gerjoii

Imaging using EM waves

summer 2017

1 Forward

Using the finite difference scheme solve for u,

$$\mathbf{v} = (H_z,\ -H_x)^\top \qquad u = E_y$$
$$\mathbf{s} = (0,\ 0,\ -J_y)^\top \qquad \mathbf{w} = (\mathbf{v},\, u)^\top$$

$$\underbrace{\begin{bmatrix} \mu_o 1_2 & 0 \\ 0 & \varepsilon \end{bmatrix}}_{A} \dot{\mathbf{w}} = \underbrace{\begin{bmatrix} 0 & \nabla^\top \\ \nabla & 0 \end{bmatrix}}_{\mathrm{D}} \mathbf{w} - \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & \sigma \end{bmatrix}}_{B} \mathbf{w} + \mathbf{s},$$
$$\mathcal{L} = A - \mathrm{D} + B.$$

over a rectangular region Ω (simulating a slice in depth of the earth) with absorbing boundary conditions everywhere. Matrix A controls *velocity*, matrix B controls *attenuation*.

Below is a brief list/explanation of what each method does. Some methods do more than explained here, take more parameters, and output more stuff. This is just for a comprehensive idea.

- Build geometric parameters from target material properties and chosen characteristic frequency,

$$\varepsilon,\, \sigma,\, \left\{ \begin{matrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{matrix} \right\} = \texttt{w\_geom}(c_o,\, \varepsilon,\, \sigma,\, f_{Ny},\, xz_{\textsf{plane}})$$

- Build M. In this case, M are just coordinates of receivers,

$$Mu = u_r.$$

- Compute wave and error,

$$u,\, e,\, \begin{Bmatrix} c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} \\ \text{cpml}\,\mathbf{E},\, \text{cpml}\,\mathbf{H} \end{Bmatrix} = \texttt{w\_fwd}\left(\varepsilon,\, \sigma,\, \mathbf{s},\, M,\, d^o,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix} \right).$$

  this method needs some lengthy constructs before it actually solves the wave. the order is as below.

  - Expand ε, σ to be in PML,

  $$\varepsilon,\, \sigma = \texttt{w\_exp2pml}\left(\varepsilon,\, \sigma,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix} \right).$$

  - Build PML and $\mathbf{H}, \mathbf{E}$ finite difference coefficients in PML,

  $$\begin{Bmatrix} \text{cpml}\,\mathbf{E} \\ \text{cpml}\,\mathbf{H} \end{Bmatrix} = \texttt{w\_cpml}\left(\varepsilon,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix} \right).$$

  - Build coefficients for fields in finite difference scheme,

  $$c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} = \texttt{w\_coeff}\left(\varepsilon,\, \sigma,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix} \right).$$

  - Solve for wave

  $$u = \texttt{w\_solve}\left(M,\, \mathbf{s},\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, \begin{Bmatrix} c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} \\ \text{cpml}\,\mathbf{E},\, \text{cpml}\,\mathbf{H} \end{Bmatrix} \right).$$

  this last method propagates the wave. At each time-step t,

  $$d_t = Mu_t \qquad \rightarrow \qquad d = \{d_t \text{ for all } t\}.$$

  - Compute e,

  $$e = d - d^o.$$

2 Inverse

Our continuous objective function is

$$\mathsf{E}(\varepsilon,\,\sigma) = \int_0^{\top} (\underbrace{d(\varepsilon,\,\sigma,\,t) - d^o(t)}_{e})^2 \,\mathrm{d}t$$

We optimize E by setting $\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathsf{E}$ and $\frac{\mathrm{d}}{\mathrm{d}\sigma}\mathsf{E}$ equal to zero and solving for the parameters,

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathsf{E} = \int_0^T \lambda\dot{u}\,\mathrm{d}t \qquad\qquad \frac{\mathrm{d}}{\mathrm{d}\sigma}\mathsf{E} = \int_0^T \lambda u\,\mathrm{d}t$$

where λ is the *adjoint* wavefield and e^* is the *time reversed error*,

$$\mathcal{L}^*\lambda = e^*, \qquad \lambda(T) = \dot{\lambda}(T) = 0.$$

Below is a brief list/explanation of what each method does. Some methods do more than explained here, take more parameters, and output more stuff. This is just for a comprehensive idea.

- Compute $\nabla_\sigma \mathsf{E}^\top$ using cross-correlation of u and its adjoint field λ,

$$g_\sigma = \texttt{w\_grad}\left(u,\, e,\, M,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, \begin{Bmatrix} c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} \\ \mathsf{cpml}\,\mathbf{E},\, \mathsf{cpml}\,\mathbf{H} \end{Bmatrix}\right).$$

  - Solve for λ. Source is error e but time-reversed ($\mathbf{s} = e^*$)

$$\lambda = \texttt{w\_solve}\left(\varepsilon,\, \sigma,\, e^*,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, \begin{Bmatrix} c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} \\ \mathsf{cpml}\,\mathbf{E},\, \mathsf{cpml}\,\mathbf{H} \end{Bmatrix}\right).$$

  At each time-step t,

$$g_t = u_t^* \odot \lambda_t \qquad\qquad g_\sigma \leftarrow g_t + g_{t-1}.$$

- Compute derivative of u with resect to time,

$$\dot{u} = \texttt{w\_dt}\left(u,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix}\right).$$

- Compute $\nabla_\varepsilon \mathsf{E}^\top$ using cross-correlation of $\dot{u}$ and its adjoint field λ,

$$g_\varepsilon = \texttt{w\_grad}\left(\dot{u},\, e,\, \varepsilon,\, \sigma,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, \begin{Bmatrix} c\mathbf{J},\, c\mathbf{E},\, c\mathbf{H} \\ \textsf{cpml}\,\mathbf{E},\, \textsf{cpml}\,\mathbf{H} \end{Bmatrix}\right).$$

- Compute step size α for update. Requires one extra forward run for each.

$$\alpha_\sigma = \texttt{w\_pica\_s.m}\left(\varepsilon,\, \sigma,\, \mathbf{s},\, M,\, d^o,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, e,\, g_\sigma,\, k_\sigma\right),$$
$$\alpha_\varepsilon = \texttt{w\_pica\_e.m}\left(\varepsilon,\, \sigma,\, \mathbf{s},\, M,\, d^o,\, \begin{Bmatrix} \Delta x,\, \Delta z,\, \Delta t \\ n,\, m,\, nt \\ x,\, z,\, t \end{Bmatrix},\, e,\, g_\varepsilon,\, k_\varepsilon\right).$$

  For example,

$$d_\bullet = d(\varepsilon,\, \sigma + k_\sigma g_\sigma) - d(\varepsilon,\, \sigma),$$
$$\alpha_\sigma = k_\sigma \frac{d_\bullet(:)^\top \cdot e_w(:)}{d_\bullet(:)^\top \cdot d_\bullet(:)},$$

  where $a(:)$ is borrowed from the Matlab command that turns matrix a to vector form.

3 File names

Constructors

- `w_geom.m` builds geometric parameters.
- `w_epsilon.m` build ε.
- `w_sigma.m` build σ.
- `w_magmat.m` build magnetic parameters (constant and homogeneous) $\sigma_{\mathsf{mag}}, \mu$.
- `w_exp2pml.m` expand ε, σ inside PML.
- `w_cpml.m` build PML and coefficients for $\mathbf{E}$, $\mathbf{H}$.
- `w_coeff.m` build finite difference coefficients for $\mathbf{E}$, $\mathbf{H}$, $\mathbf{J}$.
- `w_s.m` specify source position in index form.
- `w_M.m` specify receiver positions in index form.

Procedures

- `w_wavelet.m` compute source wavelet.
- `w_solve.m` propagates the wave u. This method has local functions,
  - `new_Ez.m` + `new_Ez_pml.m` updates u,
  - `new_H.m` + `new_H_pml.m` updates $\mathbf{v}$.
- `w_fwd.m` computes u, e.
- `w_dt.m` computes $\dot{u}$.
- `w_grad.m` computes adjoint λ and cross-correlates u with λ. This method does not reference `w_solve.m`, it redoes its work, so it also has local functions,
  - `new_Ez.m` + `new_Ez_pml.m` updates u,
  - `new_H.m` + `new_H_pml.m` updates $\mathbf{v}$.
- `w_gd_s.m` performs gradient descent on E solving for σ.
- `w_gd_e.m` performs gradient descent on E solving for ε.
- `w_pica_s.m` computes descent stepsize for g_σ.

- w_pica_e.m computes descent stepsize for g_ε.
- w_gd_s_stoch.m performs stochastic gradient descent on $\mathsf{E} = \sum_i \mathsf{E}_i$ solving for σ.
- w_gd_e_stoch.m performs stochastic gradient descent on $\mathsf{E} = \sum_i \mathsf{E}_i$ solving for ε.

Examples

- w_run.m
- w_run_many.m
- w_inv.m
- w_inv_many.m