Solving Partial Differential Equations with Python - Tentative application to Rogue Waves

Sergio Manzetti[1,2]

1. Institute for Cellular and Molecular Biology, Uppsala University, Uppsala, Sweden.
2. Fjordforsk AS, Midtun, 6894 Vangsnes, Norway.

FipY

- FiPY (FiPy: A Finite Volume PDE Solver Using Python) is an open source python program that solves numerically partial differential equations.
- It uses the solvers PySparse, SciPy, PyAMG, Trilinos and mpi4py. Visualization is done using Matplotlib and Mayavi
- FipY can solve in parallel mode, reproduce the numerical in graphical viewers, and include boundary conditions, initial conditions and solve higher order PDEs (i.e. d^4/dx^4 by splitting them into two second order PDEs).
- FiPY is also incorporated with GIT and can be run on Windows, Linux or other conventional OS.

FiPY: https://www.ctcms.nist.gov/fipy/

Topic of research

- Rogue waves are rare phenomena occurring in optic fibers, in the atmosphere and most importantly in the oceans.
- Rogue waves can be represented using the Non-linear Schrödinger equation (NLSE)[1].
- The analytical solutions to the NLSE are conjugates of polynomials with a complex exponential term[1].
- Many recent papers have solved both inhomogenous and homogenous variants of the NLSE, both analytically and numerically.

1. Manzetti S. (2017). Mathematical modelling Rogue Waves. *Subm.* Comp. And Appl. Mathematics.

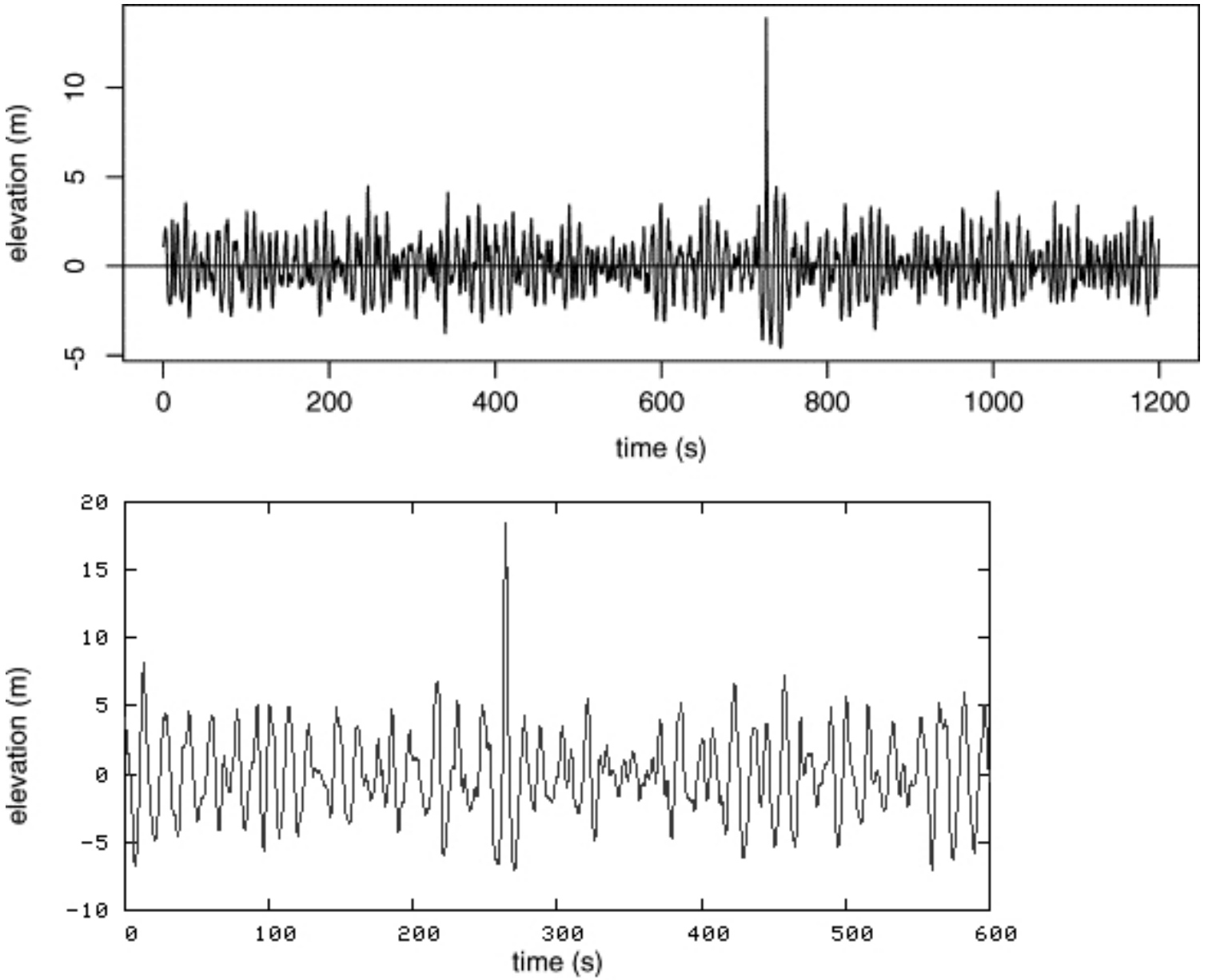


Figure. Top: Rogue wave recorded at the North Alwyn field, 500km east of Shetland islands, reaching 3X of the surrounding waves. . Bottom: The Draupner Wave recorded on 31 Dec 1995 on the Norwegian Draupner Platform, reaching 27 meters.

FiPY approach

- FiPy is called in a Python2.7 script using the command “from fipy import *”

- The interest is to solve first a simple equation:

$$\frac{\partial \phi}{\partial t} - \frac{\partial^2 \phi}{\partial x^2} = 0$$

which describes diffusion of some particles along the x-axis and tentatively try to work towards the NLSE:

$$\frac{i\partial\psi}{\partial t} + 1/2\frac{\partial^2\psi}{\partial x^2} + |\psi|^2\psi = 0$$

Python script structure – Part 1

- First we need to define a domain (with solution points) for the PDE. This requires definition of an object, Grid1D, which represents the linear structure grid. The points of the domain, which is repsented by a mesh are defined as:

  >>> mesh = Grid1D(nx=nx, dx=dx)

  >>> mesh=mesh,

  … value=0.)

- The points nx

  >>> nx = 50

  and the grid parameter dx is to to 1:

  >>> dx = 1

- Second, we need to define the initial conditions of the PDE, which are normally given as:

$$\phi = \begin{cases} 0 & \text{at } x = 1, \\ 1 & \text{at } x = 0. \end{cases}$$

  >>> valueLeft = 1
  >>> valueRight = 0

- Because the boundary conditions are represented as faces around the exterior of the mesh, we define the constraint on phi by using the names for the left and right side of the boundary using the phi.constrain() command which calls the values given above:

  >>> phi.constrain(valueLeft, mesh.facesRight)

  >>> phi.constrain(valueRight, mesh.facesLeft)

- *We then define the time-steps for the numerical computation, which is defined by 90 percent of the maximum stable timestep, which is given*

  *>>> timeStepDuration = 0.9 * dx**2 / (2 * D)*

  >>> steps = 2000

- *And then give the value of D.*

  >>> D = 1

Python script structure – Part 2

$$\frac{\partial \phi}{\partial t} - \frac{\partial^2 \phi}{\partial x^2} = 0$$

- The equation contains a transient term, d/dt, and the diffusion term, d^2/dx^2.
- In Fipy, this equation is given by:

  >>> eqX = TransientTerm() - ExplicitDiffusionTerm(coeff=D) == 0

- Then we define the variables of the equation:

  >>> x = mesh.cellCenters[0]

  >>> t = timeStepDuration * steps

- We then want to view the results and compare the results of the simulation with an analytical solution (if available):

  >>> phiAnalytical = CellVariable(name="analytical value",

  ... mesh=mesh)

  … if __name__ == '__main__':

  >>> viewer = Viewer(vars=(phi, phiAnalytical),

  ... datamin=0., datamax=1.)

  ... viewer.plot()

Python script structure – Part 3

- We then define the analytical solution, in this case, an example: $$f(x,t) = 1 - erf\frac{x}{2\sqrt{D}t}$$

- This appears in the script like:

```
>>> try:
        from scipy.special import erf
        phiAnalytical.setValue(1-erf(x / (2 * numerix.sqrt(D * t))))
>>> except ImportError:
        print ("The Scipy Library is not available to test the solution to the given transient diffusion equation")
```

- The equation is then solved by using *eqX.solve* repeatedly looping

```
>>> for step in range(steps):
      eqX.solve(var=phi,
            dt=timeStepDuration)
>>>    if __name__ == '__main__':
        viewer.plot()
>>> print (phi.allclose(phiAnalytical, atol = 7e-4)
1

>>> if __name__ == '__main__':
    raw_input("Explicit transient diffusion. Press <return> to proceed...")
```

Application to a variant of the NLSE

- We try then this structure on one candidate of the NLSE with real coefficients:

$$\frac{\partial \psi}{\partial t} + \frac{\partial^2 \psi}{\partial x^2} + |\psi|^2 \psi = 0$$

- Which is in FiPy:

```
>>> eq = TransientTerm(coeff=1., var=phi) + ExponentialConvectionTerm(coeff=(1.,), var=phi) + abs(phi_sq)*(phi) == 0.0
```

- In this test, we simulate the NLSE using a complex exponential function (e^{ix}) as a tentative form of phi:

```
>>> phiAnalytical = CellVariable(name="Analytical value",
        mesh=mesh)
>>> phiAnalytical.setValue(numerix.exp(1j*(X)))
```

Complete python script for the non-complex variant of the NLSE

```
#!/usr/bin/env python
# testing a non-complex variant of the NLSE

import numpy
import cmath as math
from fipy import *
from fipy import numerix

nx = 50
dx = 1. / float(nx)

mesh = Grid1D(nx=nx,dx=dx)
X = mesh.cellCenters[0]

phi = CellVariable(mesh=mesh, name="Solution")

vi = Viewer(vars=phi,datamin=0.0, datamax=1.0)
vi.plot()

raw_input("Initialization ...")

phi.constrain(1., mesh.facesLeft)
phi.constrain(0., mesh.facesRight)

phi_sq = CellVariable(mesh=mesh)
phi_sq.setValue( phi*phi )

#We now represent the equation, where u = phi,  du/dt + d^2u/dx^2 + |u|^2*u = 0 in fipy commands. du/dx : is a convection term with a unit scalar coefficient, i.e.
<SpecificConvectionTerm>(coeff=(1.,), var=u) ,
# du/dt : is a transient term that one can treat as before, i.e. TransientTerm(var=u). one can add a second order derivative as ExplicitDiffusionTerm(coeff=D)

eq = TransientTerm(coeff=1., var=phi) + ExponentialConvectionTerm(coeff=(1.,), var=phi) + abs((phi_sq))*(phi) == 0.0

dt = 0.01
steps = 100
for step in range(steps):
        eq.sweep(dt=dt)
        #
        phi_sq.setValue( phi * phi )
        #
        vi.plot()

# We then add a trial analytical solution to test for validity in the given PDE above, namely e^ix
phiAnalytical = CellVariable(name="Analytical value",
                mesh=mesh)


phiAnalytical.setValue(numerix.exp(1j*(X)))

vi = Viewer(vars=(phi, phiAnalytical))
vi.plot()

raw_input("Press <return> ...")
```

Output from FiPY python script

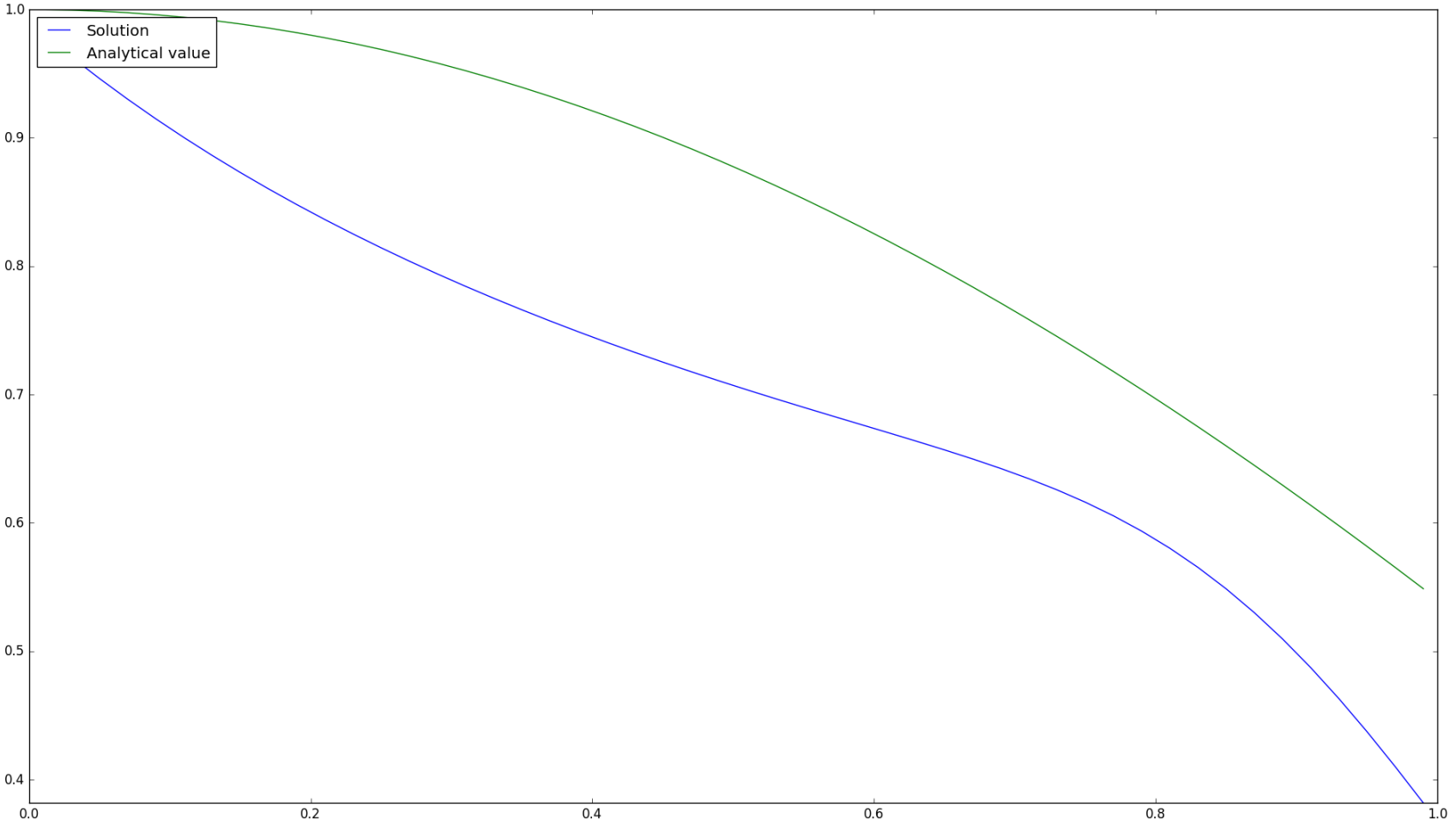


Fig1.The analytical function e^{ix} shows a similar trend as to the numerical solution of the non-complex NLSE shown in the previous slides.

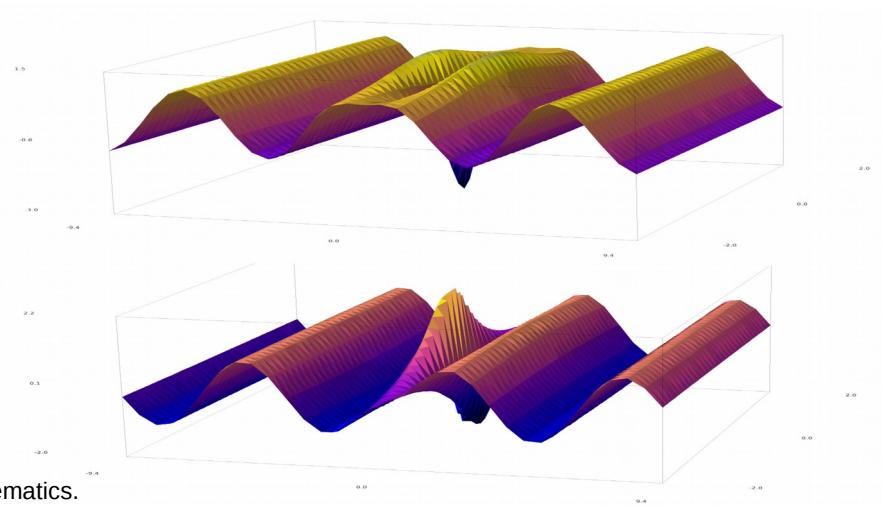
Interpretation

- The plot on the previous slide shows that e^{ix} is not a bad solution to a one dimensional case of the given PDE. However, based on the analytical solutions reviewed[1] we know that a good solution looks like:

$$\psi_j(x,t) = [(-1)^j + \frac{G_j(x,t) + ixH_j(x,t)}{D_j(x,t}]e^{ix}$$

where G, H and D are polynomials given by Akhmediev et al [2].

- The rogue wave plot from this solution in three dimensions appears as [1]:



1. Manzetti S. (2017). Mathematical modelling of Rogue Waves. Subm. Comp and App. Mathematics.
2. Akhmediev, Nail, Adrian Ankiewicz, and J. M. Soto-Crespo. "Rogue waves and rational solutions of the nonlinear Schrödinger equation." Physical Review E 80.2 (2009): 026601.

Conclusions

- FiPY has been used to make a test numerical simulation of a transient/diffusion problem in Python 2.7.
- Fipy was then tested on a non-complex variant of the NLSE.
- The result is not applicable to Rogue phenomena, and requires additional development of the FiPY script into a treatment of the complex NLSE.
- FiPY was successfully implemented as preliminary testing method for numerically solving PDEs as a basis for further research on the topic of Rogue Waves.

Mange takk