Computational Methods

Final Report: Numerical Solution of the Eikonal Equation via the Runge–Kutta Method

Haozheng JI[1] Jianfeng Zhang[1]

[1]Department of Earth and Space Science, Southern University of Science and Technology, Shenzhen, Chian
jihz2023@mail.sustech.edu.cn

Shenzhen 2026.1.1

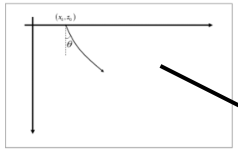
Content

Introduction

Final Project: 2D

Solution of 2D eikonal equation with initial values

$$
\begin{cases}
\frac{dx}{ds} = v(x,z)\,p_x, & \frac{dp_x}{ds} = \frac{\partial}{\partial x}\left(\frac{1}{v(x,z)}\right) \\
\frac{dz}{ds} = v(x,z)\,p_z, & \frac{dp_z}{ds} = \frac{\partial}{\partial z}\left(\frac{1}{v(x,z)}\right) \\
\frac{dT}{ds} = \frac{1}{v}
\end{cases}
$$



Initial values: x_0, z_0, p_{x0}, p_{z0}

First-Order System with x, z, p_x, p_z

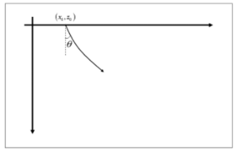
$$T[l] = \int_{(A)}^{(B)} \frac{ds}{C}.$$

RK4

➢ Traveltime and ray tracing： 3D

Solution of 3D eikonal equation with initial values

$$
\begin{cases}
\frac{dx}{ds} = v(x,y,z)\,p_x, & \frac{dp_x}{ds} = \frac{\partial}{\partial x}\left(\frac{1}{v(x,y,z)}\right) \\
\frac{dy}{ds} = v(x,y,z)\,p_y, & \frac{dp_y}{ds} = \frac{\partial}{\partial y}\left(\frac{1}{v(x,y,z)}\right) \\
\frac{dz}{ds} = v(x,y,z)\,p_z, & \frac{dp_z}{ds} = \frac{\partial}{\partial z}\left(\frac{1}{v(x,y,z)}\right) \\
\frac{dT}{ds} = \frac{1}{v}
\end{cases}
$$



Initial values: $\frac{l}{c}, \frac{m}{c}, \frac{n}{c}$ x_0, y_0, z_0

First-Order System with $x, y, z, p_x, p_y, p_z, T$

$l = \cos\theta,$
$m = \sin\theta\sin\varphi,$
$n = \sin\theta\cos\varphi$

$$T = \int_{x_0,y_0,z_0}^{x,y,z} \frac{1}{v(x,y,z)}\,ds$$

Copy from ESS207-project.pdf (31,35)

- __pycache__
- images — Appendix
- differenceregion.png
- DrawRay.py — Function (Draw pictures)
- Eikonal2d.py
- Eikonal3d.py — Class and function (Numerical solution)
- Final Project Report.ipynb — Main program
- 12311405纪浩正Final_Project_Report.pdf — Report
- 12311405纪浩正PPT.pdf — PPT

Introduction

Ensure the following files are located in the same directory :

- **Eikonal2d.py**
- **Eikonal3d.py**

Requirement

- **Python 3.10**
- **NumPy**
- **Matplotlib**

Initialize the model

```
M = Eikonal3d(v, ds=1, max_step=500000)
```

Compute a single ray path

```
rayx, rayy, rayz, rayt = M.Raypath(x0, y0, z0, toa, multi=0)
```

- **v(x, y, z)** : 3D velocity model
- **ds** : integration step size
- **max_step** : maximum number of integration steps
- **x0, y0, z0** : source location
- **toa** : take-off angle (inclination, radians)
- **theta** : azimuth angle (radians)
- **multi** : enable surface multiple reflections (1 = on, 0 = off)

DrawRay.py — Function (Draw pictures)

Eikonal2d.py

Class and function (Numerical solution)

Eikonal3d.py

Final Project Report.ipynb — Main program

Final_Project_Report.pdf — Report

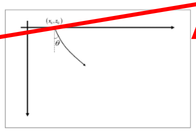
images — Appendix

Introduction-RK4

Final Project: 2D

Solution of 2D eikonal equation with initial values

$$\begin{cases} \dfrac{dx}{ds} = v(x,z)\,p_x, & \dfrac{dp_x}{ds} = \boxed{\dfrac{\partial}{\partial x}\left(\dfrac{1}{v(x,z)}\right)} \\ \dfrac{dz}{ds} = v(x,z)\,p_z, & \dfrac{dp_z}{ds} = \boxed{\dfrac{\partial}{\partial z}\left(\dfrac{1}{v(x,z)}\right)} \\ \dfrac{dT}{ds} = \dfrac{1}{v} \end{cases}$$



Initial values: x_0, z_0, p_{x0}, p_{z0}

First-Order System with x, z, p_x, p_z

$$T[l] = \int_{(A)}^{(B)} \frac{ds}{C}.$$

RK4

Auto differentiation

```python
    def _1vx(self,x,z):
        delta=self.delta
        return 0.5*(1/self.v(x+delta,z)-1/self.v(x-delta,z))/delta

    def _1vz(self,x,z):
        delta=self.delta
        return 0.5*(1/self.v(x,z+delta)-1/self.v(x,z-delta))/delta
```

Final Project

➤ Stepsize control and accuracy:

$$p_x^2 + p_y^2 + p_z^2 = \frac{1}{v^2}$$

$$\left| p_x^2 + p_y^2 + p_z^2 - \frac{1}{v^2} \right| < \delta$$

$$ds = \frac{ds}{2}$$

Each step:

$$p_x = \frac{p_x}{\left(p_x^2+p_y^2+p_z^2\right)^{1/2}}\frac{1}{v},\; p_y = \frac{p_y}{\left(p_x^2+p_y^2+p_z^2\right)^{1/2}}\frac{1}{v},$$

$$p_z = \frac{p_z}{\left(p_x^2+p_y^2+p_z^2\right)^{1/2}}\frac{1}{v}$$

$$\boxed{F(x_1,x_2,x_3,p_1,p_2,p_3) = \frac{1}{2}\left(p_1^{\,2} + p_2^{\,2} + p_3^{\,2} - \frac{1}{c^2(\mathbf{x})}\right) = 0}$$

Check steps

```python
    def _checkds(self,ds,parameter,count=0):
        if abs(parameter[2]**2+parameter[3]**2-
            1/self.v(parameter[0],parameter[1])**2)>self.tol and count<10:
            ds=ds/2
            return self._checkds(ds,parameter,count+1)
        return ds
```

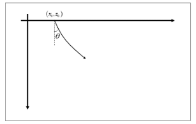
Eikonal2d.py

```python
    def _k(self,parameter):
        x,z,px,pz,_=parameter
        u=self.v(x,z)
        k11=u*px
        k12=u*pz
        k13=self._1vx(x,z)
        k14=self._1vz(x,z)
        k15=1/u
        return np.array([k11,k12,k13,k14,k15])

    def _RK4(self,parameter1,ds):
        k1=self._k(parameter1)
        parameter2=parameter1+ds/2*k1
        k2=self._k(parameter2)
        parameter3=parameter1+ds/2*k2
        k3=self._k(parameter3)
        parameter4=parameter1+ds*k3
        k4=self._k(parameter4)
        return parameter1+ds/6*(k1+2*k2+2*k3+k4)

    def _updateP(self,parameter):
        px0,pz0=parameter[2],parameter[3]
        v=self.v(parameter[0],parameter[1])
        denom=np.sqrt(px0**2+pz0**2)*v
        parameter[2]=px0/denom
        parameter[3]=pz0/denom
```

➤ Traveltime and ray tracing: 3D

Solution of 3D eikonal equation with initial valu

$$\begin{cases} \dfrac{dx}{ds} = v(x,y,z)\,p_x, & \dfrac{dp_x}{ds} = \dfrac{\partial}{\partial x}\left(\dfrac{1}{v(x,y,z)}\right) \\ \dfrac{dy}{ds} = v(x,y,z)\,p_y, & \dfrac{dp_y}{ds} = \dfrac{\partial}{\partial y}\left(\dfrac{1}{v(x,y,z)}\right) \\ \dfrac{dz}{ds} = v(x,y,z)\,p_z, & \dfrac{dp_z}{ds} = \dfrac{\partial}{\partial z}\left(\dfrac{1}{v(x,y,z)}\right) \\ \dfrac{dT}{ds} = \dfrac{1}{v} \end{cases}$$



Initial values: $\frac{l}{c}, \frac{m}{c}, \frac{n}{c}$ x_0, y_0, z_0

First-Order System with $x, y, z, p_x, p_y, p_z, T$

$$T = \int_{x_0,y_0,z_0}^{x,y,z} \frac{1}{v(x,y,z)}\,ds$$

$l = \cos\theta$,
$m = \sin\theta\sin\varphi$,
$n = \sin\theta\cos\varphi$

RK4

```python
    def _1vx(self,x,y,z):
        delta=1e-3
        return (1/self.v(x+delta,y,z)-1/self.v(x,y,z))/delta
    def _1vy(self,x,y,z):
        delta=1e-3
        return (1/self.v(x,y+delta,z)-1/self.v(x,y+delta,z))/delta
    def _1vz(self,x,y,z):
        delta=1e-3
        return (1/self.v(x,y,z+delta)-1/self.v(x,y,z))/delta
```

```python
    def _checkds(self,ds,parameter,count=0):
        if abs(parameter[3]**2+parameter[4]**2+parameter[5]**2-
            1/self.v(parameter[0],parameter[1],
                     parameter[2])**2)>self.tol and count<10:
            ds=ds/2
            return self._checkds(ds,parameter,count+1)
        return ds
```

Eikonal3d.py

```python
    def _k(self,parameter):
        x,y,z,px,py,pz,_=parameter
        u=self.v(x,y,z)
        k11=u*px
        k12=u*py
        k13=u*pz
        k14=self._1vx(x,y,z)
        k15=self._1vy(x,y,z)
        k16=self._1vz(x,y,z)
        k17=1/u
        return np.array([k11,k12,k13,k14,k15,k16,k17])

    def _RK4(self,parameter1,ds):
        k1=self._k(parameter1)
        parameter2=parameter1+ds/2*k1
        k2=self._k(parameter2)
        parameter3=parameter1+ds/2*k2
        k3=self._k(parameter3)
        parameter4=parameter1+ds*k3
        k4=self._k(parameter4)

        return parameter1+ds/6*(k1+2*k2+2*k3+k4)

    def _updateP(self,parameter):
        px0,py0,pz0=parameter[3],parameter[4],parameter[5]
        v=self.v(parameter[0],parameter[1],parameter[2])
        denom=np.sqrt(px0**2+py0**2+pz0**2)*v
        parameter[3]=px0/denom
        parameter[4]=py0/denom
        parameter[5]=pz0/denom
```

Copy from ESS207-project.pdf (31,35)

2D Ray Paths- Test board

Linearly Layered Velocity Model

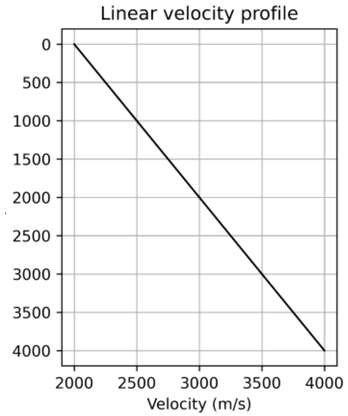
$(v(z) = v_0 + \alpha z)$

```
X_analy=2*u0/np.tan(toa)/alp
T_analy=np.log((1+np.cos(toa))/(1-np.cos(toa)))/alp
```

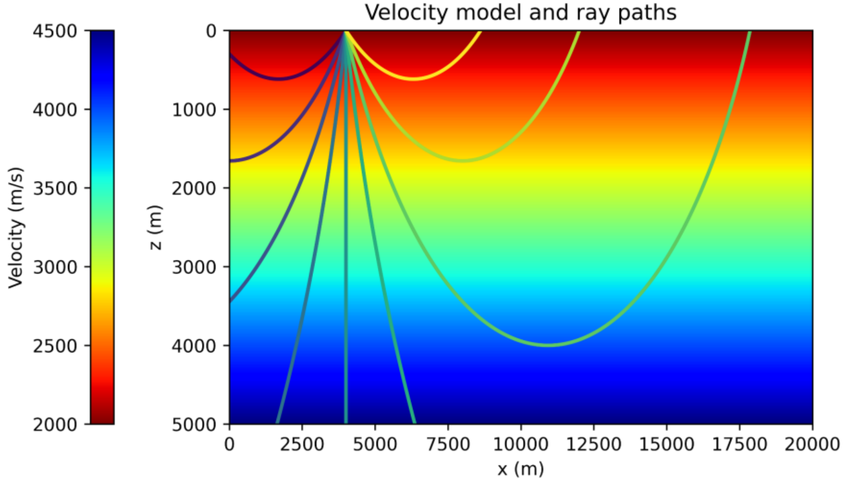
$$X = \frac{2v_0}{\alpha \tan\theta_0}$$

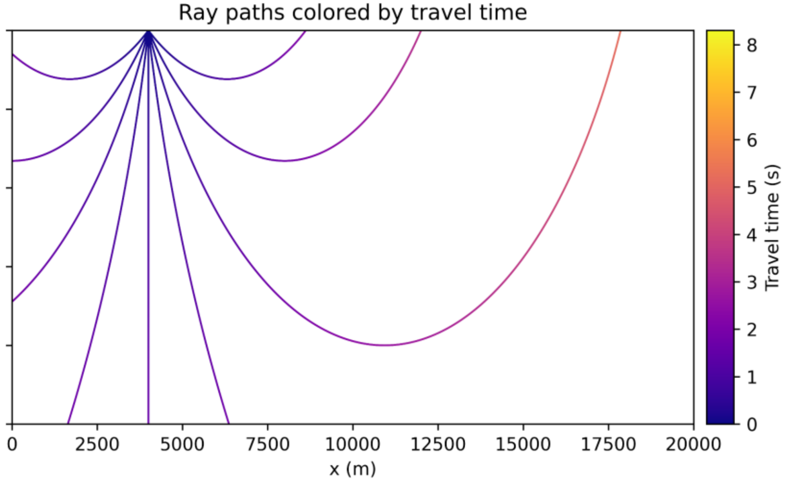
$$T = \frac{1}{\alpha}\ln\left(\frac{1+\cos\theta_0}{1-\cos\theta_0}\right)$$

(Report P4-7)

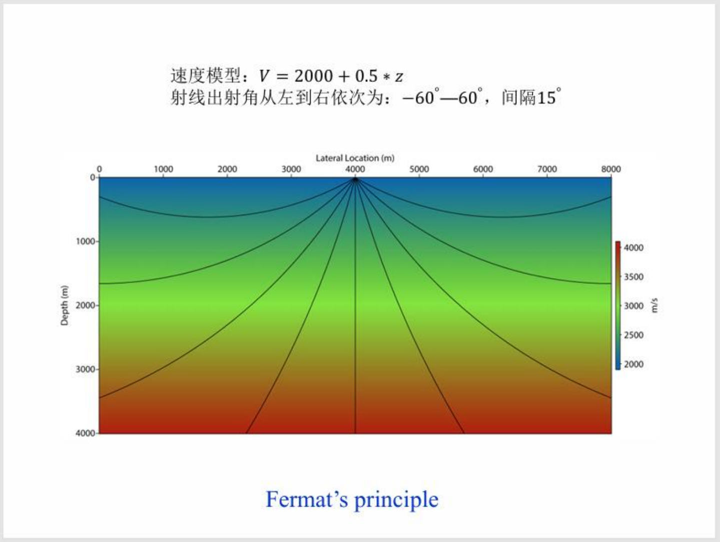


TOA (deg)	X_num (m)	X_ana (m)	RelErr_X	T_num (s)	T_ana (s)	RelErr_T
-60.00	-4621.57	-4618.80	-6.00e-04	2.199	2.197	7.29e-04
-45.00	-8002.99	-8000.00	-3.74e-04	3.528	3.525	6.01e-04
-30.00	-13858.82	-13856.41	-1.75e-04	5.270	5.268	4.60e-04
-15.00	-29857.54	-29856.41	-3.81e-05	8.113	8.110	2.71e-04
15.00	29857.54	29856.41	3.81e-05	8.113	8.110	2.71e-04
30.00	13858.82	13856.41	1.75e-04	5.270	5.268	4.60e-04
45.00	8002.99	8000.00	3.74e-04	3.528	3.525	6.01e-04
60.00	4621.57	4618.80	6.00e-04	2.199	2.197	7.29e-04

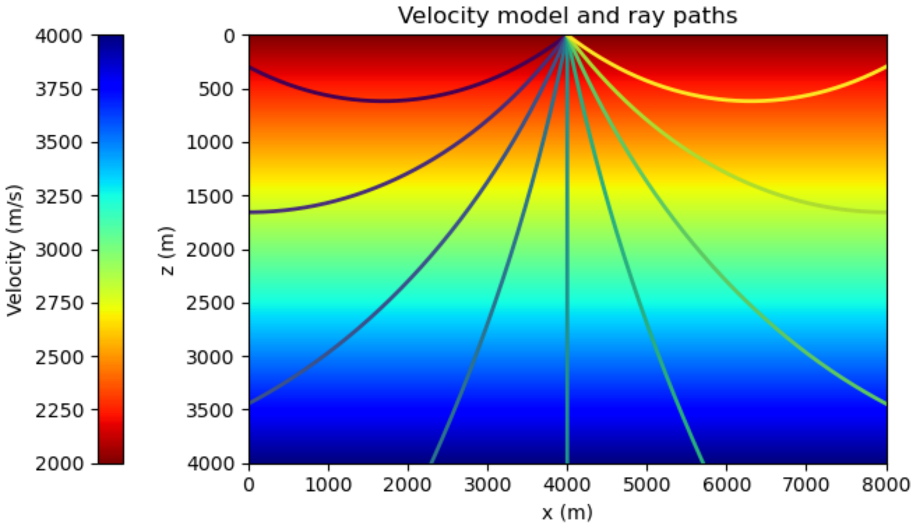


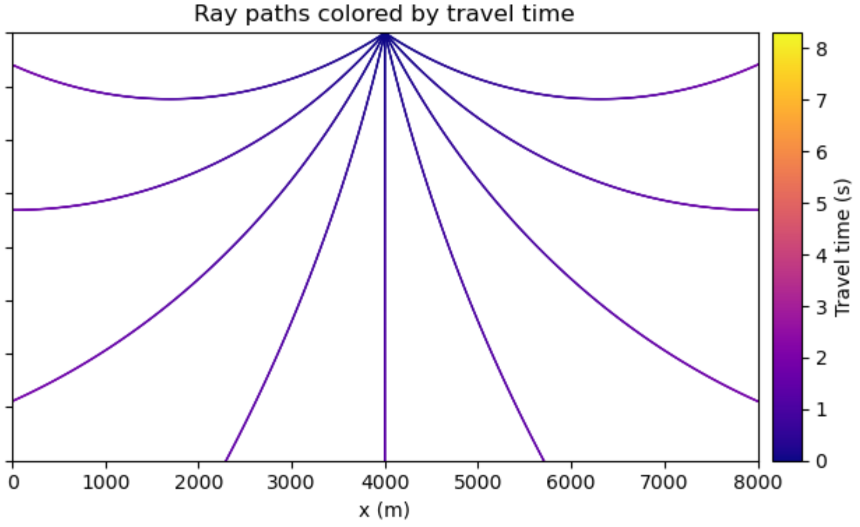


2D Ray Paths- Test board



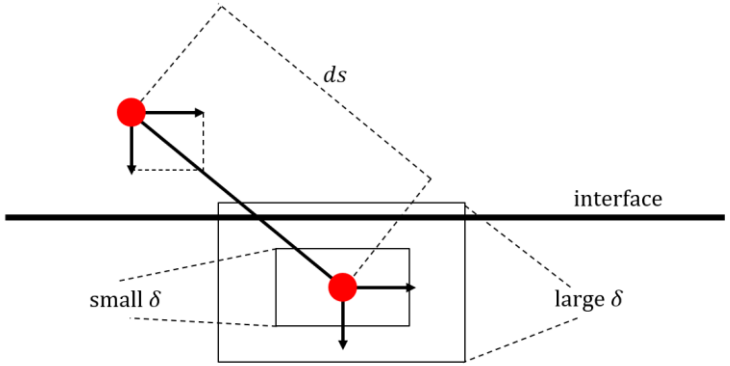
Copy from ESS207-project.pdf (33)

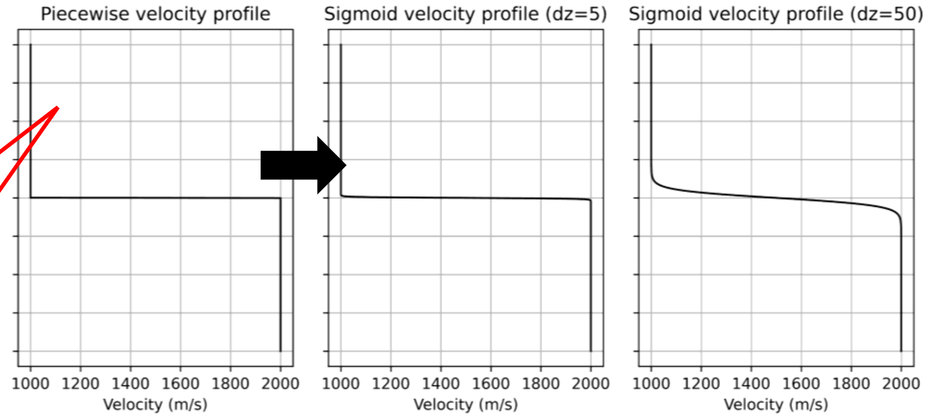


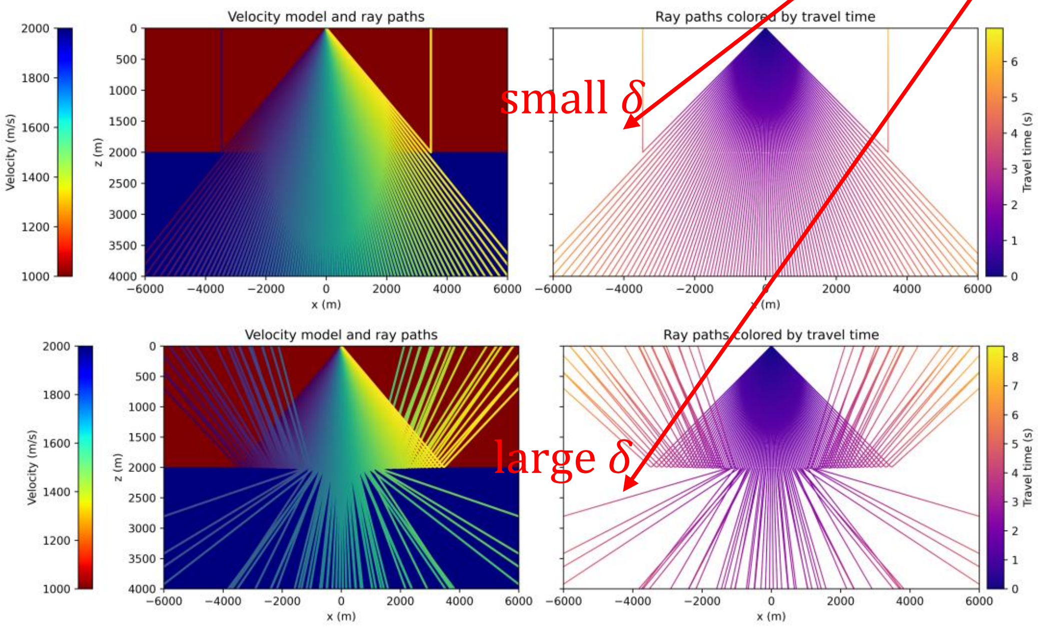


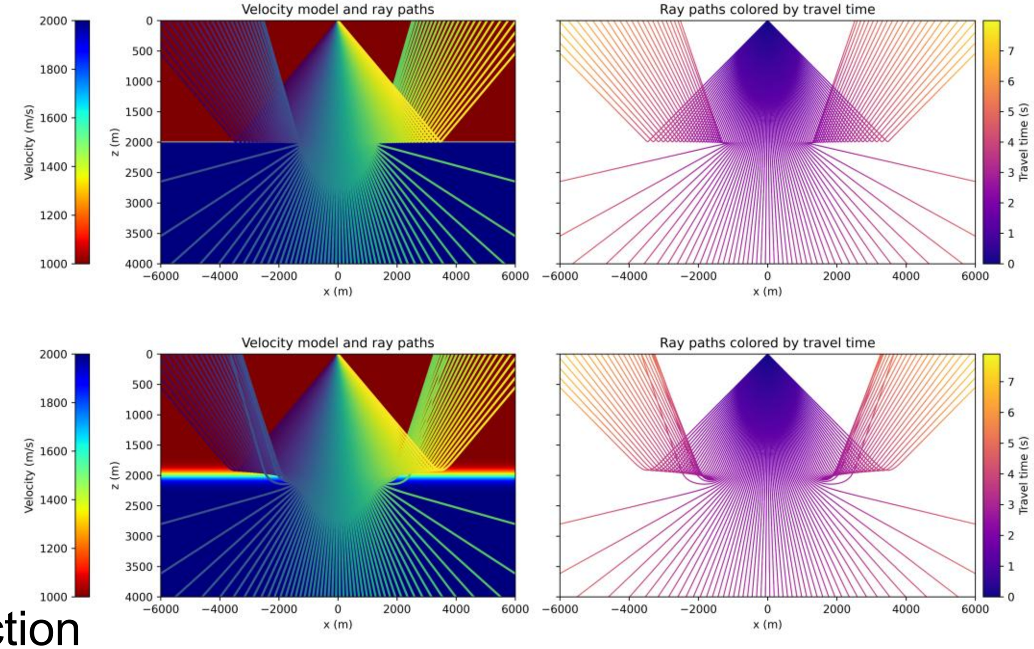
2D Ray Paths- Reflection phenomena

```python
# sigmoid velocity model
def v12(x, z):
    v1 = 1000
    v2 = 2000
    z0 = 2000
    dz = 5
    return v2 + (v1 - v2) / (1 + np.exp((z - z0) / dz))
```









small ds

large ds

(Report P7-11)

Use sigmoid function
to generate continuous step function

2D Ray Paths- Multi reflection

```
if multi==0:
    if path_z[-1]<0:
        break
```

To correctly model free-surface reflection, the surface boundary must be properly treated. In the real world, the surface corresponds to air, where the seismic wave velocity is approximately zero. However, directly setting the surface velocity to zero leads to numerical and physical inconsistencies in the Eikonal formulation. According to Snell's law,

$$\frac{\sin\theta}{v} = \frac{\sin\theta_s}{v_s}$$

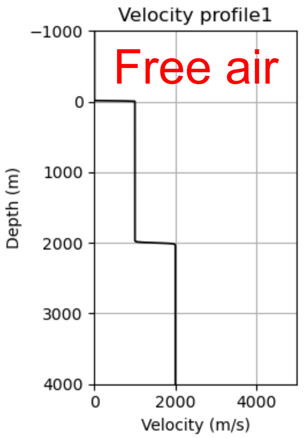
as$\theta\approx 0$, the incidence angle θ tends toward zero, which implies vertical propagation rather than reflection.

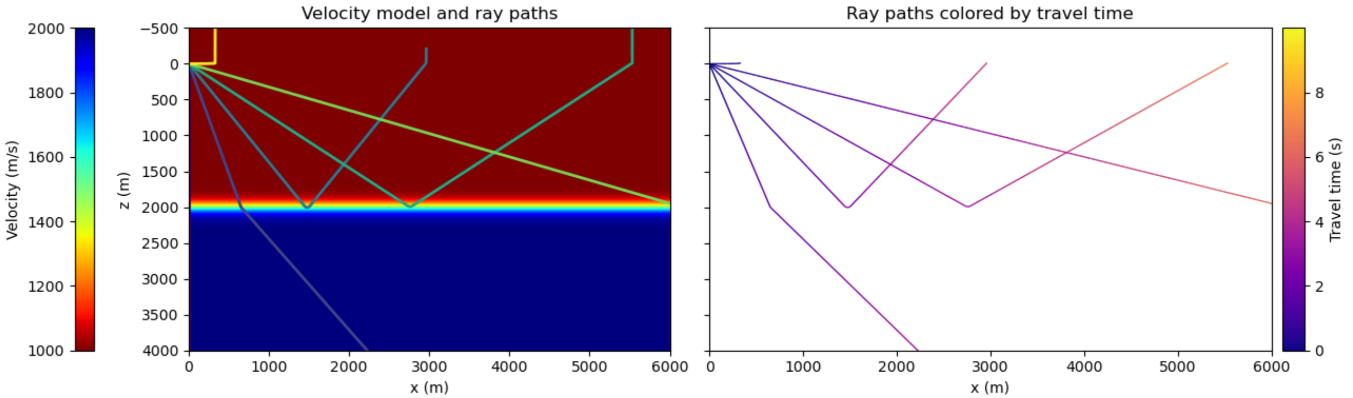
Moreover, free-surface reflection is fundamentally governed by the wave transport equation and boundary conditions, rather than the Eikonal equation itself. Although $\theta_s = 0$ is a mathematically valid solution, all incident wave energy should be reflected at the free surface.

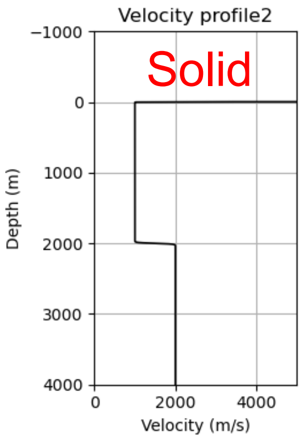
To overcome this limitation within the Eikonal-based ray-tracing framework, we artificially assign an extremely large velocity at the surface. This treatment effectively enforces ray reflection and allows free-surface reflections to be generated manually within the numerical scheme.

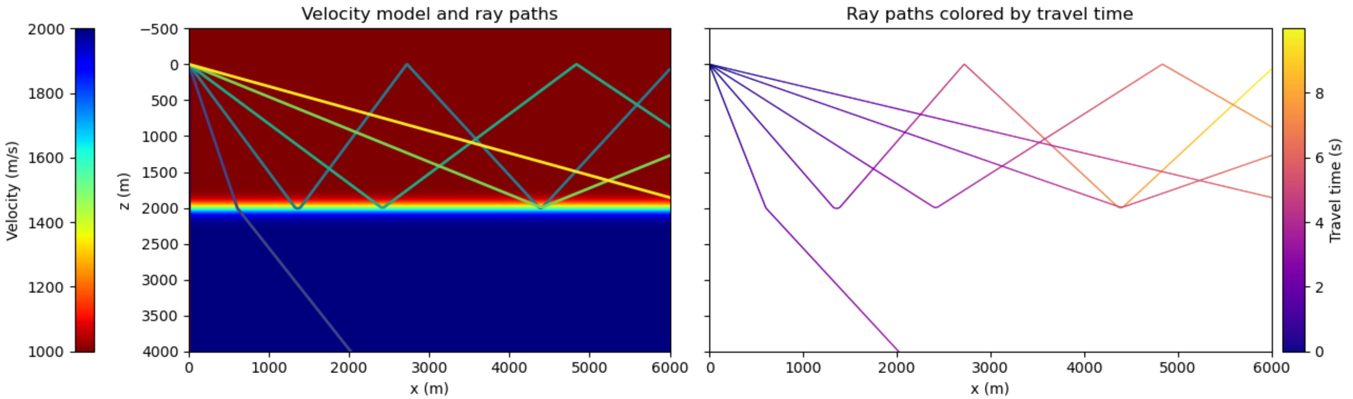
(from Report P11)

The eikonal equation describes real ray paths, but the wave energy is totally reflected at the free surface.







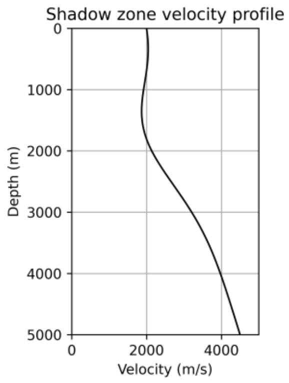


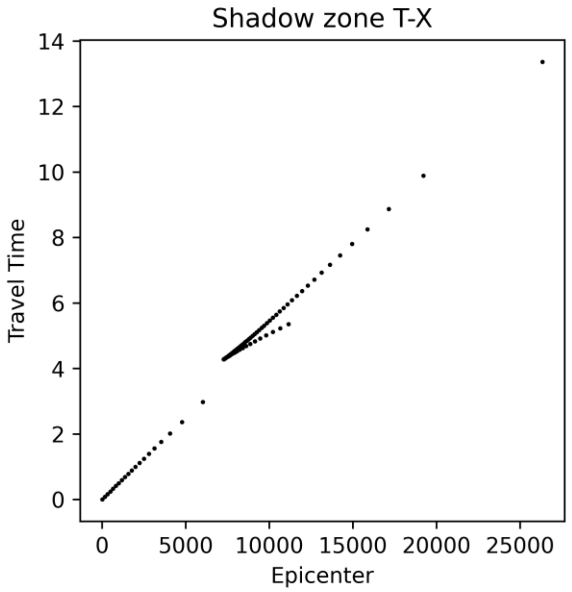
(Report P11-16)

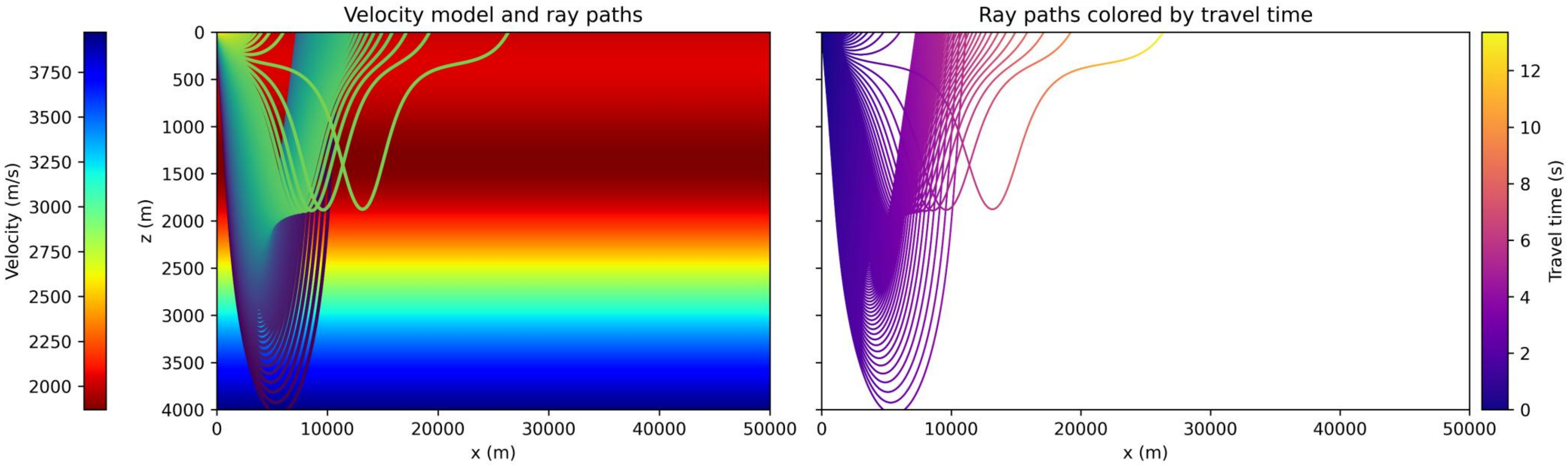
2D Ray Paths- Shadow zone

(Report P17-19)

```
# shodow zone velocity function
def v_shadow(x,z ):
    v_bg = 2000 + 0.5 * z
    dv   = 200
    zc   = 1250
    dz   = 1300
    return v_bg - 0.003*z*dv * np.exp(-((z - zc) / dz) ** 2)
toa=np.linspace(np.pi/6,np.pi/2,100)
```



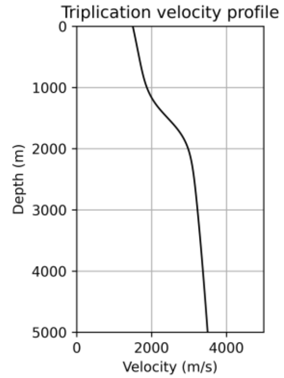


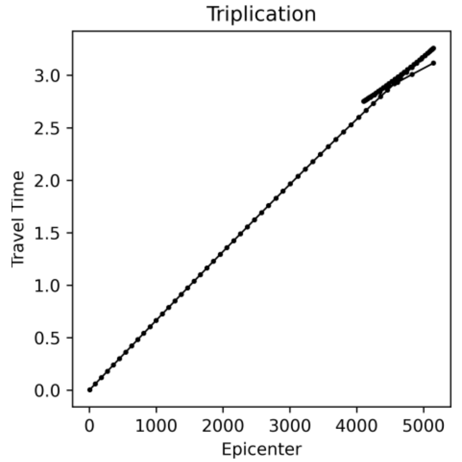


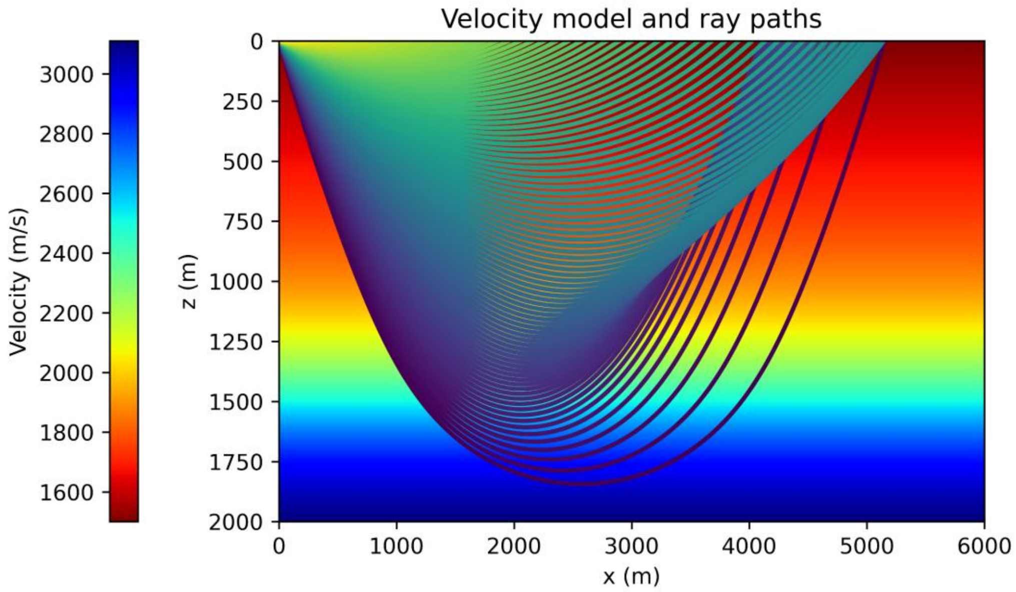
2D Ray Paths- Triplication

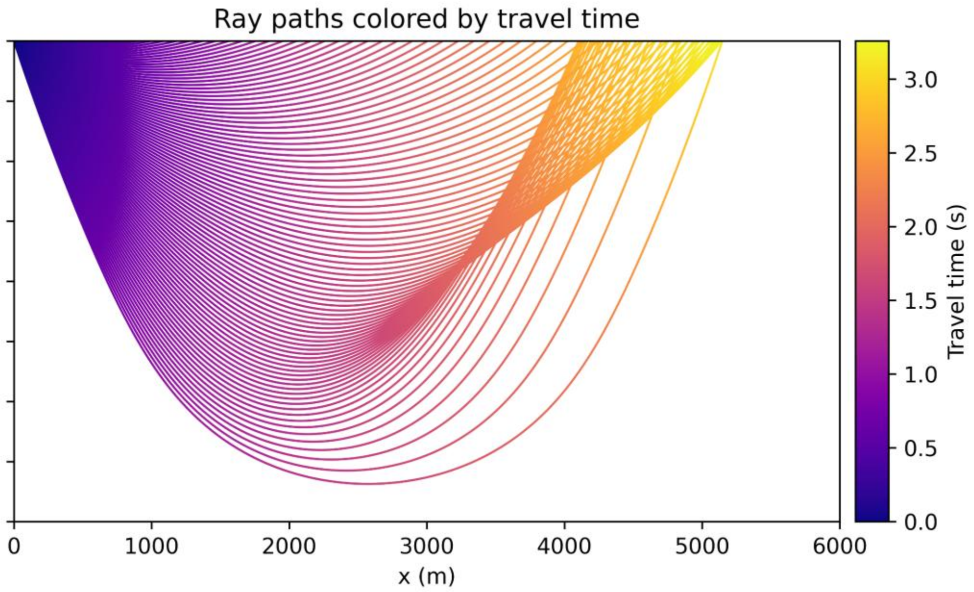
(Report P20-22)

```python
def v_triplication(x, z):
    V0 = 1500
    Vmax = 3000
    zc = 1500
    k = 0.005
    a1 = 500
    b1 = 0.0005
    slow = a1 * (1 - np.exp(-b1 * z))
    fast = (Vmax - V0 - slow.max()) / (1 + np.exp(-k * (z - zc)))
    V = V0 + slow + fast
    return V+0.1*z
```









2D Ray Paths- Class problem

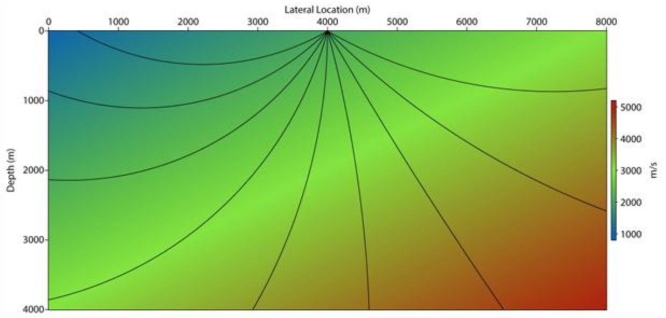
Final Project: 2D (strict)

速度模型：
$V = 1000 + 0.5 * z + 0.25 * x$

射线出射角从左到右依次为：-60°—60°，间隔15°

RK4

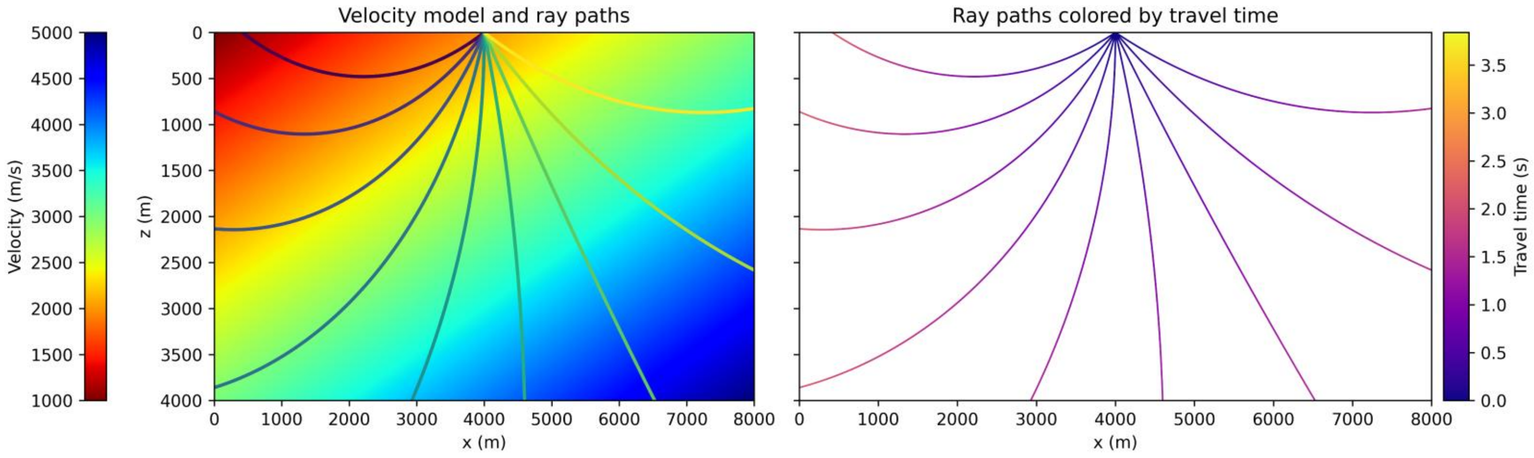
Submit a PPT report together with code



$$\frac{d\tau}{d\sigma} = \frac{1}{c(\mathbf{x})}$$

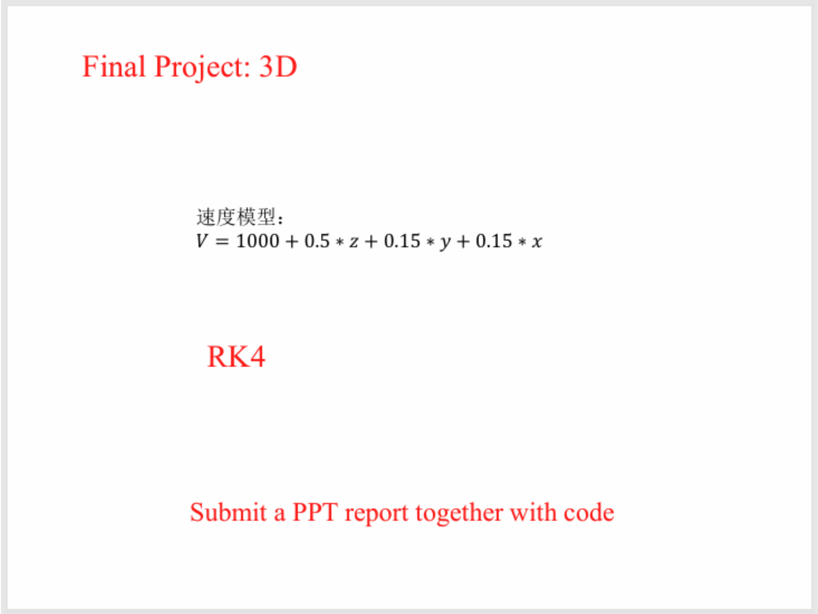
Copy from ESS207-project.pdf (27,34)

```
def v(x,z):
    return 1000+0.5*z+0.25*x

#####################  Use Eikonal2d to get numerical solutions  ####################
M=Eikonal2d(v,ds=5,max_step=1200)
d=np.pi/12
toa=np.arange(-np.pi/3,np.pi/3+d,d) # take off angle

############## Use Eikonal2d.Raypath to calculate the numerical solutions ###########
rayx_all, rayz_all, rayt_all = [], [], []
for ii in range(len(toa)):
    rayx, rayz, rayt = M.Raypath(4000, 0, toa[ii])
    rayx_all.append(rayx)
    rayz_all.append(rayz)
    rayt_all.append(rayt)
####################################################################################
```



(Report P22-23)

3D Ray Paths- Class problem


Final Project: 3D

速度模型:
$V = 1000 + 0.5 * z + 0.15 * y + 0.15 * x$

RK4

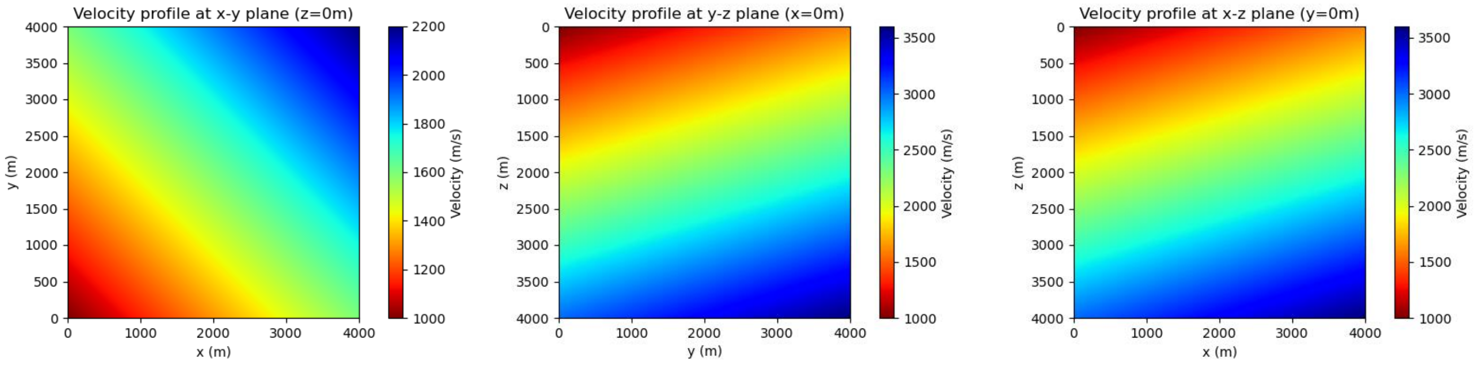
Submit a PPT report together with code

Copy from ESS207-project.pdf (37)

3D ray paths are difficult to visualize, so we only present the results relevant to the class problem.

```
def v(x,y,z):
    return 1000+0.5*z+0.15*y+0.15*x

M = Eikonal3d(v, ds=10, max_step=2000)
```

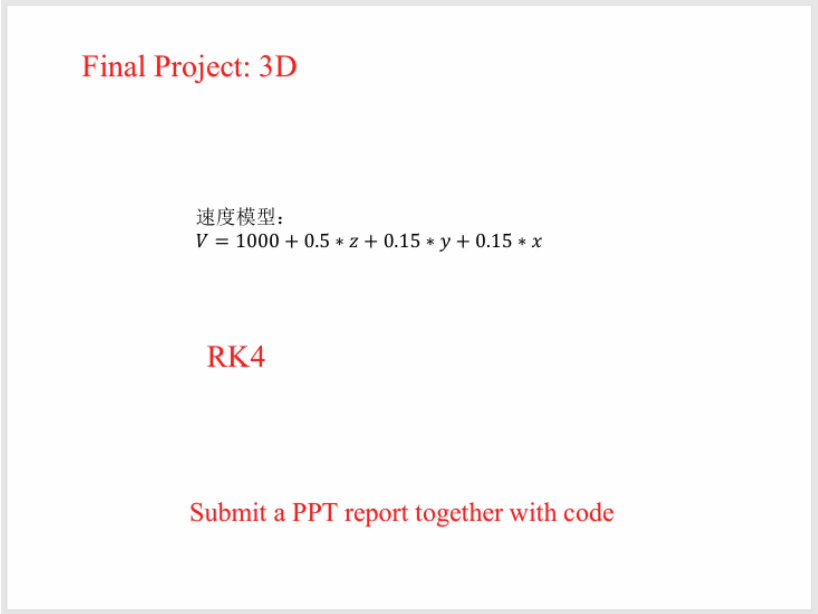
```
for toa in toa_list:
    for azi in azi_list:
        rayx, rayy, rayz, rayt = M.Raypath(x0, y0, z0, toa, azi)
```



Velocity profiles

3D Ray Paths- Class problem

3D ray paths are difficult to visualize, so we only present the results relevant to the class problem.


Final Project: 3D

速度模型：
$V = 1000 + 0.5 * z + 0.15 * y + 0.15 * x$

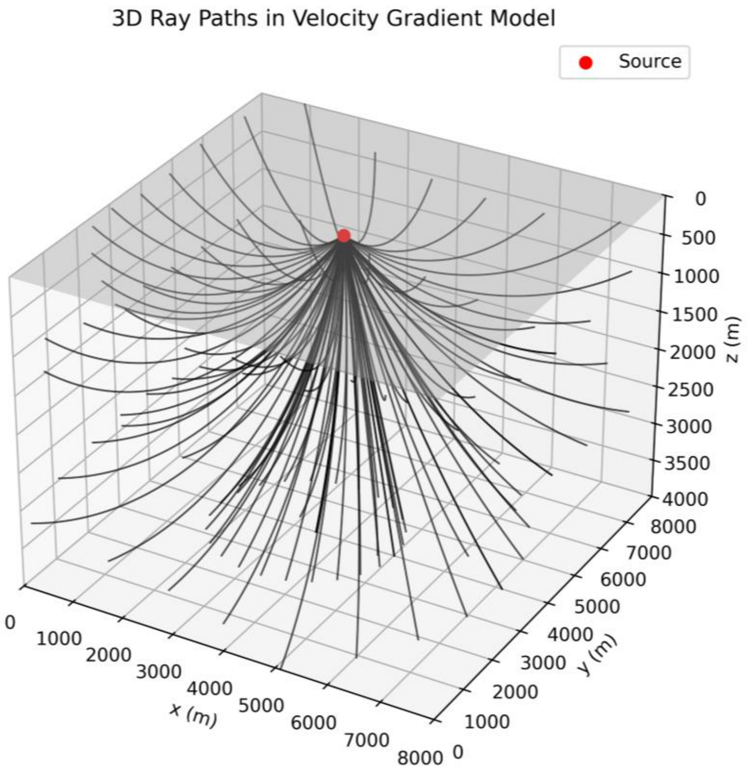
RK4

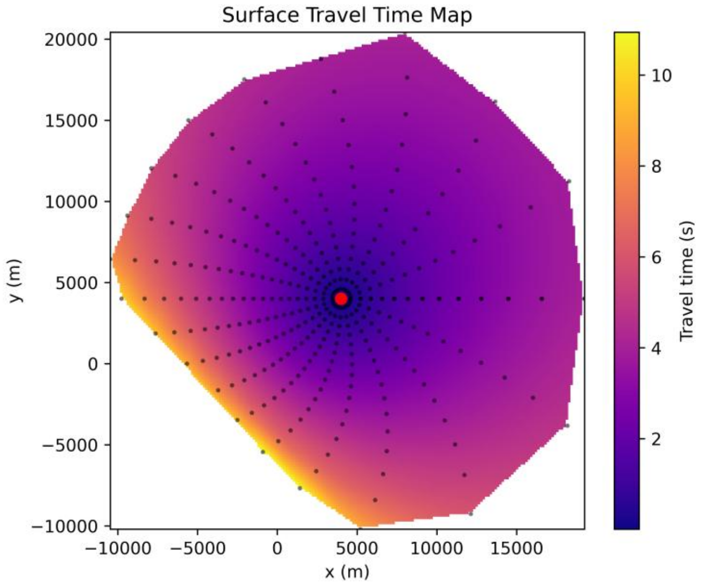
Submit a PPT report together with code

Copy from ESS207-project.pdf (37)

```
def v(x,y,z):
    return 1000+0.5*z+0.15*y+0.15*x

M = Eikonal3d(v, ds=10, max_step=2000)
```

```
for toa in toa_list:
    for azi in azi_list:
        rayx, rayy, rayz, rayt = M.Raypath(x0, y0, z0, toa, azi)
```





Thank You for Your Attention

Questions and comments are welcome.

Conclusion

- Implemented RK4 to solve ray-tracing ODEs from the Eikonal equation
- Verified accuracy using a linear velocity model (relative error $\approx 10^{-4}$)
- Analyzed reflection behavior and the trade-off between Eikonal and transport equations
- Reproduced seismological phenomena such as shadow zones and triplication
- Solved the class problem in both 2D and 3D cases