MEK4350

Oblig 2

The finite element method applied to dispersive wave propagation in channels

Greger Svenn

May 8, 2016

Introduction

As a second mandatory exercise in MEK4250 I've been granted a custom exercise, closely related to my master thesis and the course in hydrodynamic wave theory.
Some of the text is mostly for my own learning and might be redundant.

The PDEs

Dispersive propagation of long waves on constant depth may be described by the Bernoulli equation

$$\frac{\partial \phi}{\partial t} + \epsilon \frac{1}{2}(\nabla \phi)^2 + \eta - \mu^2 \frac{1}{3} \nabla^2 \frac{\partial \phi}{\partial t} = 0 \tag{1}$$

and the depth integrated continuity equation

$$\frac{\partial \eta}{\partial t} = -\nabla \cdot ((1 + \epsilon \eta) \nabla \phi) \tag{2}$$

where ∇ is the horizontal part of the gradient operator. Here η i the surface elevation and ϕ is the velocity potential which relate to the depth averaged velocity, $\mathbf{v}$, according to

$$\nabla \phi = 0$$

The equations (1) and (2) are made dimensionless by using the equilibrium depth, H, as vertical length scale, a typical wavelength, λ_c, as typical horizontal length scale and $t_c = \lambda_c / \sqrt{gH}$ as time scale. In addition, $\mu = H/\lambda_c$ and an amplitude factor, ϵ, is factorized off the surface elevation.

The numerical method

Residual Formulation

We can assume ϵ as small and linearize equations (1) and (2). This assumption is made because the amplitude is small compared to the wavelength.

Now our PDEs are reduced to the residuals

$$R_1 = \frac{\partial \phi}{\partial t} + \eta - \mu^2 \frac{1}{3} \nabla^2 \frac{\partial \phi}{\partial t} = 0 \tag{3}$$

and

$$R_2 = \frac{\partial \eta}{\partial t} + \nabla^2 \phi = 0 \tag{4}$$

A generalization of Galerkin method is to demand that R is orthogonal to some space Q but not necessarily the same space as V where we seek our unknown functions.

$$(R, v) = 0 \qquad \forall v \in Q \tag{5}$$

If $\{q_1, ..., q_N\}$ is a basis for Q, we can express the method of weighted residuals as

$$(R, q_i) = 0 \qquad i \in I_s \tag{6}$$

The result is N equations for $\{c_i\}_{i \in I_s}$, with I_s as a index set $\{1, ..., N\}$.

Introduce a continuous weighted function $W(x, y)$, the residual formulation of (3) becomes

$$\begin{aligned}
(R_1, W) &= 0 \\
(\frac{\partial \phi}{\partial t} + \eta - \mu^2 \frac{1}{3} \nabla^2 \frac{\partial \phi}{\partial t}, W) &= 0 \\
(\frac{\partial \phi}{\partial t}, W) + (\eta, W) - (\mu^2 \frac{1}{3} \nabla^2 \frac{\partial \phi}{\partial t}, W) &= 0 \\
\int_\Omega \frac{\partial \phi}{\partial t} W d\Omega + \int_\Omega \eta W d\Omega - \mu^2 \frac{1}{3} \int_\Omega \nabla^2 \frac{\partial \phi}{\partial t} W d\Omega &= 0
\end{aligned}$$

Integration by parts on the laplacian term reveals the final residual form

$$\int_\Omega (\frac{\partial \phi}{\partial t} W + \eta W + \mu^2 \frac{1}{3} \nabla \frac{\partial \phi}{\partial t} \cdot \nabla W) d\Omega - \mu^2 \frac{1}{3} \int_\Gamma W \frac{\partial^2 \phi}{\partial n \partial t} ds = 0 \tag{7}$$

(4) becomes

$$\begin{aligned}(R_2, W) &= 0 \\ (\frac{\partial \eta}{\partial t} + \nabla^2 \phi, W) &= 0\end{aligned} \tag{8}$$

Again doing integration by parts on the laplacian term, we get

$$\int_\Omega (\frac{\partial \eta}{\partial t} W - \nabla \phi \cdot \nabla W) d\Omega + \int_\Gamma W \frac{\partial \phi}{\partial n} ds = 0 \tag{9}$$

Where Γ is on the boundary.

So if we have no-flux conditions at the boundaries, $\frac{\partial \phi}{\partial n} = 0$, the initial conditions becomes the linear contribution. Else the specified normal velocity on the boundary becomes is added to the linear term.

FEM discretizetion

For both ϕ and η we employ trial functions N_i which are continuous, linear on each element, unity on node i and zero on the remaining nodes.

$$\begin{aligned}\phi &\approx \sum_{i=1}^{nm} \phi_i(t) N_i(x, y) \\ \eta &\approx \sum_{i=1}^{nm} \eta_i(t) N_i(x, y)\end{aligned} \tag{10}$$

N_i is also used as weighted function W.

Starting with a simple geometry ie. a rectangle Ω: $0 \leq x \leq L$ and $0 \leq y \leq B$. If we divide Ω in uniform rectangles of size $h_x \times h_y$, where $h_x = L/n$ and $h_y = B/m$. m and n is the total number of nodes in x-direction and y-direction respectively.
We can draw the dicretization of Ω as

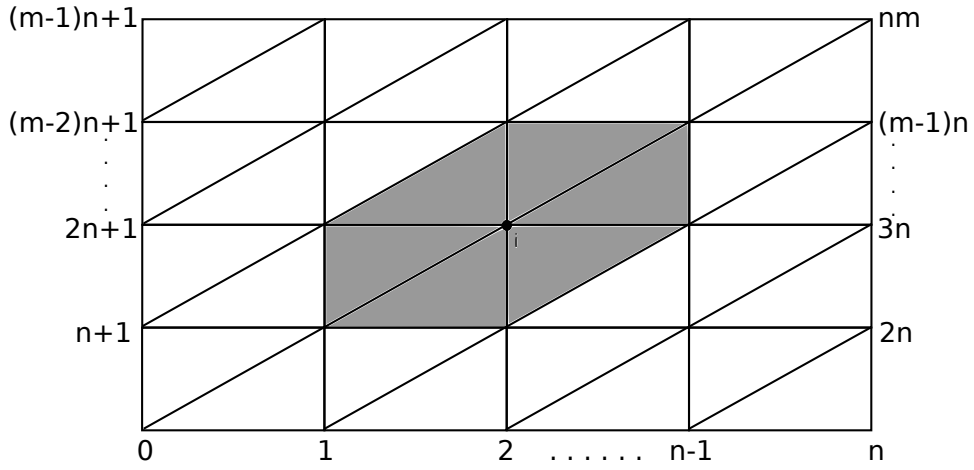


Figure 1: Elements and nodes for the rectangular geometry

Furthermore the sketch of N_i in the interior node i above looks like

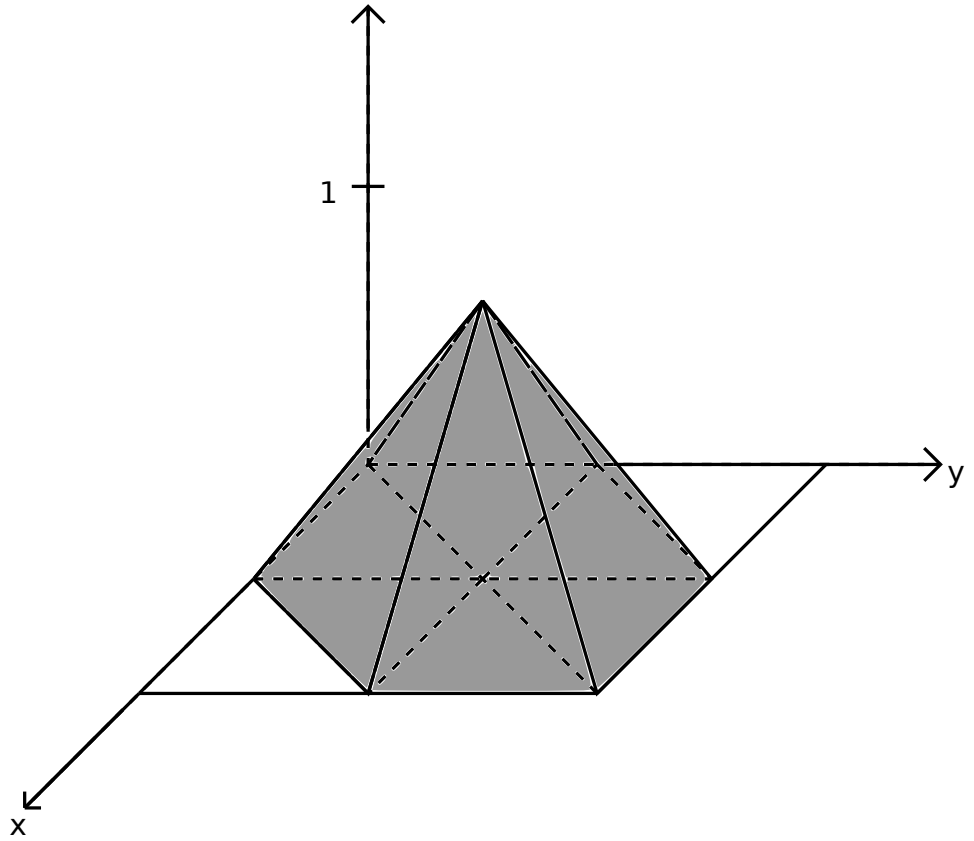


Figure 2: Sketch of N_i

Now lets plug the trial functions and test function into the variational

form and use the no-flux boundary condition.

$$\int_\Omega \left(N_i \sum_{i=1}^{nm} \dot{\phi}_i N_i + N_i \sum_{i=1}^{nm} \eta_i N_i + \frac{\mu^2}{3} \nabla (\sum_{i=1}^{nm} \dot{\phi}_i N_i) \cdot \nabla N_i \right) d\Omega = 0 \qquad (11)$$

$$\text{I} \qquad\qquad \text{II} \qquad\qquad \text{III}$$

$$\begin{aligned} \text{I}: \quad & \int_\Omega (\dot{\phi}_1 N_1 + \dot{\phi}_2 N_2 + \cdots + \dot{\phi}_k N_k) N_1 d\Omega \\ & \int_\Omega (\dot{\phi}_1 N_1 + \dot{\phi}_2 N_2 + \cdots + \dot{\phi}_k N_k) N_2 d\Omega \\ & \qquad \vdots \\ & \int_\Omega (\dot{\phi}_1 N_1 + \dot{\phi}_2 N_2 + \cdots + \dot{\phi}_k N_k) N_k d\Omega \end{aligned}$$

This gives us k functions and k unknowns. This can be written in a more compact form using a $k \times k$ mass matrix, M, and array with length k, $\dot{\mathbf{\Phi}}$ $(\{\dot{\phi}_1, \dot{\phi}_2, \cdots, \dot{\phi}_k\})$. The dot denotes time derivative.

$$\begin{aligned} \text{II}: \quad & \int_\Omega (\eta_1 N_1 + \eta_2 N_2 + \cdots + \eta_k N_k) N_1 d\Omega \\ & \int_\Omega (\eta_1 N_1 + \eta_2 N_2 + \cdots + \eta_k N_k) N_2 d\Omega \\ & \qquad \vdots \\ & \int_\Omega (\eta_1 N_1 + \eta_2 N_2 + \cdots + \eta_k N_k) N_k d\Omega \end{aligned}$$

Again we may write this as the $k \times k$ mass matrix, M, and an array with length k, $\mathbf{Y}$ $(\{\eta_1, \eta_2, \cdots, \eta_k\})$.

$$\begin{aligned} \text{III}: \quad & \int_\Omega (\dot{\phi}_1 \nabla N_1 + \dot{\phi}_2 \nabla N_2 + \cdots + \dot{\phi}_k \nabla N_k) \cdot \nabla N_1 d\Omega \\ & \int_\Omega (\dot{\phi}_1 \nabla N_1 + \dot{\phi}_2 \nabla N_2 + \cdots + \dot{\phi}_k \nabla N_k) \cdot \nabla N_2 d\Omega \\ & \qquad \vdots \\ & \int_\Omega (\dot{\phi}_1 \nabla N_1 + \dot{\phi}_2 \nabla N_2 + \cdots + \dot{\phi}_k \nabla N_k) \cdot \nabla N_k d\Omega \end{aligned}$$

III gives us the array $\dot{\mathbf{\Phi}}$ and a stiffness matrix K.

We may now write (11) in a more compact form

$$(M + K)\dot{\mathbf{\Phi}} + M\mathbf{Y} = 0 \qquad (12)$$

Similar with (9)

$$\int_\Omega \left(N_i \sum_{i=1}^{nm} \dot{\eta} N_i - (\sum_{i=1}^{nm} \eta \nabla N_i) \cdot \nabla N_i \right) d\Omega = 0 \tag{13}$$

Which gives us

$$M\dot{\mathbf{Y}} - K\mathbf{\Phi} = 0 \tag{14}$$

Temporal discretization

We use finite difference discertization in time and specify the potential and elevation in a staggered temporal grid where

$$\frac{\partial \phi_i}{\partial t} \to \frac{\phi_i^{n+1} - \phi_i^n}{\Delta t}, \qquad \eta_i \to \eta_i^{n+1/2}$$
$$\frac{\partial \eta_i}{\partial t} \to \frac{\eta_i^{n+1/2} - \phi_i^{n-1/2}}{\Delta t}, \qquad \phi_i \to \phi_i^n$$

Since the midpoint discretization $\frac{\eta^1 - \eta^0}{\Delta t}$ is at $t = 1/2$ where the initial surface elevation is located, we only have one unknown

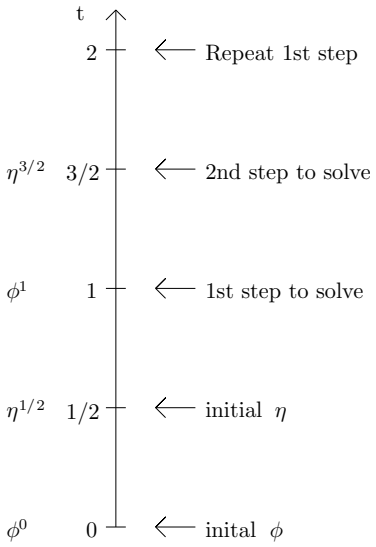


Figure 3: Staggered temporal grid visualized

Discretizing (12) and (14) step 1 becomes

$$(M + K)\mathbf{\Phi}^{n+1} + \Delta t M \mathbf{Y}^{n+1/2} - (M + K)\mathbf{\Phi}^n = 0 \tag{15}$$

and step 2 becomes

$$M\mathbf{Y}^{n+1/2} - \Delta t K \mathbf{\Phi}^n - M\mathbf{Y}^{n-1/2} \tag{16}$$

A numerical test solution

Eigenoscillations in the rectangle basin, with $\frac{\partial \phi}{\partial n} = 0$ at the boundary, are on the form

$$\begin{aligned} \phi &= B\sin(\omega t + \delta)\cos(k_x x)\cos(k_y y) \\ \eta &= A\cos(\omega t + \delta)\cos(k_x x)\cos(k_y y) \end{aligned} \tag{17}$$

where

$$\omega^2 = \frac{k^2}{1 + \frac{1}{3}\mu^2 k^2}, \qquad k^2 = k_x^2 + k_y^2 \tag{18}$$

To show that (17)are solutions to the linear PDEs, we simply test the solutions with

$$\frac{\partial \phi}{\partial t} + \eta - \frac{\mu^2}{3}\nabla^2 \frac{\partial \phi}{\partial t} = 0 \tag{19}$$

$$\frac{\partial \eta}{\partial t} + \nabla^2 \phi = 0 \tag{20}$$

where

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= B\omega\cos(\omega t + \delta)\cos(k_x x)\cos(k_y y) \\ \nabla^2 \frac{\partial \phi}{\partial t} &= -(k_x^2 + k_y^2)\frac{\partial \phi}{\partial t} = -k^2 \frac{\partial \phi}{\partial t} \end{aligned} \tag{21}$$

$$\begin{aligned} \frac{\partial \eta}{\partial t} &= -A\omega\sin(\omega t + \delta)\cos(k_x x)\cos(k_y y) \\ \nabla^2 \phi &= -(k_x^2 + k_y^2)\phi = -k^2 \phi \end{aligned} \tag{22}$$

To reduce some writing i choose to define

$$\begin{aligned} \alpha &= \sin(\omega t + \delta)\cos(k_x x)\cos(k_y y) \\ \beta &= \cos(\omega t + \delta)\cos(k_x x)\cos(k_y y) \end{aligned}$$

now (19) and (20) becomes

$$\omega B\beta + A\beta + \frac{\mu^2}{3}k^2 B\omega\beta = 0 \tag{23}$$

$$A\omega\alpha + k^2\alpha B = 0 \tag{24}$$

From (23) we may find the relation between A and B

$$A = -B\omega(1 + \frac{\mu^2}{3}k^2) \tag{25}$$

Substituting (25)into (24) we get

$$\begin{aligned} -B\omega^2(1 + \frac{\mu^2}{3}k^2)\alpha + k^2\alpha B &= 0 \\ B\alpha(-\frac{k^2}{\cancel{1 + \frac{\mu^2}{3}k^2}}\cancel{(1 + \frac{\mu^2}{3}k^2)} + k^2) &= 0 \\ B\alpha(-k^2 + k^2) &= 0 \end{aligned} \tag{26}$$

This shows that (17) is a solution of the linear PDEs as well as a relation between A and B. We can now start with the implementation in FENics.

Implementation

Testing the solver by implementing the rectangular case with initial boundary conditions derived from the eigenoscillations.
Some parameters needs to be set, and some needs to be computed from these parameters.

Test solver

Our test case is described as

$$\frac{\partial \phi}{\partial t} + \eta - \mu^2 \frac{1}{3} \nabla^2 \frac{\partial \phi}{\partial t} = 0$$

$$\frac{\partial \eta}{\partial t} = -\nabla^2 \phi$$

With initial conditions

$$\phi(x, y, 0) = 0$$
$$\eta(x, y, 0) = A\cos(k_x x)\cos(k_y y)$$

and no-flux boundary conditions

$$\frac{\partial \phi}{\partial n} = 0, \quad x, y \in \partial\Omega$$

For P1 elements we expect the H1 error norm to behave something like

$$||\phi - \phi_h||_1 \leq Ch^1 ||\phi||_2 \tag{27}$$

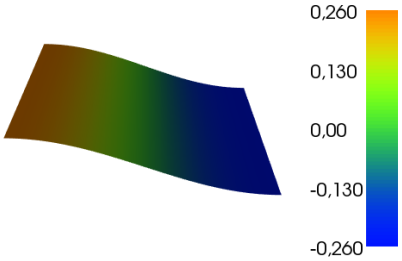
and similar for η. The L2 error norm should behave like

$$||\phi - \phi_h||_0 \leq Dh^2 ||\phi||_2 \tag{28}$$

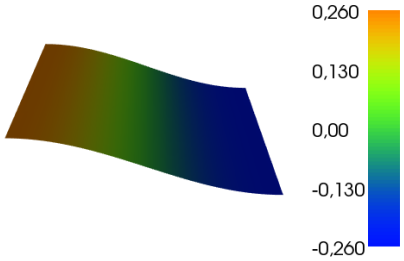
This tells us that we should expect first order convergence for the H1 error and second order convergence for L2 norm.

$\mathbf{k_x} = \pi, \mathbf{k_y} = \mathbf{0}$

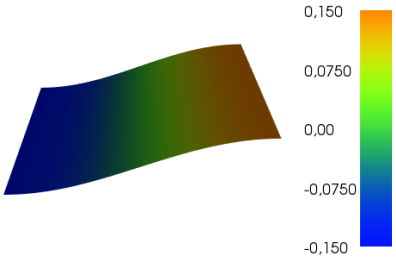
(a) Exact surface elevation at end time



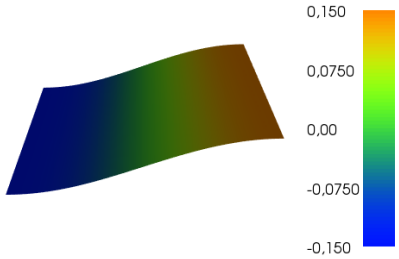
(b) Numerical surface elevation at end time



(c) Exact velocity potential at end time



(d) Numerical velocity potential at end time



N	2	4	8	16	32	64
ϕ L2 error	0.11888	0.03835	0.01038	0.00269	0.00071	0.00021
ϕ Convergence rate	-	1.63213	1.88562	1.95057	1.92461	1.75760
η L2 error	0.03112	0.00887	0.00230	0.00058	0.00014	0.00003
η Convergence rate	-	1.81105	1.94907	1.99346	2.03199	2.15310
ϕ H1 error	0.44641	0.17837	0.07377	0.03395	0.01652	0.00820
ϕ Convergence rate	-	1.32347	1.27375	1.11966	1.03888	1.01101
η H1 error	0.14509	0.07339	0.03705	0.01861	0.00932	0.00466
η Convergence rate	-	0.98334	0.98622	0.99356	0.99768	0.99918

We expected convergence rate two for the L2 error and one for the H1 error. This is alsowhat we got for both η and ϕ.

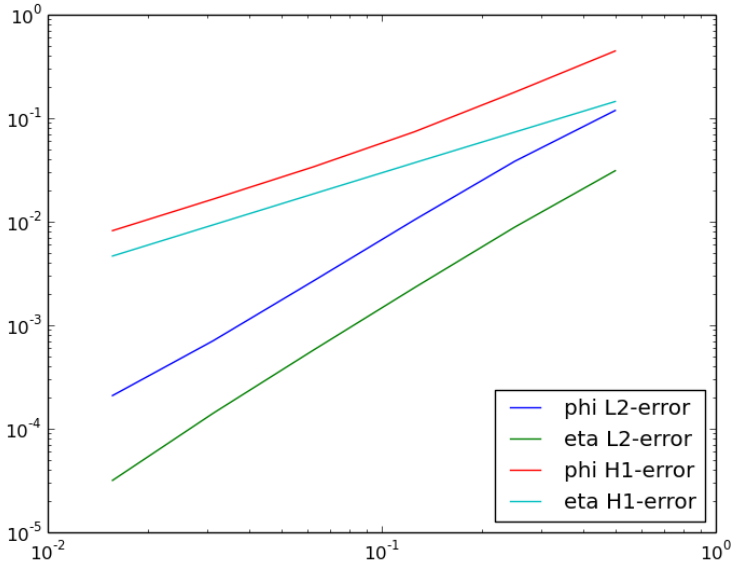
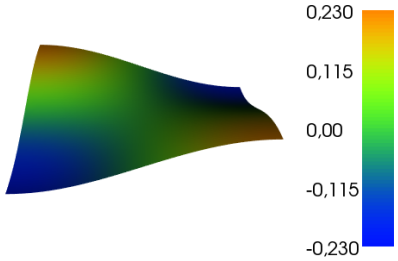


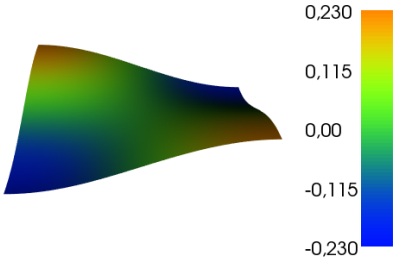
Figure 5: loglog plot: Error vs. h

$\mathbf{k_x} = \pi, \mathbf{k_y} = \pi$

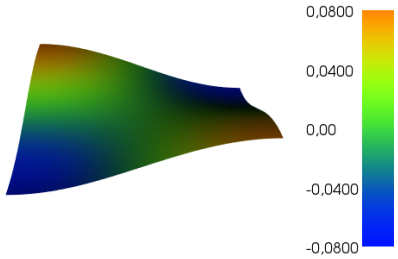
(a) Exact surface elevation at end time



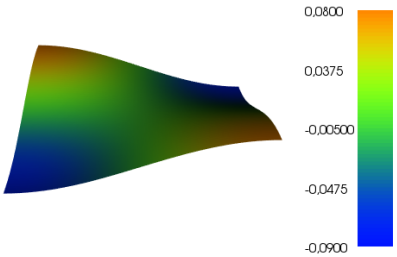
(b) Numerical surface elevation at end time



(c) Exact velocity potential at end time



(d) Numerical velocity potential at end time



N	2	4	8	16	32	64
ϕ L2 error	0.09865	0.03080	0.00883	0.00234	0.00062	0.00018
ϕ Convergence rate	-	1.67952	1.80180	1.91842	1.91661	1.74923
η L2 error	0.41738	0.10457	0.02617	0.00655	0.00164	0.00041
η Convergence rate	-	1.99684	1.99824	1.99940	1.99986	2.00010
ϕ H1 error	0.66418	0.29419	0.11748	0.05161	0.02461	0.01213
ϕ Convergence rate	-	1.17483	1.32431	1.18657	1.06839	1.02055
η H1 error	0.43474	0.12426	0.04325	0.01855	0.00885	0.00437
η Convergence rate	-	1.80678	1.52271	1.22134	1.06712	1.01770

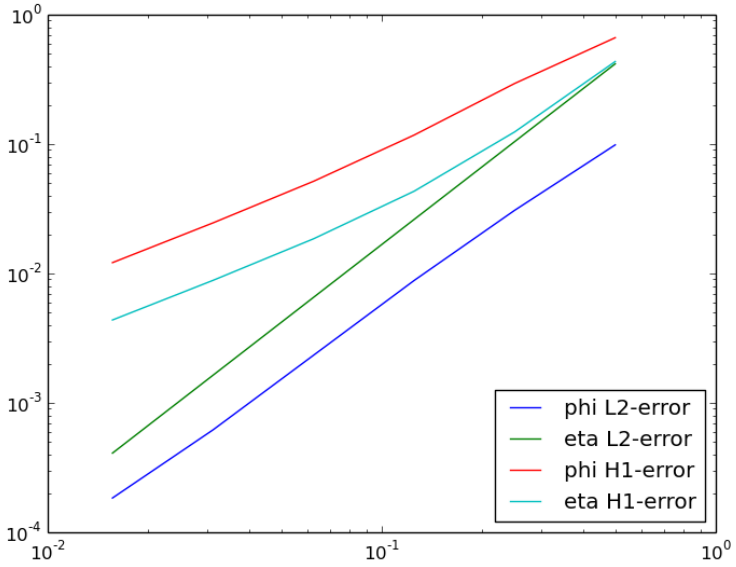


Figure 7: loglog plot: Error vs. h

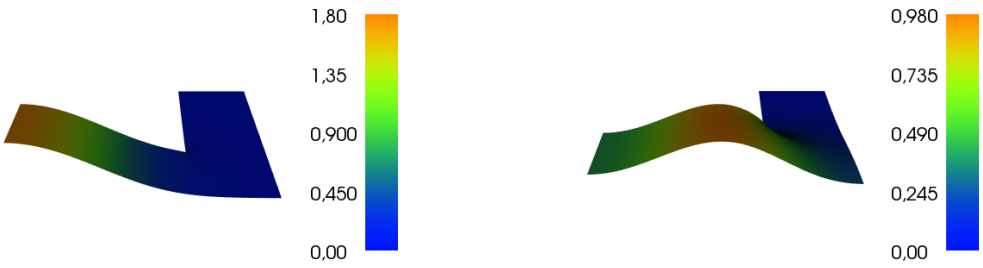
L-bend

Now that we've tested the solver and validated the results we may implement a new initial condition first in the rectangular case, then the L-bend.
The initial condition is

$$\eta(x, y, 0) = \begin{cases} 2A\cos^2(\frac{\pi x}{2l}), & x \leq l \\ 0, & x > l \end{cases} \tag{29}$$

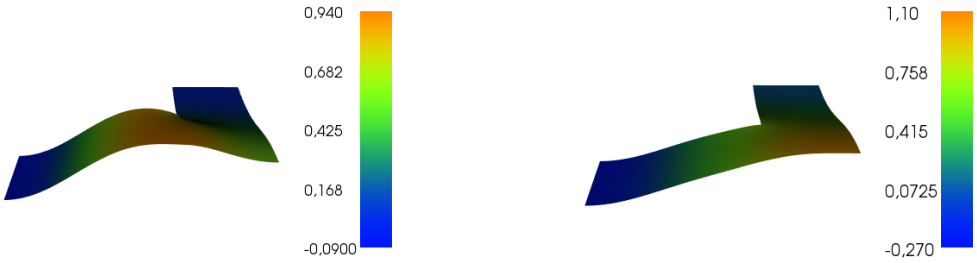
(a) t=1

(b) t=2



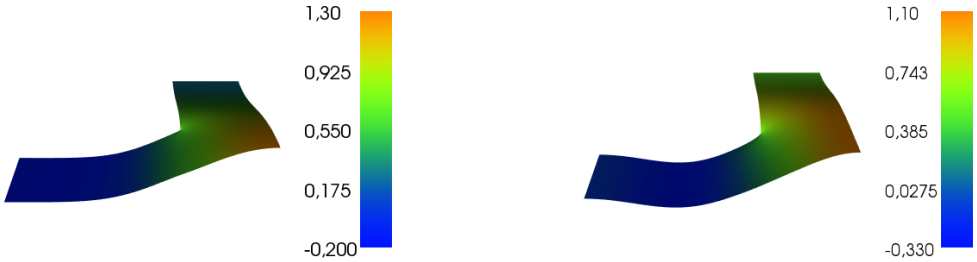
(c) t=3

(d) t=4



(e) t=5

(f) t=6



(g) t=7

(h) t=8

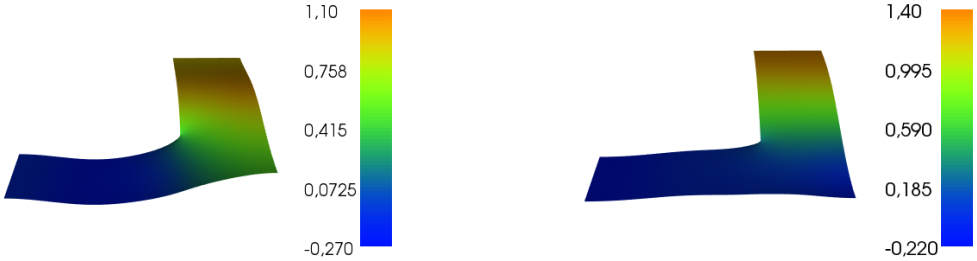


Figure 8: Wave propagation in L-bend

I would guess that the most critical points in the solution is at the bend, Figure 8(g). We see that the amplitude here is at it's largest (not including start and end).

Code

```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
set_log_active(False)

L = 5. #Lengde x-retning
Br = 1. #Lengde y-retning
def functionspace(N, kx, ky):
  a = 0
  print 'numerical error for:'
  for i in kx:
    for j in ky:
      print 'Kx:', i, 'ky:', j
      errorph= np.zeros(len(N))
      erroret=  np.zeros(len(N))
      herrorph= np.zeros(len(N))
      herroret=  np.zeros(len(N))
      h = np.zeros(len(N))
      NN = np.zeros(len(N))
      for n in range(len(N)):

        dt = 0.01
        T = 5

        eps = 1
        A = 1.
        #mesh = RectangleMesh(Point(0,0), Point(L,Br), N[n], N[n
    ])
        mesh = UnitSquareMesh(N[n], N[n])

        V = FunctionSpace(mesh, 'CG', 1)
        Q = FunctionSpace(mesh, 'CG', 4)

        mu = 1
        kxc = i*np.pi
        kyc = j*np.pi
        w2 = (kxc**2+kyc**2)/(1+mu**2*(kxc**2+kyc**2)/3.)
        B = -float(A)*np.sqrt(w2)/(kxc**2+kyc**2)
        delta = 0

```

```
41      g = Expression('A*cos(w*t+delta)*cos(kx*x[0])*cos(ky*x
    [1])', A = A, delta=delta, t=0, w=np.sqrt(w2), kx=kxc, ky=kyc
    )
        r = Expression('B*sin(w*t+delta)*cos(kx*x[0])*cos(ky*x
    [1])', B = B, delta=delta, t=0, w=np.sqrt(w2), kx=kxc, ky=kyc
    )
43
45      eta0 = project(g,V)
        phi0 = project(r,V)
47
        mu = Constant(mu)
49
        phi = TrialFunction(V)
51      eta = TrialFunction(V)
        Ni = TestFunction(V)
53
        phi_ = Function(V)
55      eta_ = Function(V)
57      dtc =Constant(dt)
        F1 = (phi-phi0)/dtc * Ni * dx + eta0 * Ni * dx + mu*mu/
    Constant(3) * inner(grad(Ni),grad((phi-phi0)/dtc))*dx
59      F2 = (eta-eta0)/dtc * Ni * dx - inner(grad(Ni), grad(
    phi0)) * dx
61      #A1 = assemble(lhs(F1))
        #A2 = assemble(lhs(F2))
63
        t = dt
65      etae = Expression('A*cos(w*t+delta)*cos(kx*x[0])*cos(ky*
    x[1])', A = A, delta=delta, t=0+dt/2., w=np.sqrt(w2), kx=kxc,
     ky=kyc)
        phie = Expression('B*sin(w*t+delta)*cos(kx*x[0])*cos(ky*
    x[1])', B = B, delta=delta, t=0, w=np.sqrt(w2), kx=kxc, ky=
    kyc)
67
        while t< T:
69
          #b1 = assemble(rhs(F1))
71        #b2 = assemble(rhs(F2))

73        etae.t = t+dt/2.
          phie.t = t
75
          #solve(A1,phi_.vector(), b1)
77        solve(lhs(F1)==rhs(F1), phi_)
          phi0.assign(phi_)
79
          #solve(A2,eta_.vector(), b2)
81        solve(lhs(F2)==rhs(F2), eta_)
          eta0.assign(eta_)
83
          #plot(eta_, rescale=False)#, interactive=True)
```

```
85            t +=dt

87          etah = project(etae, Q)
            phih = project(phie, Q)
89
            errorph[n] = errornorm(eta_, etah, 'l2')
91          erroret[n] = errornorm(phi_, phih, 'l2')

93          herrorph[n] = errornorm(eta_, etah, 'h1')
            herroret[n] = errornorm(phi_, phih, 'h1')
95          NN[n] = N[n]
            h[n] = 1./N[n] #mesh.hmin() #
97


99          if n == 0:
              print 'n=', N[n], '       l2 phi error   %.5g      l2
    eta error     %.5g      h1 phi error    %.5g          h1 eta
    error  %.5g' %(errorph[n], erroret[n], herrorph[n], herroret[
    n])
101
            else:
103           convergenceph = np.log(abs(errorph[n]/errorph[n-1]))/
    np.log(abs(h[n]/h[n-1]))
              convergenceet = np.log(abs(erroret[n]/erroret[n-1]))/
    np.log(abs(h[n]/h[n-1]))
105           hconvergenceph = np.log(abs(herrorph[n]/herrorph[n-1])
    )/np.log(abs(h[n]/h[n-1]))
              hconvergenceet = np.log(abs(herroret[n]/herroret[n-1])
    )/np.log(abs(h[n]/h[n-1]))
107           print 'n=', N[n], '       l2 phi error   %.5g      l2
     eta error     %.5g      l2 phi conv     %0.5f         l2 eta
     conv   %0.5f' %(errorph[n], erroret[n], convergenceph,
    convergenceet)
              print '            h1 phi error   %.5g     h1 eta
    error     %.5g       h1 phi conv     %0.5f         h1 eta
    conv   %0.5f' %(herrorph[n], herroret[n], hconvergenceph,
    hconvergenceet)
109
              #wiz1 = plot(eta_, interactive=False)
111           #wiz2 = plot(etah, interactive=False)
              #wiz3 = plot(phi_, interactive=False)
113           #wiz4 = plot(phih, interactive=False)

115           #wiz1.write_png('etanum%s%s'%(i,j))
              #wiz2.write_png('etaex%s%s'%(i,j))
117           #wiz3.write_png('phinum%s%s'%(i,j))
              #wiz4.write_png('phiex%s%s'%(i,j))
119           a+=2
              plt.figure(a-1)
121           plt.plot(NN, errorph, label='phi L2-error')#'phi error k =
    [%s pi, %s pi]' %(i, j))
              plt.plot(NN, erroret, label='eta L2-error')#'eta error k =
    [%s pi, %s pi]' %(i, j))
```

```
        plt.plot(NN, herrorph, label='phi H1-error')#'phi error k
    = [%s pi, %s pi]' %(i, j))
        plt.plot(NN, herroret, label='eta H1-error')#'eta error k
    = [%s pi, %s pi]' %(i, j))
        plt.legend(loc=1)
        plt.savefig('fig/l2error%s%s.png'%(i,j))

        plt.figure(a)
        plt.loglog(h, errorph, label='phi L2-error')#'phi error k
    = [%s pi, %s pi]' %(i, j))
        plt.loglog(h, erroret, label='eta L2-error')#'eta error k
    = [%s pi, %s pi]' %(i, j))
        plt.loglog(h, herrorph, label='phi H1-error')#'phi error k
     = [%s pi, %s pi]' %(i, j))
        plt.loglog(h, herroret, label='eta H1-error')#'eta error k
     = [%s pi, %s pi]' %(i, j))
        plt.legend(loc=4)
        plt.savefig('fig/loglog%s%s.png'%(i,j))

      #plt.show()

N = [2, 4, 8, 16, 32, 64]# 64, 128]
kx = [1, 2]
ky = [0, 1]

functionspace(N, kx, ky)
```

test2.py

```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
set_log_active(False)

L = 6. #Lengde x-retning
Br = 2. #Lengde y-retning
def functionspace(N):
  for n in range(len(N)):

    dt = 0.01
    T = 5

    A = 1. #Constant(1)
    mesh = RectangleMesh(Point(0,0), Point(L,Br), N[n], N[n])
    #mesh = UnitSquareMesh(N[n], N[n])

    V = FunctionSpace(mesh, 'CG', 1)
    Q = FunctionSpace(mesh, 'CG', 4)

    l= L-Br
    class MyExpression1(Expression):
      def eval_cell(self, value, x, ufc_cell):
        if x[0] < l:
          value[0] = 2*A*cos(pi*x[0]/(2*l))*cos(pi*x[0]/(2*l))
        else:
          value[0] = 0.

    g = MyExpression1(degree=1)
    #g = Expression('A*cos(w*t+delta)*cos(kx*x[0])*cos(ky*x[1])
    ', A = A, delta=delta, t=0, w=np.sqrt(w2), kx=kxc, ky=kyc)

    eta0 = project(g,V)
    phi0 = project(Constant(0),V)

    mu = Constant(1.)

    phi = TrialFunction(V)
    eta = TrialFunction(V)
    Ni = TestFunction(V)

    phi_ = Function(V)
    eta_ = Function(V)

    dtc =Constant(dt)
    F1 = (phi-phi0)/dtc * Ni * dx + eta0 * Ni * dx + mu*mu/
    Constant(3) * inner(grad(Ni),grad((phi-phi0)/dtc))*dx
    F2 = (eta-eta0)/dtc * Ni * dx - inner(grad(Ni), grad(phi0))
    * dx

    t = dt
    b = 0

```

```
    while t< T:
52      solve(lhs(F1)==rhs(F1), phi_)
        phi0.assign(phi_)
54
        solve(lhs(F2)==rhs(F2), eta_)
56      eta0.assign(eta_)
        b += 1
58      if b == 100:
          phiH1_norm = np.sqrt(assemble(phi_*phi_*dx + inner(grad(
    phi_),grad(phi_))*dx))
60        etaH1_norm = np.sqrt(assemble(eta_*eta_*dx + inner(grad(
    eta_),grad(eta_))*dx))
          phiL2_norm = np.sqrt(assemble(phi_**2*dx))
62        etaL2_norm = np.sqrt(assemble(eta_**2*dx))
          print '%1.5f %7.5f %7.5f %7.5f %7.5f' %(mesh.hmin(),
    phiL2_norm, phiH1_norm, etaL2_norm, etaH1_norm)
64      plot(eta_, rescale=False)#, interactive=True)
        t +=dt
66
N = [2, 4, 8, 16, 32, 64]# 64, 128]
68
functionspace(N)
```

new_inicond.py

```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
import meshgenerator
import os
set_log_active(False)

L = 6. #Lengde x-retning
B = 2. #Lengde y-retning

def Lmesh(L,B,grading, name, create_mesh=False):
  if create_mesh==True:
    meshgenerator.L(name+'.geo', L, B, grading)
    os.system('gmsh -2 '+name+'.geo')
    os.system('dolfin-convert '+ name+'.msh '+name+'.xml')

Lmesh(L,B,1, 'test', False)
mesh = Mesh('test.xml')#mesh1 + RectangleMesh(Point(L-B,B),
    Point(L, 20), ny, nx)

wiz1 = plot(mesh, interactive=False)
wiz1.write_png('meshL')
def functionspace(mesh, N):
  #print 'mesh size        phiL2        phiH1        etaL2
    phiH1'
  for n in range(N):
    if n == 0:
      None
    else:
      mesh = refine(mesh)

    dt = 0.01
    T = 10

    A = 1.

    V = FunctionSpace(mesh, 'CG', 1)

    l= L-B
    class MyExpression1(Expression):
      def eval_cell(self, value, x, ufc_cell):
        if x[0] < l:
          value[0] = 2*A*cos(pi*x[0]/(2*l))*cos(pi*x[0]/(2*l))
        else:
          value[0] = 0.

    g = MyExpression1(degree=1)

    eta0 = project(g,V)
    phi0 = project(Constant(0),V)

    mu = Constant(1.)

```

```
    phi = TrialFunction(V)
    eta = TrialFunction(V)
    Ni = TestFunction(V)

    phi_ = Function(V)
    eta_ = Function(V)

    dtc =Constant(dt)
    F1 = (phi-phi0)/dtc * Ni * dx + eta0 * Ni * dx + mu*mu/
    Constant(3) * inner(grad(Ni),grad((phi-phi0)/dtc))*dx
    F2 = (eta-eta0)/dtc * Ni * dx - inner(grad(Ni), grad(phi0))
    * dx

    t = dt
    a = 0
    b = 0
    while t< T:

      solve(lhs(F1)==rhs(F1), phi_)
      phi0.assign(phi_)

      solve(lhs(F2)==rhs(F2), eta_)
      eta0.assign(eta_)

      #plot(eta_, rescale=False)#, interactive=True)

      a += 1
      #b += 1
      #if b == 100:

        #phiH1_norm = np.sqrt(assemble(phi_*phi_*dx + inner(grad
    (phi_),grad(phi_))*dx))
        #etaH1_norm = np.sqrt(assemble(eta_*eta_*dx + inner(grad
    (eta_),grad(eta_))*dx))
        #phiL2_norm = np.sqrt(assemble(phi_**2*dx))
        #etaL2_norm = np.sqrt(assemble(eta_**2*dx))
        #print '%1.5f %7.5f %7.5f %7.5f %7.5f' %(mesh.hmin(),
    phiL2_norm, phiH1_norm, etaL2_norm, etaH1_norm)
      if a == 100:
        wiz = plot(eta_, interactive=False)
        wiz.write_png('timeeta%s%s' %(n,int(t)))
        a = 0
      t +=dt

N = 5

functionspace(mesh,N)
```

lbend.py