バイオセンシング特論　課題

創成科学研究科　機械工学科専攻

学籍番号：2288030381

氏名：岩城　好佑

本報告では図1に示す障害物を含む領域において，点Aに入力加重 を加えた時における波の挙動を観察する．境界条件はノイマン境界条件（自由端反射）とし，細かい仕様は自分で定義してよいものとする．

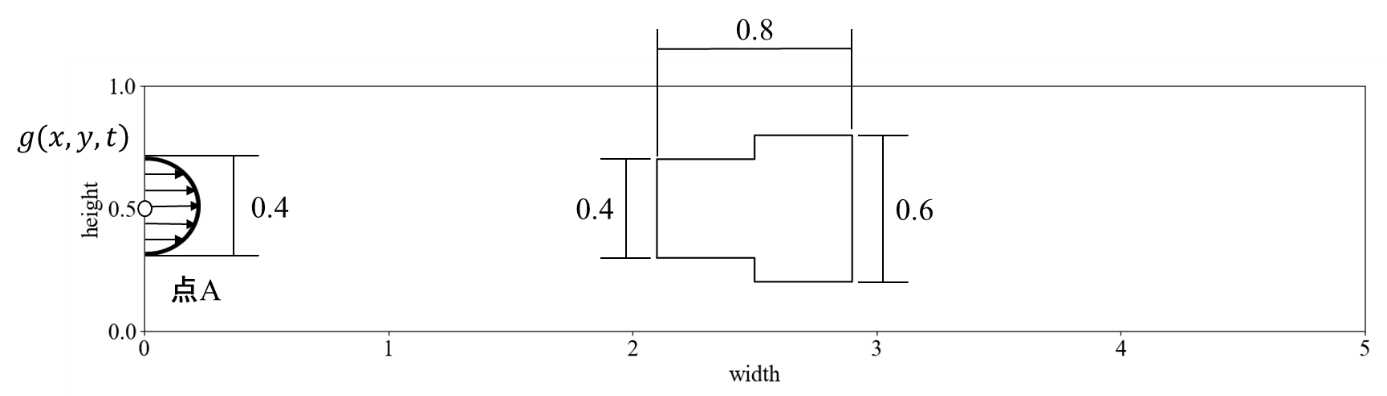


図1 観察領域

数値計算は以下に示す2次元波動方程式を用いる．

今回の報告では，式1の解法として差分法を用い．空間刻み幅をとして領域を離散化することを考える．また，図2に示すように境界付近の点によって解の導出が異なるので場合分けをすることに注意する．

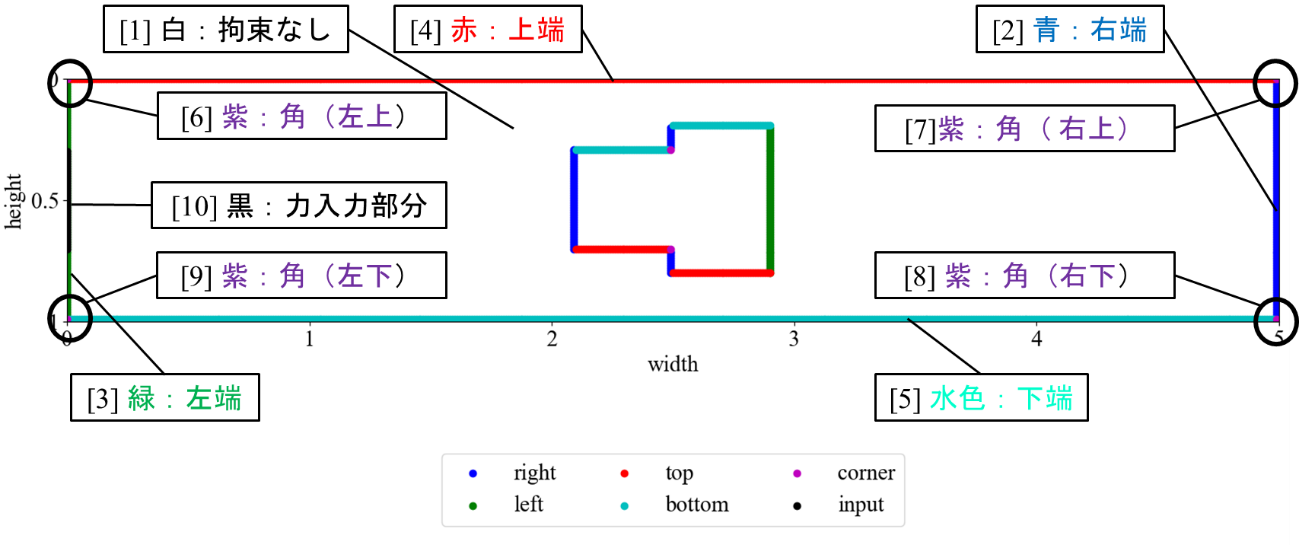


図2　境界の場所による場合分け

[1] 拘束なしの点

格子点での空間微分を考えたとき，中心差分を用いて

となり，これらの式を用いると

が成り立つ．*y*についても，同様の考え方で，

となることが分かる．式(5),(6)を式(1)に代入して整理すると，次の式が得られる．

さらに，式(7)の左辺を同様の考え方で時間領域において中心差分をとり，整理すると，次の式が得られる．

ただし，とする．

[2] 右に境界がある点

　図3のような仮想格子点を考える．

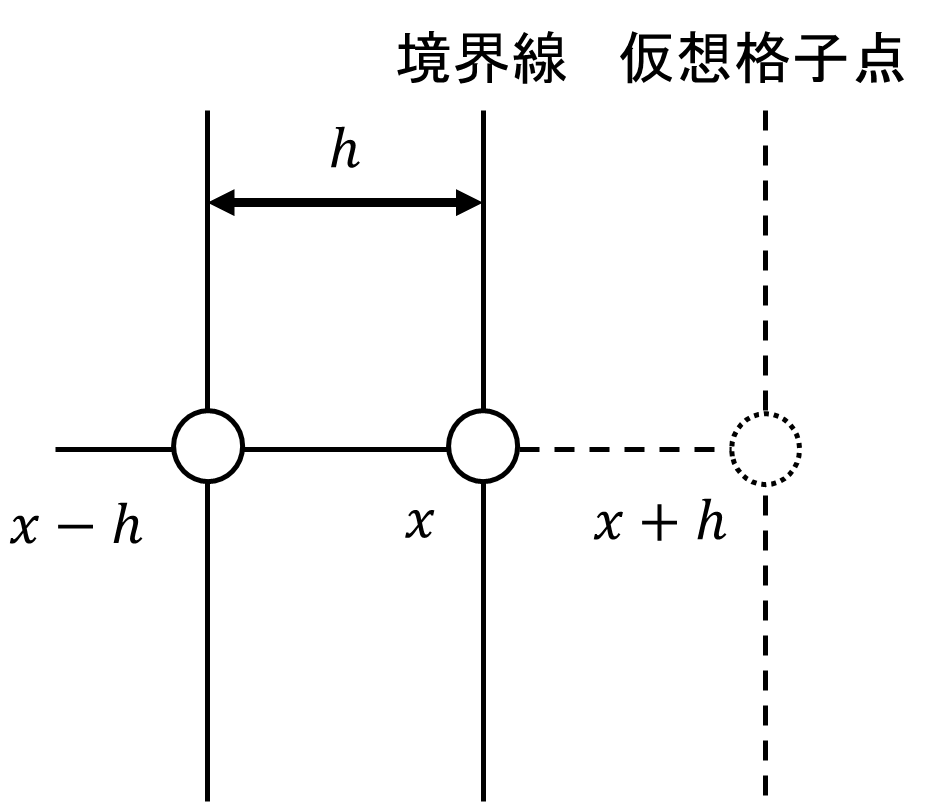


図３　仮想格子点のイメージ

境界上でひずみは

となるため

となる．式(10)を式(8)に代入すると，

[3]左に境界がある点

　[2]と同様に考えると，

[4]上に境界がある点

　[2]と同様に考えると，

[5]下に境界がある点

[2]と同様に考えると，

[6]左と上に境界がある点

[3]と[4]を同時に考慮して，

[7]右と上に境界がある点

　[2]と[4]を同時に考慮して，

[8]右と下に境界がある点

　[2]と[5]を同時に考慮して，

[9]左と下に境界がある点

　[3]と[5]を同時に考慮して，

[10] 左に力入力の境界がある点

　ひずみが生じる変位勾配をとすると，

となるため

が成り立つ．これを式（8）に代入すると，

以上挙げた式を用いてPythonで波動方程式の数値計算を行った．ソースコードは本報告の末尾に記載する．表1に数値解析を行った際のパラメータを示す．クーラン条件より，時間刻み幅を空間刻み幅より小さくさせることに留意する．

表1　解析条件

|  |  |
| --- | --- |
| 時間刻み | 0.005 [s] |
| 空間刻み幅 *h* | 0.01 [m] |
| 速度 | 1 [ |
| 解析時間範囲 | 0 ~ 10 [s] |

また，変異勾配 は以下のように定義した．

周期 は [s]とした．

次の図4~23に表1と式(22)の条件下におけるの分布を0.5秒ごとに示す．

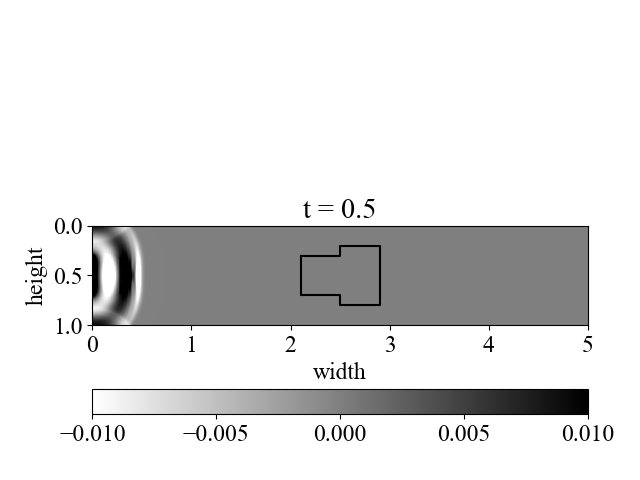


図4 t = 0.5[s]の分布

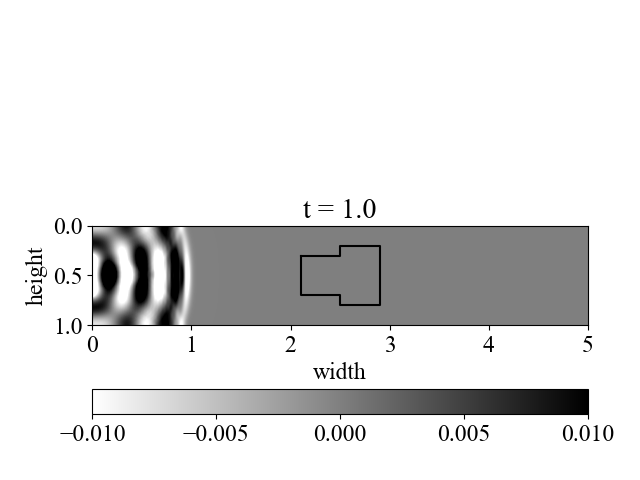


図5 t = 1.0 [s] の分布

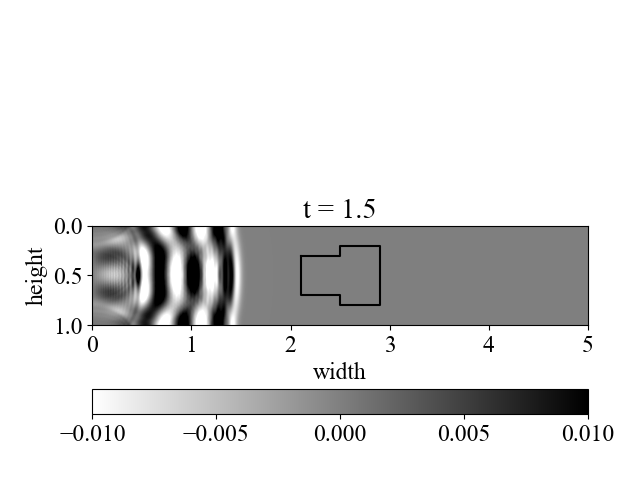


図6 t = 1.5 [s] の分布

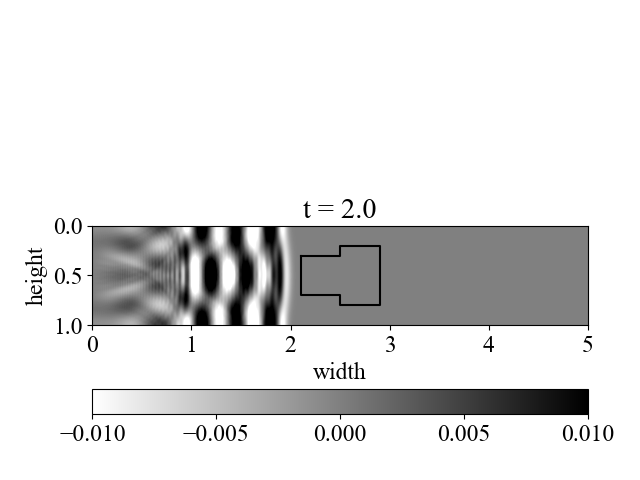


図7 t = 2.0 [s] の分布

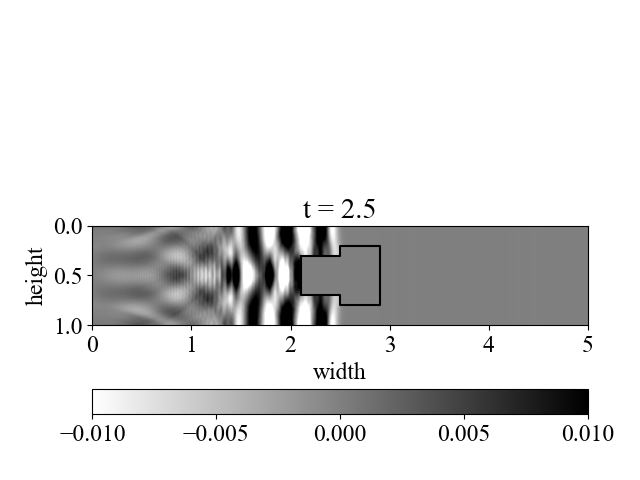


図8 t = 2.5 [s] の分布

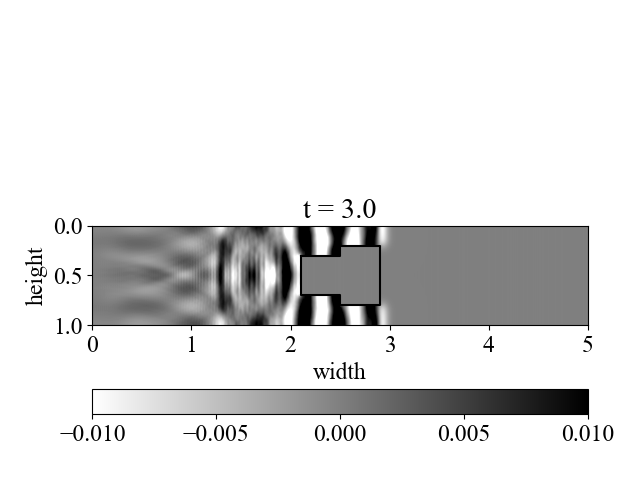


図9 t = 3.0 [s] の分布

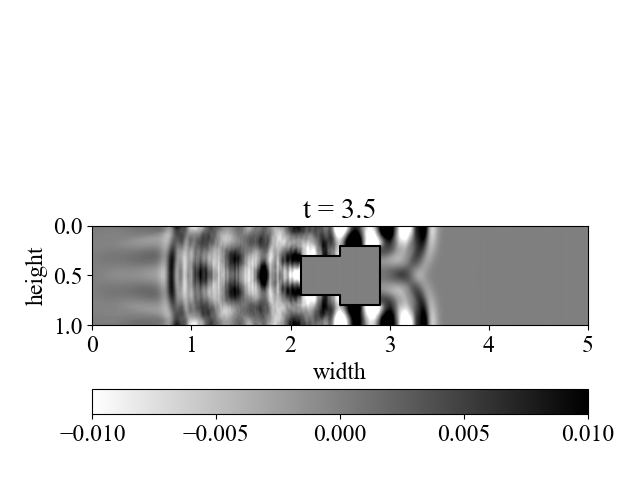


図10 t = 3.5 [s] の分布

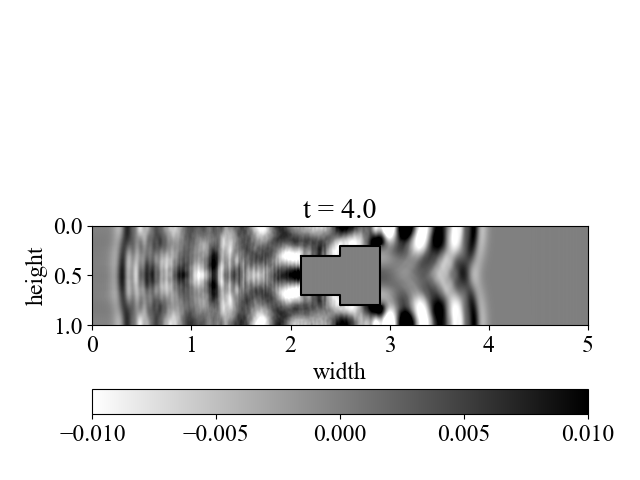


図11 t = 4.0 [s] の分布

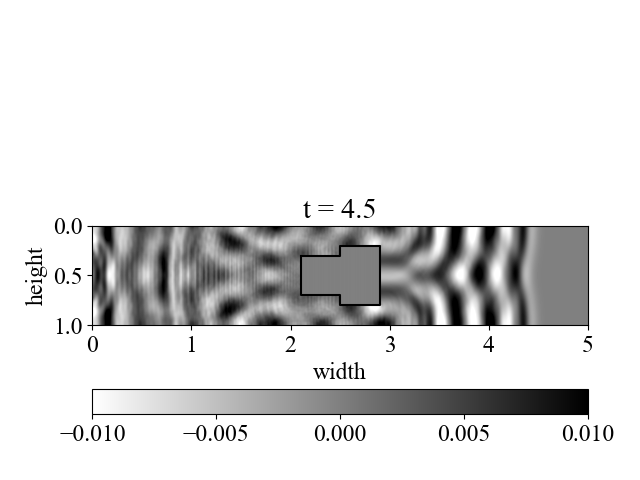


図12 t = 4.5 [s] の分布

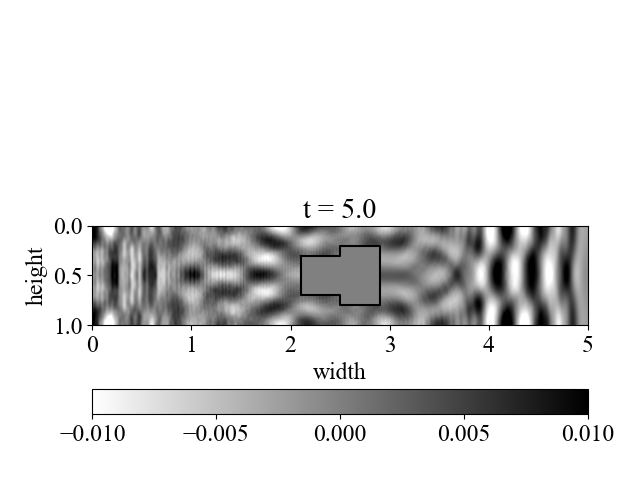


図13 t = 5.0 [s] の分布

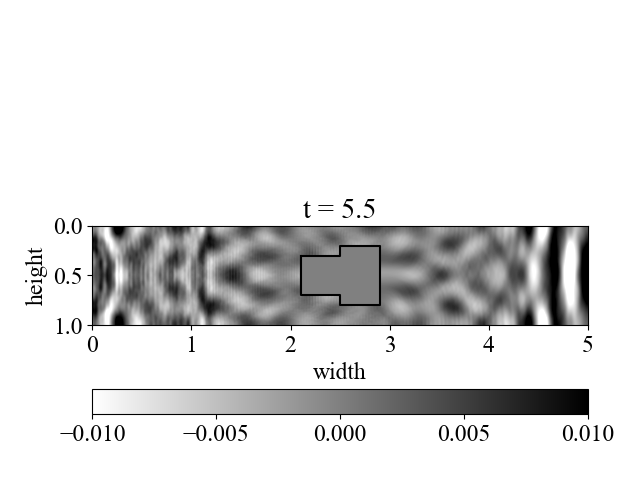


図14 t = 5.5 [s] の分布

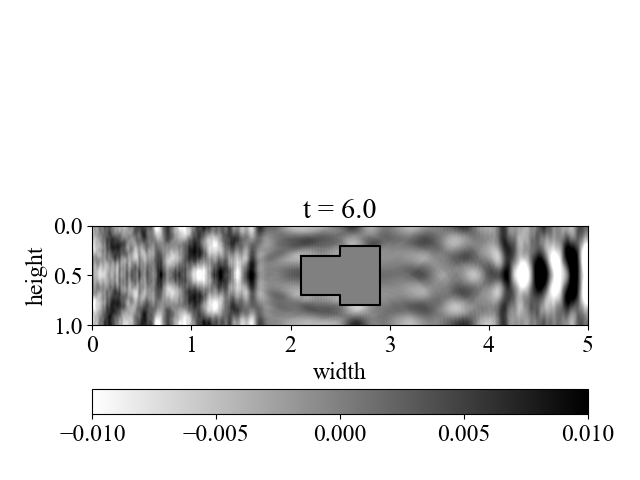


図15 t = 6.0 [s] の分布

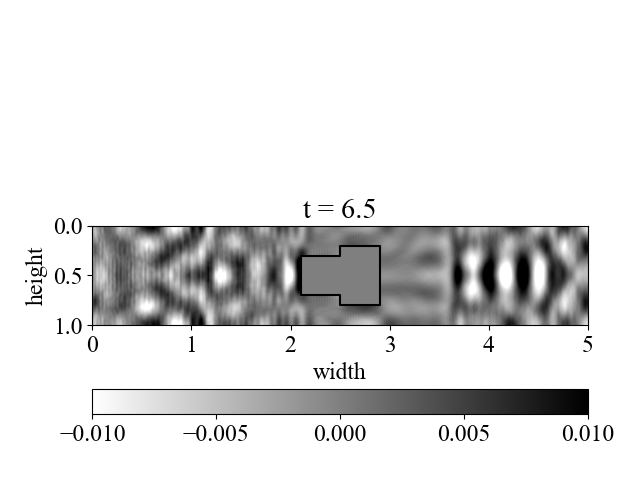


図16 t = 6.5 [s] の分布

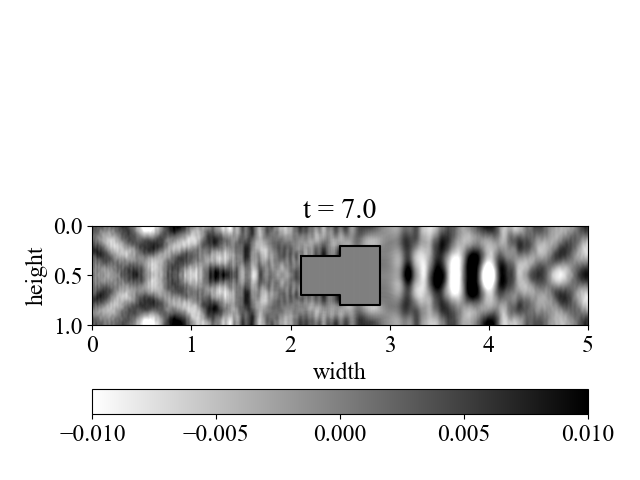


図17 t = 7.0 [s] の分布

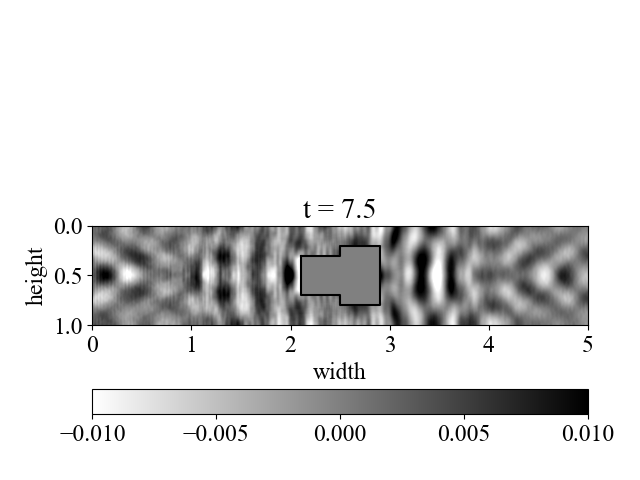


図18 t = 7.5 [s] の分布

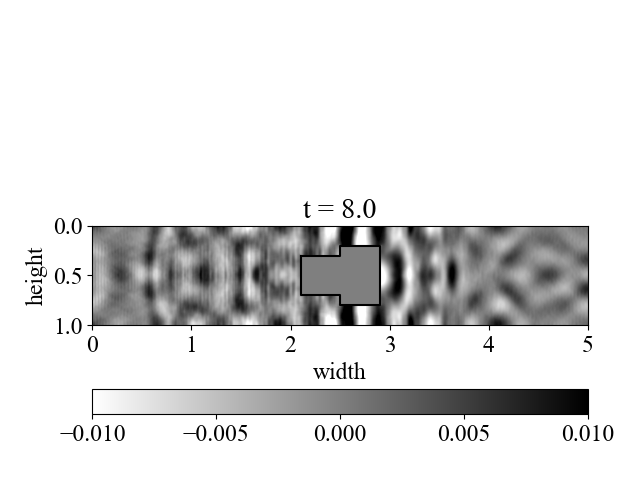


図19 t = 8.0 [s] の分布

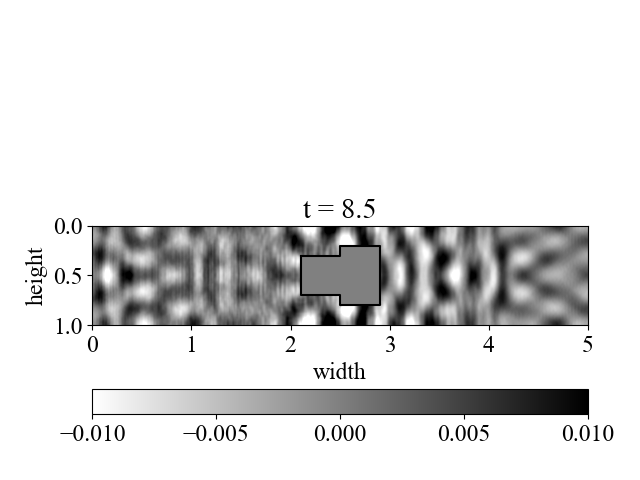


図20 t = 8.5 [s] の分布

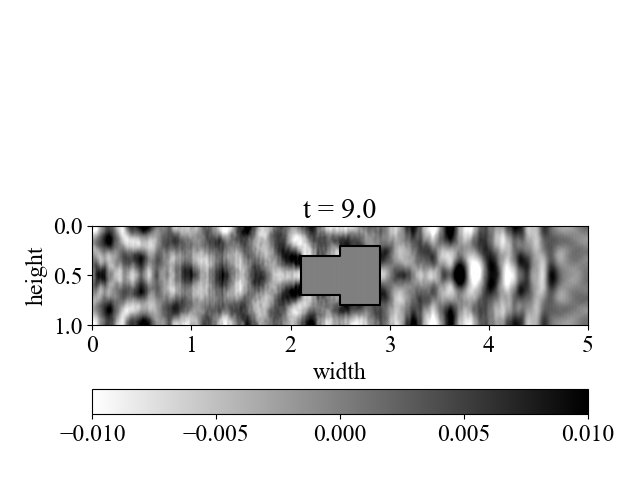


図21 t = 9.0 [s] の分布

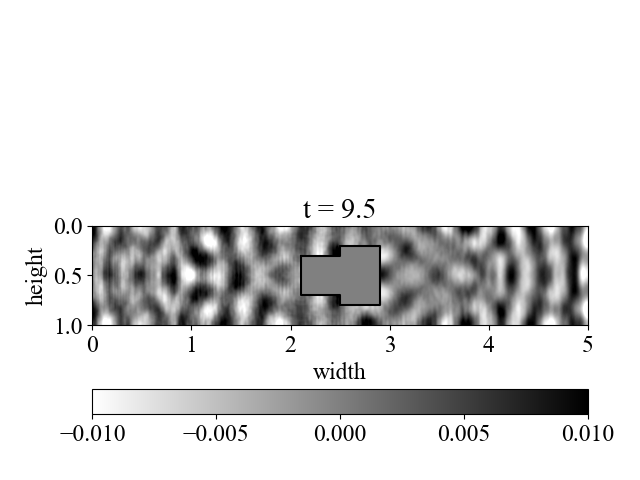


図22 t = 9.5 [s] の分布

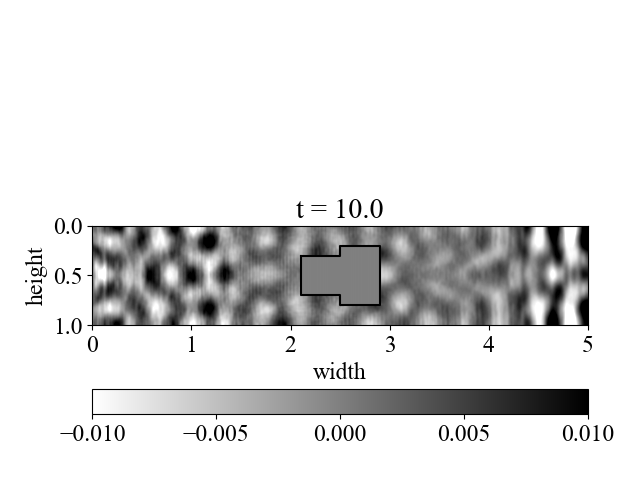


図23 t = 10.0 [s] の分布

これらの図を観察すると，点Aで*cos*波のひずみ入力が加わり，その部分から分布が変化していることが分かる．そして， のときに入力側の境界と反対側の境界に波が到達していることがわかる．今回は波の速度を1 [m/s]としているため，数値計算はうまく計算できていると考えることができる．また， は障害物と衝突したときの反射， は右壁と衝突したときの反射の様子が確認できる．比較してみると，前者は波の一部のみ反射し，反射後の波は振幅が弱くなるのに対して，後者の反射は，多少波の配置が崩れるものの，振幅が弱まらずに反射していることが分かる．また，障害物通貨直後の波 () を見ると障害物に振幅を吸い取られるような波の形をしていることが分かる．

図3～23の一連の流れを0.01秒刻みで動画として閲覧できるgif画像を作成した．Pdfやword上ではgifは機能しないため，今回の課題で作成したものはこの報告書を除き，すべてGithub上のリポジトリに配置している．図24にQRコードを示す．

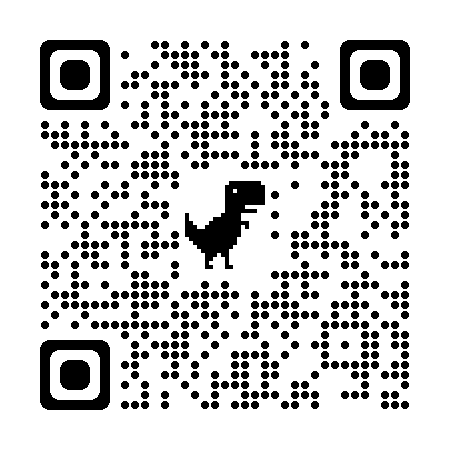


図24 リポジトリに移動するQRコード

リポジトリには課題とは違う条件で数値実験した結果もgifとして載せている．

最後に，今回の課題で作成したpythonスクリプトを示す．

|  |
| --- |
| import os  import random  from typing import Callable, List  import numpy as np  import matplotlib.pyplot as plt  from matplotlib import animation  class SimulatePropagation:      def \_\_init\_\_(self,width:int,height:int,h:float,dt:float,border\_vecs:np.ndarray=None,prop\_grads:List[str]=None,\                  distorted\_vec:np.ndarray=None,distorted\_func:Callable[[float,float,float],np.ndarray]=None,condition:str='neumann'):          if condition not in {'neumann','diricre'}:              raise ValueError('argument of condition must be \"neumann\" or \"diricre\"')          self.width = width          self.height = height          self.dt = dt          self.h = h          self.time = 0.          self.alpha = (dt/h)\*\*2          self.\_u = np.full((int(self.width/h),int(self.height/h)),0)          self.\_u\_pre = self.\_u.copy()          self.condition = condition          self.g = distorted\_func            #各境界付近のインデックス番号[X,Y]の配列          self.T\_idxes:List[List[int],List[int]] = [list(range(1,self.\_u.shape[0]-1))] + [[0]\*(self.\_u.shape[0]-2)] #上          self.B\_idxes:List[List[int],List[int]] = [list(range(1,self.\_u.shape[0]-1))] + [[self.\_u.shape[1]-1]\*(self.\_u.shape[0]-2)] #下          self.L\_idxes:List[List[int],List[int]] = [[0]\*(self.\_u.shape[1]-2)] + [list(range(1,self.\_u.shape[1]-1))] #左          self.R\_idxes:List[List[int],List[int]] = [[self.\_u.shape[0]-1]\*(self.\_u.shape[1]-2)] + [list(range(1,self.\_u.shape[1]-1))] #右          self.LT\_idxes:List[List[int],List[int]] = [[0],[0]]          self.RT\_idxes:List[List[int],List[int]] = [[self.\_u.shape[0]-1],[0]] #右上角          self.RB\_idxes:List[List[int],List[int]] = [[self.\_u.shape[0]-1],[self.\_u.shape[1]-1]] #右下角          self.LB\_idxes:List[List[int],List[int]] = [[0],[self.\_u.shape[1]-1]] #左下角          #障害物がある場合,障害物の境界付近のインデックス番号[X,Y]を追加する          if border\_vecs is not None:              border\_vecs = np.round(border\_vecs/h).astype(int)              for border\_vec,prop\_grad in zip(border\_vecs,prop\_grads):                  for i,vec in enumerate(border\_vec):                      x1 , y1 = vec[0]                      x2 , y2 = vec[1]                      xmin = x1 if x1 <= x2 else x2                      xmax = x1 if x1 >= x2 else x2                      ymin = y1 if y1 <= y2 else y2                      ymax = y1 if y1 >= y2 else y2                      grad = prop\_grad[i]                      next\_grad = prop\_grad[i+1] if i+1 < len(prop\_grad) else prop\_grad[0]                      prev\_grad = prop\_grad[i-1] if i-1 >= 0 else prop\_grad[-1]                      if x1 == x2:                          #self.\_u[xmin,ymin:ymax+1] = 0                          if grad == 'right':                              if prev\_grad == 'bottom':                                  [[self.R\_idxes[0].append(xmin-1),self.R\_idxes[1].append(y)] for y in range(ymin,ymax-1)]                                  self.RB\_idxes[0].append(xmin-1); self.RB\_idxes[1].append(ymax-1)                              if prev\_grad == 'top':                                  [[self.R\_idxes[0].append(xmin-1),self.R\_idxes[1].append(y)] for y in range(ymin-1 if next\_grad == 'bottom' else ymin+1,ymax+1)]                          if grad == 'left':                              if prev\_grad == 'bottom':                                  [[self.L\_idxes[0].append(xmin),self.L\_idxes[1].append(y)] for y in range(ymin,ymax+1)]                              if prev\_grad == 'top':                                  [[self.L\_idxes[0].append(xmin),self.L\_idxes[1].append(y)] for y in range(ymin-1,ymax)]                      if y1 == y2:                          #self.\_u[xmin:xmax+1,ymin] = 0                          if grad == 'bottom':                              if prev\_grad == 'left':                                  [[self.B\_idxes[0].append(x),self.B\_idxes[1].append(ymin-1)] for x in range(xmin,xmax)]                              if prev\_grad == 'right':                                  [[self.B\_idxes[0].append(x),self.B\_idxes[1].append(ymin-1)] for x in range(xmin,xmax+1 if next\_grad == 'left' else xmax-1)]                          if grad == 'top':                              if prev\_grad == 'left':                                  [[self.T\_idxes[0].append(x),self.T\_idxes[1].append(ymin)] for x in range(xmin,xmax+1)]                              if prev\_grad == 'right':                                  [[self.T\_idxes[0].append(x),self.T\_idxes[1].append(ymin)] for x in range(xmin,xmax-1)]                                  self.RT\_idxes[0].append(xmax-1); self.RT\_idxes[1].append(ymin)            #境界にひずみがある場合、その境界のインデックス番号[X,Y]を取得しておく          if distorted\_vec is not None:              distorted\_vec = np.round(distorted\_vec/h).astype(int)              for vec in distorted\_vec:                  if vec[0][0] == 0:                      init\_y ,end\_y = min(vec[:,1]) , max(vec[:,1])                      self.D\_idxes = [[0]\*(end\_y-init\_y+1)] + [list(range(init\_y,end\_y+1))]                  else:                      init\_x , end\_x = min(vec[:,0]) , max(vec[:,0])                      self.D\_idxes = [list(range(init\_x,end\_x+1))] + [[0]\*(end\_x-init\_x+1)]          else:              self.D\_idxes = []      def plot\_model(self,ax):          for XY,color,label in zip([self.R\_idxes,self.L\_idxes,self.T\_idxes,self.B\_idxes,self.LT\_idxes,self.RT\_idxes,self.RB\_idxes,self.LB\_idxes,self.D\_idxes],\                            ['b','g','r','c','m','m','m','m','k'],                            ['right','left','top','bottom','corner',None,None,None,'input']):              if not XY:                  break              ax.scatter(\*XY,color=color,label=label)              ax.legend(bbox\_to\_anchor=(0.5, -0.5), loc='upper center',ncol=3)              ax.set(xlim=[0,self.\_u.shape[0]],ylim=[0,self.\_u.shape[1]],aspect='equal',xlabel='x grid num',ylabel='y grid num')              ax.invert\_yaxis()      def input\_gauss(self,x0,y0,rad):          x = np.linspace(0,self.width,int(self.width/self.h)).reshape(-1,1)          y = np.linspace(0,self.height,int(self.height/self.h))          z = np.exp(-((x-x0)\*\*2)\*rad\*\*2) \* np.exp(-((y-y0)\*\*2)\*rad\*\*2)          self.\_u = self.\_u + z          if self.time == 0.:              self.\_u\_pre = self.\_u.copy()              self.time = dt      def update(self):          uR = np.roll(self.\_u,-1,1)          uL = np.roll(self.\_u,1,1)          uB = np.roll(self.\_u,-1,0)          uT = np.roll(self.\_u,1,0)          #一旦全ての点を拘束なしの条件でまとめて計算          new\_u = 2\*self.\_u - self.\_u\_pre + self.alpha\*(uL+uR+uB+uT-4\*self.\_u)          if self.condition == 'neumann': #ノイマン境界条件              #左端              X , Y = np.array(self.L\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X+1,Y]+self.\_u[X,Y-1]+self.\_u[X,Y+1]-4\*self.\_u[X,Y])              o\_idxes = X>0              new\_u[X[o\_idxes]-1,Y[o\_idxes]] = 0 #障害物内部に波が侵入しないようにする処理              #上端              X , Y = np.array(self.T\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(self.\_u[X-1,Y]+self.\_u[X+1,Y]+2\*self.\_u[X,Y+1]-4\*self.\_u[X,Y])              o\_idxes = Y>0              new\_u[X[o\_idxes],Y[o\_idxes]-1] = 0 #障害物内部に波が侵入しないようにする処理              #右端              X , Y = np.array(self.R\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X-1,Y]+self.\_u[X,Y-1]+self.\_u[X,Y+1]-4\*self.\_u[X,Y])              o\_idxes = X+1<self.\_u.shape[0]              new\_u[X[o\_idxes]+1,Y[o\_idxes]] = 0 #障害物内部に波が侵入しないようにする処              #下端              X , Y = np.array(self.B\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(self.\_u[X-1,Y]+self.\_u[X+1,Y]+2\*self.\_u[X,Y-1]-4\*self.\_u[X,Y])              o\_idxes = Y+1<self.\_u.shape[1]              new\_u[X[o\_idxes],Y[o\_idxes]+1] = 0 #障害物内部に波が侵入しないようにする処理              #左上端              X , Y = np.array(self.LT\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X+1,Y]+2\*self.\_u[X,Y+1]-4\*self.\_u[X,Y])              o\_idxes\_1 , o\_idxes\_2 = X>0, Y>0              new\_u[X[o\_idxes\_1]-1,Y[o\_idxes\_1]] = 0              new\_u[X[o\_idxes\_2],Y[o\_idxes\_2]-1] = 0 #障害物内部に波が侵入しないようにする処理              #右上              X , Y = np.array(self.RT\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X-1,Y]+2\*self.\_u[X,Y+1]-4\*self.\_u[X,Y])              o\_idxes\_1 , o\_idxes\_2 = X+1<self.\_u.shape[0] , Y>0              new\_u[X[o\_idxes\_1]+1,Y[o\_idxes\_1]] = 0              new\_u[X[o\_idxes\_2],Y[o\_idxes\_2]-1] = 0 #障害物内部に波が侵入しないようにする処理              #右下              X , Y = np.array(self.RB\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X-1,Y]+2\*self.\_u[X,Y-1]-4\*self.\_u[X,Y])              o\_idxes\_1 , o\_idxes\_2 = X+1<self.\_u.shape[0] , Y+1<self.\_u.shape[1]              new\_u[X[o\_idxes\_1]+1,Y[o\_idxes\_1]] = 0              new\_u[X[o\_idxes\_2],Y[o\_idxes\_2]+1] = 0 #障害物内部に波が侵入しないようにする処理              #左下              X , Y = np.array(self.LB\_idxes)              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(2\*self.\_u[X+1,Y]+2\*self.\_u[X,Y-1]-4\*self.\_u[X,Y])              o\_idxes\_1 , o\_idxes\_2 = X>0, Y+1<self.\_u.shape[1]              new\_u[X[o\_idxes\_1]-1,Y[o\_idxes\_1]] = 0              new\_u[X[o\_idxes\_2],Y[o\_idxes\_2]+1] = 0 #障害物内部に波が侵入しないようにする処理          elif self.condition == 'diricre': #ディリクレ境界条件              for XY in (self.L\_idxes,self.R\_idxes,self.B\_idxes,self.T\_idxes,self.LT\_idxes,self.RT\_idxes,self.RB\_idxes,self.LB\_idxes):                  new\_u[XY] = 0          if self.D\_idxes:              X , Y = np.array(self.D\_idxes) #ひずみの境界              new\_u[X,Y] = 2\*self.\_u[X,Y] - self.\_u\_pre[X,Y] + self.alpha\*(self.\_u[X+1,Y]+self.\_u[X,Y+1]+self.\_u[X,Y-1]-4\*self.\_u[X,Y]-2\*self.h\*self.g(X,Y,self.time))          self.\_u\_pre = self.\_u.copy()          self.\_u = new\_u.copy()          self.time += self.dt      @property      def u(self):          return self.\_u.T  if \_\_name\_\_ == '\_\_main\_\_':        def distorted\_func(x:float,y:float,t:float) -> np.ndarray:          f = 3          period = 1/f          return np.cos(2\*np.pi\*f\*t) if  t <= 3\*period else 0        for option,value in zip(['font.family','font.size'],['Times New Roman',20]):          plt.rcParams[option] = value        width = 5 #幅      height = 1 #高さ      obstacle\_height\_1 = 0.4      obstacle\_height\_2 = 0.6      obstacle\_width = 0.8      #障害物のx座標      obstacle\_x = [          width/2 - obstacle\_width/2,          width/2 ,          width/2 ,          width/2 + obstacle\_width/2,          width/2 + obstacle\_width/2,          width/2,          width/2,          width/2 - obstacle\_width/2,          width/2 - obstacle\_width/2,      ]      #障害物のy座標      obstacle\_y = [          height/2 - obstacle\_height\_1/2,          height/2 - obstacle\_height\_1/2,          height/2 - obstacle\_height\_2/2,          height/2 - obstacle\_height\_2/2,          height/2 + obstacle\_height\_2/2,          height/2 + obstacle\_height\_2/2,          height/2 + obstacle\_height\_1/2,          height/2 + obstacle\_height\_1/2,          height/2 - obstacle\_height\_1/2,      ]      #障害物をベクトル表示      obstacle\_vec = np.array([[[(obstacle\_x[i],obstacle\_y[i]),(obstacle\_x[i+1],obstacle\_y[i+1])] for i in range(len(obstacle\_x)-1)]]) #[[(x1,y1),(x2,y2)]]      #obstacle\_vecとセット，obstacle\_vecに垂直で波がぶつかる方向を示す配列 bottom or top or left or right      grad = [['bottom','right','bottom','left','top','right','top','right']]      #ひずみがある境界座標ベクトル      distorted\_vec = np.array([[(0,height/2+0.2),(0,height/2-0.2)]])        h = 0.01 #空間刻み幅      dt = 0.005 #時間刻み      tend = 10 #計測時間      simulator = SimulatePropagation(width, #幅                                      height, #高さ                                      h, #空間刻み                                      dt, #時間刻み                                      obstacle\_vec, #障害物                                      grad, #障害物の向き                                      distorted\_vec, #歪がある境界                                      distorted\_func, #歪の関数                                      condition='neumann' #neumann:自由端反射 , diricre:固定端反射になる                                      )        fig , ax = plt.subplots()      ims = []      while True:          simulator.update() #dtだけ更新          im = ax.imshow(simulator.u,cmap='binary',extent=[0,width,height,0],vmin=-0.01,vmax=0.01)          title = ax.text(0.5, 1.01, f'Time = {round(simulator.time,2)}',                       ha='center', va='bottom',                       transform=ax.transAxes, fontsize='large')          ims.append([im,title])          if simulator.time > tend:              break      ax.plot(obstacle\_x,obstacle\_y,'k')      ax.set(yticks=[0,0.5,1])      anim = animation.ArtistAnimation(fig,ims,interval=30)      fig.colorbar(im,orientation='horizontal')      plt.show() |