Full-Waveform Inversion, a hands-in tutorial

## Introduction

Full-waveform inversion (FWI) gained tremendous attention in geophysical exploration since it was first introduced [@Pratt]. However, the literature mostly contains specific and technical paper about applications and advanced method but lacks simple introductory resources for geophysics newcomers. Mathematical and geophysical overviews exist, as in [@Virieux], but do not highlight the implementation side of the problem. In this two part tutorial, we propose a hands-in walkthrough of FWI with Devito as a support. Devito is a finite-difference DSL that provides a concise and straightforward interface for the discretization and operator creation and allow the user to concentrate on the geophysical problem to solve instead of the low-level tedious implementation of a wave-equation simulator.We will only concentrate on conventional adjoint-state FWI. Other method exist to improve the convergence properties of the algorithm, but most still rely on forward/adjoint pairs an correlation-based gradient and should use the proposed framework.

FWI relies on two main components:

* A modelling operator to simulate synthetic data that can be compared to field recorded data
* A adjoint operator to compute the cross-correlation update direction.

We will illustrate the workflow on a very simplistic 2D model to guaranty everyone can run it on any computer. Larger and more realistic models come at a computational and memory price that wis not fit for a tutorial. The workflow we describe here translate however to any model no matter it's size, and to any wave-equation, provided the adjoint is known. The workflow for full-waveform inversion, and the corresponding notebooks, is the following:

* Simulate synthetic data and save the corresponding wavefield (**modelling.ipynb**).
* Compute the data residual, difference between the synthetic and observed data (**FWI.ipynb**).
* Back-propagate the residual (adjoint solve with the residual as a source term, **FWI.ipynb**).
* Compute the cross-correlation FWI gradient over time (**FWI.ipynb**).
* Repeat for all sources, at every iteration of the optimization algorithm in (**FWI.ipynb**).

We start with the description of a modelling operator then move onto adjoint operators and gradient for FWI. A complete tutorial on how to solve the seismic inversion problem once the computational framework is in place will be covered in the second part. This tutorials is linked to three notebooks that detail each step of the implementation from modelling to FWI. For clarity purposes, some details will be left out of the article but are fully detailed in the corresponding notebooks.

## Modelling

The acoustic wave equation for the square slowness m defined as m=\frac{1}{c^2}, where c is the speed of sound in the given physical media, and a source q is given by:

math {#WE} m \frac{d^2 u(x,t)}{dt^2} - \nabla^2 u(x,t) + \eta \frac{d u(x,t)}{dt}&=q \ \text{in } \Omega \\ u(.,0) &= 0 \\ \frac{d u(x,t)}{dt}|\_{t=0} &= 0

with the zero initial conditions to guarantee unicity of the solution. The boundary conditions are Dirichlet conditions :

u(x,t)|\_\delta\Omega = 0

where \delta\Omega is the surface of the boundary of the model \Omega.

In the field, the seismic wave propagates in every directions to an "infinite" distance. However, solving the wave equation in a mathematically/discrete infinite domain is not feasible. In order to compensate, Absorbing Boundary Conditions (ABC) or Perfectly Matched Layers (PML) are required to mimic an infinite domain. These two method allow to approximate an infinite media by damping and absorbing the waves at the limit of the domain to avoid reflections.

The simplest of these methods is the absorbing damping mask. The core idea is to extend the physical domain and to add a Sponge mask in this extension that will absorb the incident waves. In our case, we use ABC where \eta is the damping mask equal to 0 inside the physical domain and increasing inside the sponge layer. Multiple choice of profile can be chosen for \eta from linear to exponential.

We discretize the wave equation with Devito, a finite-difference DSL that solves the discretized wave-equation on a Cartesian grid. The finite-difference approximation is derived from Taylor expansions of the continuous field after removing the error term. ###Time discretization

We only consider the second order time discretization for now. From the Taylor expansion of the continuous wavefield \vd{u} in time, the second order discrete approximation of the second order time derivative is:

math {#timedis} \frac{d^2 u(x,t)}{dt^2} = \frac{\vd{u}(\vd{x},\vd{t+\Delta t}) - 2 \vd{u}(\vd{x},\vd{t}) + \vd{u}(\vd{x},\vd{t-\Delta t})}{\vd{\Delta t}^2} + O(\vd{\Delta t}^2).

and the finite-difference approximation is \frac{\vd{d^2u(x,t)}}{\vd{dt^2}} = \frac{\vd{u}(\vd{x},\vd{t+\Delta t}) - 2 \vd{u}(\vd{x},\vd{t}) + \vd{u}(\vd{x},\vd{t-\Delta t})}{\vd{\Delta t}^2} where \vd{u} is the discrete wavefield, \vd{\Delta t} is the discrete time-step (distance between two consecutive discrete times) and O(\vd{\Delta t}^2) is the discretization error term. The discretized approximation of the second order time derivative is then given by dropping the error term. This derivative is represented in Devito by u.dt2.

#### Spatial discretization

We define the discrete Laplacian as the sum of the second order spatial derivatives in the three dimensions. Each second spatial derivative is discretized with a k^{th} order finite-difference scheme also derived from Taylor expansion. math {#spacedis} \Delta \vd{u}(\vd{x},\vd{y},\vd{z},\vd{t})= \sum\_{j=1}^{j=\frac{k}{2}} \Bigg[\alpha\_j \Bigg(& \vd{u}(\vd{x+jdx},\vd{y},\vd{z},\vd{t})+\vd{u}(\vd{x-jdx},\vd{y},\vd{z},\vd{t}) + \\ &\vd{u}(\vd{x},\vd{y+jdy},\vd{z},\vd{t})+\vd{u}(\vd{x},\vd{y-jdy},\vd{z}\vd{t}) + \\ &\vd{u}(\vd{x},\vd{y},\vd{z+jdz},\vd{t})+\vd{u}(\vd{x},\vd{y},\vd{z-jdz},\vd{t})\Bigg) \Bigg] + \\ &3\alpha\_0 \vd{u}(\vd{x},\vd{y},\vd{z},\vd{t}).

This derivative is represented in Devito by u.laplace.

### Wave equation

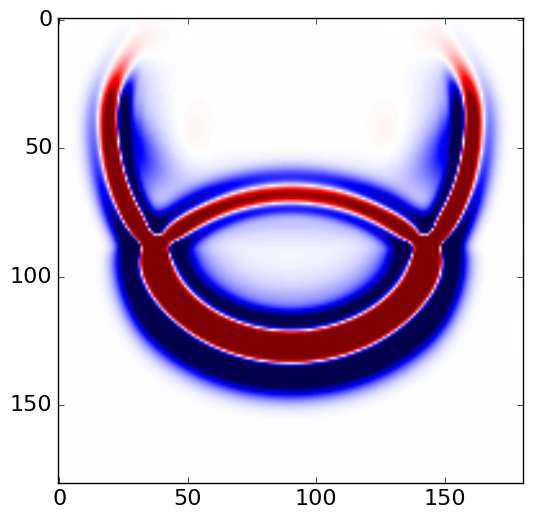
With the space and time discretization defined, we can fully discretize the wave-equation with the combination of time and space discretizations and obtain the following second order in time and k^{th} order in space discrete stencil to update one grid point at position \vd{x}, \vd{y},\vd{z} at time \vd{t}, i.e. math {#WEdis} \vd{u}(\vd{x},\vd{y},\vd{z},\vd{t+\Delta t}) = &2\vd{u}(\vd{x},\vd{y},\vd{z},\vd{t}) - \vd{u}(\vd{x},\vd{y}, \vd{z},\vd{t-\Delta t}) +\\ & \frac{\vd{\Delta t}^2}{\vd{m(\vd{x},\vd{y},\vd{z})}} \Big(\Delta \vd{u}(\vd{x},\vd{y},\vd{z},\vd{t}) + \vd{q}(\vd{x},\vd{y},\vd{z},\vd{t}) \Big).

The complete implementation of the wave equation including detailed handling of the source and receiver injection/measurment operators \vd{P}\_s, \vd{P}\_r is highlighted in modelling.ipynb.

#### Figure:

{width=100%}

#### Figure:

{width=100%}

## FWI

Full-waveform inversion (FWI) aims then to invert an accurate model of the discrete wave velocity, \vd{c}, or equivalently the square slowness of the wave, \vd{m} = \frac{1}{\vd{c}^2}, from a given set of measurements of the pressure wavefield \vd{u}. This can be expressed as the following optimization problem [@LionsJL1971, @Virieux, @haber10TRemp]:

math {#FWI} \mathop{\hbox{minimize}}\_{\vd{m}} \Phi(\vd{m})&=\frac{1}{2}\left\lVert\vd{P}\_r \vd{A}(\vd{m})^{-1} \vd{P}\_s^T \vd{q} - \vd{d}\right\rVert\_2^2 \\

The gradient of the FWI objective \Phi(\vd{m}) with respect to the square slowness \vd{m} is defined in a linear algebra way as

math {#FWIgradLA} \nabla\Phi\_s(\vd{m})= - \left(\frac{d \vd{A}(\vd{m}) \vd{u}}{dm}\right)^T \vd{A}(\vd{m})^{-T} \vd{P}\_r^T \delta \vd{d} =\vd{J}^T\delta\vd{d},

and can be rewritten as a sum over time of zero-time offset correlation of the forward and adjoint wavefield.

math {#FWIgrad} \nabla\Phi\_s(\vd{m})= - \sum\_{\vd{t} =1}^{n\_t}\vd{u}[\vd{t}] \vd{v}\_{tt}[\vd{t}].

n\_t is the number of computational time steps, \delta\vd{d} = \left(\vd{P}\_r \vd{u} - \vd{d} \right) is the data residual (difference between the measured data and the modelled data), \vd{J} is the Jacobian operator and \vd{v}\_{tt} is the second-order time derivative of the adjoint wavefield solving #adjWE.

### Implementation

As we explained in the introduction of this tutorial and showed in Equation #FWIgradLA, this method is based on back-propagation. If we go back to the modelling part, we can rewrite the simulation as a linear system solve: math {#linWE} \vd{A}(\vd{m}) \vd{u} = \vd{P}\_s^T \vd{q}

and we require its adjoint to solve

math {#adjWE} \vd{A}(\vd{m})^T \vd{v} = \vd{P}\_r^T \delta \vd{d}

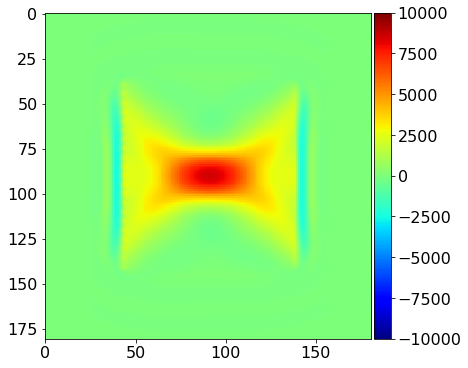
Solving the wave-equation is equivalent to solving a linear system \vd{Au}=\vd{q} where the vector \vd{u} is the discrete wavefield solution of the discrete wave-equation, \vd{q} is the source term and \vd{A} is the matrix representation of the discrete wave-equation. From Equation #WEdis we can see that the matrix \vd{A} is a lower triangular matrix that reflects the time-marching structure of the stencil. Simulation of the wavefield is equivalent to a forward elimination on the lower triangular matrix \vd{A}. The adjoint of \vd{A}, denoted as \vd{A}^T, is then an upper triangular matrix and the solution \vd{v} of the discrete adjoint wave-equation \vd{A}^\top\vd{v}=\vd{q}\_a for an adjoint source \vd{q}\_a is equivalent to a backward elimination on the upper triangular matrix \vd{A}\top and is simulated backward in time starting from the last time-step. These matrices are never explicitly formed, but are instead matrix free operators with implicit implementation of \vd{u}=\vd{A}^{-1}\vd{q}.

The accuracy of the adjoint and its rigorousness can and should be tested. One simple way to do so is the **dottest** that implements the mathematical definition of and adjoint and writes in the Devito frawework

# Run forward and adjoint operators  
 rec, \_, \_ = solver.forward(save=False) # computes rec = P\_r A^{-1} P\_s^T q  
 solver.adjoint(rec=rec, srca=srca) # computes q\_a = P\_s A^{-1} P\_r^T rec  
 # Adjoint test: Verify <Ax,y> matches <x, A^Ty> closely  
 term1 = np.dot(srca.data.reshape(-1), solver.source.data)  
 term2 = linalg.norm(rec.data) \*\* 2  
 info('<Ax,y>: %f, <x, A^Ty>: %f, difference: %12.12f, ratio: %f'  
 % (term1, term2, term1 - term2, term1 / term2))  
 assert np.isclose(term1, term2, rtol=1.e-5)

The complete implementation of the gradient operator is defined in FWI.ipynb and produces the following update direction for a simple transmission camembert model.

#### Figure:

{width=100%}

## Conclusion

We have now a simple setup to compute one gradient of the FWI objective. We will detail in the following tutorial how to fully run FWI with multiple sources and an optimization toolbox/algorithm.

### Installation

This tutorial and the coming second part are based on Devito version 3.0.3. It also require to install the full software with examples, not only the code generation API. To install devito

git clone https://github.com/opesci/devito/tree/v3.0.3  
 cd devito  
 conda env create -f environment.yml  
 source activate devito  
 pip install -e .  
 export PYTHONPATH=$PYTHONPATH:path-to-devito

### Usefull links

* [Devito documentation](http://www.opesci.org/)
* [Devito source code and examples](https://github.com/opesci/Devito)
* [Tutorial notebooks with latest Devito/master](https://github.com/opesci/Devito/examples/seismic/tutorials)

\def\argmin{\mathop{\rm arg\,min}}  
\def\vec{\mbox{``\mathrm{vec}``}}  
\def\ivec{\mbox{``\mathrm{vec}^{-1}``}}  
\newcommand{\m}{{\mathsf{m}}}  
\newcommand{\PsDO}{\mbox{PsDO\,}}  
\newcommand{\Id}{\mbox{``\tensor{I}\,``}}  
\newcommand{\R}{\mbox{``\mathbb{R}``}}  
\newcommand{\Z}{\mbox{``\mathbb{Z}``}}  
\newcommand{\DE}{:=}  
\newcommand{\Order}{\mbox{``{\cal O}``}} \def\bindex#1{{\mathcal{#1}}}  
\def\pector#1{\mathrm{\mathbf{#1}}}   
\def\cector#1{#1}   
\def\censor#1{#1}   
\def\vd#1{\mathbf{#1}}  
\def\fvector#1{{\widehat{\vd{#1}}}}  
\def\evector#1{{\widetilde{\vd{#1}}}}  
\def\pvector#1{{\breve{\vd{#1}}}}  
\def\pector#1{\mathrm{#1}}  
\def\ctensor#1{\bm{\mathcal{#1}}}  
\def\tensorm#1{\bm{#1}}  
\def\tensor#1{\vd{#1}}  
\def\hensor#1{\tensor{#1}}  
\def\uensor#1{\underline{\bm{#1}}}  
\def\hector#1{\vd{#1}}  
\def\ftensor#1{{\widehat{\tensor{#1}}}}  
\def\calsor#1{{\boldsymbol{\mathcal{#1}}}}  
\def\optensor#1{{\boldsymbol{\mathcal{#1}}}}  
\def\hvector#1{\hat{\boldsymbol{\mathbf{#1}}}}  
\def\minim{\mathop{\hbox{minimize}}}  
\newcommand{\norm}[1]{\left\lVert#1\right\rVert\_2^2}  
\newcommand{\overbar}[1]{\mkern 1.5mu\overline{\mkern-1.5mu#1\mkern-1.5mu}\mkern 1.5mu}