

Aulas práticas

Objetivos

- Mostrar aplicações da teoria
- Para que serve tudo isso
- Coisas misteriosas acontecem na prática
- Força a descobrir coisas novas

Software

- Projeto ***Fatiando a Terra***
- Código aberto (open source)
- Gratuito
- Vários tipos de modelagem e inversão
- Para ensino e pesquisa

Fatiando a Terra

- www.fatiando.org
- Blog
- Documentação
- Teoria
- Como e onde baixar
- Instalar



Fatiando a Terra

Geophysical direct and inverse modeling



An open source modeling toolkit written in **Python**

Fatiando a Terra provides command-line tools and an API for a variety of geophysical modeling applications. It makes heavy use of **Numpy** for linear algebra operations, provides a high level interface to visualise your data and results using **Matplotlib** for 2D and **Mayavi2** for 3D. Optimized code is written in Fortran and C and wrapped to Python using Numpy's f2py.

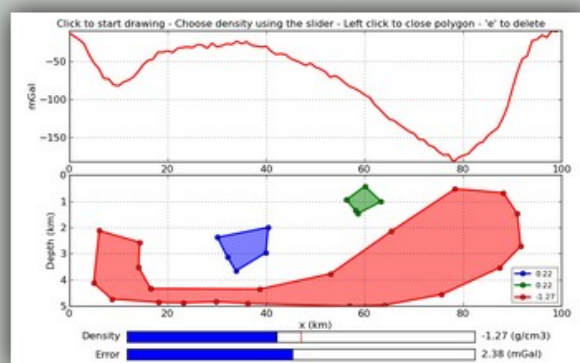
Documentation

The **online documentation** is available with a complete **API reference** and a **Cookbook** with some recipes to get you started. Be aware that since we're under development the documentation (specially the API reference) changes very fast (though mostly by adding new things). If you find any mistakes or think that some section of the docs could be better explained, please **give us a shout!**

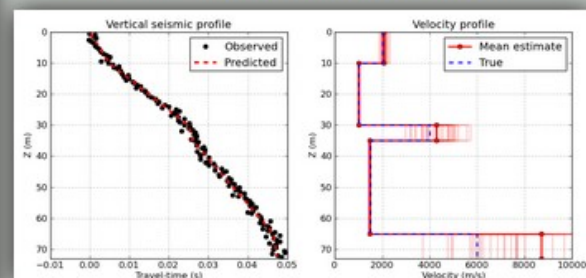
Example Gallery

Some of the functionality already implemented:

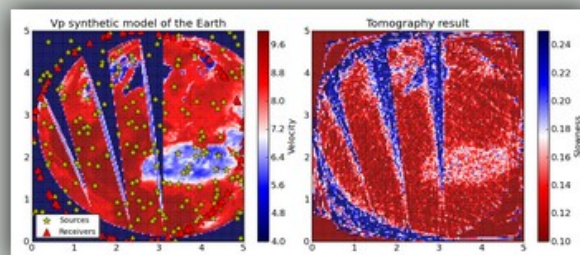
Moulder - 2D gravimetric direct modeling



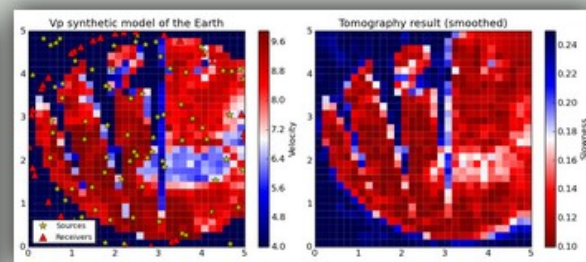
Inversion of synthetic vertical seismic profile data



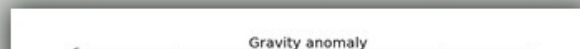
Straight-ray travel-time tomography of large models using sparse linear algebra



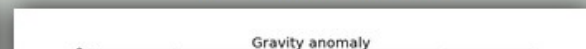
Straight-ray travel-time tomography



Simple inversion for the relief of the 2D triangular basin



... and also a trapezoidal basin





Previous topic

Cookbook

Next topic

Potential Fields
([fatiando.potential](#))

This Page

Show Source

Quick search

Go

Enter search terms or a module, class
or function name.

API Reference

Package `fatiando` contains tools for geophysical direct and inverse modeling as well as data processing, gridding, and visualization.

The API is divided into modules and subpackages, each for a different geophysical method or type of operation:

- **Potential Fields ([fatiando.potential](#))**
 - Upward continuation, derivatives, etc. ([fatiando.potential.transform](#))
 - Direct modeling with right rectangular prisms ([fatiando.potential.prism](#))
 - Direct modeling with 3D polygonal prisms ([fatiando.potential.polyprism](#))
 - Direct modeling with 2D polygons ([fatiando.potential.talwani](#))
 - Inversion for the relief of 2D basins ([fatiando.potential.basin2d](#))
- **Seismics and seismology ([fatiando.seismic](#))**
 - Direct modeling of seismic wave travel times ([fatiando.seismic.traveltime](#))
 - Epicenter determination ([fatiando.seismic.epicenter](#))
 - Seismic profiling ([fatiando.seismic.profile](#))
 - Straigh-ray 2D tomography with SrTomo ([fatiando.seismic.srtomo](#))
- **Geothermics ([fatiando.heat](#))**
 - Climate change signal in well temperature logs ([fatiando.heat.climatesignal](#))
- **Meshing ([fatiando.mesher](#))**
 - 3D meshes and objects ([fatiando.mesher.ddd](#))
 - 2D meshes and objects ([fatiando.mesher.dd](#))
- **Inverse problem solving tools ([fatiando.inversion](#))**
 - Solvers for linear problems ([fatiando.inversion.linear](#))
 - Gradient solvers ([fatiando.inversion.gradient](#))
 - Heuristic solvers ([fatiando.inversion.heuristic](#))
 - Regularizing functions ([fatiando.inversion.regularizer](#))
 - Base data module class ([fatiando.inversion.datamodule](#))
- **Gridding ([fatiando.gridder](#))**
- **Visualization ([fatiando.vis](#))**
 - Map grids and plot 2D objects ([fatiando.vis.map](#))

Next topic

Solvers for linear problems
(`fatiano.inversion.linear`)

This Page

Show Source

Quick search

Go

Enter search terms or a module, class
or function name.

fatiano.org/docs/api/inversion.html

See the `fatiano.inversion.regularizer.Regularizer` class for the general structure the solvers expect from regularizer classes.

INVERSE PROBLEM SOLVERS

- `fatiano.inversion.linear`
- `fatiano.inversion.gradient`
- `fatiano.inversion.heuristic`

These modules have factory functions that generate generic inverse problem solvers. Instead of solving the inverse problem, they return a solver function with they solver parameters (step size, maximum iterations, etc) already set. The solver functions are actually Python [generators](#). Generators are Python functions that have a `yield` statement instead of a `return`. The difference is that generators should be used in `for` loops and return one iteration of the solving process per loop iteration. The solver functions receive only two parameters: a list of data modules (`fatiano.inversion.datamodule`) and a list of regularizers (`fatiano.inversion.regularizer`). Data modules know how to calculate things like the Hessian matrix and gradient vector for a given data set and parametrization (specific to a given inverse problem). Regularizers know how to calculate the same things but for a given regularizing function and parametrization. This way, the user can combine any number of data sets and regularizers as he/she wants. So we can program the solvers and regularizers once and use them in any inverse problem!

A typical factory function for an iterative solver looks like:

```
def factory(initial, step, maxit):  
    # Define the solver generator. The parameters need by this specific  
    # solver are passed to it as optional parameters.  
    # dms is a list of data modules and regs is a list of regularizers  
    def solver(dms, regs, initial=initial, step=step, maxit=maxit):  
        # Initialize the solver parameters  
        ...  
        for it in xrange(maxit):  
            # Do an iteration and find an estimate p  
            ...  
            # Now comes the cool part! Spit out a changeset with the result  
            # of this iteration in a dictionary. goals is a list of the  
            # goal function values until this iteration. misfits is the same  
            # but for the data-misfit function. residuals is a list of the  
            # residual vectors of each data module  
            yield {'estimate':p, 'goals':goals, 'misfits':misfits,  
                  'residuals':residuals}  
    # The factory function returns the solver function (Python magic) which  
    # can be passed to a particular inverse problem.  
    return solver
```

Lets say I have a module that solves a particular inverse problem called `myinverse.py`. This module would look something like this:

References.

- Kelley, C. T., 1999, Iterative methods for optimization: Raleigh: SIAM.

`fatiando.inversion.gradient.steepest(initial, step=0.1, maxit=500, tol=1e-05, armijo=True, maxsteps=20)`

Factory function for the non-linear inverse problem solver using the Steepest Descent algorithm.

The increment to the parameter vector \bar{p} is calculated by

$$\Delta \bar{p} = -\lambda \bar{g}$$

where λ is the step size and \bar{g} is the gradient vector.

Optionally, the step size can be determined through a line search algorithm using the Armijo rule. In this case

$$\lambda = \beta^m$$

where $1 > \beta > 0$ and $m \geq 0$ is an integer that controls the step size. The line search finds the smallest m that satisfies the condition (Armijo rule)

$$\Gamma(\bar{p} + \Delta \bar{p}) - \Gamma(\bar{p}) < \alpha \beta^m \|\bar{g}(\bar{p})\|^2$$

where $\Gamma(\bar{p})$ is the goal function evaluated for parameter vector \bar{p} , $\alpha = 10^{-4}$, and $\bar{g}(\bar{p})$ is the gradient vector of $\Gamma(\bar{p})$.

Parameters:

- initial
The initial estimate of the parameters
- step
The step size (if using Armijo, β , else λ)
- maxit
Maximum number of iterations
- tol
Relative tolerance for decreasing the goal function to before terminating
- armijo
If True, will use the Armijo rule to determine the best step size at each iteration. It's highly recommended to use this.

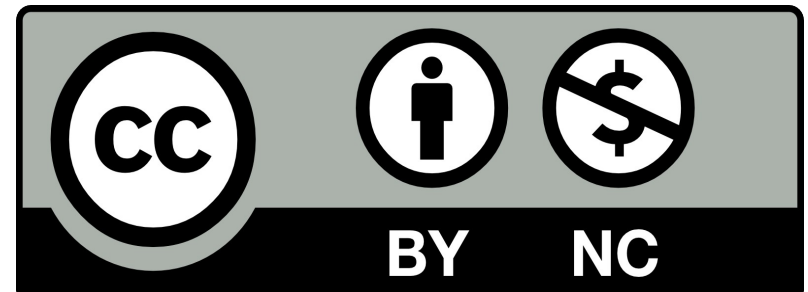
Apostila e slides

- <http://code.google.com/p/inverse-problems/>
- Apostila
 - PDF
 - Código Latex
 - Figuras
- Slides
- Práticas

Apostila

Creative Commons:

- Livre para:
 - Compartilhar
 - Editar
- Contanto que:
 - Atribuição
 - Não comercial



inverse-problems

Apostila de "Tópicos em inversão geofísica"

 Search projects

Project Home | [Downloads](#) | [Wiki](#) | [Issues](#) | [Source](#)


Summary | [Updates](#) | [People](#)

Project Information


[Activity](#)  Medium
[Project feeds](#)

Code license
[Other Open Source](#)
See source for details

Labels
[geophysics](#)

 **Members**
[leouieda](#)
[2 committers](#)

Featured

 **Downloads**
[apostila-v1.0.1.pdf](#)
[Show all »](#)

Repositório da apostila de "Tópicos em inversão geofísica".



This work is licensed under a [Creative Commons Attribution-NonCommercial](#) 3.0 Unported License.