本報告では図1に示す領域において，点Aに入力を加えた時における波の挙動を観察する．

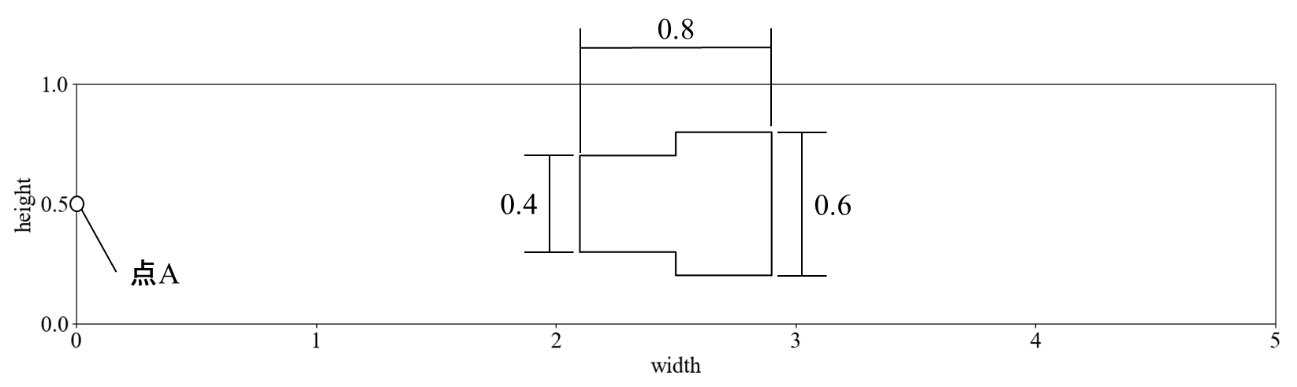


図1 観察領域

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る．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．

|  |
| --- |
|  |
|  |

図2　境界による場合分け

[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．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．

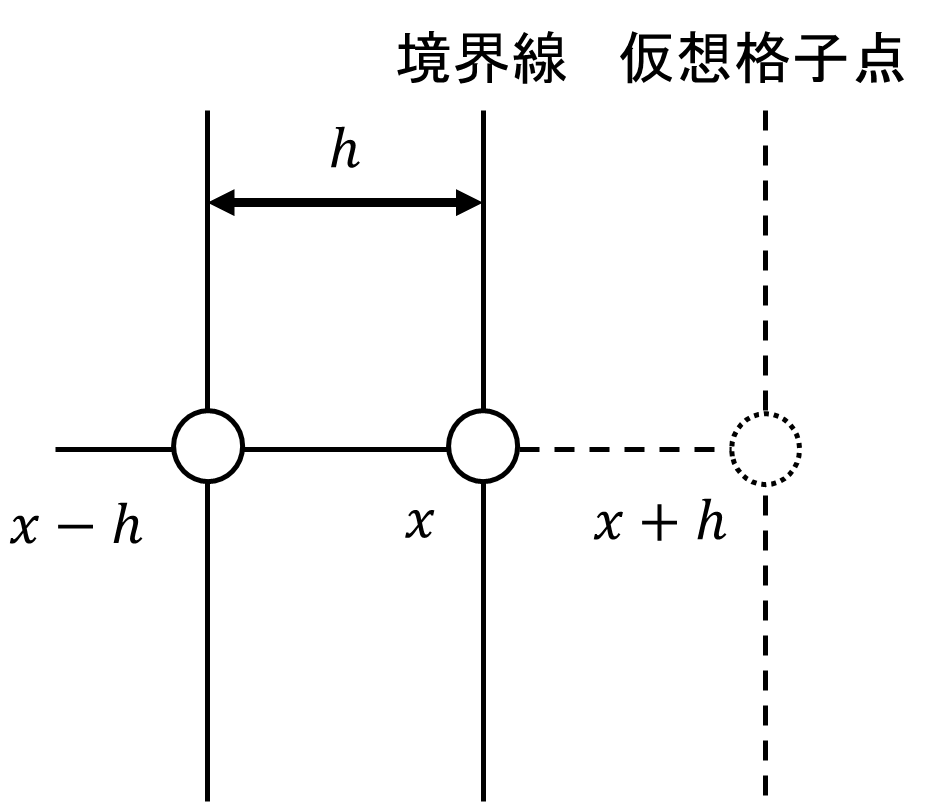


図３　仮想格子点のイメージ

境界上でひずみは

となるため

となる．式(10)を式(6)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，

([2]~[9])の別解

　方向の総格子点数をそれぞれ，とおいたとき，*,*については前進差分，*,*については後退差分すると考えると，

と考えることができる．式(19)~(26)はそれぞれ式(11)~(18)の別解に相当する．今回の解析ではこちらの式を用いることとする.

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 |
| 速度 | 1 [ |
| 解析時間 | 10 [s] |

また，初期条件として点A に2次元ガウス布関数を設置する．

としたときの分布を次の図3に示す．

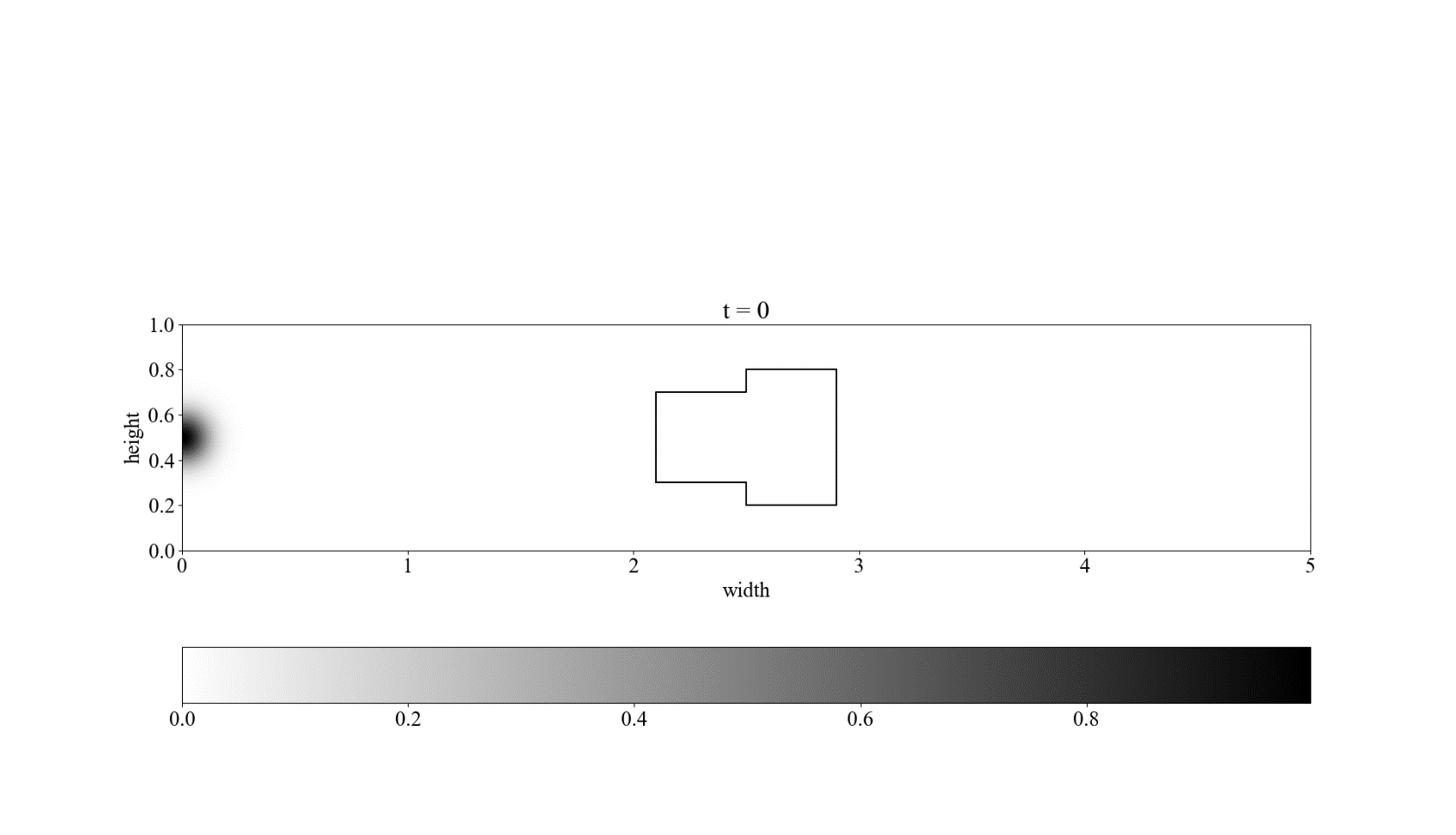


図3　ときのuの分布

そして，0.5秒ごとのの分布を図に示す．

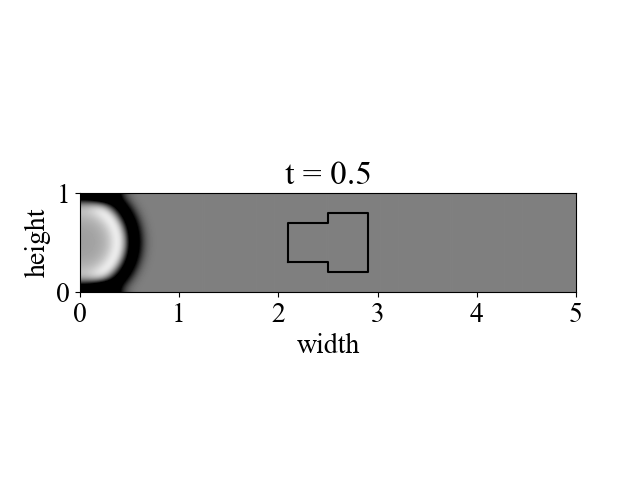


図4 t = 0.5[s]の分布

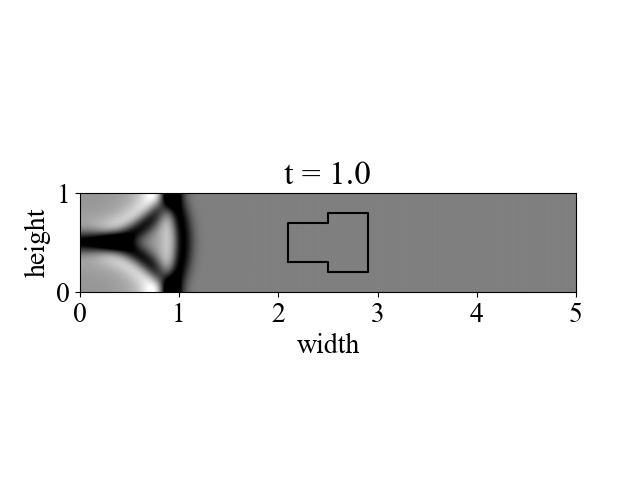


図5 t = 1.0 [s] の分布

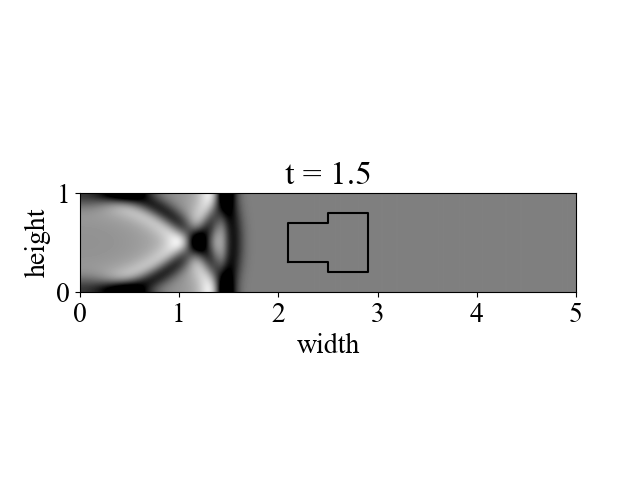


図6 t = 1.5 [s] の分布

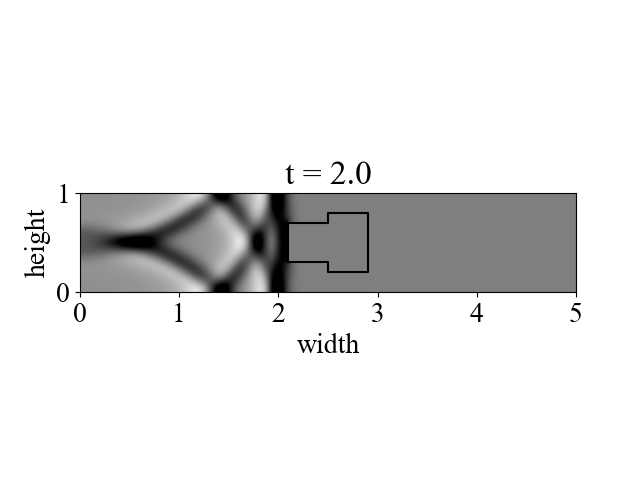


図7 t = 2.0 [s] の分布

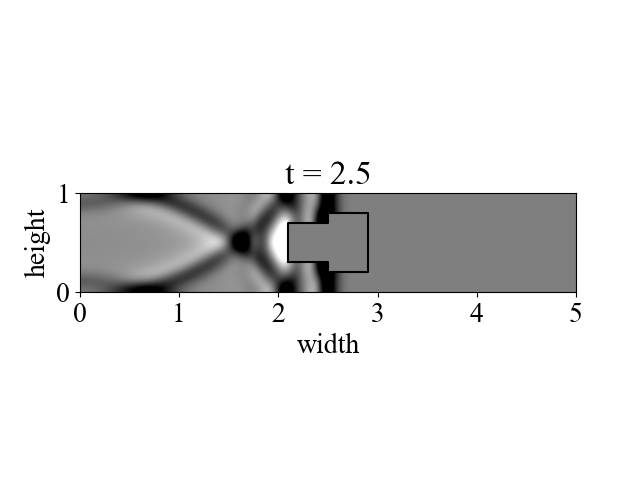


図8 t = 2.5 [s] の分布

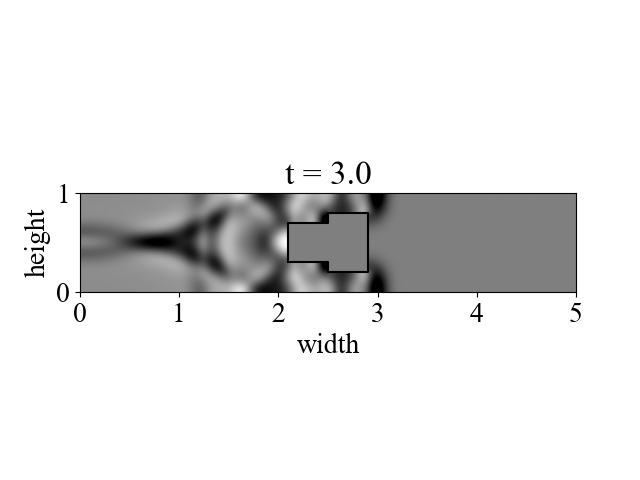


図9 t = 3.0 [s] の分布

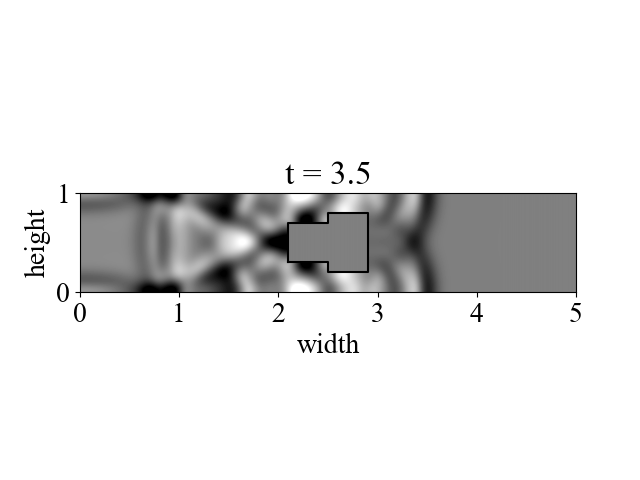


図10 t = 3.5 [s] の分布

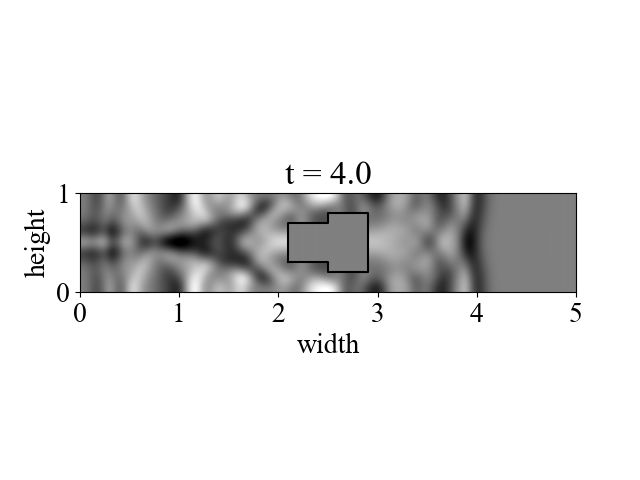


図11 t = 4.0 [s] の分布

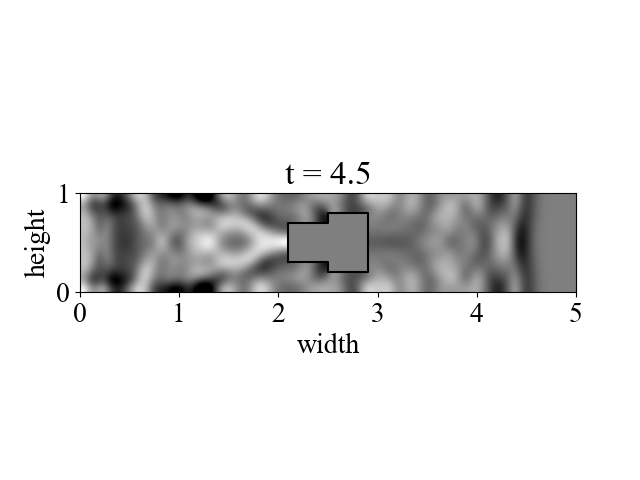


図12 t = 4.5 [s] の分布

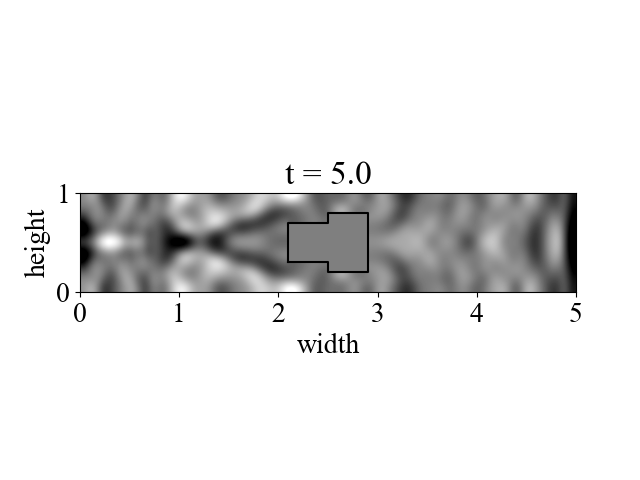


図13 t = 5.0 [s] の分布

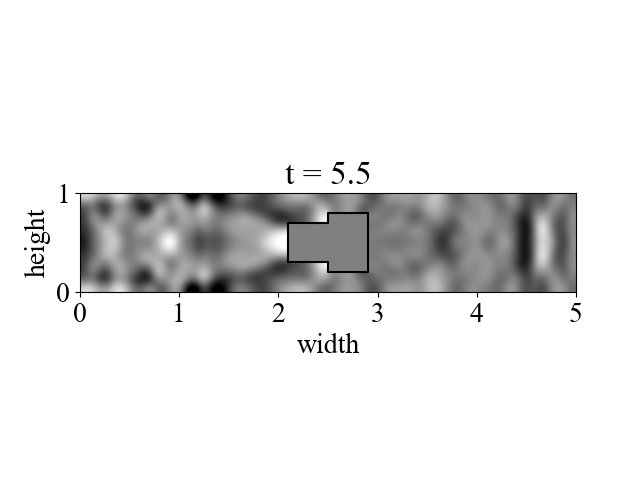


図14 t = 5.5 [s] の分布

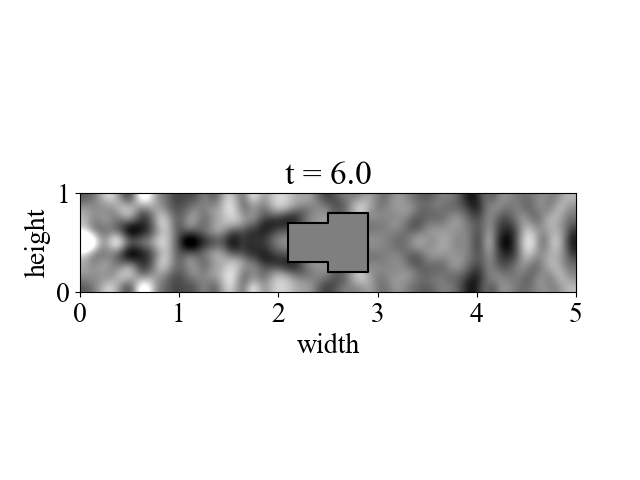


図15 t = 6.0 [s] の分布

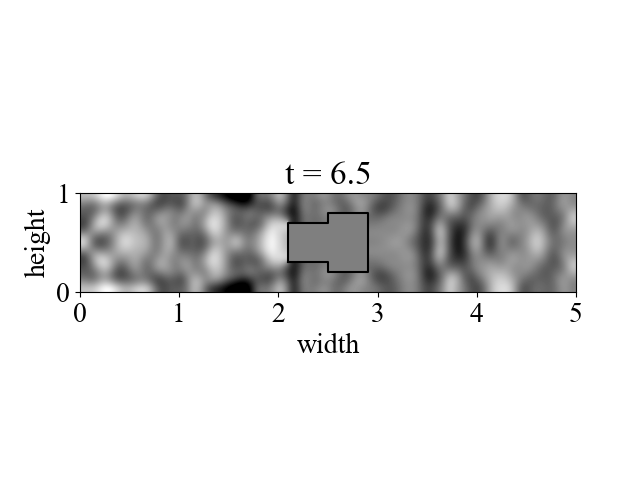


図16 t = 6.5 [s] の分布

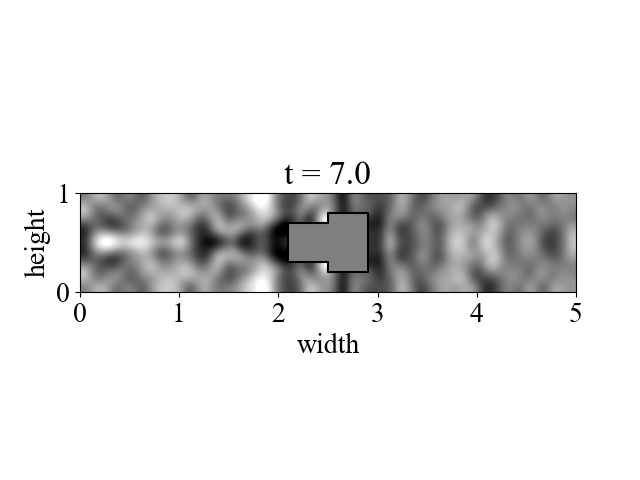


図17 t = 7.0 [s] の分布

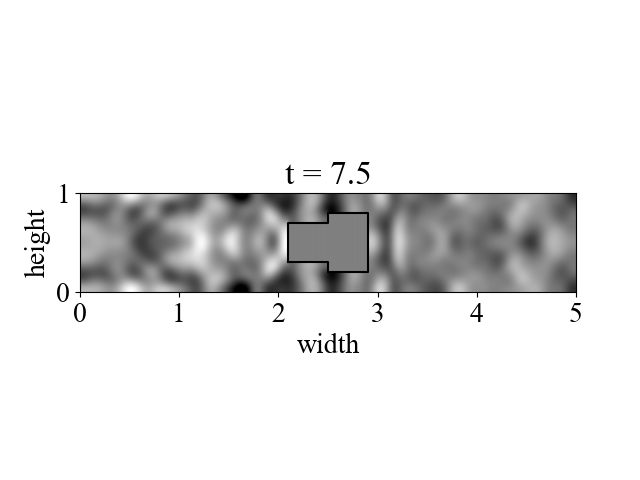


図18 t = 7.5 [s] の分布

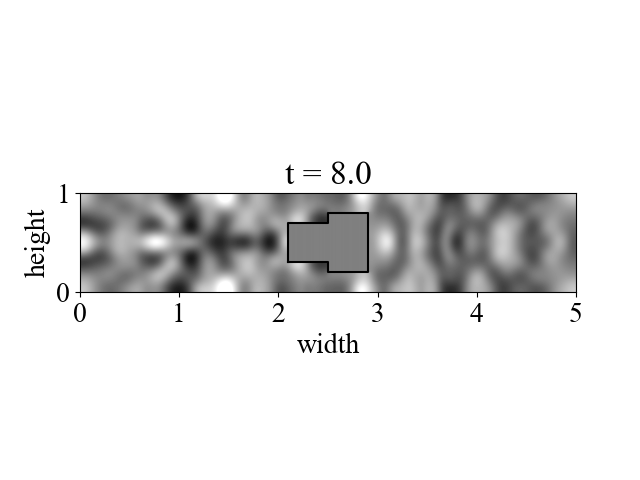


図19 t = 8.0 [s] の分布

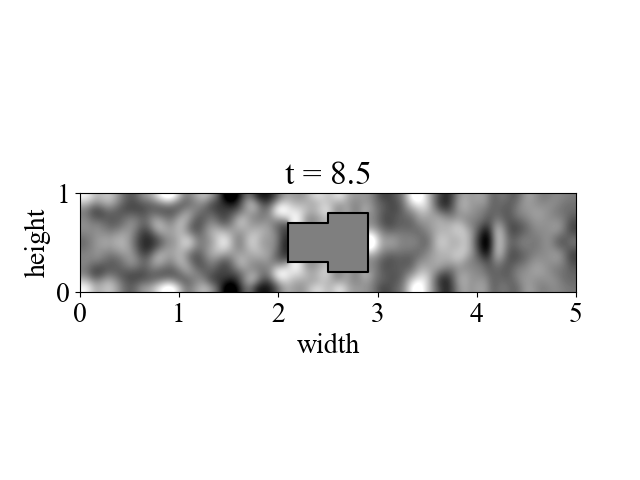


図20 t = 8.5 [s] の分布

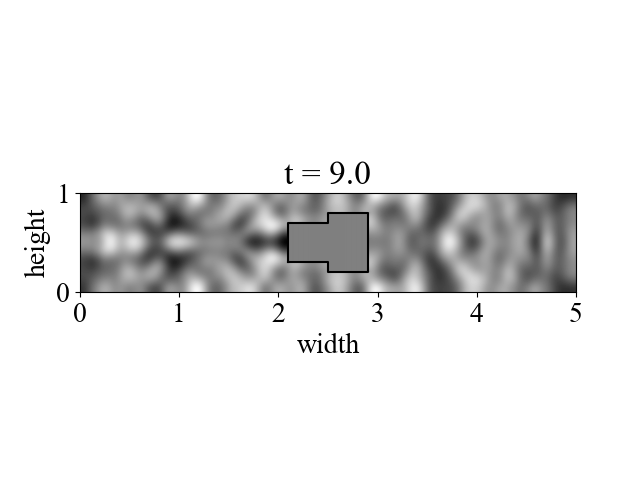


図21 t = 9.0 [s] の分布

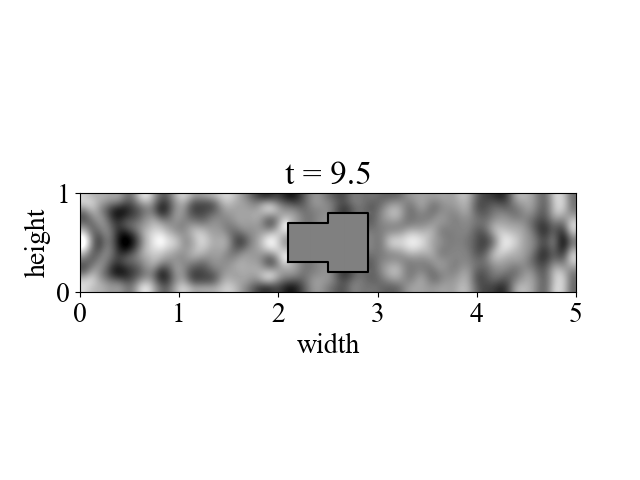


図22 t = 9.5 [s] の分布

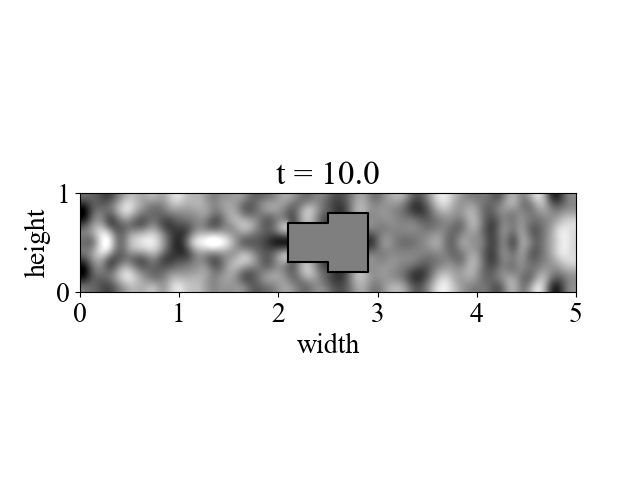


図23 t = 10.0 [s] の分布

また，図3～23の一連の流れをgif画像として記録したものが[図24](https://github.com/kou1046/simulate_propagation/blob/main/%E5%9B%B324.gif) である．また，次に図24を表示するpythonスクリプトを示す．

|  |
| --- |
| import os  from typing import 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\_vec:np.ndarray=None,prop\_grad:List[str]=None,condition='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 = dt          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          #各境界付近のインデックス番号(i,j)の配列          self.T\_idxes = [(x,0) for x in range(1,self.u.shape[0]-1)] #上          self.B\_idxes = [(x,self.u.shape[1]-1) for x in range(1,self.u.shape[0]-1)] #下          self.L\_idxes = [(0,y) for y in range(1,self.u.shape[1]-1)] #左          self.R\_idxes = [(self.u.shape[0]-1,y) for y in range(1,self.u.shape[1]-1)] #右          self.LT\_idxes = [(0,0)] #左上角          self.RT\_idxes = [(self.u.shape[0]-1,0)] #右上角          self.RB\_idxes = [(self.u.shape[0]-1,self.u.shape[1]-1)] #右下角          self.LB\_idxes = [(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(i,j)を追加する          if border\_vec is not None:              border\_vec = [[np.round((xy1)/h).astype(int),np.round((xy2)/h).astype(int)] for xy1,xy2 in border\_vec]              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 += [(xmin-1,y) for y in range(ymin,ymax-1)]                              self.RB\_idxes += [(xmin-1,ymax-1)]                          if prev\_grad == 'top':                              self.R\_idxes += [(xmin-1,y-1) for y in range(ymin if next\_grad == 'bottom' else ymin+2,ymax+1)]                      if grad == 'left':                          if prev\_grad == 'bottom':                              self.L\_idxes += [(xmin,y) for y in range(ymin,ymax+1)]                          if prev\_grad == 'top':                              self.L\_idxes += [(xmin,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 += [(x,ymin-1) for x in range(xmin,xmax)]                          if prev\_grad == 'right':                              self.B\_idxes += [(x,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 += [(x,ymin) for x in range(xmin,xmax+1)]                          if prev\_grad == 'right':                              self.T\_idxes += [(x,ymin) for x in range(xmin,xmax-1)]                              self.RT\_idxes += [(xmax-1,ymin)]      def plot\_model(self,ax):          model = self.u.copy()          for i,XY in enumerate([self.R\_idxes,self.L\_idxes,self.T\_idxes,self.B\_idxes,self.LT\_idxes,self.RT\_idxes,self.RB\_idxes,self.LB\_idxes]):              for x,y in XY:                  model[x,y] = 1 if i ==0 else 2 if i==1 else 3 if i==2 else 4 if i==3 else 5          ax.imshow(model.T,extent=[0,self.width,0,self.height],cmap='jet')      def set\_input(self,x0,y0,kind='gauss'):          def gauss(x0:float,y0:float,rad,X:np.ndarray,Y:np.ndarray) -> np.ndarray:              return np.exp(-((X-x0)\*\*2)\*rad\*\*2) \* np.exp(-((Y-y0)\*\*2)\*rad\*\*2)          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))          if kind == 'gauss':              self.u = self.u + gauss(x0,y0,9,x,y)          self.u\_pre = self.u.copy()      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)          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(自由端反射)                for x,y in self.L\_idxes: #左端は右の波                  new\_u[x,y] = new\_u[x+1,y]                  if x > 0 : #障害物内部に波が侵入しないようにする処理                      new\_u[x-1,y] = 0              for x,y in self.R\_idxes: #右端は左の波                  new\_u[x,y] = new\_u[x-1,y]                  if x+1 < self.u.shape[0]: #障害物内部に波が侵入しないようにする処理                      new\_u[x+1,y] = 0              for x,y in self.T\_idxes: #上端は下の波                  new\_u[x,y] = new\_u[x,y+1]                  if y > 0: #障害物内部に波が侵入しないようにする処理                      new\_u[x,y-1] = 0              for x,y in self.B\_idxes: #下端は上の波                  new\_u[x,y] = new\_u[x,y-1]                  if y+1 < self.u.shape[1]: #障害物内部に波が侵入しないようにする処理                      new\_u[x,y+1] = 0              for x,y in self.LT\_idxes: #左上は右と下の波平均                  new\_u[x,y] = (new\_u[x+1,y] + new\_u[x,y+1]) / 2                  #障害物内部に波が侵入しないようにする処理                  if x > 0 :                      new\_u[x-1,y] = 0                  if y > 0:                      new\_u[x,y-1] = 0              for x,y in self.RT\_idxes: #右上は左と下の波平均                  new\_u[x,y] = (new\_u[x-1,y] + new\_u[x,y+1]) / 2                   #障害物内部に波が侵入しないようにする処理                  if x+1 < self.u.shape[0]:                      new\_u[x+1,y] = 0                  if y > 0:                      new\_u[x,y-1] = 0              for x,y in self.RB\_idxes: #右下は左と上の波平均                  new\_u[x,y] = (new\_u[x-1,y] + new\_u[x,y-1]) / 2                  #障害物内部に波が侵入しないようにする処理                  if x+1 < self.u.shape[0]:                      new\_u[x+1,y] = 0                  if y+1 < self.u.shape[1]:                      new\_u[x,y+1] = 0              for x,y in self.LB\_idxes: #左下は右と上の波平均                  new\_u[x,y] = (new\_u[x+1,y] + new\_u[x,y-1]) / 2                  #障害物内部に波が侵入しないようにする処理                  if x > 0 :                      new\_u[x-1,y] = 0                  if y+1 < self.u.shape[1]:                      new\_u[x,y+1] = 0            if self.condition == 'diricre': #ディリクレ境界条件(固定端反射)              for XY in [self.L\_idxes,self.R\_idxes,self.T\_idxes,self.B\_idxes,self.LT\_idxes,self.RT\_idxes,self.RB\_idxes,self.LB\_idxes]:                  for x,y in XY:                      new\_u[x,y] = 0 #端は全て零          self.u\_pre = self.u.copy()          self.u = new\_u.copy()          self.time += self.dt      def reset(self):          self.u = 0          self.u\_pre = 0          self.time = 0  if \_\_name\_\_ == '\_\_main\_\_':      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']          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, #障害物ベクトル                                      grad, #障害物に当たる直線の波の向き                                      condition='neumann' #newumann:自由端反射 , diricre:固定端反射になる                                      )        fig , ax = plt.subplots()      ax.plot(obstacle\_x,obstacle\_y,'k') #境界線の描画      simulator.set\_input(0,height/2,kind='gauss') #初期値の入力 kind='gauss'でガウス布関数      ims = []        #gif表示      while True:          simulator.update()          im = ax.imshow(simulator.u.T,cmap='binary',vmin=-0.1,vmax=0.1,extent=[0,width,0,height])          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      anim = animation.ArtistAnimation(fig,ims,interval=30)      plt.show()        #png保存      #while True:      #    simulator.update()      #    if np.round(simulator.time\*100) % 50 == 0:      #        fig , ax = plt.subplots()      #      #        ax.imshow(simulator.u.T,cmap='binary',vmin=-0.1,vmax=0.1,extent=[0,width,0,height])      #        ax.set(title=f't = {round(simulator.time,2)}',xlabel='width',ylabel='height')      #        fig.savefig(os.path.join(os.path.dirname(\_\_file\_\_),f'time\_{round(simulator.time,2)}\_propagation.png'))      #    if simulator.time > tend:      #        break |

今回の課題で作成したものはgithubにアップロードしている．

[GitHub - kou1046/simulate\_propagation](https://github.com/kou1046/simulate_propagation)