A parallel matrix-free framework for frequency-domain seismic modelling, imaging and inversion in Matlab

Tristan van Leeuwen

Dept. of Earth and Ocean sciences, University of British Columbia, 6339 stores road, Vancouver, BC, Canada, V6T 1Z4

Abstract

I present a parallel matrix-free framework for frequency-domain seismic modeling, imaging and inversion. The framework provides basic building blocks for designing and testing optimization-based formulations of both linear and non-linear seismic inverse problems. By overloading standard linear-algebra operations, such as matrix-vector multiplications, standard optimization packages can be used to work with the code without any modification. This leads to a scalable testbed on which new methods can be rapidly prototyped and tested on medium-sized 2D problems. I present some numerical examples on both linear and non-linear seismic inverse problems.

Keywords: Seismic imaging, optimization, Matlab, object-oriented programming

1. Introduction

Seismic data are a rich source of information on the subsurface. Large amounts of data are acquired for both academic and industrial purposes, either through active experiments or by recording earth-quake responses. Over the years many different formulations and algorithms have been proposed to estimate subsurface properties based on such data. Over the past two decades these algorithms have come to rely heavily on high-performance computing. A successful seismic inversion framework needs to combine a modeling engine to solve a wave-equation (e.g., finite differences or finite elements), (non-) linear optimization algorithms and facilitate i/o of large amounts of data and the corresponding meta data. Due to the large scale of typical problems, a computer code that implements these algorithms and concepts has to be extremely efficient, scalable and robust. On the other hand, the interdisciplinary nature of such projects requires that experts from different fields can contribute to the code. Needless to say, some form of unit-testing needs to be incorporated to facilitate de-bugging. The challenge is in meeting these, seemingly mutually exclusive, requirements.

A necessary (but not sufficient) condition is that the basic building blocks of the code, such as the modeling operator, its Jacobian and their basic operations (forward modeling, action of the Jacobian and its adjoint) are exposed. This allows the units to be tested separately and re-used for different purposes. Then, these building blocks should be cast in pre-defined classes, such as (non-) linear operators and functionals.

This allows us to have our code reflect the underlying mathematical principles and serves a bridge to existing code. We do not have to sacrifice efficiency when following these principles because the individual blocks can still contain highly optimized code.

An added bonus of such a framework is that it allows for rapid prototyping and increases reproducibility. Because the separate building blocks expose the underlying mathematics, it should be relatively straightforward to understand and implement the algorithm using a different modeling kernel (given that is also exposes the basic functionality as described above). Unit tests can ensure that any given implementation is correct.

For further reading on this design philosophy I refer the reader to [1] and [2] who illustrates the approach for time-stepping codes and C++. A few existing libraries that facilitate such approaches to various degrees are PETSC [3], Trilinos [4], the Rice Vector Library [5] and SPOT [6].

Framework

To illustrate the design principles sketched above, I present a *parallel matrix-free* framework for frequency-domain wavefield modelling, imaging and inversion in Matlab. The basic building blocks of the framework are the finite-difference modeling operator $\mathbf{d} = \mathsf{F}(\mathbf{m})$ that maps a model of the subsurface $\mathbf{m}$ to the observed data $\mathbf{d}$, and its Jacobian $\nabla\mathsf{F}(\mathbf{m})$, implemented as matrix-free linear operator. Using these building blocks, we can formulate algorithms to solve the basic inverse problem

> *for given observations* $\mathbf{d}_{\text{obs}}$*, find a model* $\mathbf{m}$ *for which the predicted data* $F(\mathbf{m})$ *fits the observed data.*

In particular, I will focus on optimization-based formulations of this problem

$$\min_{\mathbf{m}} W(\mathsf{F}(\mathbf{m}), \mathbf{d}_{\text{obs}}), \tag{1}$$

where W measures the misfit between the observed and predicted data (e.g., for non-linear least-squares $W(\cdot,\cdot) = \|\cdot - \cdot\|_2^2$).

Matlab code

The Matlab code is available for academic purposes. The code to reproduce the examples presented in this paper is also included. See `https://www.slim.eos.ubc.ca/releases` for download instructions.

Outline

First, I discuss the modeling kernel, which consists of the forward modeling operator and its Jacobian. Then I show how to use a framework for matrix-free linear algebra in Matlab to implement functionality needed for various imaging and inversion purposes. I discuss how the different units of code should be tested and finally, give some numerical examples on both linear and non-linear seismic inverse problems.

Notation

- x - lowercase, italic symbols denote scalars
- $\mathbf{x}$ - lowercase, boldface symbols denotes column vectors whose elements are denoted as x_i
- A - capital italic symbols denote matrices or linear operators and $\cdot^\dagger$ denotes their (conjugate) transpose or adjoint.
- $\mathsf{vec}(\cdot)$ denotes vectorization, $\mathsf{vec}^{-1}(\cdot)$ denotes reshaping into the original size.
- $\mathrm{diag}(\cdot)$ when applied to a matrix extracts the diagonal from a matrix, when applied to a vector constructs a diagonal matrix.
- $\mathsf{F}(\cdot)$ - capital sans-serif symbols denote non-linear operators, and $\nabla\mathsf{F}$ denotes the Jacobian.
- $\mathsf{f}(\cdot)$ - lower case sans-serif symbols denote functions and $\nabla\mathsf{f}$ denotes the gradient.
- $\mathcal{S}$ - curly capital letters denote sets/lattices/grids.

2. Modeling

Wave-propagation in the earth is assumed to obey a scalar Helmholtz equation:

$$A_\omega(\mathbf{m})\mathbf{u} = \mathbf{q}. \tag{2}$$

Here $A_\omega(\mathbf{m})$ is the discretized Helmholtz operator $\omega^2\mathbf{m} + \nabla^2$ – with some form of absorbing boundary conditions to simulate an infinite domain – for frequency ω and squared slowness $\mathbf{m}$ in seconds^2/meters^2, $\mathbf{u}$ is the wavefield and $\mathbf{q}$ represents the source.

Grids

I define 4 distinct grids that are used during the computations:

- The physical grid $\mathcal{G}$. This grid represents the region of interest. The medium paramerer $\mathbf{m}$ is discretized on this grid.
- The computational grid $\mathcal{C}$, which is an extension of the physical grid. The Helmholtz operator is discretized on this grid. The extended area is used for the absorbing boundary conditions.
- The source grid $\mathcal{S}$. This grid is a subset of the physical grid and is used to define the source functions.
- The receiver grid $\mathcal{R}$. This grid is a subset of the physical grid and is used to define the receiver array.

Figure 1 illustrates the different grids. Quantities are mapped from one grid to the other via interpolation or padding:

- $P_\mathcal{X}$ is a 2D interpolation operator that interpolates from $\mathcal{C}$ to $\mathcal{X}$,
- $E_\mathcal{X}$ padds the input, defined on grid $\mathcal{X}$ with its boundary values to the computational grid $\mathcal{C}$.

Source representation

I use adjoint interpolation to represent point-sources. This ensures that the point-source will behave like a discretized delta function to some extent. Consider the following 1D example. The source-grid is a single point $\mathcal{S} = \{\frac{1}{2}\}$ while the computational grid is given by $C = \{0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\}$. Define the interpolation operator $P_{\mathcal{S}}$ to interpolate from C to $\mathcal{S}$. The source function on the source grid is a single scalar $\mathbf{q} = 1$. The resulting source functions for linear, quadratic and cubic interpolation are shown in figure 2. For given polynomials $\mathbf{p}^l = [0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1]^l$, we want our discretized delta function to satisfy

$$\langle \mathbf{p}_l, {P_{\mathcal{S}}}^\dagger \mathbf{q} \rangle_C \quad = \quad \left(\tfrac{1}{2}\right)^l, \tag{3}$$

up to some order. We can write this requirement as $\langle P_C \mathbf{p}^l, \mathbf{q} \rangle_{\mathcal{S}} = P_C \mathbf{p}^l = (\frac{1}{2})^l$, from which it follows that the order to which this requirement will be satisfied is determined by the accuracy of the interpolation operator. A numerical verification is shown in table 1.

Modeling operator

The data for one frequency and source is obtained via

$$\mathbf{d}_{ij} = P_{\mathcal{R}} A_{\omega_i} (E_{\mathcal{G}} \mathbf{m})^{-1} {P_{\mathcal{S}}}^\dagger \mathbf{q}_{ij}. \tag{4}$$

The modeling operator F solves (4) for several frequencies $i = 1, \ldots, n_f$ and sources $j = 1, \ldots, n_s$ and organizes the result in a vector.

Linearized modeling opertor

The Jacobian of the modeling operator, $\nabla\mathsf{F}$, for one source and frequency is given by

$$\frac{\partial \mathbf{d}_{ij}}{\partial \mathbf{m}} = P_{\mathcal{R}} A_{\omega_i} (E_{\mathcal{G}} \mathbf{m})^{-1} G_{\omega_i}(\mathbf{m}, \mathbf{u}_{ij}) E_{\mathcal{G}}, \tag{5}$$

where $\mathbf{u}_{ij} = A_{\omega_i} (E_{\mathcal{G}} \mathbf{m})^{-1} {P_{\mathcal{S}}}^\dagger \mathbf{q}_{ij}$ and $G_\omega(\mathbf{m}, \mathbf{u}) = \left(\frac{\partial A_\omega \mathbf{u}}{\partial \mathbf{m}}\right)$. In practice, the Jacobian matrix is never formed explicitly. Instead, its action is computed by solving 2 Helmholtz systems per source and frequency.

Analytic solutions

For some specific cases, analytic expressions exist for the Greens function. For a constant medium with velocity c_0 we have

$$\mathbf{u} = \frac{\iota}{4} H_0^1(\omega \mathbf{r}/c_0), \tag{6}$$

where H_0^1 is a Hankel function of the first kind and $r_i = \sqrt{(x_i - x_s)^2 + (z_i - z_s)^2}$.

For linearly increasing velocity $c_0 + \alpha z$, the analytic solution is given by [7]

$$\mathbf{u} = Q_{\nu - \frac{1}{2}}(\mathbf{s}), \tag{7}$$

where Q_μ is a Legendre function and $\nu = \imath\sqrt{(\omega/\alpha)^2 - \frac{1}{4}}$ and

$$s_i = 1 + \frac{1}{2}\frac{r_i^2}{(z_s + c_0/\alpha)(z_i + c_0/\alpha)}.$$

3. Matrix-free framework

The (linearized) modeling operator is readily implemented in Matlab, but interaction with existing code, especially for linear operators, is not always straightforward. To facilitate incorporation of these blocks into exisisting code, I adopt the SPOT toolbox for matrix-free linear algebra [6]. This allows us, for example, to perform matrix-free matrix-vector products by overloading the Matlab multiplication operation. Furthermore, by insisting on a certain structure of our non-linear operators we can easily adopt ideas form algorithmic differentiation to calculate derivatives of combinations of such operators.

3.1. Linear operators

To illustrate the workings of SPOT, let us consider the Fourier transform. The following code fragment defines an operator that performs the Fourier transform on a vector of length 1000 and performs a forward and inverse FFT on a random vector.

```
n = 1000;          % length
A = opDFT(n);      % define SPOT operator
x = randn(n,1);    % random vector of length n
y = A*x;           % forward FFT
x = A'*y;          % inverse/adjoint FFT
```

Upon multiplying the operator `A` with the vector `x`, Matlab will call the function `A.multiply(x,1)`, wich in turn will perform the Fourier transform using the Matlab built-in function `fft`. Similarly, when multiplying the adjoint with `y`, Matlab will call the function `A.multiply(x,-1)`, which in turn calls `ifft`.

Custom linear operators can be implemented in a similar manner by supplying a function that performs forward and adjoint multiplication. This function can be directly wrapped into a standard SPOT operator using `opFunction`, or the user can define a custom SPOT operator using the `opSpot` class. The latter allows one to overload other Matlab built-in functions, such as `mldivide`, `mrdivide`, `display`, `norm` etc. Composition and addition of operators is also supported. For example, `C = A*B`, where `A` and `B` are SPOT operators, will return a spot operator that multiplies with `B` and subsequently with `A`. The Kronecker product is supported via `C = opKron(A,B)`. A parallel extension of SPOT, pSPOT [8, 9] allows us to define linear operators that work on distributed arrays.

The linear operators can be tested for consistency via the *dot-test*

$$\langle A\mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{x}, A^\dagger \mathbf{y}\rangle, \tag{8}$$

where $\mathbf{y}$ has to be in the range of A. Typically, one takes $\mathbf{x}$ to be a random vector and we can generate $\mathbf{y}$ by acting on another random vector with A. The two should be the same up to numerical precision.

3.2. Non-linear operators

For non-linear operators and functions I use the convection that they return their output (vector or scalar) and the Jacobian or gradient. This structure makes it extremely easy to implement complictated chains of such operations. Furthermore, individual components can be tested seperately to facilitate debugging. One popular way of testing gradients is based on the Taylor series

$$\mathsf{f}(\mathbf{x}_0 + h\delta\mathbf{x}) = \mathsf{f}(\mathbf{x}_0) + h\langle\nabla\mathsf{f}(\mathbf{x}_0), \delta\mathbf{x}\rangle + O(h^2), \tag{9}$$

where $\mathbf{x}_0$ and $\delta\mathbf{x}$ are suitable testvectors. Alternatively, we can use the Taylor series at two different points

$$\mathsf{f}(\mathbf{x}_2) - \mathsf{f}(\mathbf{x}_1) \approx \frac{1}{2}\langle\nabla\mathsf{f}(\mathbf{x}_1) + \nabla\mathsf{f}(\mathbf{x}_2), \mathbf{x}_2 - \mathbf{x}_1\rangle \tag{10}$$

Simimarly, for the non-linear operators we have

$$\mathsf{F}(\mathbf{x}_0 + h\delta\mathbf{x}) = \mathsf{F}(\mathbf{x}_0) + h\nabla\mathsf{F}(\mathbf{x}_1)\delta\mathbf{x} + O(h^2). \tag{11}$$

It is easily verified that the error decays with h as predicted.

4. Implementation and testing

The modeling operator is implemented as a Matlab function `[d,J] = F(m,Q,model)`, where each column of the matrix `Q` represents a gridded source function, `model` is a struct that contains descriptions of the grids and specifies the frequencies. The output is the data `d` and the Jacobian as a SPOT operator `J`, which calls the function `output = DF(m,Q,input,flag,model)` to compute the forward (`flag=1`) or adjoint (`flag=-1`) action of the Jacobian on the vector `input`.

I use a 9-point mixed-grid stencil [10] to discretize the Helmholtz operator and use a sponge boundary condition. The resulting sparse matrix is inverted for all right-hand-sides simultaneously using Matlab's `mldivide`, which (for sparse matrices) is based on an LU decomposition. The loop over frequencies is done in parallel using the Parallel Computing Toolbox, as will be discussed later.

Other building blocks, like interpolation, basic i/o and optimization methods are part of the package. A brief description can be found in the appendix.

Accuracy of the modeling code

The modeling operator is tested against analytic solutions for constant and linearly increasing velocity profiles $v = v_0 + \alpha z$. Figure 3 (a) shows the error for $v_0 = 2000$ m/s, $\alpha = 0$ 1/s and a frequency of 10 Hz as a function of the gridspacing. A slice through the wavefield is shown in figure 3 (b). The analytical and numerical soltions for $v_0 = 2000$ m/s, $\alpha = 0.7$ 1/s and a frequency of 10 Hz is shown in figures 3 (c) and (d) respectively.

Accuracy of the Jacobian

The accuracy of the Jacobian is tested according to (11) for a constant velocity and a random perturbation. The resulting error is shown in figure 4.

Consistency of the Jacobian

The consistency of the Jacobian is tested with the dottest (cf. eq. 8). The result on 10 random vectors is shown in table 2.

Scalability

The modelling code is parallelized over frequencies using Matlab's Parallel Computing Toolbox. We use the *Single Program Multiple Data* (SPMD) functionality of this tool box to divide the computation over the available processors. The basic algorithm is as follows:

```
% distribute frequencies using standard distribution
freq = distributed(model.freq);
spmd
    % define distribution of data array
    codistr  = ...
       codistributor1d(2,[],[nsrc*nrec,nfreq]);
    % get local frequencies
    freqloc  = getLocalPart(freq);
    nfreqloc = length(freqloc);
    % loop over local frequencies
    for k = 1:nfreqloc
        % solve Helmholtz equation for local ...
           frequencies and store resulting data in ...
           Dloc
    end
    % build distributed array from local parts
    D = codistributed.build(Dloc,codistr);
end
```

To test the performance of the parallelization, I compute data for a model of size 1200×404, for 10 sources and 64 frequencies. The compute cluster consists of 36 IBM x3550 nodes, each with 2 quad core Intel 2.6Ghz CPUs, 16 GB RAM and a Volterra x4 inter-processor network. We used 1 processor per node plus one master node (used by Matlab to manage the computation and not counted in the total number of CPUs used). The results for various numbers of CPUs are shown in table 3.

5. Examples

5.1. Seismic Waveform Inversion

Waveform inversion is a procedure to obtain gridded medium parameters form seimic data by solving e a non-linear data-fitting problem of the form

$$\min_{\mathbf{m}} \mathsf{w}\left(\mathsf{F}(\mathbf{m}) - \mathbf{d}\right), \tag{12}$$

where w is some differentiable penalty function. While the classical formulation of this problem for seismic applications uses a least-squares penalty [11], many other penalties have been proposed in this context.

A standard optimization-procedure relies on iterative updating of the model:

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k, \tag{13}$$

where the search direction $\mathbf{s}_k$ is obtained from the gradient of the objective

$$\nabla \mathsf{f}(\mathbf{m}_k) = \nabla \mathsf{F}(\mathbf{m}_k)^\dagger \nabla \mathsf{w}\left(\mathsf{F}(\mathbf{m}_k) - \mathbf{d}\right) \tag{14}$$

in some way (e.g., Gauss-Newton, Quasi-Newton or CG).

Alternative model representations are easily included by replacing $\mathbf{m}$ with $B\mathbf{x}$, where B is some basis (e.g., splines). The corresponding gradient is obtained easily by projecting the gradient w.r.t. $\mathbf{m}$ onto this basis with $B^\dagger$.

The least-squares penalty is be implemented as

```
function [f,g] = w_ls(r);
  f = .5*norm(r)^2; % two-norm of residual
  g = r;            % gradient
end
```

The least-squares misfit of the form

$$\mathsf{f}_{\mathsf{LS}}(\mathbf{m}) = \frac{1}{2}\|\mathsf{F}(\mathbf{m}) - \mathbf{d}\|_2^2,$$

is then implemented as

```
function [val,grad] = f_ls(m,Q,dt,model);
  [d,DF]     = F(m,Q,model); % predicted data
  r          = d - dt;       % residual
  [val,df]   = w_ls(r);      % penalty
  grad       = DF'*df;       % gradient
end
```

This function can be passed on to any black-box optimization method that uses only function-values and gradients. A Gauss-Newton method can be easily implemented by having the function return the Gauss-Newton Hessian as SPOT operator.

To illustrate the approach we use the BG Compass velocity model depicted in figure 5 (a). We generate data for this model for 71 sources and 281 receivers, all equi-spaced and located at the top of the model. For the inversion we use a *multi-scale* strategy [12], inverting the data in consecutive frequency bands $f_1 = \{2.5, 3, 3.5\}$, $f_2 = \{5.5, 6, 6.5\}$, $f_3 = \{11.5, 12, 12.5\}$, $f_4 = \{17.5, 18, 18.5\}$, each time using the end result as initial guess for the next frequency band. The initial model for the first frequency band is depicted in figure 5 (b). We use 10 L-BFGS iterations for each frequency band. The results are depicted in figure 6.

5.2. Reverse time migration

A seismic image can be obtained from reflection data via backpropagation

$$\delta\mathbf{m} = \nabla\mathsf{F}^\dagger\delta\mathbf{d}. \tag{15}$$

This is referred to as *reverse-time migration* in the geophysical literature. The fact that this simple backprojection yields a useable image at all is due to the fact that the Normal operator $\nabla\mathsf{F}^\dagger\nabla\mathsf{F}$ acts as local scaling and filtering operation only [13]. Many approaches have been developed to estimate the inverse of this filter without explicitly inverting the normal operator. Following [14] we estimate the image as

$$\delta\mathbf{m} = W_\mathbf{x}\nabla\mathsf{F}^\dagger\mathrm{diag}(\mathbf{w}_\omega)\mathbf{d}, \tag{16}$$

where $W_\mathbf{x}$ represents the spatial weighting and $\mathbf{w}_\omega$ represents a frequency-weighting $|\omega|^{-1}$. Herrmann et al. [14] take $W_\mathbf{x}$ to be a diagonal weighting in the Curvelet domain. For the sake of this example, however, we use a simple spatial weighting that corrects for the geometric spreading: $W_\mathbf{x} = \mathrm{diag}(\sqrt{z})$.

The following code implements this procedure:

```
% reverse time migration for given data d and ...
   background model m0

% get data and Jacobian
[d0,DF] = F(m0,Q,model);

% weighting with |\omega|^{-1}
Wf = oppKron2Lo(opDiag(1./freq),opDirac(nsrc*nrec));

% depth weighting
Wz = opKron(opDirac(n(2)),opDiag(sqrt(z)));

% reverse-time migration
dm = Wz*DF'*(Wf*(d - d0));
```

Note the use of the parallel Kronecker product `oppKron2Lo` which works on arrays that are distributed along the last dimension (frequency, in this case). To avoid artifacts due to non-reflected events in the data, I subtract data for the background model. I illustrate the algorithm on the Marmousi model, depicted in figure 7 (a), using 101 sources, 201 receivers (equispaced at the top of the model) and frequencies between 2.5 and 20 Hz with 0.5 Hz increments. The background model, depicted in figure 7 (b), is a smooth version of this model. The true perturbation (i.e., the difference between the true and background model) is shown in figure 7 (c). The resulting image is shown in figure 7 (d).

5.3. Seismic tomography

Tomographic inversion relies on traveltime information to invert for the velocity. Traditionally, traveltimes are manually picked from the data and predicted traveltimes are computed using a geometric-optics approximating as is also used x-ray tomography. The last two decades there has been an increasing interest in generalizing this approach to finite-frequency regime where wave propagation is no longer accurately modeled via rays. One way to define the *finite-frequency* traveltime is to consider the correlation of the observed data with the data for some reference model, and pick the maximum [15]. The observed traveltime *difference* w.r.t. to the reference model for source i and receiver j is then given by

$$\delta t_{ij}^{\text{obs}} = \underset{t}{\text{argmax}} \sum_{\omega} \left(d_{ij}(\omega)\right)^{*} d_{ij}^{\text{obs}}(\omega) e^{\imath\omega t}, \tag{17}$$

where $d_{ij}^{\text{obs}}(\omega)$ is the observed data and $d_{ij}(\omega)$ is the modeled data for the reference model (i.e., $\mathbf{d} = \mathsf{F}(\mathbf{m}_0)$). By correlating the data for the reference model with predicted data for a model $\mathbf{m}$ we can compute the predicted traveltime difference $\delta\mathbf{t}(\mathbf{m})$ in a similar manner. The goal then is to find a model $\mathbf{m}$ such that $\delta\mathbf{t}(\mathbf{m}) \approx \delta\mathbf{t}^{\text{obs}}$. For the purpose of inversion the relation between the traveltime difference and the model perturbation $\delta\mathbf{m} = \mathbf{m} - \mathbf{m}_0$ is usually linearized, yielding

$$\delta t_{ij} = \frac{\sum_{\omega} \imath\omega \left(d_{ij}(\omega)\right)^{*} \delta d_{ij}(\omega)}{\sum_{\omega} \omega^2 |d_{ij}(\omega)|^2}, \tag{18}$$

where $\delta d_{ij}(\omega)$ is the linearized data for the model perturbation $\delta\mathbf{m}$ (i.e., $\delta\mathbf{d} = \nabla\mathsf{F}(\mathbf{m}_0)\delta\mathbf{m}$). We can write this relation as $\delta\mathbf{t} = K\delta\mathbf{m}$, where the rows of K are the so-called *sensitivity kernels*. For an in-depth overview and further references we refer to [16].

In case of constant reference velocity we can implement the (linearized) scattering operator using the Greens function:

$$\delta\mathbf{d}(\omega) = \omega^2 G_r(\omega)\text{diag}(\delta\mathbf{m})G_s(\omega), \tag{19}$$

where G_r denotes the Greens function from the receivers to the subsurface and G_s denotes the Greens function from the subsurface to the sources.

```
function output = Ktomo(v0,input,flag)
```

```
% get reference wavefield and scale factor
for k = 1:nfreq
    d_ref(:,:,k) = ... % reference data
    scale = scale + omega(k)^2*abs(d_ref(:,:,k));
end

% input
if flag > 0
    output = zeros(nrec,nsrc);
else
    input  = reshape(input,[nrec,nsrc])./scale;
    output = zeros(nz*nx,1);
end


for k = 1:nfreq % loop over frequencies
    Gs  = ...   % source wavefield
    Gr  = ...   % receiver wavefield

    if flag > 0 % forward mode
        d_scat = omega(k)^2*Gr*diag(input)*Gs;
        output = output + ...
           1i*omega(k)*conj(d_ref(:,:,k)).*d_scat;
    else        % adjoint mode
        input_k = -1i*omega(k)*d_ref(:,:,k).*input;
        output  = output + ...
           omega(k)^2*diag(Gr'*input_k*Gs');
    end
end

% output
if flag > 0
    output = vec(output./scale);
end

end
```

To illustrate the method, I use the velocity model depicted in figure 8 (a). The data are modeled for frequencies 0.5 Hz to 20 Hz with 0.5 Hz increments and 21 sources and 21 receivers which are equally distributed along vertical lines on the left and right side of the model at $x = 10$ and $x = 990$ respectively. As outlined above, the traveltime data are obtained by correlating this data with the data for the reference model (a constant velocity of 2000 m/s in this case), and picking the maximum value for each source-receiver pair. The resulting traveltime data is shown in figure 8 (b). An example of

the corresponding waveform data for the true and reference model is shown in figures 8 (c,d). The correlation panel and the picked traveltimes are shown in figure 8 (c). The scattering operator is implemented using the analytic expression for the Greens function discussed earlier. An example of the sensitivity kernel for one source and receiver is shown in figure 9 (a). Smoothness is imposed on the solution by representing the model on a coarser grid with cubic splines. For this example, the spline grid is defined as $\mathcal{B}_x = \{100, 300, 500, \ldots, 900\}$ and $\mathcal{B}_z = \{100, 200, 300, \ldots, 900\}$. For the the spline evaluation I use `opSpline1D` with Dirichlet boundary conditions in the z direction and Neumann boundary conditions in the x direction. Matlab's `lsqr` is used to solve the resulting linear system. The results after 10 iterations is shown in figure 9 (b). The code to perform this experiment is given by

```
% spline grid
xs = [100:200:900];
zs = [100:100:900];

% spline operators
Bx = opSpline1D(xs,x,[1 1]);
Bz = opSpline2D(zs,z,[0 0]);
B  = opKron(Bx,Bz);

% wrap function Ktomo in SPOT operator
K  = opFunction(...

% call LSQR
dm = lsqr(K*B,T,1e-6,10);
```

6. Conclusion and discussion

I present a flexible and scalable framework for prototyping algorithms for seismic imaging and inversion. The code can be useful for other applications, such as Medical imaging [17].

Separate pieces of the code can be easily swapped out. For example, we can use a different wave-equation and leave the rest of the code unchanged. The use of operator overloading allows us to easily interface with existing optimization codes.

The object-oriented approach, of which this framework is an example, allows us to design complicated algorithms from relatively simple, easy to maintain, building blocks. Although the current implementation is done in Matlab, a similar framework can be realized in lower level languages using existing libraries such the Rice Vector Library.

Future work will be aimed at further abstraction of data-structures. Matlab already allows us to work with distributed arrays with relative ease by taking care of communication automatically. In a similar fashion, we can use the `memmap` functionality to

acces data that is stored on disk as if it was a Matlab array. Such an approach would allow us to scale the same algorithms to very large data-sets, for example by adopting Google's MapReduce model [18].

Acknowledgments

I thank everybody at the Seismic Laboratory for Imaging and Modeling (SLIM) at UBC for their feedback and help with developing this software. The BG Compass velocity model was made available by the BG Group. This work was in part financially supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant (22R81254) and the Collaborative Research and Development Grant DNOISE II (375142-08). This research was carried out as part of the SINBAD II project with support from the following organizations: BG Group, BP, Chevron, ConocoPhillips, Petrobras, Total SA, and WesternGeco.

[1] J. Claerbout, S. Fomel, IMAGE ESTIMATION BY EXAMPLE: Geophysical Soundings Image Construction, 2012.
URL http://sepwww.stanford.edu/sep/prof/gee8.08.pdf

[2] W. W. Symes, D. Sun, M. Enriquez, From modelling to inversion: designing a welladapted simulator, Geophysical Prospecting 59 (5) (2011) 814–833. doi:10.1111/j.1365-2478.2011.00977.x.

[3] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.
URL http://dl.acm.org/citation.cfm?id=266469.266486

[4] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley, An overview of the trilinos project, ACM Trans. Math. Softw. 31 (3) (2005) 397–423. doi:10.1145/1089014.1089021.

[5] A. D. Padula, S. D. Scott, W. W. Symes, A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms, ACM Transactions on Mathematical Software 36 (2) (2009) 1–36. doi:10.1145/1499096.1499097.

[6] E. van den Berg, M. P. Friedlander, Spot A Linear-Operator Toolbox (2009).
URL http://www.cs.ubc.ca/labs/scl/spot/

[7] B. N. Kuvshinov, W. A. Mulder, The exact solution of the time-harmonic wave equation for a linear velocity profile, Geophysical Journal International 167 (2) (2006) 659–662. doi:10.1111/j.1365-246X.2006.03194.x.

[8] H. Modzelewski, T. Lai, S. Jalali, pSPOT, a parallel extension of SPOT (2010).
URL https://github.com/slimgroup/pSPOT

[9] S. Pacteau, pSPOT, a parallel linear operator toolbox, Tech. rep. (2010).
URL http://slim.eos.ubc.ca/Publications/Public/TechReports/Internship_report_last.pdf

[10] C.-H. Jo, An optimal 9-point, finite-difference, frequency-space, 2-D scalar wave extrapolator, Geophysics 61 (2) (1996) 529. doi:10.1190/1.1443979.

[11] A. Tarantola, A. Valette, Generalized nonlinear inverse problems solved using the least squares criterion, Reviews of Geophysics and Space Physics 20 (2) (1982) 129–232.

[12] L. Sirgue, R. G. Pratt, Efficient waveform inversion and imaging: a strategy for selecting temporal frequencies, Geophysics 69 (1) (2004) 231–248. doi:10.1190/1.1649391.

[13] W. W. Symes, Approximate linearized inversion by optimal scaling of prestack depth migration, Geophysics 73 (2) (2008) R23–R35. doi:10.1190/1.2836323.

[14] F. J. Herrmann, P. P. Moghaddam, C. C. Stolk, Sparsity- and continuity-promoting seismic image recovery with curvelet frames, Applied and Computational Harmonic Analysis 24 (2) (2008) 150–173. doi:10.1016/j.acha.2007.06.007.

[15] Y. Luo, Wave-equation traveltime inversion, Geophysics 56 (5) (1991) 645. doi:10.1190/1.1443081.

[16] A. Fichtner, Full Seismic Waveform Modelling and Inversion, Springer Verlag, 2010.
URL http://www.springer.com/earth+sciences+and+geography/geophysics/book/978-3-642-15806-3

[17] F. Natterer, F. Wübbeling, Mathematical Methods in Image Reconstruction, Mathematical Modeling and Computation, SIAM, 2001. doi:10.1137/1.9780898718324.

[18] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04, USENIX Association, Berkeley, CA, USA, 2004, pp. 10–10.
URL http://dl.acm.org/citation.cfm?id=1251254.1251264

	linear	quadratic	cubic
0	0.000000e+00	0.000000e+00	0.000000e+00
1	0.000000e+00	5.551115e-17	5.551115e-17
2	1.000000e-02	0.000000e+00	0.000000e+00
3	1.500000e-02	3.000000e-03	0.000000e+00
4	1.510000e-02	5.100000e-03	9.000000e-04
5	1.275000e-02	5.550000e-03	2.250000e-03

Table 1: Accuracy of the discretized delta function for $l = 0, 1, \ldots, 5$ using linear, quadratic and cubic interpolation.

$\Re\langle A\mathbf{x}, \mathbf{y}\rangle$	$\Re\langle \mathbf{x}, A^\dagger\mathbf{y}\rangle$	error
2.453000e+07	2.453000e+07	2.220400e-14
1.762400e+08	1.762400e+08	3.552700e-15
1.896000e+08	1.896000e+08	1.421100e-14
-4.902800e+07	-4.902800e+07	7.194200e-14
1.295700e+08	1.295700e+08	3.197400e-14
-4.529400e+08	-4.529400e+08	3.197400e-14
-1.711600e+08	-1.711600e+08	3.206300e-13
-3.416800e+08	-3.416800e+08	0.000000e+00
-8.159100e+08	-8.159100e+08	1.421100e-14
3.125000e+07	3.125000e+07	7.993600e-15

Table 2: Dot-test of Jacobian on random test vectors.

# of procs	time [s]	efficiency
1.00	2459.00	1.00
2.00	1224.00	1.00
4.00	615.00	1.00
8.00	313.00	0.98
16.00	188.00	0.82

Table 3: We compute data for a 1200x404 model for 64 frequencies and 10 sources using different numbers of CPUs. The times are averages over 5 runs.

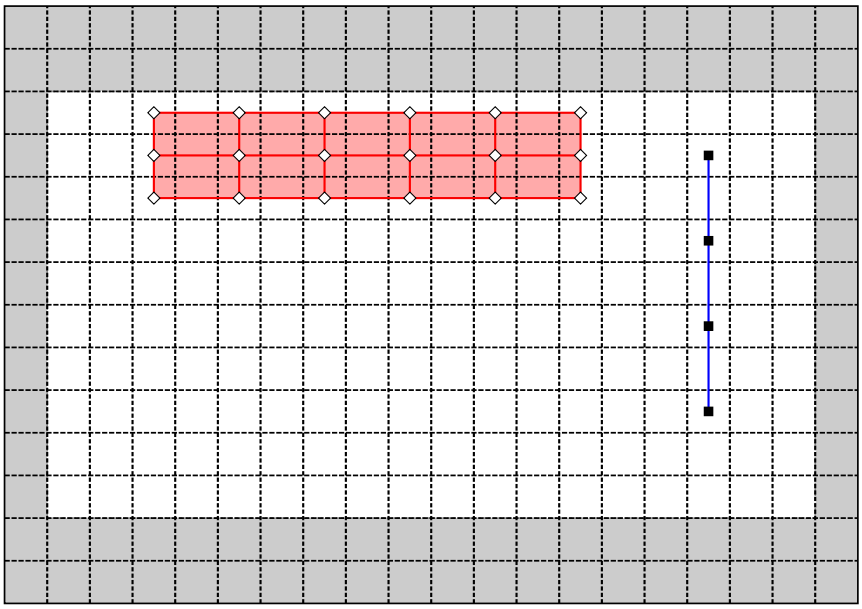

Figure 1: Illustration of grids used for modeling. The physical grid (white) is extended on all sides (grey) to define the computational grid. Examples of the source and receiver grids are also shown.

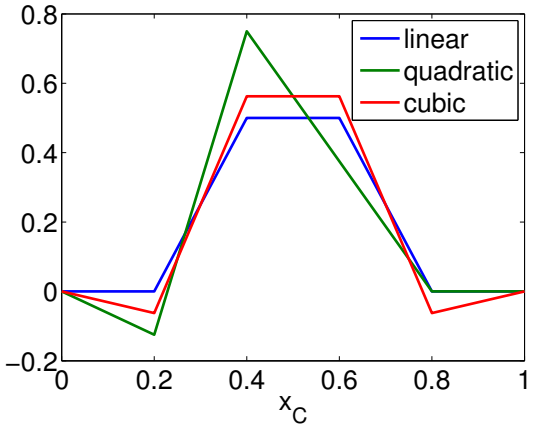

Figure 2: Discretization of a delta function $\delta(x - \frac{1}{2})$ using linear, quadratic and cubic interpolation.

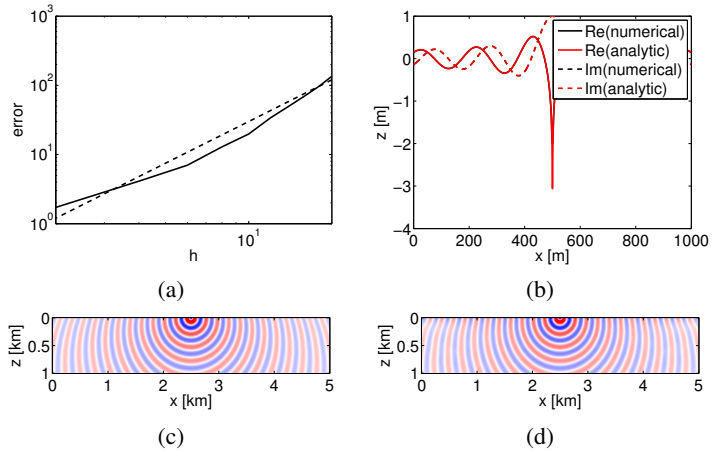

Figure 3: Accuracy of the modeling operator. (a) error between analytic and numerical solution for constant velocity $v_0 = 2000$ m/s and a frequency of 10 Hz. on a domain of 1000 m× 1000 m with a point source in the center. (b) Wavefields at $z = 500$m. (c-d) Analytic and numerical solution for linear velocity profile $2000 + 0.7z$ and a frequency of 10 Hz.

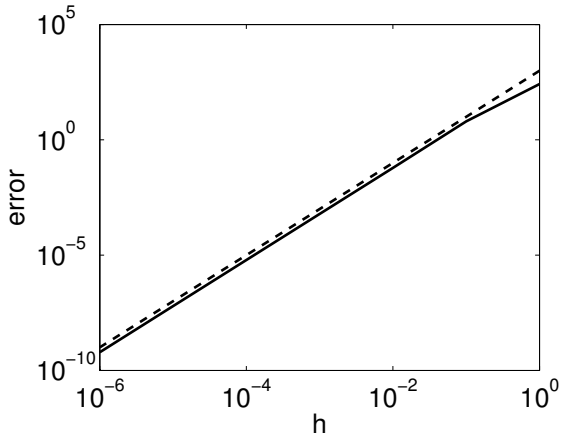

Figure 4: bla

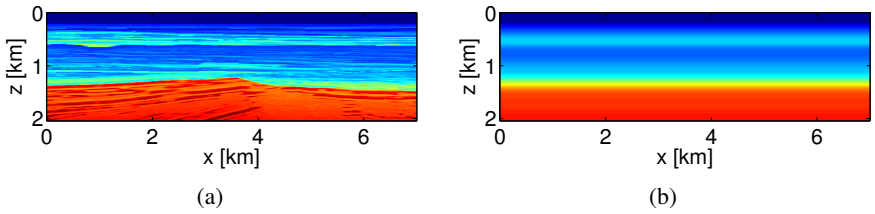

Figure 5: (a) BG Compass velocity model and (b) initial model for FWI example.

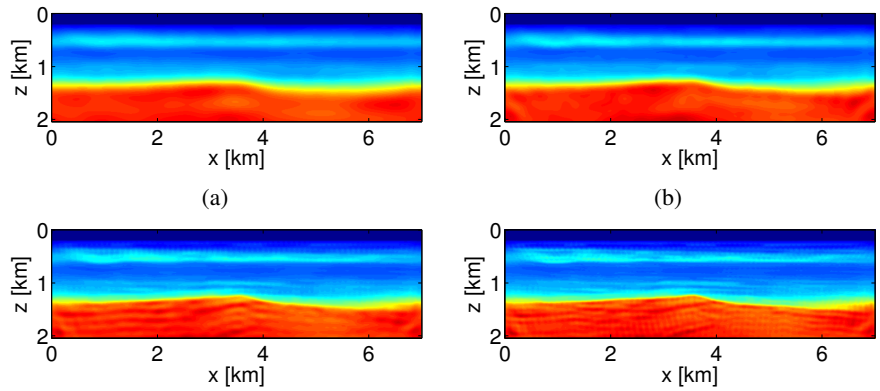

Figure 6: Result of the inversion of subsequent frequency bands: (a) 2.5-3.5 Hz, (b) 5.5-6.5 Hz, (c) 11.5-12.5 Hz and (d) 17.5-18.8 Hz, each after 10 L-BFGS iterations using the previous result as starting model.

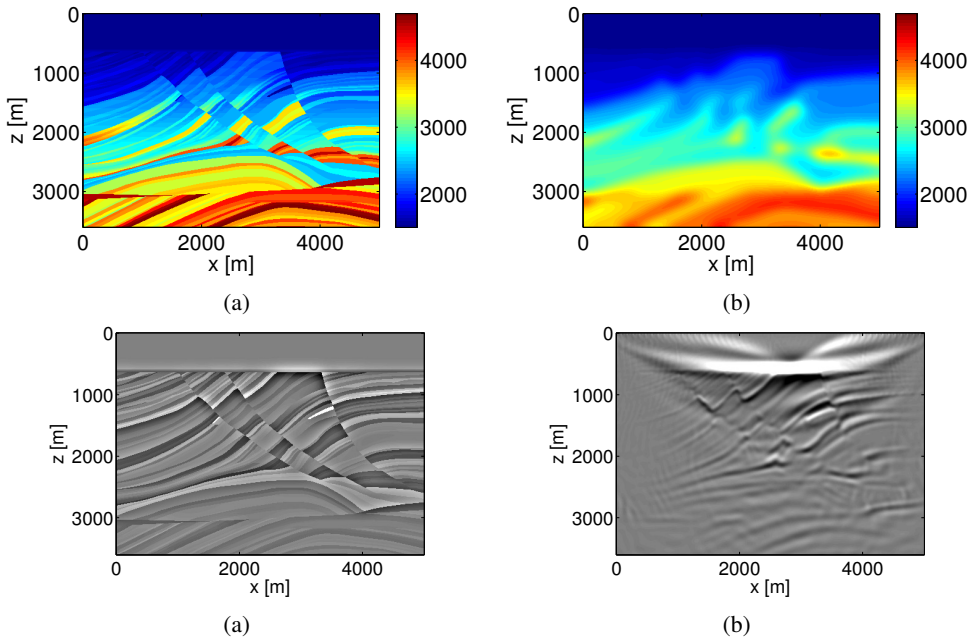

Figure 7: (a) Marmousi velocity model (m/s). (b) Background model used for imaging (m/s). (c) true perturbation. (d) image.

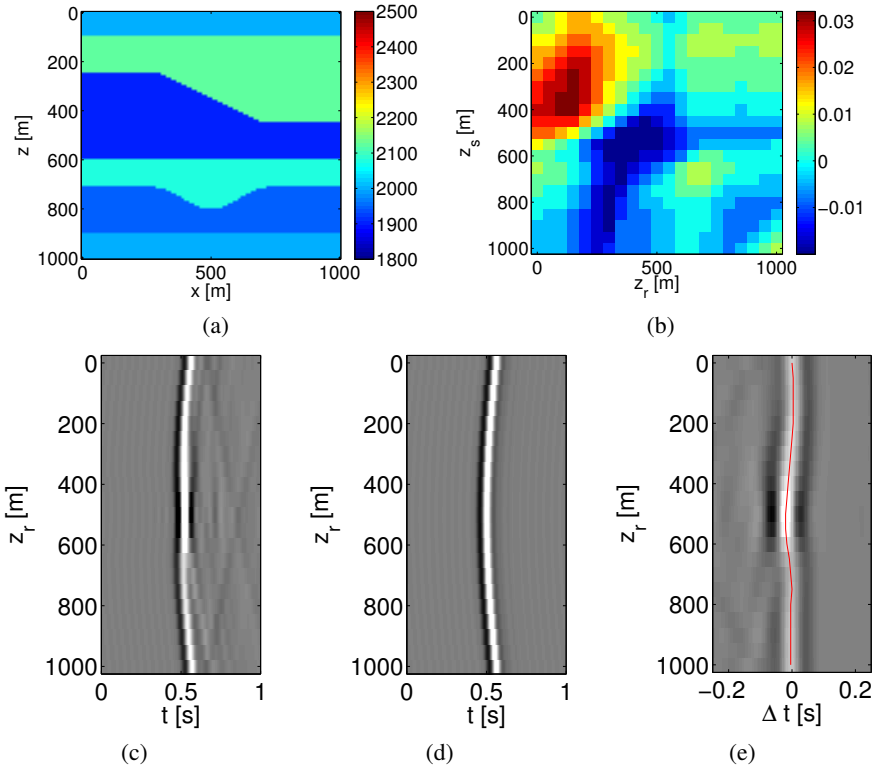

Figure 8: The velocity model (m/s) used in tomographic example is shown in (a), the corresponding traveltime data (s) for each source-receive pair is shown in (b). The waveform data for a source located at $x = 10, z = 500$ is depicted in (b); (c) shows the data for the initial homogeneous velocity (2000 m/s) and (d) shows the correlation of these two and the picked traveltimes.

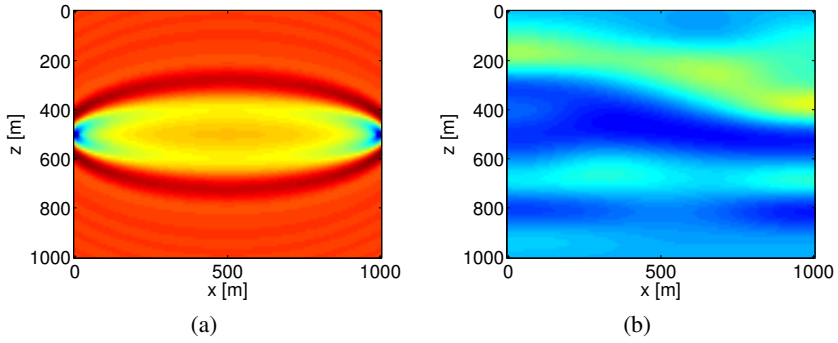

Figure 9: (a) sensitivity kernel for one source-receiver pair. (b) reconstruction after 20 LSQR iterations on the same colorscale as figure 8 (a).

Appendix A. Matlab Package

The Matlab code can be downloaded from `https://www.slim.eos.ubc.ca/releases`. It has been tested with Matlab 2011b. The Parallel Computing Toolbox is needed to take advantage of the parallelism, though the code will run in serial model without the toolbox as well.

Below is a short description of the contents of the package.

tools/utilities/

- `SPOT-SLIM` – SPOT toolbox
- `pSPOT` – pSPOT toolbox

tools/algorithms/2DFreqModeling/

- `F` – Modelling operator
- `DF` – Jacobian of `F`
- `oppDF` – pSPOT constructor of Jacobian, uses `DF`
- `G` – Modelling operator for velocity of the form $v(z) = v_0 + \alpha z$
- `legendreQ` – evaluates the Legendre Q function $Q_\nu(z)$
- `Helm2D` – discretized Helmholtz operator

tools/operators/misc/

- `opLinterp1D` – 1D cubic Lagrange interpolation
- `opLinterp2D` – 2D linear Lagrange interpolation
- `opSpline1D` – 1D Cubic Spline evaluation
- `opSpline2D` – 2D Cubic Spline evaluation
- `opExtension` – 1D padding
- `opDFTR` – Fourier transform for real signals; returns only positive frequencies.

tools/functions/misc/

- `vec`, `invvec` – vectorize (distributed) array and reshape (distributed) – vector back to (distributed) n-dimensional array.
- `odn2grid, grid2odn` – convert offset-increment-size vectors to grid vectors and vice-versa.
- `odnread`, `odnwrite` – read/write for gridded data.

tools/solvers/QuasiNewton/

- `lbfgs` – L-BFGS method with weak Wolfe linesearch

applications/SoftwareDemos/2DSMII

- `mbin/f_ls` – Least-squares misfit and gradient
- `mbin/w_ls` – compute two-norm of vector and its gradient.
- `mbin/Ktomo` – Scattering operator for traveltime tomography example
- `scripts` – scripts to produce examples
- `doc` – documentation in html format