Finite difference simulations of 2D waves The mathematical model Our 2D PDE is given by  $\frac{\partial^2 u}{\partial t^2} + b \frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left( q(x, y) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( q(x, y) \frac{\partial u}{\partial y} \right) + f(x, y, t)$ with Neumann boundary conditions  $\frac{\partial u}{\partial n} = 0,$ and initial conditions u(x, y, 0) = I(x, y) $u_t(x, y, 0) = V(x, y)$ **Discretization** We discretize the domain such that we have  $N_x + 1$  points in the x-direction,  $N_y + 1$  points in the y-direction and  $N_t$  points in the time domain. Hence we have the spatial mesh  $0 < x_1 < \cdots < x_{N_t} < L_x$  $0 < y_1 < \cdots < y_{N_t} < L_{v}$ for some length L and  $0 = t_0 < t_1 < \cdots < t_{N_t} = T$ for some total time T. **Ghost cells** We anticipate that we'll use ghost cells, such that we let  $i=0,1,\ldots,N_x,N_x+1$ , in the x-direction where i=0 and  $i=N_x+1$  correspond to ghost cells. Similarly, we set  $j=0,1,\ldots,N_y,N_y+1$  in the y-direction where j=0 and  $j=N_y+1$  correspond to ghost cells. This means we'll operate with i=1 and  $i=N_x$  for the boundaries in the x-direction and j=1 and  $j=N_y$  as the boundaries in the ydirection Discretization in the time domain We discretize in the time domain in the following way:  $(u_{tt})_{i,j}^n \approx \frac{u_{i,j}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}}{\Delta t^2},$  $(u_t)_{i,j}^n \approx \frac{u_{i,j}^{n+1} - u_{i,j}^n}{2^{n+1}}$ Discretizing the spatial terms We can discretize the spatial terms in the following. Let's first define  $\phi \equiv q(x, y)u_x$  $\partial_x (q(x, y)u_x) = \phi_x.$ We discretize this with centered difference at x using a stepsize  $h = \Delta x/2$ . We then obtain the centered difference scheme  $(\phi_x)_{i,j}^n \approx \frac{\phi_{i+1/2,j}^n - \phi_{i-1/2,j}^n}{\Lambda}$ where we approximate the two terms using another round of centered difference at  $x_{i\pm 1/2}$  as  $\phi_{i+1/2,j}^n = q_{i+1,j}(u_x)_{i+1/2,j}^n \approx \frac{q_{i+1,j} + q_{i,j}}{2\Lambda x} \left( u_{i+1,j}^n - u_{i,j}^n \right),$ where we approximated  $q_{i+1/2,j} \approx (q_{i+1,j} + q_{i,j})/2$ . A similar argument leads to  $\phi_{i-1/2,j}^n = q_{i-1/2,j}(u_x)_{i-1/2,j}^n \approx \frac{q_{i,j} + q_{i-1,j}}{2 \Lambda r} \left( u_{i,j}^n - u_{i-1,i}^n \right)$ This ultimately lead to the discretization formula  $\left[\partial_x \left(q u_x\right)\right]_{i,j}^n = \frac{(q_{i+1,j} + q_{i,j})(u_{i+1,j}^n - u_{i,j}^n) - (q_{i,j} + q_{i-1,j})(u_{i,j}^n - u_{i-1,j}^n)}{2\Delta x^2}$ Discretization in the y direction leads to an analogue expression:  $\left[\partial_{y}\left(qu_{y}\right)\right]_{i,j}^{n} = \frac{(q_{i,j+1} + q_{i,j})(u_{i,j+1}^{n} - u_{i,j}^{n}) - (q_{i} + q_{i,j-1})(u_{i,j}^{n} - u_{i,j-1}^{n})}{2\Delta v^{2}}$ As a final note to fix our notation, we define  $f(x_i, y_j, t_n) \equiv f_{i,j}^n$ . Formula for the next timestep on the inner mesh points  $u_{i,j}^{n+1}$ Combining all the terms gives  $\frac{u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}}{\frac{1}{2} \sqrt{2}} + b \frac{u_{i,j}^{n+1} - u_{i,j}^{n-1}}{\frac{1}{2} \sqrt{2}} = \frac{(q_{i+1,j} + q_{i,j})(u_{i+1,j}^{n} - u_{i,j}^{n}) - (q_{i,j} + q_{i-1,j})(u_{i,j}^{n} - u_{i-1,j}^{n})}{\frac{1}{2} \sqrt{2}}$  $+ \frac{(q_{i,j+1} + q_{i,j})(u_{i,j+1}^n - u_{i,j}^n) - (q_i + q_{i,j-1})(u_{i,j}^n - u_{i,j-1}^n)}{2\Delta v^2} + f_{i,j}^n$ Which can be rearranged into  $u_{i,j}^{n+1} = A \left[ (b\Delta t - 2)u_{i,j}^{n-1} + 4u_{i,j}^n + C_x^2 \left[ (q_{i+1,j} + q_{i,j})(u_{i+1,j}^n - u_{i,j}^n) - (q_{i,j} + q_{i-1,j})(u_{i,j}^n - u_{i-1,j}^n) \right] \right]$  $+ C_y^2 \left[ (q_{i,j+1} + q_{i,j})(u_{i,j+1}^n - u_{i,j}^n) - (q_{i,j} + q_{i,j-1})(u_{i,j}^n - u_{i,j-1}^n) \right] + 2\Delta t^2 f_{i,j}^n \right].$ with  $A \equiv 1/(1 + 2b\Delta t)$ ,  $C_x \equiv \Delta t/\Delta x$  and  $C_y \equiv \Delta t/\Delta y$ . Imposing boundary conditions Imposing the Neumann boundary conditions  $\partial u/\partial n = 0$  effectively means  $u_x(0, y, t) = 0,$   $u_x(L_x, y, t) = 0,$ and  $u_{\nu}(x, 0, t) = 0,$   $u_{\nu}(x, L_{\nu}, t) = 0$ Using a centered difference scheme at x=0 and  $x=L_{x}$  gives  $(u_x)_{1,j}^n \approx \frac{u_{2,j}^n - u_{0,j}^n}{2\Delta x} = 0, \qquad (u_x)_{N_x,j}^n \approx \frac{u_{N_x+1,j}^n - u_{N_x-1,j}^n}{2\Delta x} = 0,$ meaning  $u_{N_x+1,j}^n = u_{N_x-1,j}^n$ The same argument holds for the boundaries y = 0 and  $y = L_v$  such that  $u_{i,0}^n = u_{i,2}^n$  $u_{i,N,+1}^n = u_{i,N,-1}^n$ At the boundaries, we'll run into trouble with the evaluation of q(x, y) outside the mesh. To circumvent this, we we Taylor expand in space:  $q_{i+1,j} = q_{i,j} + \Delta x (q_x)_{i,j} + \mathcal{O}(\Delta x^2)$ and  $q_{i-1,i} = q_{i,i} - \Delta x (q_x)_{i,i} + \mathcal{O}(\Delta x^2)$ which when added together yields  $q_{i+1,i} + q_{i-1,i} = 2q_{i,i} + \mathcal{O}(\Delta x^2),$ meaning we can approximate q at the boundary i = 1 with Thus we can approximate  $2q_{i,j} \approx q_{i+1,j} + q_{i-1,j}$ , and similarly in the *y*-direction. With this approximation, we can summarize four equations relevant for their respective boundaries boundary:  $q_{0,j} \approx 2q_{1,j} - q_{2,j}, \qquad x = 0,$  $q_{N_x+1,j} \approx 2q_{N_x,j} - q_{N_x-1,j}, \qquad x = L_x, \\ q_{i,0} \approx 2q_{i,1} - q_{i,2}, \qquad y = 0, \\ q_{i,N_y+1} \approx 2q_{i,N_y} - q_{i,N_y-1}, \qquad y = L_y,$ where i = 0 and  $i = N_x + 1$  are ghost cells in the x-direction and j = 0 and  $j = N_y + 1$  are ghost cells in the y-direction. Formula for the first time step  $u_{i,j}^1$ The initial conditions  $u(x, y, 0) = I(x, y) \rightarrow u_{i,j}^{0} = I_{i,j}$ and  $u_t(x, y, 0) = V(x, y) \rightarrow (u_t)_{i,i}^0 = V_{i,j}$ Using centered difference in the time domain, we get  $(u_t)_{i,j}^0 \approx \frac{u_{i,j}^1 - u_{i,j}^{-1}}{2\Delta t} = V_{i,j},$ which gives us  $u_{i,j}^{-1} = u_{i,j}^{1} - 2\Delta t V_{i,j}$ Inserting n = 0 and rearranging gives us  $u_{i,j}^{1} = \frac{1}{4} \left[ 2\Delta t (2 - b\Delta t) V_{ij} + 4u_{i,j}^{0} + C_{x}^{2} \left[ (q_{i+1,j} + q_{i,j}) (u_{i+1,j}^{0} - u_{i,j}^{0}) - (q_{i,j} + q_{i-1,j}) (u_{i,j}^{0} - u_{i-1,j}^{0}) \right] \right]$  $+ C_y^2 \left[ (q_{i,j+1} + q_{i,j})(u_{i,j+1}^0 - u_{i,j}^0) - (q_{i,j} + q_{i,j-1})(u_{i,j}^0 - u_{i,j-1}^0) \right] + 2\Delta t^2 f_{i,j}^0 \, .$ **Verification Constant solution** Assuming that u(x, y, t) = U for some real constant U, we get that  $u_{v} = 0$ ,  $u_{x} = 0$ , and  $u_t = 0, \qquad u_{tt} = 0,$ The generic wave equation PDE may be compactly written as  $u_{tt} + bu_t = \partial_x(qu_x) + \partial_y(qu_y) + f.$ This equation imply that f(x, y, t) = 0 since all other terms are zero. The initial conditions imply that u(x, y, 0) = U = I(x, y), $u_t(x, y, 0) = 0 = V(x, y).$ On "physical" grounds, we must require b=0 since a non-zero damping coefficient will necessarily be inconsistent with a constant solution. We further set q(x, y) = 1. Constant solution using scalar code from scalar\_wavesolver2D import scalar\_wavesolver2D In [1]: import numpy as np In [2]: U = 1. I = lambda x, y: UV =**lambda** x, y: 0.f = lambda x, y, t: 0.q = lambda x, y: 1.In [3]: b = 0. Nx = 10Ny = 10T = 10Lx = 1.Ly = 1.In [4]: | my\_solver = scalar\_wavesolver2D(b, Nx, Ny, Lx, Ly, T) In [5]: my\_solver.set\_function\_conditions(I, V, f, q) my\_solver.set\_conditions() my\_solver.solve() my\_solver.print\_solution() Timeused = 0.1191709041595459u = [[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.][1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]] Constant solution using vectorized code In [6]: from vector\_wavesolver2D import vector\_wavesolver2D In [7]: my\_solver = vector\_wavesolver2D(b, Nx, Ny, Lx, Ly, T) In [8]: my\_solver.set\_function\_conditions(I, V, f, q) my solver.set conditions() my\_solver.solve() my\_solver.print\_solution() [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] In [ ]: Standing wave solution Let  $q(x, y) = q_0 \in \mathbb{R} \ \forall (x, y) \in [0, L_x] \times [0, L_y]$  and f(x, y, t) = 0. A solution set is then given by standing waves  $k_x = \frac{m_x \pi}{L_x}, \qquad k_y = \frac{m_y \pi}{L_y}$  $u(x, y, t) = A\cos(k_x x)\cos(k_y y)\cos(\omega t),$ with  $I(x, y) = u(x, y, 0) = A\cos(k_x x)\cos(k_y y),$ and  $V(x, y) = u_t(x, y, 0) = 0.$ To determine  $\omega$ , we can differentiate u in the simpler PDE  $u_{tt}=c^2(u_{xx}+u_{yy})$ . Clearly,  $u_{tt}=-\omega^2 u$ ,  $u_{xx}=-k_x^2 u$  and  $u_{yy}=-k_y^2 u$ , and inserting into the PDE, we obtain  $\omega = c\sqrt{k_x^2 + k_y^2}$ Convergence rate In the following we test the vectorized solver for convergence rate which is expected to be r=2. In [9]: **from vector wavesolver2D import** vector wavesolver2D from scalar\_wavesolver2D import scalar wavesolver2D import numpy as np def q(x,y): return 1. **def** I(x,y,n=1, m=1, Lx = 1, Ly = 1): return np.cos(n\*np.pi\*x/Lx)\*np.cos(m\*np.pi\*y/Ly) **def** f(x, y, t): return 0 **def** V(x, y): return 0 **def** analytical(x,y,t, n=1, m=1, Lx = 1, Ly = 1): c = 1.omega = np.sqrt((n\*np.pi/Lx)\*\*2 + (m\*np.pi/Ly)\*\*2)\*creturn np.cos(n\*np.pi\*x/Lx)\*np.cos(m\*np.pi\*y/Ly)\*np.cos(omega\*t) In [10]: Lx = 1 Ly = 1Nx = 10Ny = 10T = 1In [11]: | #Compute convergence rate: E = []h = []N = [2\*\*i for i in range(1, 9)]for n in N: Nx = nNy = n $my\_solver = vector\_wavesolver2D(b=0., Nx = Nx, Ny=Ny, Lx = Lx, Ly = Ly, T = T)$  $my_solver.set_function_conditions(I = I, V = V, f = f, q = q)$ my\_solver.set\_conditions() my solver.solve() linf norm, dx = my solver.compute error(analytical) E.append(linf norm) h.append(dx) for i in range (len (E) - 1): r.append(np.log10(E[i+1]/E[i])/np.log10(h[i+1]/h[i]))print(r) [2.864294200331711, 2.2002134929367103, 1.9669160761079238, 2.0367989520050878, 2.0186919282738858,1.9934508410621041, 2.0045646140390505] The test shows that the convergence rate is approximately r = 2. Damped waves with variable velocity Let  $u(x, y, t) = (A\cos\omega t + B\sin\omega t)e^{-dt}\cos(k_x x)\cos(k_y y), \qquad k_x = \frac{m_x \pi}{L_x}, \qquad k_y = \frac{m_y \pi}{L_v}.$ To simplify matters, we set B = 0 such that  $u(x, y, t) = A\cos\omega t e^{-dt}\cos(k_x x)\cos(k_y y),$ and assuming that q(x, y) = y, we obtain  $f(x, y, t) = (k^2 y + k_y \tan(k_y y) - \omega^2 - d^2)u(x, y, t),$ where we've also assumed that b = 2d. Clearly, the expressions for I(x, y) and V(x, y) are easily obtained to be  $I(x, y) = A\cos(k_x x)\cos(k_y y),$ and  $V(x, y) = -Ad\cos(k_x x)\cos(k_y y).$ In [12]: from vector\_wavesolver2D import vector\_wavesolver2D import numpy as np def q(x,y): return y **def** I(x,y,nx=1, my=1, Lx = 1, Ly = 1): kx = nx\*np.pi/Lxky = my\*np.pi/Lyreturn np.cos(kx\*x)\*np.cos(ky\*y) **def** f(x, y, t, nx=1, my=1, omega=1., d=1., Lx = 1, Ly = 1):kx = nx\*np.pi/Lxky = my\*np.pi/Lyk squared = kx\*\*2 + ky\*\*2u = np.cos(omega\*t)\*np.exp(-d\*t)\*np.cos(kx\*x)\*np.cos(ky\*y)return (k\_squared\*y + ky\*np.tan(ky\*y)-2\*omega\*\*2)\*u **def** V(x, y, nx=1, my=1, Lx = 1, Ly = 1, d=1): kx = nx\*np.pi/Lxky = my\*np.pi/Lyreturn -d\*np.cos(kx\*x)\*np.cos(ky\*y) def analytical(x, y, t, nx=1, my=1, omega=1., d=1, Lx = 1, Ly = 1): kx = nx\*np.pi/Lxky = my\*np.pi/Lyk squared = kx\*\*2 + ky\*\*2return np.cos(omega\*t)\*np.exp(-d\*t)\*np.cos(kx\*x)\*np.cos(ky\*y) In [13]: |Lx = 1Ly = 1T = 1#Compute convergence rate: r = []E = []h = []N = [2\*\*i for i in range(1, 8)]print(N) for n in N: Nx = nNy = nprint("n = ", n)my\_solver = vector\_wavesolver2D(b=2., Nx = Nx, Ny=Ny, Lx = Lx, Ly = Ly, T = T)  $my_solver.set_function_conditions(I = I, V = V, f = f, q = q)$ my\_solver.set\_conditions() my solver.solve() #my\_solver.plot\_solution(analytical) #linf\_norm, dx = my\_solver.compute\_error(analytical) linf\_norm, dx = my\_solver.compute\_error(analytical) #my solver.plot solution(analytical) E.append(linf\_norm) h.append(dx) for i in range (len (E) - 1): r.append(np.log10(E[i+1]/E[i])/np.log10(h[i+1]/h[i]))print(r) [2, 4, 8, 16, 32, 64, 128] n = 2n = 4n = 8n = 16n = 32n = 64n = 128[2.5337057950642277, 1.408729101409308, 1.967827379302148, 1.980274832254245, 1.986922921578017, 1.99 5906926550047] So clearly, again, the convergence rate is approximately found to be r=2. Investigating a physical problem In the following we simulate a wave for several choices of bottom surface B(x, y) and one initial wave configuration I(x, y).

my_so my_so my_so 0.8 - 0.6 - > 0.4 -	Approxima		- 1.80 - 1.65 - 1.50 - 1.35 - 1.20			
0.0 + 0.0 1.0 - 0.8 - 0.6 - >	0.2 0.4 x Approxima		1.05 0.90 1.0 1.8 1.7 1.6 1.5 1.4			
0.4 - 0.2 - 0.0 - 0.0 1.0 - 0.8 -	0.2 0.4 x		-1.3 -1.2 -1.1 -1.0 1.0			
0.4 - 0.2 - 0.0 - 0.0 - 0.0 - 0.0 - 0.8 -	0.2 0.4 x		-1.28 -1.20 -1.12 -1.04 -0.96			
0.6 - 0.4 - 0.2 - 0.0 + 0.0	0.2 0.4 x		-1.36 -1.28 -1.20 -1.12 -1.04 -0.96			
0.8 - 0.6 - > 0.4 - 0.2 - 0.0 - 0.0	0.2 0.4		1.52 -1.44 -1.36 -1.28 -1.20 -1.12 -1.04 -0.96			
0.8 - 0.6 - > 0.4 -	Approxima		1.52 -1.44 -1.36 -1.28 -1.20 -1.12 -1.04			
0.0 + 0.0 + 0.6 + 0.6 + 0.4 +	0.2 0.4 x Approxima		1.04 0.96 1.0 1.52 - 1.44 - 1.36 - 1.28 - 1.20			
0.2 -	0.2 0.4 x Approxima		1.12 -1.04 -0.96 1.0 -1.52 -1.44 -1.36 -1.28			
0.4 - 0.2 - 0.0 - 0.0 - 0.0	0.2 0.4 x Approxima		1.12 1.04 0.96			
0.6 -	0.2 0.4 x		-136 -128 -120 -112 -104 -0.96			
0.8 - 0.6 - 0.4 - 0.2 - 0.0 0.0	0.2 0.4 x	0.6 0.8	-17 -16 -15 -14 -13 -12 -11 -10 -0.9			
0.8 - 0.6 - > 0.4 -	Approxima	ation	2.10 -1.95 -1.80 -1.65 -1.50 -1.35 -1.20 -1.05			
0.0	0.2 0.4 x Approxima		1.0			
0.2 - 0.0 - 0.0 - 0.0 - 0.6 - 0	0.2 0.4 x Approxima		1.05 1.05 0.90 1.0 1.52 -1.44 -1.36 -1.28			
0.4 - 0.2 - 0.0 - 0.0 - 0.0 - 0.8 - 0	0.2 0.4 x Approxima		1.20 -1.12 -1.04 -0.96			
0.6 - 0.4 - 0.2 - 0.0 - 0.0	0.2 0.4 x Approxima		-1.36 -1.28 -1.20 -1.12 -1.04 -0.96			
0.8 - 0.6 - >- 0.4 - 0.2 -	х		-1.44 -1.36 -1.28 -1.20 -1.12 -1.04 -0.96			
0.8 - 0.6 - > 0.4 - 0.2 -	Approxima		1.52 -1.44 -1.36 -1.28 -1.20 -1.12 -1.04 -0.96			
0.0 + 0.0 - 0.8 - 0.6 - > 0.4 -	0.2 0.4 x Approxima		1.0			
0.2 - 0.0 + 0.0 - 0.8 - 0.6 -	0.2 0.4 x		1.04 0.96 1.0 1.52 -1.44 -1.36 -1.28			
0.4 -	0.2 0.4 x		1.20 -1.12 -1.04 -0.96 1.0 -1.52 -1.44 -1.36			
0.6 - > 0.4 - 0.2 - 0.0 - 0.0	0.2 0.4 x		-128 -120 -112 -104 -0.96			
0.8 - 0.6 - >- 0.4 - 0.2 -	х		-16 -15 -14 -13 -12 -11 -10 -0.9			
0.0 + 0.0	Approxima	ation	2.10 - 1.95 - 1.80 - 1.65			
0.8 - 0.6 - 0.4 - 0.2 - 0.0 -	0.2 0.4  (, 10 = 1, Ia = n 10 + Ia*np.e:  n 0.  (,t): n 0.  (,g=9.81, H0 = n g*(H0 - B(x,;  , B0=0., Ba = <= np.sqrt(x**: eturn B0 + Ba*: eturn B0  ctorize(B)	<pre>1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) &lt;= np.cos(np.pi*  in range(21)] wavesolver2D( ion_condition tions() tion(analytic</pre>	(b=b, Nx = Nx sis(I = I, V)	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 -  0.0 -  0.0 -  def I(x,y retur def (x,y	0.2	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions() tion(analytic  ation  06 08	135 120 105 0.90 10 10 10 10 10 10 10 10 10 1	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 -  0.6 f I(x,y)  def I(x,y)  def Q(x,y)  def Q(x,y)  def Q(x,y)  def B(x,y)  else:  r  s = np.ve  Ix = 50  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 1.  Ix = 50  Ix = 50  Ix = 1.  I	0.2	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D( ion_condition tions() tion(analytic  ation  0.6 0.8	1.35 1.20 1.05 0.90 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 0  def I(x,y)  def (x,y)  d	0.2	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_conditions()) tion(analytic  ation  0.6 0.8	-1.35 -1.20 -1.05 -0.90 -1.0 -1.05 -0.90 -1.0 -1.05 -0.90 -1.0 -1.05 -1.090 -1.05 -1	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 -  0.0 -  0.0 -  0.1 -  0.2 -  0.4 -  0.5 -  0.4 -  0.5 -  0.5 -  0.5 -  0.6 -  0.7 -  0.8 -  0.9 -  0.0 -  0.	0.2	1., Im = 0.5 xp(-((x-Im)/I)  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions() tion(analytic  ation  0.6 0.8  ation  0.6 0.8	1.35 1.20 1.05 1.090 1.0 1.15 1.090 1.0 1.15 1.0 1.15 1.10 1.10 1.10 1.10	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 0.0  def I(x,y)  def Y(x,y)  def Y(x,y)  def Q(x,y)	0.2 0.4 x  , I0 = 1, Ia = n I0 + Ia*np.e: ): n 0. , t): n 0. , g=9.81, H0 = n g*(H0 - B(x, y, B0=0., Ba = <= np.sqrt(x**x, eturn B0 + Ba*x; eturn B0 ctorize(B)  0.005*i for i times: lver = vector_lver.set_functilver.set_condilver.set_vector_lver.set_functilver.set_vector_lver.set_lv	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions()) tion(analytic stion  0.6	1.35 1.20 1.05 1.090 1.0 1.15 1.090 1.0 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.15 1.10 1.10	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	0.2 0.4	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions()) tion (analytic tion)  0.6	b=b, Nx = N (2	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0  0.8  0.6  0.4  0.2  0.0  0.0  def I(x,y)  def Q(x,y)  def Q(x,y)  def A(x,y)  def A(x	0.2 0.4 x  , IO = 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia = n IO + Ia*np.e:  ): n 0. , d= 1, Ia*np.e:  ]: n 0. , d= 1, Ia*	1., Im = 0.5 xp(-(x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions() tion(analytic tion)  0.6 0.8  ation  0.6 0.8  ation  0.6 0.8  ation	1.35 1.20 1.05 1.09 1.0 1.15 1.09 1.0 1.15 1.0 1.15 1.10 1.0 1.15 1.10 1.0 1.15 1.10 1.0 1.10 1.0 1.10 1.0 1.10 1.0 1.10 1.0 1.	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -	0.2 0.4 x  , IO = 1, Ia = n	1., Im = 0.5 xp(-(x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions()) tion (analytic tion)  1., Im = 0.5 xp(-(x-Im)/I  1., Im	135	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -  0.8 -  0.6 -  0.4 -  0.2 -  0.0 -  0.6 f (x, y)  def (x, y)  retur  def (x, y)  re	0.2 0.4 x  , IO = 1, Ia = n IO + Ia*np.e; ); n O. , t): n O. , g=9.81, HO = n g*(HO - B(x, y, BO=0., Ba = 0.0); eturn BO + Ba*; eturn BO + Ba*; eturn BO + Ba*; lver = vector_iver.set_functliver.set_condiver.solve() lver.plot_soluver.plot_s	1., Im = 0.5 xp(-((x-Im)/I)  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions()) tion(analyticutions()) tion(analyticutions()) tion(analyticutions()) tion(analyticutions()) 0.6	1.35	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0  0.8  0.6  0.4  0.2  0.6  1.0  1.0  1.0  1.0  1.0  1.0  1.0	0.2 0.4 x  , IO = 1, Ia = n 10 + Ia*np.e; ): n 0. , r): n 0. , g=9.81, H0 = m g*(H0 - B(x); , B0=0., Bat = steturn B0 + Ba*; eturn B0 + Ba*; eturn B0 ctorize(B)  0.005*i for i itimes; eturn B0 ctorize(B)  0.005*i for i itimes; liver = vector, liver.set_condi; l	1., Im = 0.5 xp(-((x-Im)/I)  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(ion_conditions()) tion(analyticutions()) tion(analyticutions)  0.6	1.35	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0	0.2 0.4 x  , IO = 1, Ia = n 10 + Ia*np.e; ): n 0. , r): n 0. , g=9.81, H0 = m g*(H0 - B(x); , B0=0., Bat = steturn B0 + Ba*; eturn B0 + Ba*; eturn B0 ctorize(B)  0.005*i for i itimes; eturn B0 ctorize(B)  0.005*i for i itimes; liver = vector, liver.set_condi; l	1., Im = 0.5 xp(-((x-Im)/I)  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D(itions(on) tion(analyticutions(on) tion(analyticution) 0.6 0.8  ation  0.6 0.8  ation  0.6 0.8  ation  0.6 0.8  ation	1.35	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0	0.2 0.4 x  , IO = 1, Ia = n IO + Ia*np.e. ): n O. , t): n O. , g=9.81, HO = g (HO - B(x, y)) , BO=0., Ba = <= n p. sqrt (x**x**x**x**x**x**x**x**x**x**x**x**x**	1., Im = 0.5 xp(-((x-Im)/I  10): y)) 9., Bmx = 0.5 2. + y*2) <= inp.cos(np.pi*  in range(21)] wavesolver2D(ion_condition tions()) tion (analytic tion)  0.6		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -	0.2 0.4 x  , 10 = 1, 1a = n 10 + Ia*np.e: ): n 0. , g=9.81, H0 = n 0. , g=9.81, H0 = set	1., Im = 0.5 xp(-((x-Im)/I)  10): y)) 9., Bmx = 0.5 2 + y**2) <= np.cos(np.pi*  in range(21)] wavesolver2D( inions()  inion(analytic office of the cost of the cos	belon, Nx = None  (x - Bmx) / (2  belon,	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0 -	0.2 0.4 x  n 10 = 1, Ia = e n 10 + Ia*np.es ): n 0. , t): n 0. , g=9.81, H0 = e n g*(H0 - B(x); , B0=0., Ba = e n seturn B0 ctorize(B)  0.005*i for i times: lever set_condi lver.set_condi lver.set_condi lver.set_condi lver.set_condi lver.solve() lver.plot_solve  0.2 0.4 x  Approxima	1., Im = 0.5 xp(-(x-Im)/I  10): y)) 9., Bmx = 0.5 2 + y**2) <= inp.cos(np.pi*  in range(21)] wavesolver20(ion_condition tion() tion(analytic tion  0.6 0.8  ation		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0	0.2 0.4 x  n 10 = 1, Ia = n 10 + Ia*np.e:  n 0.	10, Im = 0.5 xp(-(x-Im)/I  10): y), Bmx = 0.5 yp, Cy 2), pix  in range(21)] invocondition invocondit		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
1.0	02 04 x  , 10 = 1, 1a = n in + 1a + np. e. ) 1 0. , 10 = 0. , 10 = 1, 1a = n in + 1a + np. e. ) 1 0. , 2 = 0.4   Satisfies   eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Condition   condition	10, Im = 0.5 xp(-(x-Im)/I xp(-(x-Im)/I y); y); y), yo, Bmx = 0.5 xp, pi*  in range(21); wavesolver20( ion_condition tions() tion (analytic tion  06		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02 04 x  , 10 = 1, 1a = n in + 1a + np. e. ) 1 0. , 10 = 0. , 10 = 1, 1a = n in + 1a + np. e. ) 1 0. , 2 = 0.4   Satisfies   eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Ba*: eturn B0 + Condition   condition	10, Im = 0.5 xp(-((x-Im)/I  10): y1): y2): y3.		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02	11., Im = 0.5 xp(-((x-Im)/I) 10): y1): y2, Bmx = 0.5 y2 + y*2 <= np.cos (np.pi*) in range (21) ] wave colverizion ticon (analytic ticon (analy	b=b, Nx = N	<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02 04 x  , IC = 1, Ia	1., Im = 0.5 xp(-((x-Im)/I)  10): yp) el, Bmx = 0.5 2n + y**2) = p yn) os (np.pi*  in range(21)   wavesolver20 in condition tion (nallytic tion  06		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	10	in range (21) ]  y) ,		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02	1., Im = 0.5  xp(-((x-Im)/I  10):		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02	1., Im = 0.5  xp(-((x-Im)/I  10): yp) yp) yp, Pmx = 0.5 2.		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	
10	02	1., Im = 0.5 xp(-((x-Im)/I  10): y1); y2, Emx = 0.5 2.		<pre>, Bs = 1.): *Bs)))*np.cos x, Ny = Ny, Example 1.</pre>	ux = Lx, Ly =	