empyscripts Documentation

Release 0.3.2

Dieter Werthmüller

22 May 2018

Contents

Version: 0.3.2 ~ Date: 22 May 2018

The **empyscripts** are *add-ons* for the electromagnetic modeller **empymod**. These add-ons provide some very specific, additional functionalities:

- `tmtemod`: Return up- and down-going TM/TE-mode contributions for x-directed electric sources and receivers, which are located in the same layer.
- `fdesign`: Design digital linear filters for the Hankel and Fourier transforms.

There is also `empyscripts.versions()`, which can be used to show date, time, and package version information at the end of a notebook or script:

- `versions('HTML')` for Jupyter Notebooks, and
- `versions()` for IPython, QT, and Python consoles.

See https://empymod.github.io/#features for a complete list of features of empymod.

CHAPTER 1

More information

For more information regarding installation, usage, add-ons, contributing, roadmap, bug reports, and much more, see

- **Website**: https://empymod.github.io,
- **Documentation empymod**: https://empymod.readthedocs.io,
- **Documentation add-ons**: https://empyscripts.readthedocs.io,
- **Source Code**: https://github.com/empymod,
- **Examples**: https://github.com/empymod/example-notebooks.

CHAPTER 2

License information

2.1 Manual

Documentation for `empyscripts`, the add-ons for `empymod`.

The add-ons are all independent of each other, and have their own documentation. You can find the information of each add-on in the respective *Code*-section.

For more information regarding installation, usage, add-ons, contributing, roadmap, bug reports, and much more, see https://empymod.github.io.

2.1.1 License

2.1.2 References

2.2 Changelog

2.2.1 v0.3.2 - *2018-05-22*

Last release of `empyscripts`. From `empymod v1.7.0` onwards `empyscripts` is included in `empymod` as `empymod.scripts`.

- Make `matplotlib`, `IPython`, and `numexpr` soft dependencies.

2.2.2 v0.3.1 - *2018-02-24*

- Add sphinx-docs, https://empyscripts.readthedocs.io

2.2.3 v0.3.0 - *2018-01-25*

- New utility `printinfo`.

2.2.4 v0.2.0 - *2018-01-09*

- New add-on `fdesign`.
- Now has tests and TravisCI, coveralls integration.

2.2.5 v0.1.1 - *2017-09-22*

- Initial release; add-on `tmtemod`.

2.3 Credits

Thanks to

- **Evert Slob** for the feedback and interaction during the creation of the add-on `tmtemod`, which was developed for the creation of github.com/empymod/csem-ziolkowski-and-slob.
- **Kerry Key** and **Evert Slob** for their inputs and feedback during the development of the add-on `fdesign` (see github.com/empymod/article-fdesign).

2.4 Code

2.4.1 Digital Linear Filter (DLF) design

Add-on for `empymod`, *[Werthmuller_2017]*.

The add-on fdesign can be used to design digital linear filters for the Hankel or Fourier transform, or for any linear transform (*[Ghosh_1970]*). For this included or provided theoretical transform pairs can be used. Alternatively, one can use the EM modeller empymod to use the responses to an arbitrary 1D model as numerical transform pair.

More information can be found in the following places:

- The article about fdesign is in the repo https://github.com/empymod/article-fdesign

- Example notebooks to design a filter can be found in the repo https://github.com/empymod/example-notebooks

This filter designing tool uses the direct matrix inversion method as described in *[Kong_2007]* and is based on scripts by *[Key_2012]*. The whole project of `fdesign` started with the Matlab scripts from Kerry Key, which he used to design his filters for *[Key_2009]*, *[Key_2012]*. Fruitful discussions with Evert Slob and Kerry Key improved the add-on substantially.

Note that the use of empymod to create numerical transform pairs is, as of now, only implemented for the Hankel transform.

Implemented analytical transform pairs

The following tables list the transform pairs which are implemented by default. Any other transform pair can be provided as input. A transform pair is defined in the following way:

```
from empyscripts.fdesign import Ghosh

def my_tp_pair(var):
    '''My transform pair.'''

    def lhs(l):
        return func(l, var)

    def rhs(r):
        return func(r, var)

    return Ghosh(name, lhs, rhs)
```

Here, `name` must be one of `j0`, `j1`, `sin`, or `cos`, depending what type of transform pair it is. Additional variables are provided with `var`. The evaluation points of the `lhs` are denoted by `l`, and the evaluation points of the `rhs` are denoted as `r`. As an example here the implemented transform pair `j0_1`

```
def j0_1(a=1):
    '''Hankel transform pair J0_1 ([Anderson_1975]_).'''

    def lhs(l):
        return l*np.exp(-a*l**2)

    def rhs(r):
        return np.exp(-r**2/(4*a))/(2*a)

    return Ghosh('j0', lhs, rhs)
```

Implemented Hankel transforms

- `j0_1` *[Anderson_1975]*

$$\int_0^\infty l \exp\left(-al^2\right) J_0(lr) dl = \frac{\exp\left(\frac{-r^2}{4a}\right)}{2a}$$

- `j0_2` *[Anderson_1975]*

$$\int_0^\infty \exp\left(-al\right) J_0(lr) dl = \frac{1}{\sqrt{a^2 + r^2}}$$

- `j0_3` *[Guptasarma_and_Singh_1997]*

$$\int_0^\infty l \exp\left(-al\right) J_0(lr) dl = \frac{a}{(a^2 + r^2)^{3/2}}$$

- `j0_4` *[Chave_and_Cox_1982]*

$$\int_0^\infty \frac{l}{\beta} \exp\left(-\beta z_v\right) J_0(lr) dl = \frac{\exp\left(-\gamma R\right)}{R}$$

- `j0_5` *[Chave_and_Cox_1982]*

$$\int_0^\infty l \exp\left(-\beta z_v\right) J_0(lr) dl = \frac{z_v(\gamma R + 1)}{R^3} \exp\left(-\gamma R\right)$$

- `j1_1` *[Anderson_1975]*

$$\int_0^\infty l^2 \exp\left(-al^2\right) J_1(lr) dl = \frac{r}{4a^2} \exp\left(-\frac{r^2}{4a}\right)$$

- `j1_2` *[Anderson_1975]*

$$\int_0^\infty \exp\left(-al\right) J_1(lr) dl = \frac{\sqrt{a^2 + r^2} - a}{r\sqrt{a^2 + r^2}}$$

- `j1_3` *[Anderson_1975]*

$$\int_0^\infty l \exp\left(-al\right) J_1(lr) dl = \frac{r}{(a^2 + r^2)^{3/2}}$$

- `j1_4` *[Chave_and_Cox_1982]*

$$\int_0^\infty \frac{l^2}{\beta} \exp\left(-\beta z_v\right) J_1(lr) dl = \frac{r(\gamma R + 1)}{R^3} \exp\left(-\gamma R\right)$$

- `j1_5` *[Chave_and_Cox_1982]*

$$\int_0^\infty l^2 \exp\left(-\beta z_v\right) J_1(lr) dl = \frac{r z_v(\gamma^2 R^2 + 3\gamma R + 3)}{R^5} \exp\left(-\gamma R\right)$$

Where

$$a > 0, r > 0$$

$$z_v = |z_{rec} - z_{src}|$$

$$R = \sqrt{r^2 + z_v^2}$$

$$\gamma = \sqrt{2j\pi\mu_0 f/\rho}$$

$$\beta = \sqrt{l^2 + \gamma^2}$$

Implemented Fourier transforms

- `sin_1` *[Anderson_1975]*

$$\int_0^\infty l \exp\left(-a^2 l^2\right) \sin(lr) dl = \frac{\sqrt{\pi} r}{4a^3} \exp\left(-\frac{r^2}{4a^2}\right)$$

- `sin_2` *[Anderson_1975]*

$$\int_0^\infty \exp\left(-al\right) \sin(lr) dl = \frac{r}{a^2 + r^2}$$

- `sin_3` *[Anderson_1975]*

$$\int_0^\infty \frac{l}{a^2+l^2}\sin(lr)dl = \frac{\pi}{2}\exp(-ar)$$

- `cos_1` *[Anderson_1975]*

$$\int_0^\infty \exp\left(-a^2l^2\right)\cos(lr)dl = \frac{\sqrt{\pi}}{2a}\exp\left(-\frac{r^2}{4a^2}\right)$$

- `cos_2` *[Anderson_1975]*

$$\int_0^\infty \exp(-al)\cos(lr)dl = \frac{a}{a^2+r^2}$$

- `cos_3` *[Anderson_1975]*

$$\int_0^\infty \frac{1}{a^2+l^2}\cos(lr)dl = \frac{\pi}{2a}\exp(-ar)$$

`empyscripts.fdesign.`**`design`**(*n*, *spacing*, *shift*, *fI*, *fC=False*, *r=None*, *r_def=(1, 1, 2)*, *reim=None*, *cvar='amp'*, *error=0.01*, *name=None*, *full_output=False*, *finish=False*, *save=True*, *verb=2*, *plot=1*)

Digital linear filter (DLF) design

This routine can be used to design digital linear filters for the Hankel or Fourier transform, or for any linear transform (*[Ghosh_1970]*). For this included or provided theoretical transform pairs can be used. Alternatively, one can use the EM modeller empymod to use the responses to an arbitrary 1D model as numerical transform pair.

This filter designing tool uses the direct matrix inversion method as described in *[Kong_2007]* and is based on scripts by *[Key_2012]*. The tool is an add-on to the electromagnetic modeller empymod *[Werthmuller_2017]*. Fruitful discussions with Evert Slob and Kerry Key improved the add-on substantially.

Example notebooks of its usage can be found in the repo github.com/empymod/example-notebooks.

Parameters **n** : int

Filter length.

spacing: float or tuple (start, stop, num)

Spacing between filter points. If tuple, it corresponds to the input for np.linspace with endpoint=True.

shift: float or tuple (start, stop, num)

Shift of base from zero. If tuple, it corresponds to the input for np.linspace with endpoint=True.

fI, fC : transform pairs

Theoretical or numerical transform pair(s) for the inversion (I) and for the check of goodness (fC). fC is optional. If not provided, fI is used for both fI and fC.

r : array, optional

Right-hand side evaluation points for the check of goodness (fC). Defaults to r = np.logspace(0, 5, 1000), which are a lot of evaluation points, and depending on the transform pair way too long r's.

r_def : tuple (add_left, add_right, factor), optional

Definition of the right-hand side evaluation points r of the inversion. r is derived from the base values, default is (1, 1, 2).

- rmin = log10(1/max(base)) - add_left
- rmax = log10(1/min(base)) + add_right
- r = logspace(rmin, rmax, factor*n)

reim : np.real or np.imag, optional

Which part of complex transform pairs is used for the inversion. Defaults to np.real.

cvar : string {'amp', 'r'}, optional

If 'amp', the inversion minimizes the amplitude. If 'r', the inversion maximizes the right-hand side evaluation point r. Defaults is 'amp'.

error : float, optional

Up to which relative error the transformation is considered good in the evaluation of the goodness. Default is 0.01 (1 %).

name : str, optional

Name of the filter. Defaults to dlf_+str(n).

full_output : bool, optional

If True, returns best filter and output from scipy.optimize.brute; else only filter. Default is False.

finish : None, True, or callable, optional

If callable, it is passed through to scipy.optimize.brute: minimization function to find minimize best result from brute-force approach. Default is None. You can simply provide True in order to use scipy.optimize.fmin_powell(). Set this to None if you are only interested in the actually provided spacing/shift-values.

save : bool, optional

If True, best filter is saved to ./filters/name.dir/.bak/.dat with shelve. Can be loaded with fdesign.load_filter(name).

verb : {0, 1, 2}, optional

Level of verbosity, default is 2:

- 0: Print nothing.
- 1: Print warnings.
- 2: Print additional time, progress, and result

plot : {0, 1, 2, 3}, optional

Level of plot-verbosity, default is 1:

- 0: Plot nothing.
- 1: Plot brute-force result
- 2: Plot additional theoretical transform pairs, and best inv.
- **3: Plot additional inversion result** (can result in lots of plots depending on spacing and shift) If you are using a notebook, use %matplotlib notebook to have all inversion results appear in the same plot.

Returns **filter** : empymod.filter.DigitalFilter instance

Best filter for the input parameters.

full : tuple

Output from scipy.optimize.brute with full_output=True. (Returned when `full_output` is True.)

empyscripts.fdesign.**save_filter**(*name*, *filt*, *full=None*)
Save DLF-filter to shelve.

empyscripts.fdesign.**load_filter**(*name*, *full=False*)
Load saved DLF-filter from shelve.

empyscripts.fdesign.**plot_result**(*filt*, *full*, *cvar='amp'*, *prntres=True*)
QC the inversion result.

Parameters **- filt, full as returned from fdesign.design with full_output=True**

- cvar as used for fdesign.design.

- If prntres is True, it calls fdesign.print_result as well.

empyscripts.fdesign.**print_result**(*filt*, *full=None*, *cvar='amp'*)
Print best filter information.

Parameters **- filt, full as returned from fdesign.design with full_output=True**

- cvar as used for fdesign.design.

class empyscripts.fdesign.**Ghosh**(*name*, *lhs*, *rhs*)
Simple Class for Theoretical Transform Pairs.

Named after D. P. Ghosh, honouring his 1970 Ph.D. thesis with which he introduced the digital filter method to geophysics (*[Ghosh_1970]*).

empyscripts.fdesign.**j0_1**(*a=1*)
Hankel transform pair J0_1 (*[Anderson_1975]*).

empyscripts.fdesign.**j0_2**(*a=1*)
Hankel transform pair J0_2 (*[Anderson_1975]*).

empyscripts.fdesign.**j0_3**(*a=1*)
Hankel transform pair J0_3 (*[Guptasarma_and_Singh_1997]*).

empyscripts.fdesign.**j0_4**(*f=1*, *rho=0.3*, *z=50*)
Hankel transform pair J0_4 (*[Chave_and_Cox_1982]*).

Parameters **f** : float

Frequency (Hz)

rho : float

Resistivity (Ohm.m)

z : float

Vertical distance between source and receiver (m)

empyscripts.fdesign.**j0_5**(*f=1*, *rho=0.3*, *z=50*)
Hankel transform pair J0_5 (*[Chave_and_Cox_1982]*).

Parameters **f** : float

Frequency (Hz)

rho : float

Resistivity (Ohm.m)

z : float

Vertical distance between source and receiver (m)

empyscripts.fdesign.**j1_1**(*a=1*)
Hankel transform pair J1_1 (*[Anderson_1975]*).

empyscripts.fdesign.**j1_2**(*a=1*)
Hankel transform pair J1_2 (*[Anderson_1975]*).

empyscripts.fdesign.**j1_3**(*a=1*)
Hankel transform pair J1_3 (*[Anderson_1975]*).

empyscripts.fdesign.**j1_4**(*f=1*, *rho=0.3*, *z=50*)
Hankel transform pair J1_4 (*[Chave_and_Cox_1982]*).

Parameters **f** : float

Frequency (Hz)

rho : float

Resistivity (Ohm.m)

z : float

Vertical distance between source and receiver (m)

empyscripts.fdesign.**j1_5**(*f=1*, *rho=0.3*, *z=50*)
Hankel transform pair J1_5 (*[Chave_and_Cox_1982]*).

Parameters **f** : float

Frequency (Hz)

rho : float

Resistivity (Ohm.m)

z : float

Vertical distance between source and receiver (m)

empyscripts.fdesign.**sin_1**(*a=1*)
Fourier sine transform pair sin_1 (*[Anderson_1975]*).

empyscripts.fdesign.**sin_2**(*a=1*)
Fourier sine transform pair sin_2 (*[Anderson_1975]*).

empyscripts.fdesign.**sin_3**(*a=1*)
Fourier sine transform pair sin_3 (*[Anderson_1975]*).

empyscripts.fdesign.**cos_1**(*a=1*)
Fourier cosine transform pair cos_1 (*[Anderson_1975]*).

empyscripts.fdesign.**cos_2**(*a=1*)
Fourier cosine transform pair cos_2 (*[Anderson_1975]*).

empyscripts.fdesign.**cos_3**(*a=1*)
Fourier cosine transform pair cos_3 (*[Anderson_1975]*).

empyscripts.fdesign.**empy_hankel**(*ftype*, *zsrc*, *zrec*, *res*, *freqtime*, *depth=[]*, *aniso=None*, *epermH=None*, *epermV=None*, *mpermH=None*, *mpermV=None*, *htarg=None*, *verblhs=0*, *verbrhs=0*)
Numerical transform pair with empymod.

All parameters except `ftype`, `verblhs`, and `verbrhs` correspond to the input parameters to `empymod.dipole`. See there for more information.

Note that if depth=[], the analytical full-space solutions will be used (much faster).

Parameters **ftype** : str or list of strings

Either of: {'j0', 'j1', 'j2', ['j0', 'j1']}

- 'j0': Analyze J0-term with ab=11, angle=45°
- 'j1': Analyze J1-term with ab=31, angle=0°
- 'j2': Analyze J0- and J1-terms jointly with ab=12, angle=45°
- **['j0', 'j1']: Same as calling empy_hankel twice, once with 'j0' and** one with 'j1'; can be provided like this to fdesign.design.

verblhs, verbrhs: int

verb-values provided to empymod for lhs and rhs.

Note that ftype='j2' only works for fC, not for fI.

2.4.2 Calculate up- and down-going TM and TE modes

Add-on for `empymod`, *[Werthmuller_2017]*: Adjust *[Hunziker_et_al_2015]* for TM/TE-split. The development was initiated by the development of https://github.com/empymod/csem-ziolkowski-and-slob (*[Ziolkowski_and_Slob_2018]*).

This is a stripped-down version of `empymod` with a lot of simplifications but an important addition. The modeller `empymod` returns the total field, hence not distinguishing between TM and TE mode, and even less between up- and down-going fields. The reason behind this is simple: The derivation of *[Hunziker_et_al_2015]*, on which `empymod` is based, returns the total field. In this derivation each mode (TM and TE) contains non-physical contributions. The non-physical contributions have opposite signs in TM and TE, so they cancel each other out in the total field. However, in order to obtain the correct TM and TE contributions one has to remove these non-physical parts.

This is what this routine does, but only for an x-directed electric source with an x-directed electric receiver, and in the frequency domain (src and rec in same layer). This version of `dipole` returns the signal separated into TM++, TM+-, TM-+, TM–, TE++, TE+-, TE-+, and TE– as well as the direct field TM and TE contributions. The first superscript denotes the direction in which the field diffuses towards the receiver and the second superscript denotes the direction in which the field diffuses away from the source. For both the plus-sign indicates the field diffuses in the downward direction and the minus-sign indicates the field diffuses in the upward direction. It uses `empymod` wherever possible. See the corresponding functions in `empymod` for more explanation and documentation regarding input parameters. There are important limitations:

- `ab` == 11 [=> x-directed el. source & el. receivers]
- `signal` == None [=> only frequency domain]
- `xdirect` == False [=> direct field calc. in wavenr-domain]
- `ht` == 'fht'
- `htarg` == 'key_201_2012'
- Options `ft`, `ftarg`, `opt`, and `loop` are not available.
- `lsrc` == `lrec` [=> src & rec are assumed in same layer!]
- Model must have more than 1 layer
- Electric permittivity and magnetic permeability are isotropic.
- Only one frequency at once.

This script is tested and works with `empymod v1.4.4` onwards.

Theory

The derivation of *[Hunziker_et_al_2015]*, on which `empymod` is based, returns the total field. Internally it also calculates TM and TE modes, and sums these up. However, the separation into TM and TE mode introduces a singularity at $\kappa = 0$. It has no contribution in the space-frequency domain to the total fields, but it introduces non-physical events in each mode with opposite sign (so they cancel each other out in the total field). In order to obtain the correct TM and TE contributions one has to remove these non-physical parts.

To remove the non-physical part we use the file `tmtemod.py` in this directory. This routine is basically a heavily simplified version of `empymod` with the following limitations outlined above.

So `tmtemod.py` returns the signal separated into TM++, TM+-, TM-+, TM–, TE++, TE+-, TE-+, and TE– as well as the direct field TM and TE contributions. The first superscript denotes the direction in which the field diffuses towards the receiver and the second superscript denotes the direction in which the field diffuses away

from the source. For both the plus-sign indicates the field diffuses in the downward direction and the minus-sign indicates the field diffuses in the upward direction. The routine uses `empymod` wherever possible, see the corresponding functions in `empymod` for more explanation and documentation regarding input parameters.

Please note that the notation in *[Hunziker_et_al_2015]* differs from the notation in *[Ziolkowski_and_Slob_2018]*. I specify therefore always, which notification applies, either *Hun15* or *Zio18*.

We start with equation (105) in *Hun15*:

$$\hat{G}^{ee}_{xx}(\boldsymbol{x},\boldsymbol{x'},\omega) = \hat{G}^{ee;i}_{xx;s}(\boldsymbol{x}-\boldsymbol{x'},\omega) + \frac{1}{8\pi}\int_{\kappa=0}^{\infty}\left(\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s} - \frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}\right)J_0(\kappa r)\kappa d\kappa$$

$$-\frac{\cos(2\phi)}{8\pi}\int_{\kappa=0}^{\infty}\left(\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s} + \frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}\right)J_2(\kappa r)\kappa d\kappa.$$

Ignoring the incident field, and using $J_2 = \frac{2}{\kappa r}J_1 - J_0$ to avoid J_2-integrals, we get

$$\hat{G}^{ee}_{xx}(\boldsymbol{x},\boldsymbol{x'},\omega) = \frac{1}{8\pi}\int_{\kappa=0}^{\infty}\left(\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s} - \frac{\zeta_s ilde g^{te}_{zz;s}}{\bar{\Gamma}_s}\right)J_0(\kappa r)\kappa\,\mathrm{d}\kappa$$

$$+\frac{\cos(2\phi)}{8\pi}\int_{\kappa=0}^{\infty}\left(\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s} + \frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}\right)J_0(\kappa r)\kappa\,\mathrm{d}\kappa$$

$$-\frac{\cos(2\phi)}{4\pi r}\int_{\kappa=0}^{\infty}\left(\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s} + \frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}\right)J_1(\kappa r)\,\mathrm{d}\kappa.$$

From this the TM- and TE-parts follow as

$$\mathrm{TE} = \frac{\cos(2\phi)-1}{8\pi}\int_{\kappa=0}^{\infty}\frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}J_0(\kappa r)\kappa\,\mathrm{d}\kappa - \frac{\cos(2\phi)}{4\pi r}\int_{\kappa=0}^{\infty}\frac{\zeta_s\tilde{g}^{te}_{zz;s}}{\bar{\Gamma}_s}J_1(\kappa r)\,\mathrm{d}\kappa,$$

$$\mathrm{TM} = \frac{\cos(2\phi)+1}{8\pi}\int_{\kappa=0}^{\infty}\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s}J_0(\kappa r)\kappa\,\mathrm{d}\kappa - \frac{\cos(2\phi)}{4\pi r}\int_{\kappa=0}^{\infty}\frac{\Gamma_s\tilde{g}^{tm}_{hh;s}}{\eta_s}J_1(\kappa r)\,\mathrm{d}\kappa.$$

Equations (108) and (109) in Hun15 yield the required parameters $\tilde{g}^{tm}_{hh;s}$ and $\tilde{g}^{te}_{zz;s}$,

$$\tilde{g}^{tm}_{hh;s} = P^{u-}_s W^u_s + P^{d-}_s W^d_s,$$

$$\tilde{g}^{te}_{zz;s} = \bar{P}^{u+}_s \bar{W}^u_s + \bar{P}^{d+}_s \bar{W}^d_s\ .$$

The parameters $P^{u\pm}_s$ and $P^{d\pm}_s$ are given in equations (81) and (82), $\bar{P}^{u\pm}_s$ and $\bar{P}^{d\pm}_s$ in equations (A-8) and (A-9); W^u_s and W^d_s in equation (74) in Hun15. This yields

$$\tilde{g}^{te}_{zz;s} = \frac{\bar{R}^+_s}{\bar{M}_s}\left\{\exp[-\bar{\Gamma}_s(z_s - z + d^+)] + \bar{R}^-_s\exp[-\bar{\Gamma}_s(z_s - z + d_s + d^-)]\right\}$$

$$+\frac{\bar{R}^-_s}{\bar{M}_s}\left\{\exp[-\bar{\Gamma}_s(z - z_{s-1} + d^-)] + \bar{R}^+_s\exp[-\bar{\Gamma}_s(z - z_{s-1} + d_s + d^+)]\right\},$$

$$= \frac{\bar{R}^+_s}{\bar{M}_s}\left\{\exp[-\bar{\Gamma}_s(2z_s - z - z')] + \bar{R}^-_s\exp[-\bar{\Gamma}_s(z' - z + 2d_s)]\right\}$$

$$+\frac{\bar{R}^-_s}{\bar{M}_s}\left\{\exp[-\bar{\Gamma}_s(z + z' - 2z_{s-1})] + \bar{R}^+_s\exp[-\bar{\Gamma}_s(z - z' + 2d_s)]\right\},$$

where $d^\pm$ is taken from the text below equation (67). There are four terms in the right-hand side, two in the first line and two in the second line. The first term in the first line is the integrand of TE+-, the second term in the first line corresponds to TE++, the first term in the second line is TE-+, and the second term in the second line is TE–.

If we look at TE+-, we have

$$\tilde{g}^{te+-}_{zz;s} = \frac{\bar{R}^+_s}{\bar{M}_s}\exp[-\bar{\Gamma}_s(2z_s - z - z')]\ ,$$

and therefore

$$\mathrm{TE}^{+-} = \frac{\cos(2\phi)-1}{8\pi}\int_{\kappa=0}^{\infty}\frac{\zeta_s\bar{R}_s^+}{\bar{\Gamma}_s\bar{M}_s}\exp[-\bar{\Gamma}_s(2z_s-z-z')]J_0(\kappa r)\kappa\,\mathrm{d}\kappa$$
$$-\frac{\cos(2\phi)}{4\pi r}\int_{\kappa=0}^{\infty}\frac{\zeta_s\bar{R}_s^+}{\bar{\Gamma}_s\bar{M}_s}\exp[-\bar{\Gamma}_s(2z_s-z-z')]J_1(\kappa r)\,\mathrm{d}\kappa.$$

We can compare this to equation (4.165) in Zio18, with $\hat{I}_x^e = 1$ and slightly re-arranging it to look more alike, we get

$$\hat{E}_{xx;H}^{+-} = \frac{y^2}{4\pi r^2}\int_{\kappa=0}^{\infty}\frac{\zeta_1}{\Gamma_1}\frac{R_{H;1}^-}{M_{H;1}}\exp(-\Gamma_1 h^{+-})J_0(\kappa r)\kappa d\kappa$$
$$+\frac{x^2-y^2}{4\pi r^3}\int_{\kappa=0}^{\infty}\frac{\zeta_1}{\Gamma_1}\left(\frac{R_{H;1}^-}{M_{H;1}}-\frac{R_{H;1}^-(\kappa=0)}{M_{H;1}(\kappa=0)}\right)\exp(-\Gamma_1 h^{+-})J_1(\kappa r)d\kappa$$
$$-\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{H;1}^-(\kappa=0)}{M_{H;1}(\kappa=0)}\exp(-\gamma_1 R^{+-}).$$

The notation in this equation follows Zio18.

The difference between the two previous equations is that the first one contains non-physical contributions. These have opposite signs in TM+- and TE+-, and therefore cancel each other out. But if we want to know the specific contributions from TM and TE we have to remove them. The non-physical contributions only affect the J_1-integrals, and only for $\kappa = 0$.

The following lists for all 8 cases the term that has to be removed, in the notation of Zio18 (for the notation as in Hun15 see the implementation in `tmtemod.py`):

$$TE^{++} = +\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{\exp(-\gamma_1|h^-|)}{M_{H;1}(\kappa=0)},$$
$$TE^{-+} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{H;1}^+(\kappa=0)\exp(-\gamma_1 h^{-+})}{M_{H;1}(\kappa=0)},$$
$$TE^{+-} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{H;1}^-(\kappa=0)\exp(-\gamma_1 h^{+-})}{M_{H;1}(\kappa=0)},$$
$$TE^{--} = +\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{H;1}^+(\kappa=0)R_{H;1}^-(\kappa=0)\exp(-\gamma_1 h^{--})}{M_{H;1}(\kappa=0)},$$
$$TM^{++} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{\exp(-\gamma_1|h^-|)}{M_{V;1}(\kappa=0)},$$
$$TM^{-+} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{V;1}^+(\kappa=0)\exp(-\gamma_1 h^{-+})}{M_{V;1}(\kappa=0)},$$
$$TM^{+-} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{V;1}^-(\kappa=0)\exp(-\gamma_1 h^{+-})}{M_{V;1}(\kappa=0)},$$
$$TM^{--} = -\frac{\zeta_1(x^2-y^2)}{4\pi\gamma_1 r^4}\frac{R_{V;1}^+(\kappa=0)R_{V;1}^-(\kappa=0)\exp(-\gamma_1 h^{--})}{M_{V;1}(\kappa=0)}.$$

Note that in the first and fourth equations the correction terms have opposite sign as those in the fifth and eighth equations because at $\kappa = 0$ the TM and TE mode correction terms are equal. Also note that in the second and third equations the correction terms have the same sign as those in the sixth and seventh equations because at $\kappa = 0$ the TM and TE mode reflection responses in those terms are equal but with opposite sign: $R_{V;1}^{\pm}(\kappa=0) = -R_{V;1}^{\pm}(\kappa=0)$.

Hun15 uses ϕ, whereas Zio18 uses x, y, for which we can use

$$\cos(2\phi) = -\frac{x^2-y^2}{r^2}\ .$$

empyscripts.tmtemod.**dipole**(*src*, *rec*, *depth*, *res*, *freqtime*, *aniso=None*, *eperm=None*, *mperm=None*, *verb=2*)

Return the electromagnetic field due to a dipole source.

This is a modified version of `empymod.model.dipole()`. It returns the separated contributions of TM–, TM-+, TM+-, TM++, TMdirect, TE–, TE-+, TE+-, TE++, and TEdirect.

Parameters **src, rec** : list of floats or arrays

Source and receiver coordinates (m): [x, y, z]. The x- and y-coordinates can be arrays, z is a single value. The x- and y-coordinates must have the same dimension.

Sources or receivers placed on a layer interface are considered in the upper layer.

Sources and receivers must be in the same layer.

depth : list

Absolute layer interfaces z (m); #depth = #res - 1 (excluding +/- infinity).

res : array_like

Horizontal resistivities rho_h (Ohm.m); #res = #depth + 1.

freqtime : float

Frequency f (Hz). (The name `freqtime` is kept for consistency with `empymod.model.dipole()`. Only one frequency at once.

aniso : array_like, optional

Anisotropies lambda = sqrt(rho_v/rho_h) (-); #aniso = #res. Defaults to ones.

eperm : array_like, optional

Relative electric permittivities epsilon (-); #eperm = #res. Default is ones.

mperm : array_like, optional

Relative magnetic permeabilities mu (-); #mperm = #res. Default is ones.

verb : {0, 1, 2, 3, 4}, optional

Level of verbosity, default is 2:

- 0: Print nothing.
- 1: Print warnings.
- 2: Print additional runtime and kernel calls
- 3: Print additional start/stop, condensed parameter information.
- 4: Print additional full parameter information

Returns **TM, TE** : list of ndarrays, (nfreq, nrec, nsrc)

Frequency-domain EM field [V/m], separated into TM = [TM–, TM-+, TM+-, TM++, TMdirect] and TE = [TE–, TE-+, TE+-, TE++, TEdirect].

However, source and receiver are normalised. So the source strength is 1 A and its length is 1 m. Therefore the electric field could also be written as [V/(A.m2)].

The shape of EM is (nfreq, nrec, nsrc). However, single dimensions are removed.

2.4.3 Tools to print date, time, and version information

Add-on for `empymod`, *[Werthmuller_2017]*.

Print or return date, time, and package version information in any environment (Jupyter notebook, IPython console, Python console, QT console), either as html-table (notebook) or as plain text (anywhere).

This script was heavily inspired by

- `ipynbtools.py` from https://github.com/qutip, and
- `watermark.py` from https://github.com/rasbt/watermark,

Always shown are the OS, number of CPU(s), `numpy`, `scipy`, `empymod`, `empyscripts`, `sys.version`, and time/date.

Additionally shown are, if they can be imported, `IPython`, `matplotlib`, and `numexpr`. If `numexpr` can be imported it shows additionally VML information.

All modules provided in `add_pckg` are also shown. They have to be imported before `versions` is called.

`empyscripts.printinfo.`**`versions`**(*mode='print'*, *add_pckg=[]*, *ncol=3*)

Return date, time, and version information.

Print or return date, time, and package version information in any environment (Jupyter notebook, IPython console, Python console, QT console), either as html-table (notebook) or as plain text (anywhere).

This script was heavily inspired by:

- ipynbtools.py from qutip https://github.com/qutip
- watermark.py from https://github.com/rasbt/watermark

This is a wrapper for `versions_html` and `versions_text`.

Parameters **mode** : string, optional; {<'print'>, 'HTML', 'Pretty', 'plain', 'html'}

Defaults to 'print':

- 'print': Prints text-version to stdout, nothing returned.
- 'HTML': Returns html-version as IPython.display.HTML(html).
- 'html': Returns html-version as plain text.
- 'Pretty': Returns text-version as IPython.display.Pretty(text).
- 'plain': Returns text-version as plain text.

'HTML' and 'Pretty' require IPython.

add_pckg : packages, optional

Package or list of packages to add to output information (must be imported beforehand).

ncol : int, optional

Number of package-columns in html table; only has effect if `mode='HTML'` or `mode='html'`. Defaults to 3.

Returns Depending on `mode` (HTML-instance; plain text; html as plain text; or

nothing, only printing to stdout).

Examples

```
>>> import pytest
>>> import dateutil
>>> from empyscripts import versions
>>> versions()                  # Default values
>>> versions('plain', pytest)   # Provide additional package
>>> versions('HTML', [pytest, dateutil], ncol=5)   # HTML
```

empyscripts.printinfo.**versions_html**(*add_pckg=[]*, *ncol=3*)
HTML version.

See `versions` for details.

empyscripts.printinfo.**versions_text**(*add_pckg=[]*)
Plain-text version.

See `versions` for details.

Bibliography

[Anderson_1975] Anderson, W. L., 1975, Improved digital filters for evaluating Fourier and Hankel transform integrals: USGS, PB242800; pubs.er.usgs.gov/publication/70045426.

[Chave_and_Cox_1982] Chave, A. D., and C. S. Cox, 1982, Controlled electromagnetic sources for measuring electrical conductivity beneath the oceans: 1. forward problem and model study: Journal of Geophysical Research, 87, 5327-5338; DOI: 10.1029/JB087iB07p05327.

[Ghosh_1970] Ghosh, D. P., 1970, The application of linear filter theory to the direct interpretation of geoelectrical resistivity measurements: Ph.D. Thesis, TU Delft; UUID: 88a568bb-ebee-4d7b-92df-6639b42da2b2.

[Guptasarma_and_Singh_1997] Guptasarma, D., and B. Singh, 1997, New digital linear filters for Hankel J0 and J1 transforms: Geophysical Prospecting, 45, 745–762; DOI: 10.1046/j.1365-2478.1997.500292.x.

[Hunziker_et_al_2015] Hunziker, J., J. Thorbecke, and E. Slob, 2015, The electromagnetic response in a layered vertical transverse isotropic medium: A new look at an old problem: Geophysics, 80(1), F1–F18; DOI: 10.1190/geo2013-0411.1; Software: software.seg.org/2015/0001.

[Key_2009] Key, K., 2009, 1D inversion of multicomponent, multifrequency marine CSEM data: Methodology and synthetic studies for resolving thin resistive layers: Geophysics, 74(2), F9–F20; DOI: 10.1190/1.3058434. Software: marineemlab.ucsd.edu/Projects/Occam/1DCSEM.

[Key_2012] Key, K., 2012, Is the fast Hankel transform faster than quadrature?: Geophysics, 77, F21–F30; DOI: 10.1190/GEO2011-0237.1; Software: software.seg.org/2012/0003.

[Kong_2007] Kong, F. N., 2007, Hankel transform filters for dipole antenna radiation in a conductive medium: Geophysical Prospecting, 55, 83–89; DOI: 10.1111/j.1365-2478.2006.00585.x.

[Werthmuller_2017] Werthmüller, D., 2017, An open-source full {3D} electromagnetic modeler for 1D VTI media in Python: empymod: Geophysics, 82, WB9–WB19.; DOI: 10.1190/geo2016-0626.1.

[Ziolkowski_and_Slob_2018] Ziolkowski, A., and E. Slob, 2018, Introduction to Controlled-Source Electromagnetic Methods: Cambridge University Press; expected to be published late 2018.

Python Module Index

e

Index