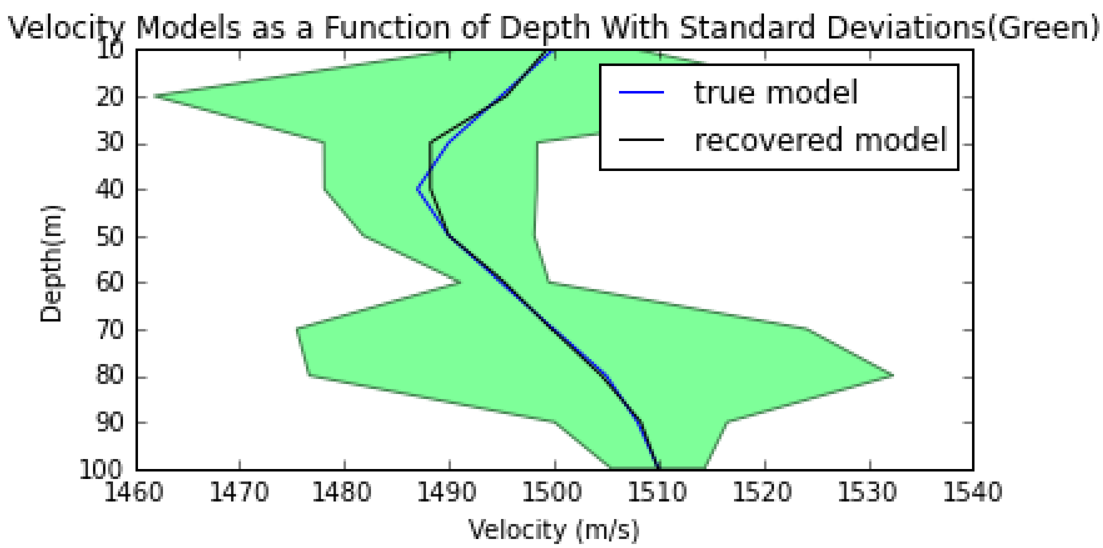
```
        for j in range(0,10):
            Rc[i,j] = cm[i,j]/(np.sqrt(cm[i,i]*cm[j,j]))


    norm = np.max(abs(Rc))
    A = Rc/(norm)
    T = range(A.shape[1])
    plt.figure(figsize=(4,3))
    for i in range(A.shape[0]):
        base = A[i,]*0.
        base.fill(A.shape[1]-i)
        plt.fill_between(T, base, A[i,]+base)
    plt.ylim(0,A.shape[1]+1)
    plt.title('Rows of Correlation matrix normalized by max')
    plt.axis('off')
    plt.show()

    #vi)
    #resolution matrix)
    Rm = np.dot(V,V.T)

    norm = np.max(abs(Rm))
    A = Rm/(norm)
    T = range(A.shape[1])
    plt.figure(figsize=(4,3))
    for i in range(A.shape[0]):
        base = A[i,]*0.
        base.fill(A.shape[1]-i)
        plt.fill_between(T, base, A[i,]+base)
    plt.ylim(0,A.shape[1]+1)
    plt.title('Rows of Model Resolution matrix normalized by max')
    plt.axis('off')
    plt.show()

    #vii)
    print('the SVD pseudo inverse did a good job of reconstructing the true model. There was 9 singu
    print(' the model correlation and resolution matrices resembled those in part c). The standard d
    print('distance from the source, at 0m,50m,90m the deviation was smallest.')
```

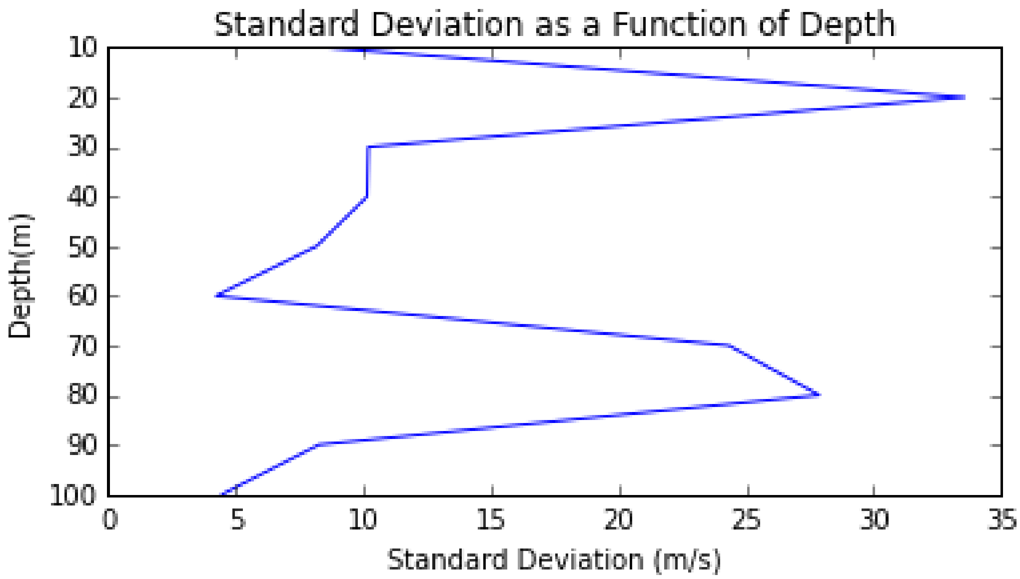
```
d) mixed determined
 predicted slowness model for sources at 0,50,90m without 30m reciever
[[ 0.00066695]
 [ 0.00066868]
 [ 0.00067193]
 [ 0.00067193]
 [ 0.00067114]
 [ 0.00066874]
 [ 0.00066675]
 [ 0.00066465]
 [ 0.00066299]
 [ 0.00066227]]
```

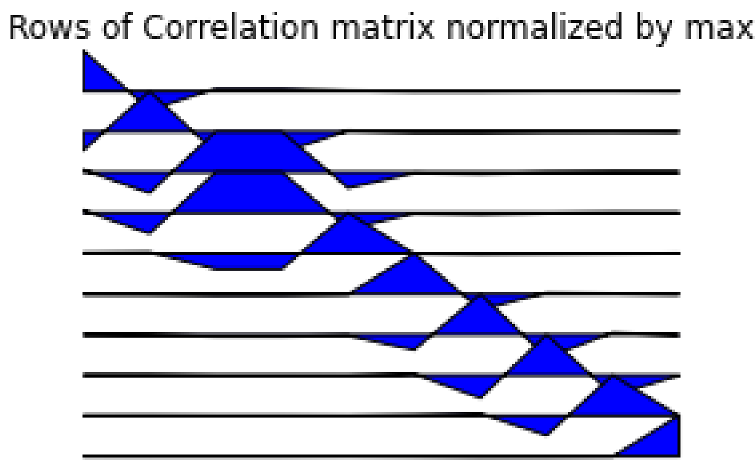

Rows 1-27 of sensitivity matrix normalized by max
Rows 1-9
Rows 10-18
Rows 19-27

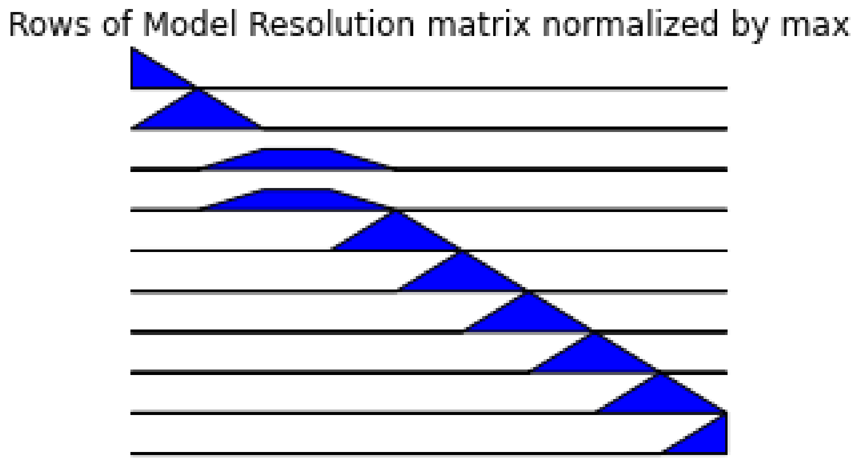



```
[  8.47764225  33.56998242  10.19009931  10.14910666   8.16627981
   4.22770806  24.33654021  27.87467854   8.25470384   4.45561484]
```




Standard Deviation as a Function of Depth
Depth(m)
Standard Deviation (m/s)
10
20
30
40
50
60
70
80
90
100
0
5
10
15
20
25
30
35


Rows of Correlation matrix normalized by max




```
the SVD pseudo inverse did a good job of reconstructing the true model. There was 9 singular values for
 the model correlation and resolution matrices resembled those in part c). The standard deviation varie
distance from the source, at 0m,50m,90m the deviation was smallest.
```

In [ ]: