simpegEM1D Documentation

Release 0.0.1

Seogi Kang

August 04, 2014

Contents

SimPEG (Simulation and Parameter Estimation in Geophysics) is a python package for simulation and gradient based parameter estimation in the context of geoscience applications.

simpegEM1D uses SimPEG as the framework for the forward and inverse problems for electromagnetic (EM) problemes assuming layered earth. Here, we focus on airbonre EM application.

CHAPTER 1

Contents

1.1 AirEM1D

1.1.1 Motivation

Airborne Electromagnetic (EM) methods in geophysical applications has been successfully applied for several decades to map interesting geological structure of the earth in large scale. A natural way to categorize this airborne EM methods might be frequency domain and time domain EM systems. Famous frequency domain systems are DIGEHM and RESOLVE of CGG; time domain systems are VTEM of Geotech and AeroTEM of Aeroquest. Each instrument has its own advantage and disadvantage depends on purposes of geophysical survey so that indentifying those are crucial for successful geophysical application.

One of the most used interpretation tools of these airborne EM data is 1D inversion, which assumes the earth structure as layers. Since we can derive solutions for this case pseudo-analytically, this can be evaluated relatively fast compared to solving differential equations in 2D or 3D. Therefore, this is really useful tool that we can use for first order survey design and interpretation in reaility. Furthermore, this is really nice education tool for students who are studying geophysics, since they can play with EM responses by manipulating conductivity or susceptibiltiy of the layered earth. While they are playing with this tool, if they want to recognize EM responses more seriously, then they can see how we derived these responses.

However, although it has been more than ten years since these tools were developed, as far as I know, there are no avaiable open source, modular, well-documented 1D EM forward modeling and inversion program that we can use for airborn EM applications. Therefore, here, we try to make this program applicable for both

- Practical applications for most airborne EM system (real data inversion)
- Education tools (easy implentation and well-documented)

In order to satisfy those components, first we derive solutions of frequency and time domain EM problems, and develop some modules that we can compute forward EM responses. We use SimPEG's frame work, to make this algorithm modular. Next, we apply inversion frame in SimPEG to our forward problem.

1.1.2 Forward problem

Freqeuncy domain EM

Maxwell's equations in frequency domain can be written as

$$= -\imath\omega \tag{1.1}$$

$$\amalg =_s \tag{1.2}$$

where \(\E\)

Euler's identity, equation (**??**), was elected one of the most beautiful mathematical formulas

Time domain EM

1.1.3 Inverse problem

1.2 AirEM1D code

Here, we used SimPEG's frame work so that we have following modules:

- Problem: EM1D
- Survey: BaseEM1Dsurvey
- Mapping: BaseEM1Dmap

1.2.1 EM1D problem

class simpegem1d.EM1D.**EM1D**(*mesh*, *mapping=None*, ***kwargs*)
Bases: SimPEG.Problem.BaseProblem

Pseudo analytic solutions for frequency and time domain EM problems assuming Layered earth (1D).

CondType = 'Real'

HzKernel_layer(*lamda*, *f*, *nlay*, *sig*, *chi*, *depth*, *h*, *z*, *flag*)
Kernel for vertical magnetic component (Hz) due to vertical magnetic diopole (VMD) source in (kx,ky) domain

HzkernelCirc_layer(**args*, ***kwargs*)
Kernel for vertical magnetic component (Hz) at the center due to circular loop source in (kx,ky) domain

$$H_z = \frac{Ia}{2}\int_0^\infty [e^{-u_0|z+h|} + r_{TE}e^{u_0|z-h|}]\frac{\lambda^2}{u_0}J_1(\lambda a)]d\lambda$$

To use *HzkernelCirc_layer* method, SimPEG requires that the survey be specified.

Jtvec(**args*, ***kwargs*)
Computing Jacobian^T multiplied by vector.

To use *Jtvec* method, SimPEG requires that the survey be specified.

Jtvec_approx(*m*, *v*, *u=None*)
Approximate effect of transpose of J(m) on a vector v.

Parameters

- **m** (*numpy.array*) – model
- **v** (*numpy.array*) – vector to multiply
- **u** (*numpy.array*) – fields

Return type numpy.array

Returns JTv

Jvec(**args*, ***kwargs*)
Computing Jacobian^T multiplied by vector.

To use *Jvec* method, SimPEG requires that the survey be specified.

Jvec_approx(*m*, *v*, *u=None*)
Approximate effect of J(m) on a vector v

Parameters

- **m** (*numpy.array*) – model
- **v** (*numpy.array*) – vector to multiply
- **u** (*numpy.array*) – fields

Return type numpy.array

Returns approxJv

M00 = None

M01 = None

M10 = None

M11 = None

Solver
alias of `spsolve_Wrapped`

WT0 = None

WT1 = None

YBASE = None

chi = None

counter = None

curModel
Sets the current model, and removes dependent mass matrices.

deleteTheseOnModelUpdate = []

fields(**args*, ***kwargs*)
Return Bz or dBzdt

To use *fields* method, SimPEG requires that the survey be specified.

ispaired
True if the problem is paired to a survey.

jacSwitch = False

mapPair
alias of BaseEM1DMap

mapping = None

mesh = None

pair(*d*)
Bind a survey to this problem instance using pointers.

solverOpts = {}

survey
The survey object for this problem.

surveyPair
alias of BaseEM1DSurvey

unpair()
Unbind a survey from this problem instance.

Computing reflection coefficients

simpegem1d.RTEfun.**matmul**(*a00*, *a10*, *a01*, *a11*, *b00*, *b10*, *b01*, *b11*)
Compute 2x2 matrix mutiplication in vector way C = A*B C = [a00 a01] * [b00 b01] = [c00 c01] [a10 a11] [b10 b11] [c10 c11]

simpegem1d.RTEfun.**rTEfunfwd**(*nlay*, *f*, *lamda*, *sig*, *chi*, *depth*, *HalfSwitch*)
Compute reflection coefficients for Transverse Electric (TE) mode. Only one for loop for multiple layers. Do not use for loop for lambda, which has 801 times of loops (actually, this makes the code really slow).

simpegem1d.RTEfun.**rTEfunjac**(*nlay*, *f*, *lamda*, *sig*, *chi*, *depth*, *HalfSwitch*)
Compute reflection coefficients for Transverse Electric (TE) mode. Only one for loop for multiple layers. Do not use for loop for lambda, which has 801 times of loops (actually, this makes the code really slow).

Digital filtering

simpegem1d.DigFilter.**EvalDigitalFilt**(*base*, *weight*, *fun*, *r*)
Evaluating Digital filtering based on given base and weight

simpegem1d.DigFilter.**LoadWeights**()

simpegem1d.DigFilter.**setFrequency**(*time*)

simpegem1d.DigFilter.**transFilt**(*hz*, *wt*, *tbase*, *omega_int*, *t*, *tol=1e-12*)
Compute Step-off responses by Fast Hankel Transform (FHT) with cosine filters

simpegem1d.DigFilter.**transFiltImpulse**(*hz*, *wt*, *tbase*, *omega_int*, *t*, *tol=1e-12*)
Compute Impulse responses by Fast Hankel Transform (FHT) with cosine filters

simpegem1d.DigFilter.**transFiltImpulseInterp**(*hz*, *wt*, *tbase*, *omega*, *omega_int*, *t*, *tol=1e-12*)
Compute Impulse responses by Fast Hankel Transform (FHT) with cosine filters

simpegem1d.DigFilter.**transFiltInterp**(*hz*, *wt*, *tbase*, *omega*, *omega_int*, *t*, *tol=1e-12*)
Compute Step-off responses by Fast Hankel Transform (FHT) with cosine filters

Transmitter Waveform

simpegem1d.Waveform.**CausalConv**(*array1*, *array2*, *time*)
Evaluate convolution for two causal functions. Input

•array1: array for \(\ f_1(t)\)

•array2: array for \(\ f_2(t)\)

•time: array for time

$$Out(t) = \int_0^t f_1(a) f_2(t-a) da$$

simpegem1d.Waveform.**CenDiff**(*f*, *tin*)
Evaluating central difference of given array (f) and provide funtion handle for interpolation

simpegem1d.Waveform.**RectFun**(*time*, *ta*, *tb*)
Rectangular Waveform

•time: 1D array for time

•ta: time for transition from (+) to (-)

•tb: time at step-off

$$I(t) = 1, 0 < t \leq t_a$$
$$I(t) = -1, t_a < t < t_b$$
$$I(t) = 0, t \leq t_a \text{ or } t \geq t_b$$

simpegem1d.Waveform.**TriangleFun**(*time*, *ta*, *tb*)
Triangular Waveform * time: 1D array for time * ta: time at peak * tb: time at step-off

simpegem1d.Waveform.**TriangleFunDeriv**(*time*, *ta*, *tb*)
Derivative of Triangular Waveform

simpegem1d.Waveform.**VTEMFun**(*time*, *ta*, *tb*, *a*)
VTEM Waveform * time: 1D array for time * ta: time at peak of exponential part * tb: time at step-off

1.2.2 EM1D survey

class simpegem1d.BaseEM1D.**BaseEM1DSurvey**(***kwargs*)
Bases: SimPEG.Survey.BaseSurvey

Base EM1D Survey

I = 1.0
Tx loop current

a = None
Tx loop radius

dpred(**args*, ***kwargs*)

dpred(m, u=None)

Create the projected data from a model. The field, u, (if provided) will be used for the predicted data instead of recalculating the fields (which may be expensive!).

$$d_{\text{pred}} = P(u(m))$$

Where P is a projection of the fields onto the data space.

Note: To use survey.dpred(), SimPEG requires that a problem be bound to the survey. If a problem has not been bound, an Exception will be raised. To bind a problem to the Data object:

```
survey.pair(myProblem)
```

---

fieldtype = None
total or secondary fields

h = None
Tx heights at local coordinate

isSynthetic
Check if the data is synthetic.

makeSyntheticData(*m*, *std=0.05*, *u=None*, *force=False*)
Make synthetic data given a model, and a standard deviation.

Parameters

- **m** (*numpy.array*) – geophysical model
- **std** (*numpy.array*) – standard deviation
- **u** (*numpy.array*) – fields for the given model (if pre-calculated)
- **force** (*bool*) – force overwriting of dobs

mesh
Mesh of the paired problem.

nD
Number of data

nTx
Number of Transmitters

nlay = None
The # of layer (fixed for all soundings)

pair(*p*)
Bind a problem to this survey instance using pointers

prob
The geophysical problem that explains this survey, use:

```
survey.pair(prob)
```

projectFields(*u*)
This function projects the fields onto the data space.

$$d_{\text{pred}} = \mathbf{P}u(m)$$

projectFieldsDeriv(*u*)
This function s the derivative of projects the fields onto the data space.

$$\frac{\partial d_{\text{pred}}}{\partial u} = \mathbf{P}$$

residual(*m*, *u=None*)

Parameters

- **m** (*numpy.array*) – geophysical model
- **u** (*numpy.array*) – fields

Return type numpy.array

Returns data residual

The data residual:

$$\mu_{\text{data}} = \mathbf{d}_{\text{pred}} - \mathbf{d}_{\text{obs}}$$

txList
Transmitter List

txPair
alias of `BaseTx`

unpair()
Unbind a problem from this survey instance

vnD
Vector number of data

z = None
Rx heights at local coordinate

Frequency domain survey

class simpegem1d.BaseEM1D.**EM1DSurveyFD**(***kwargs*)
Bases: simpegem1d.BaseEM1D.BaseEM1DSurvey

docstring for EM1DSurveyFD

projectFields(*u*)
Decompose frequency domain EM responses as real and imaginary components

Time domain survey

class simpegem1d.BaseEM1D.**EM1DSurveyTD**(***kwargs*)
Bases: simpegem1d.BaseEM1D.BaseEM1DSurvey

docstring for EM1DSurveyTD

projectFields(*u*)
Transform frequency domain responses to time domain responses

setWaveform(***kwargs*)
Set parameters for Tx Waveform

1.2.3 EM1D analaytic solutions

simpegem1d.EM1DAnal.**BzAnalCircT**(*a*, *t*, *sigma*)
Hz component of analytic solution for half-space (Circular-loop source) Tx and Rx are on the surface and receiver is located at the center of the loop. Tx waveform here is step-off.

$$h_z = \frac{I}{2a}\left(\frac{3}{\sqrt{\pi}\theta a}e^{-\theta^2 a^2} + (1 - \frac{3}{2\theta^2 a^2})erf(\theta a)\right)$$

$$\theta = \sqrt{\frac{\sigma\mu}{4t}}$$

simpegem1d.EM1DAnal.**BzAnalCircTCole**(*a*, *t*, *sigma*)

simpegem1d.EM1DAnal.**BzAnalT**(*r*, *t*, *sigma*)

simpegem1d.EM1DAnal.**ColeCole**(*f*, *sig_inf=0.01*, *eta=0.1*, *tau=0.1*, *c=1*)

Computing Cole-Cole model in frequency domain

$$\sigma(\omega) = \sigma_\infty - \frac{\sigma_\infty \eta}{1 + (1-\eta)(\imath\omega\tau)^c}$$

where \(\sigma_{\infty}\) is conductivity at infinte frequency, \(\eta\) is chargeability, \(\tau\) is chargeability, \(\ c\) is chargeability.

simpegem1d.EM1DAnal.**Hzanal**(*sig*, *f*, *r*, *flag*)

Hz component of analytic solution for half-space (VMD source) Tx and Rx are on the surface

$$H_z = \frac{m}{2\pi k^2 r^5}\left(9 - (9 + \imath\, kr - 4k^2r^2 - \imath k^3 r^3)e^{-\imath kr}\right)$$

- r: Tx-Rx offset
- m: magnetic dipole moment
- k: propagation constant

$$k = \omega^2\epsilon\mu - \imath\omega\mu\sigma$$

simpegem1d.EM1DAnal.**HzanalCirc**(*sig*, *f*, *I*, *a*, *flag*)

Hz component of analytic solution for half-space (Circular-loop source) Tx and Rx are on the surface and receiver is located at the center of the loop.

$$H_z = -\frac{I}{k^2 a^3}\left(3 - (3 + \imath\, ka - k^2a^2)e^{-\imath ka}\right)$$

- a: Tx-loop radius
- I: Current intensity

simpegem1d.EM1DAnal.**dBzdtAnalCircT**(*a*, *t*, *sigma*)

Hz component of analytic solution for half-space (Circular-loop source) Tx and Rx are on the surface and receiver is located at the center of the loop. Tx waveform here is step-off.

$$\frac{\partial h_z}{\partial t} = -\frac{I}{\mu_0\sigma a^3}\left(3erf(\theta a) - \frac{2}{\sqrt{\pi}}\theta a(3 + 2\theta^2a^2)e^{-\theta^2a^2}\right)$$

$$\theta = \sqrt{\frac{\sigma\mu}{4t}}$$

simpegem1d.EM1DAnal.**dBzdtAnalCircTCole**(*a*, *t*, *sigma*)

simpegem1d.EM1DAnal.**dHzdsiganalCirc**(*sig*, *f*, *I*, *a*, *flag*)

Compute sensitivity for HzanalCirc by using perturbation

$$\frac{\partial H_z}{\partial \sigma} = \frac{H_z(\sigma + \triangle\sigma) - H_z(\sigma - \triangle\sigma)}{2\triangle\sigma}$$

1.3 Contact

Please contact me, if you have some questions or problems while using this program.

Seogi Kang

PhD student

Geophysical Inversion Facility, University of Britich Columbia, Canada

email: sgkang09@gmail.com

1.4 License

- **Master Branch**

CHAPTER 2

Project Index & Search

- *genindex*
- *modindex*
- *search*

S