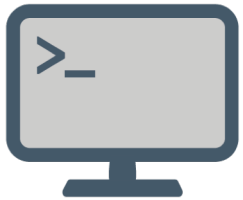
Computational challenges

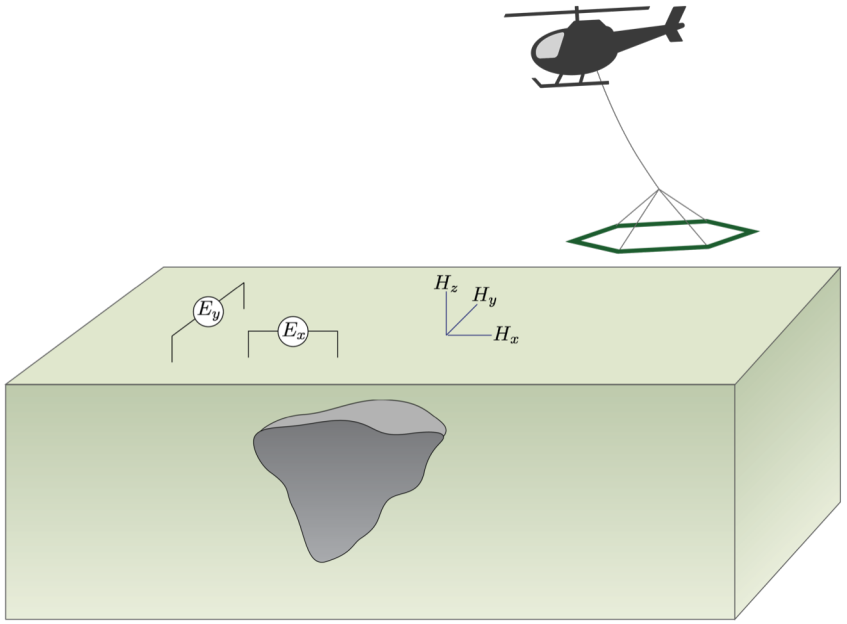


Forward Problem

Fundamental component of survey design

- ability to excite a target
- observe fields, fluxes, charges
- compute expected observations

Essential component of the inverse problem



Maxwell's equations:

- Faraday's law:

$$\nabla \times \mathbf{e} = -\frac{\partial \mathbf{b}}{\partial t}$$

- Ampere's law:

$$\nabla \times \mathbf{h} = \mathbf{j} + \mathbf{s}$$

- Other important relationships:

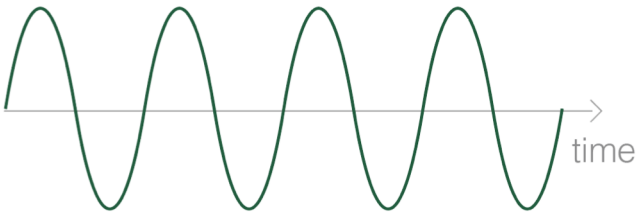
$$\nabla \cdot \mathbf{b} = 0$$

$$\mathbf{j} = \sigma \mathbf{e}$$

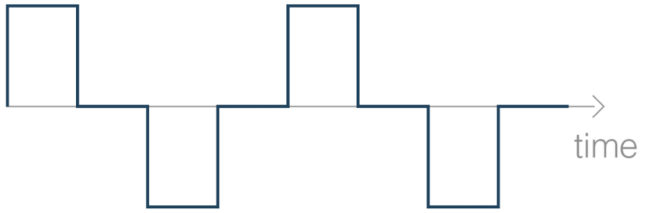
$$\mathbf{b} = \mu \mathbf{h}$$

- Solution depends upon the sources and boundary conditions

Frequency:



Time-domain:

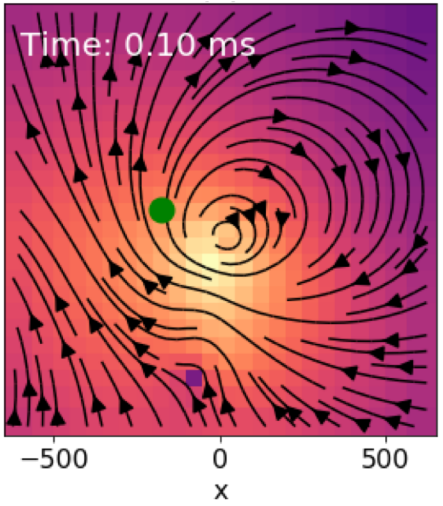


Forward modelling

What is desired of the forward simulation software?

- **High accuracy:** even under extreme conditions such as high conductivity contrast or rugged topography

- **Efficiency:** the forward problem needs to be solved many times when computing the inverse solution



- **Flexibility:** Want to access and visualize EM fields, fluxes, and charges

Forward modelling approaches

- Maxwell's equations can be solved as:
  - Integral equation (IE)

$$\vec{E}(\vec{r}) = \vec{E}_p(\vec{r}) + \int_V G(\vec{r}, \vec{r}_s)\sigma_a(\vec{r}_s)\vec{E}(\vec{r}_s)dv_s$$

  - Differential equation (DE)

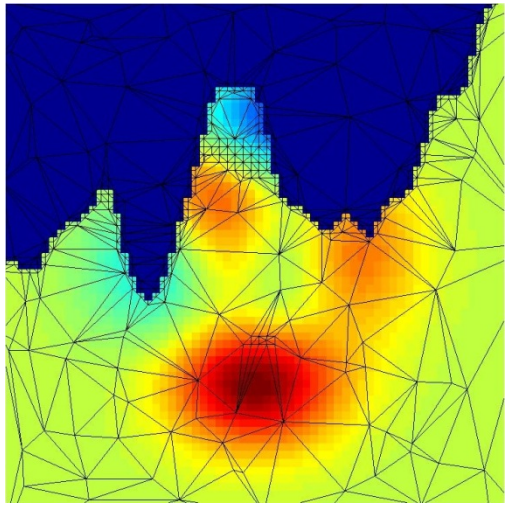
$$\nabla \times \mu^{-1}\nabla \times \vec{E} + \imath\omega\sigma\vec{E} = -\imath\omega\vec{J}_s$$

Integral Equation or Differential Equation

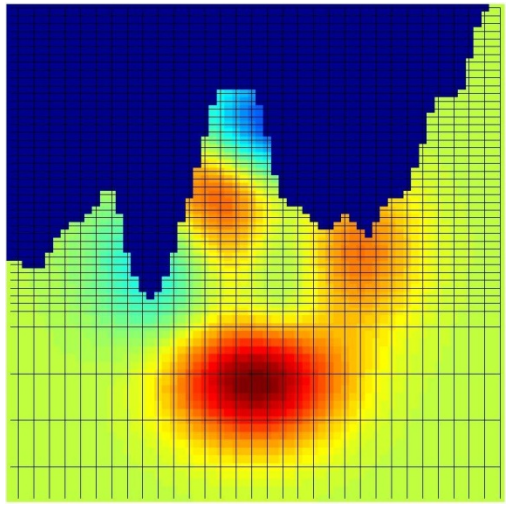
	Integral Equation	Differential Equation
Computational domain	Closed volumes (handles infinity)	Entire volume
Matrix system	Dense	Sparse
Highly variable discontinuous coefficients (eg topography)	With difficulty	Yes

What type of mesh?

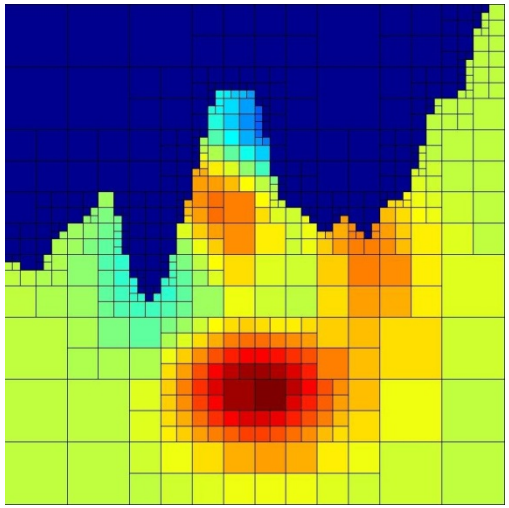
Unstructured



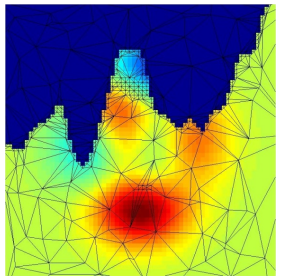
Structured



Semi-structured



Unstructured meshes



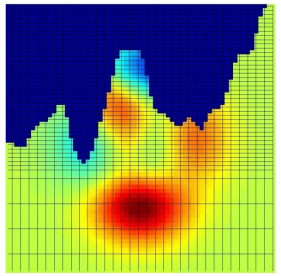
Advantages:

- Handle complex geometry
- Small matrix (but unstructured)

Disadvantages:

- Algebra is more difficult
- Programming is difficult
- Special software tools needed for meshing and handling ill-conditioning
- Visualization and interacting with results requires advanced graphical tools.

Structured meshes



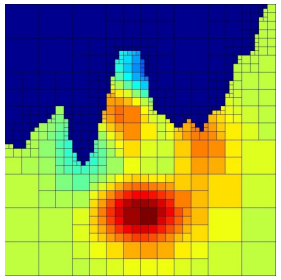
Advantages:

- Straightforward to implement
  - Discretize Maxwell's eqns
  - Solve in Matlab, Python
  - Visualize fields
  - Set up inversion
  - Visualize results

Disadvantages:

- Requires large number of cells to handle
  - infinity
  - discretized topography

Semi-structured meshes



Advantages:

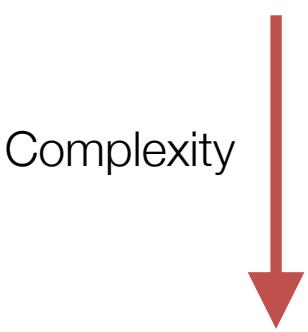
- Yields structured matrices
- “Standard” linear algebra works well
- Relatively easy to visualize
- Reduced mesh size as compared to structured meshes

Disadvantages

- Somewhat harder to implement than structured
- Need to be careful when changing cell size

Solving the Differential Equations

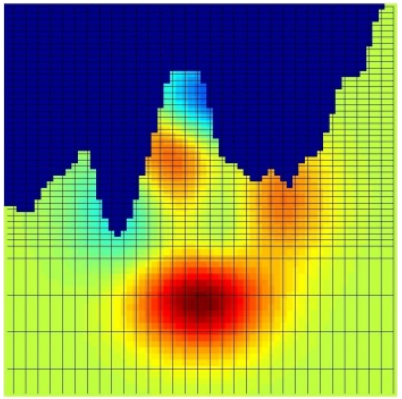
Problems on unstructured or structured meshes can be solved using

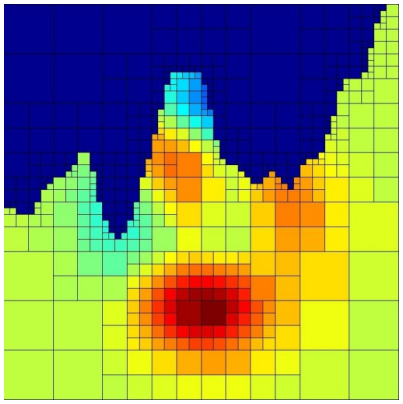


- Finite Difference Method (FDM)
- Finite Volume Method (FVM)
- Finite Element Method (FEM)

Choices for our work

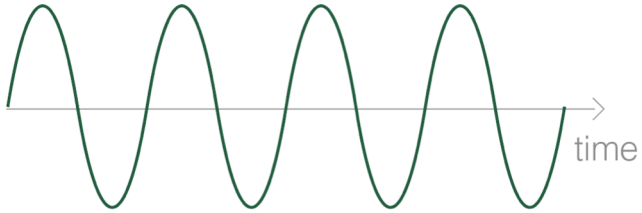
- Differential equations (DE)
- Finite volume method (FVM)
- Structured grids
- Semi-structured grid (OcTree)



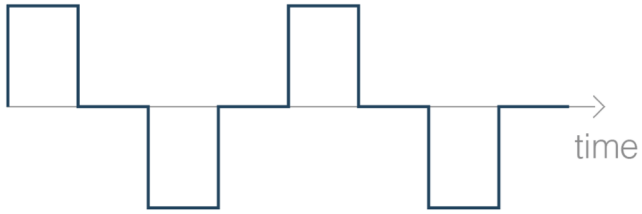


Data: frequency or time

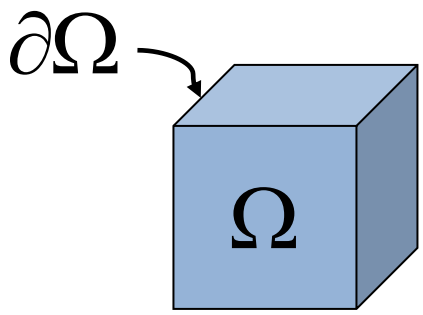
- Frequency Domain Data:
  - Solve Maxwell's equations in frequency domain



- Time Domain Data:
  - Solve Maxwell's equations in frequency domain and Fourier transform
  - Solve Maxwell's equations with time stepping



Frequency Domain: Mathematical Setup



Maxwell's equations

$$\nabla \times \mathbf{E} + i\omega\mathbf{B} = 0$$
$$\nabla \times \mu^{-1}\mathbf{B} - \sigma\mathbf{E} = \mathbf{s}$$

Boundary conditions

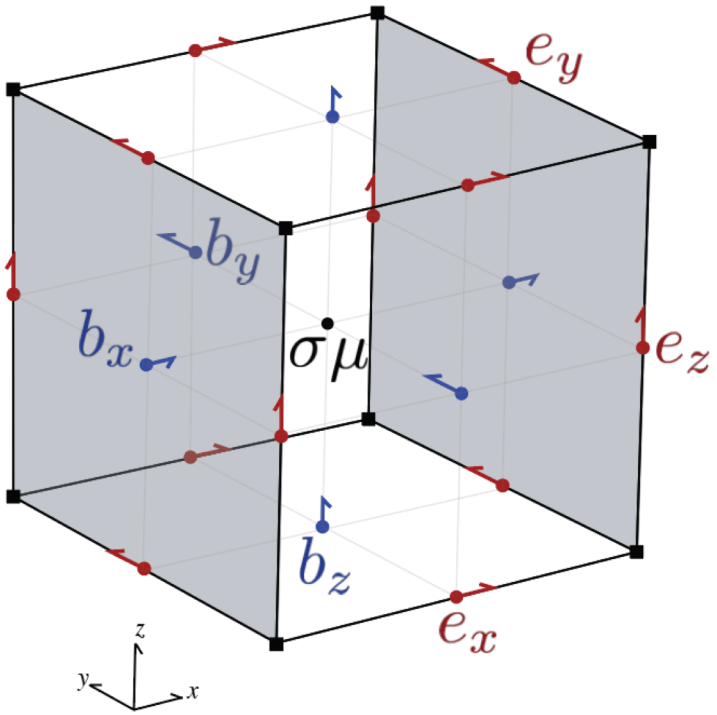
$$\mathbf{n} \times \mathbf{B} = 0$$

Need to solve in space for each frequency

Staggered Grid Discretization (in space)

Staggered Grid

- Physical properties: cell centers
- Fields: edges
- Fluxes: faces



Continuous second-order equations

$$\nabla \times \mu^{-1} \nabla \times \mathbf{E} + i\omega\sigma\mathbf{E} = -i\omega\mathbf{s}$$

Discrete second order equations

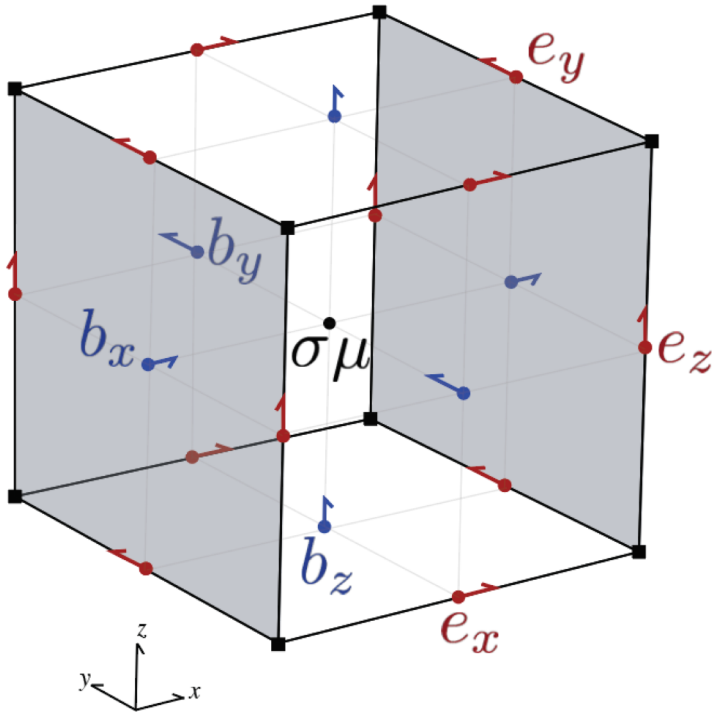
$$(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + i\omega \mathbf{M}^E_\sigma)\mathbf{E} = -i\omega\mathbf{s}$$

Solving an FDEM Problem

$$\underbrace{(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + i\omega \mathbf{M}^e_\sigma)}_{\mathbf{A}(\sigma,\omega)} \underbrace{\mathbf{E}}_{\mathbf{u}} = \underbrace{-i\omega \mathbf{s}}_{\mathbf{q}(\omega)}$$

$\mathbf{A}(\sigma,\omega)$

- Complex
- Symmetric
- Factor once for each frequency (solve for multiple sources)
- Needs to be refactored on each model update
- Problem separable over frequencies



Solving an FDEM Problem

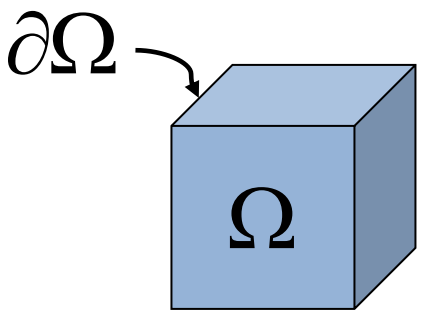
$$\underbrace{(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + i\omega \mathbf{M}^e_\sigma)}_{\mathbf{A}(\sigma,\omega)} \underbrace{\mathbf{E}}_{\mathbf{u}} = \underbrace{-i\omega \mathbf{s}}_{\mathbf{q}(\omega)}$$

```
u = zeros(n_edges, n_sources)  # initalize storage for solution

for f in frequencies:
    A = getA(sigma, f)
    Ainv = solver(A)  # e.g. Pardiso, MUMPS

    rhs = getRHS(f)  # all sources at the frequency f
    sources = get_sources_by_frequency(f)  # indices of sources
    u[:, sources] = Ainv * rhs
```

Time Domain: Mathematical Setup



time: $[0, t_f]$

Maxwell's equations

$$\nabla \times \mathbf{e} + \frac{\partial \mathbf{b}}{\partial t} = 0$$
$$\nabla \times \mu^{-1}\mathbf{b} - \sigma\mathbf{e} = \mathbf{s}(t)$$

Boundary conditions

$$\mathbf{n} \times \mathbf{b} = 0$$

Initial conditions

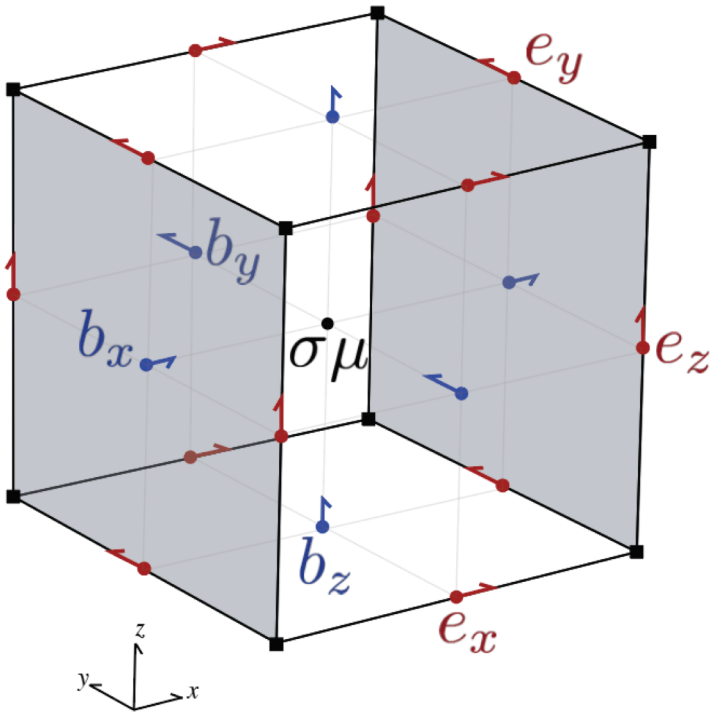
$$\mathbf{e}(x, y, z, t = 0) = \mathbf{e}_0$$
$$\mathbf{b}(x, y, z, t = 0) = \mathbf{b}_0$$

Need to solve in space and time

Semi-discretization in space

Staggered Grid

- Physical properties: cell centers
- Fields: edges
- Fluxes: faces



Continuous second-order equations

$$\nabla \times \mu^{-1} \times \mathbf{e} + \sigma \frac{\partial \mathbf{e}}{\partial t} = -\frac{\partial \mathbf{s}}{\partial t}$$

Semi-discretized second order equations

$$\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C}\mathbf{e} + \mathbf{M}^e_\sigma \frac{\partial \mathbf{e}}{\partial t} = -\frac{\partial \mathbf{s}}{\partial t}$$

Discretizing in time

First order backwards difference (implicit)

- $\mathbf{e}^{n+1}$ depends upon $\mathbf{e}^{n}$

$$\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C}\mathbf{e} + \mathbf{M}^e_\sigma \frac{\partial \mathbf{e}}{\partial t} = -\frac{\partial \mathbf{s}}{\partial t}$$

- Time-step: $\Delta t = t_{n+1} - t_n$

$$\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C}\mathbf{e}^{n+1} + \mathbf{M}^e_\sigma \frac{\mathbf{e}^{n+1} - \mathbf{e}^n}{\Delta t} = -\frac{\mathbf{s}^{n+1} - \mathbf{s}^n}{\Delta t}$$

$$\left(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + \frac{1}{\Delta t} \mathbf{M}^e_\sigma \right) \mathbf{e}^{n+1} = -\frac{\mathbf{s}^{n+1} - \mathbf{s}^n}{\Delta t} + \frac{1}{\Delta t} \mathbf{M}^e_\sigma \mathbf{e}^n$$

Discretizing in time

First order backwards difference (implicit)

$$\underbrace{\left(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + \frac{1}{\Delta t}\mathbf{M}^e_\sigma\right)}_{\mathbf{A}_{n+1}(\sigma,\Delta t)} \underbrace{\mathbf{e}^{n+1}}_{\mathbf{u}_{n+1}} = \underbrace{-\frac{\mathbf{s}^{n+1} - \mathbf{s}^n}{\Delta t}}_{\mathbf{q}_{n+1}} + \underbrace{\frac{1}{\Delta t}\mathbf{M}^e_\sigma}_{\mathbf{B}_n(\sigma,\Delta t)} \underbrace{\mathbf{e}^n}_{\mathbf{u}_n}$$

Arrange in a big matrix

$$\begin{pmatrix} \mathbf{A}_0 & & & & & \\ \mathbf{B}_1 & \mathbf{A}_1 & & & & \\ & \mathbf{B}_2 & \mathbf{A}_2 & & & \\ & & \ddots & \ddots & & \\ & & & \mathbf{B}_{n-1} & \mathbf{A}_{n-1} & \\ & & & & \mathbf{B}_n & \mathbf{A}_n \end{pmatrix} \begin{pmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n-1} \\ \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{n-1} \\ \mathbf{q}_n \end{pmatrix}$$

Solving a TDEM Problem

Solve with forward elimination

- Initial conditions provide $\mathbf{u}_0$
- To propagate forward, solve

$$\mathbf{A}_{n+1}\mathbf{u}_{n+1} = -\mathbf{B}_n\mathbf{u}_n + \mathbf{q}_{n+1}$$

$$\begin{pmatrix} \mathbf{A}_0 & & & & & \\ \mathbf{B}_1 & \mathbf{A}_1 & & & & \\ & \mathbf{B}_2 & \mathbf{A}_2 & & & \\ & & \ddots & \ddots & & \\ & & & \mathbf{B}_{n-1} & \mathbf{A}_{n-1} & \\ & & & & \mathbf{B}_n & \mathbf{A}_n \end{pmatrix} \begin{pmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n-1} \\ \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{n-1} \\ \mathbf{q}_n \end{pmatrix}$$

$$\mathbf{A}_{n+1}(\sigma, \Delta t)$$

- Symmetric
- Need to solve many times
- Only changes if $(\sigma, \Delta t)$ change → store factors

$$\underbrace{\left(\mathbf{C}^\top \mathbf{M}^f_{\mu^{-1}} \mathbf{C} + \frac{1}{\Delta t}\mathbf{M}^e_\sigma\right)}_{\mathbf{A}_{n+1}(\sigma, \Delta t)}$$

Solving a TDEM Problem

$$\begin{pmatrix} \mathbf{A}_0 & & & & & \\ \mathbf{B}_1 & \mathbf{A}_1 & & & & \\ & \mathbf{B}_2 & \mathbf{A}_2 & & & \\ & & \ddots & \ddots & & \\ & & & \mathbf{B}_{n-1} & \mathbf{A}_{n-1} & \\ & & & & \mathbf{B}_n & \mathbf{A}_n \end{pmatrix} \begin{pmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n-1} \\ \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{n-1} \\ \mathbf{q}_n \end{pmatrix}$$

```
u = zeros(n_edges, n_timesteps)  # initialize storage for solutions

u[:, 0] = get_initial_condition(sigma)

for i in range([1,n_timesteps]):
    dt = timesteps(i)

    # only re-factor if timesteps change
    if i==1 or dt != timesteps(i-1):
        A = getA(sigma, dt)
        Ainv = solver(A)  # e.g. Pardiso, MUMPS

    # build the RHS
    B = getB(sigma, timesteps(i-1))
    q = get_q(i)
    rhs = (q - B*u[:, i-1])

    u[:, i] = Ainv * rhs
```

Using Direct Solvers

$$\mathbf{A}\mathbf{u} = \mathbf{q}$$

- $\mathbf{A}$ is symmetric
- Decompose using MUMPS, Pardiso (both freely available)

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\top}$$

- Refactor only if $(\sigma, \Delta t)$ changes
- Divide modelling time into P partitions

- Total computation time

$$T = P(N_{\Delta t} N_{TX} t_{\text{solve}} + t_{\text{factor}})$$

Time to solve factored system

Time to factor system

Solution times for a direct solver

- 70x70x70 mesh
- 60 time steps, 10^{-5} to 10^{-1} s
- t_f=165s $N_{factorizations}$=5
- Mem=40Gb, Dual hex core

- Direct: t_{solve} = 0.8s
- Iterative: 1.3m per Maxwell solution (12 processors)

Num. Transmitters	Time Direct (min)	Time Iterative (min)
1	17	82
10	19	82
100	44	683
1000	290	6833

Importance of time difference is exacerbated in solving the Inverse problem as many forward modellings are needed

FDEM and TDEM Simulations

FDEM

- $\mathbf{A}(\sigma,\omega)$
  - Complex, symmetric
  - Factor for each frequency
  - Inversion: sensitivity derivation straight-forward

TDEM

- $\mathbf{A}_{n+1}(\sigma,\Delta t)$
  - Real, symmetric
  - Factor for each unique, Δt
  - Inversion: sensitivity derivation more involved

FDEM and TDEM Simulations

FDEM

- $\mathbf{A}(\sigma,\omega)$
  - Complex, symmetric
  - Factor for each frequency

- e.g. RESOLVE
  - 1000 source locations
  - 5 frequencies
    - 5 factorizations of complex system

$$T = N_\omega(N_{TX}t_{\text{solve}} + t_{\text{factor}})$$
$$= 5 \cdot (10^3 \cdot t_{\text{solve}} + t_{\text{factor}})$$

TDEM

- $\mathbf{A}_{n+1}(\sigma,\Delta t)$
  - Real, symmetric
  - Factor for each unique, Δt

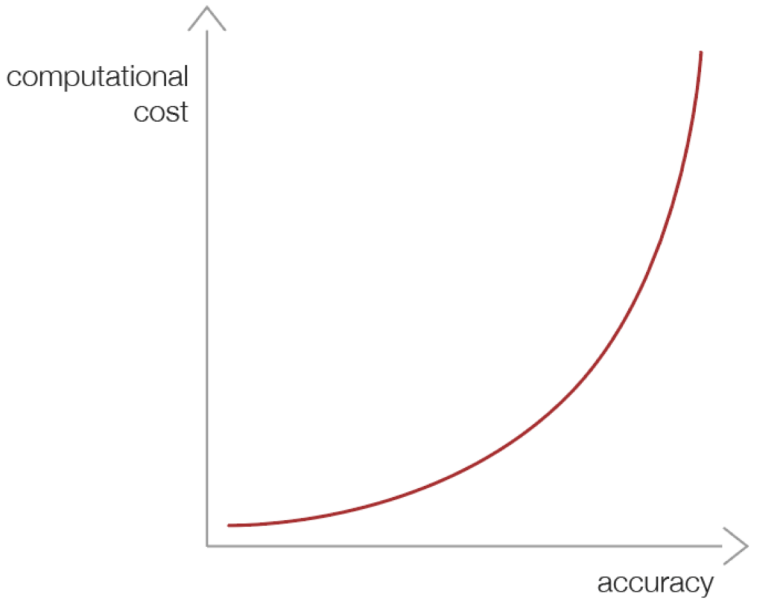
- e.g. Co-located loop time domain
  - 1000 source locations
  - 60 timesteps, 6 unique Δt
    - 6 factorizations of real system

$$T = P(N_{\Delta t}N_{TX}t_{\text{solve}} + t_{\text{factor}})$$
$$= 6 \cdot (10 \cdot 10^3 \cdot t_{\text{solve}} + t_{\text{factor}})$$

Setting up a forward simulation

Trade-off between computation cost and accuracy

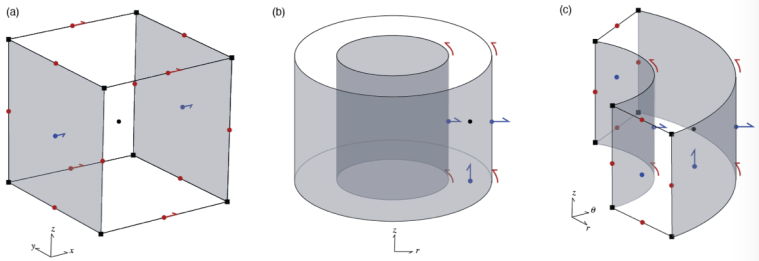
- Mesh design
  - Smallest cell
    - Capture shortest time / highest frequency and highest conductivity
  - Extent of domain / number of padding cells
    - Beyond skin depth / diffusion distance
- Time discretization
  - Capture short-timescale variations near shut-off
  - Extend to latest time channel



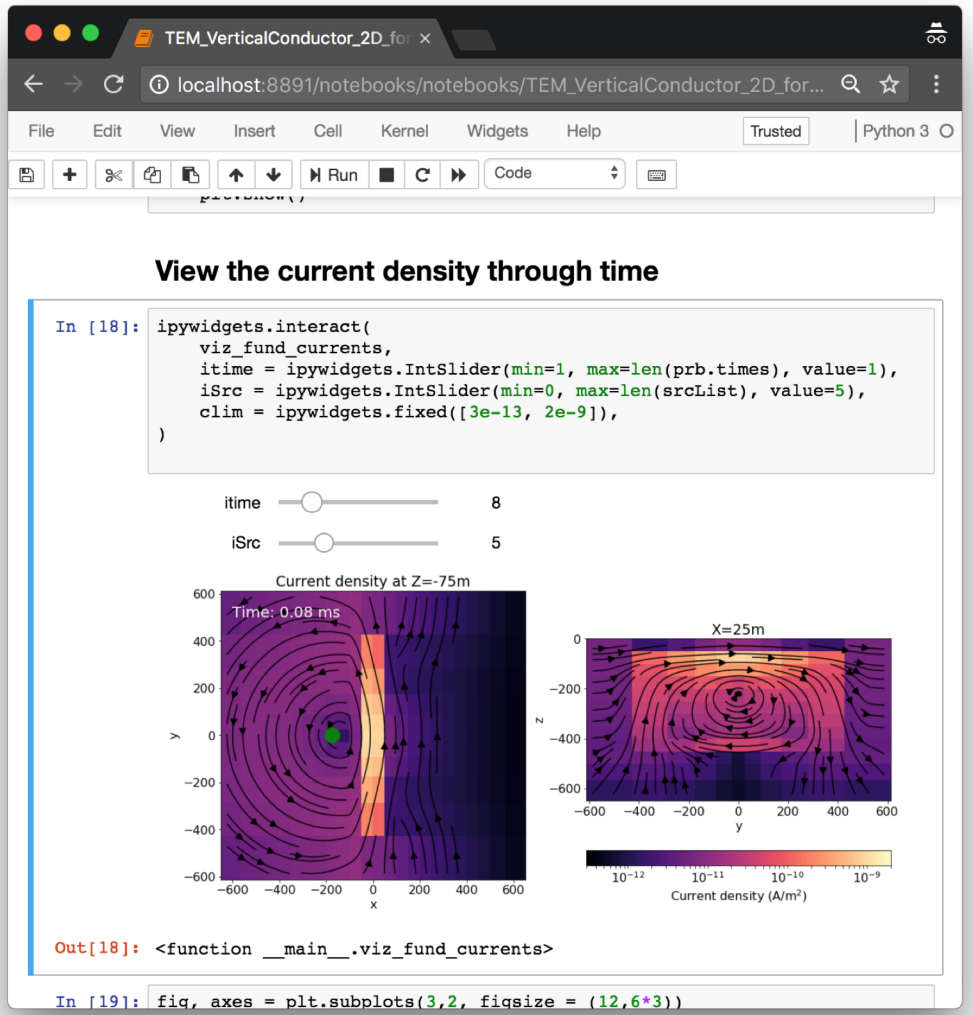
3D EM forward modelling

SimPEG EM module

- FDEM and TDEM
- Tensor, 2D and 3D cylindrical meshes, OcTree meshes



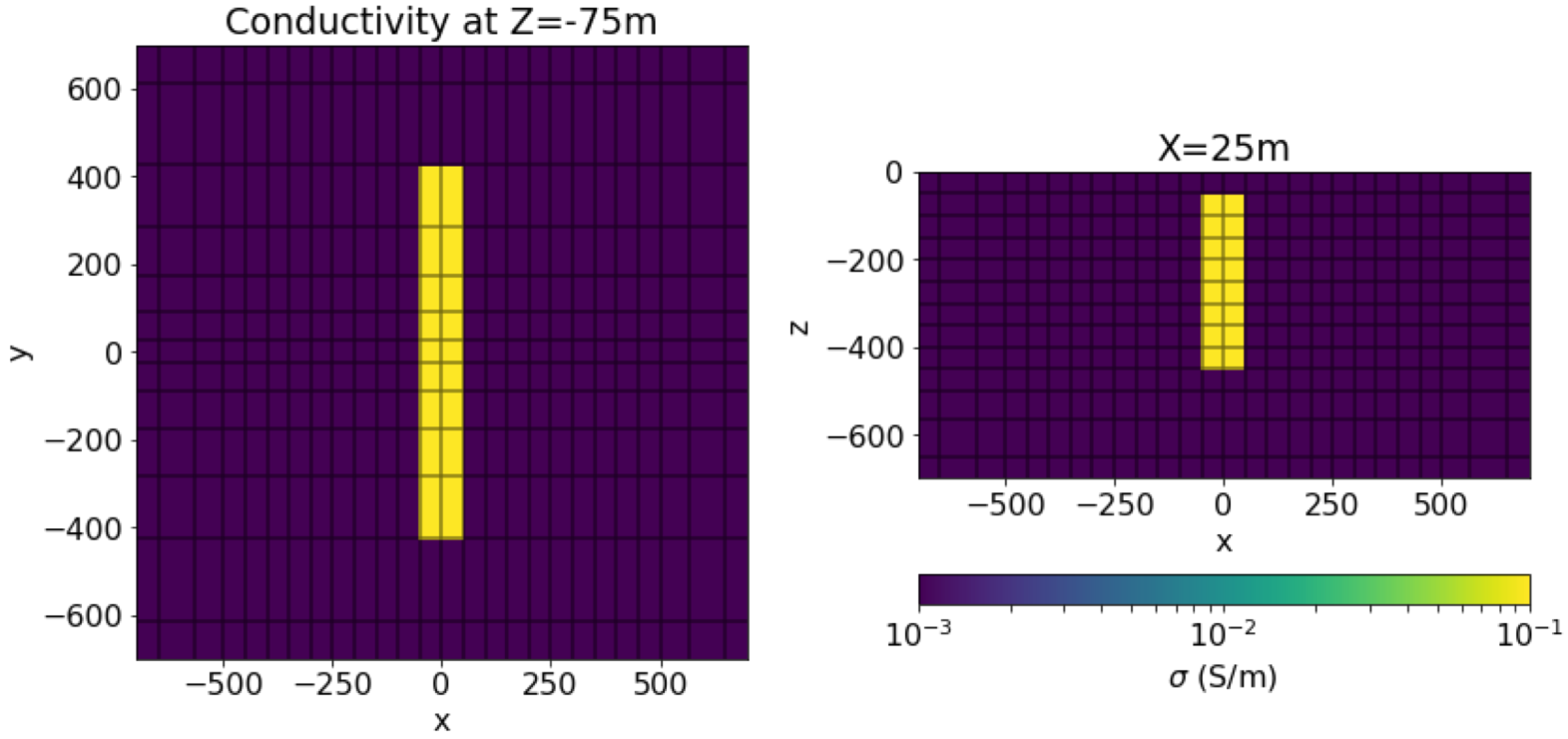
- Readily visualize fields, fluxes, charges
- Connected to inversion machinery



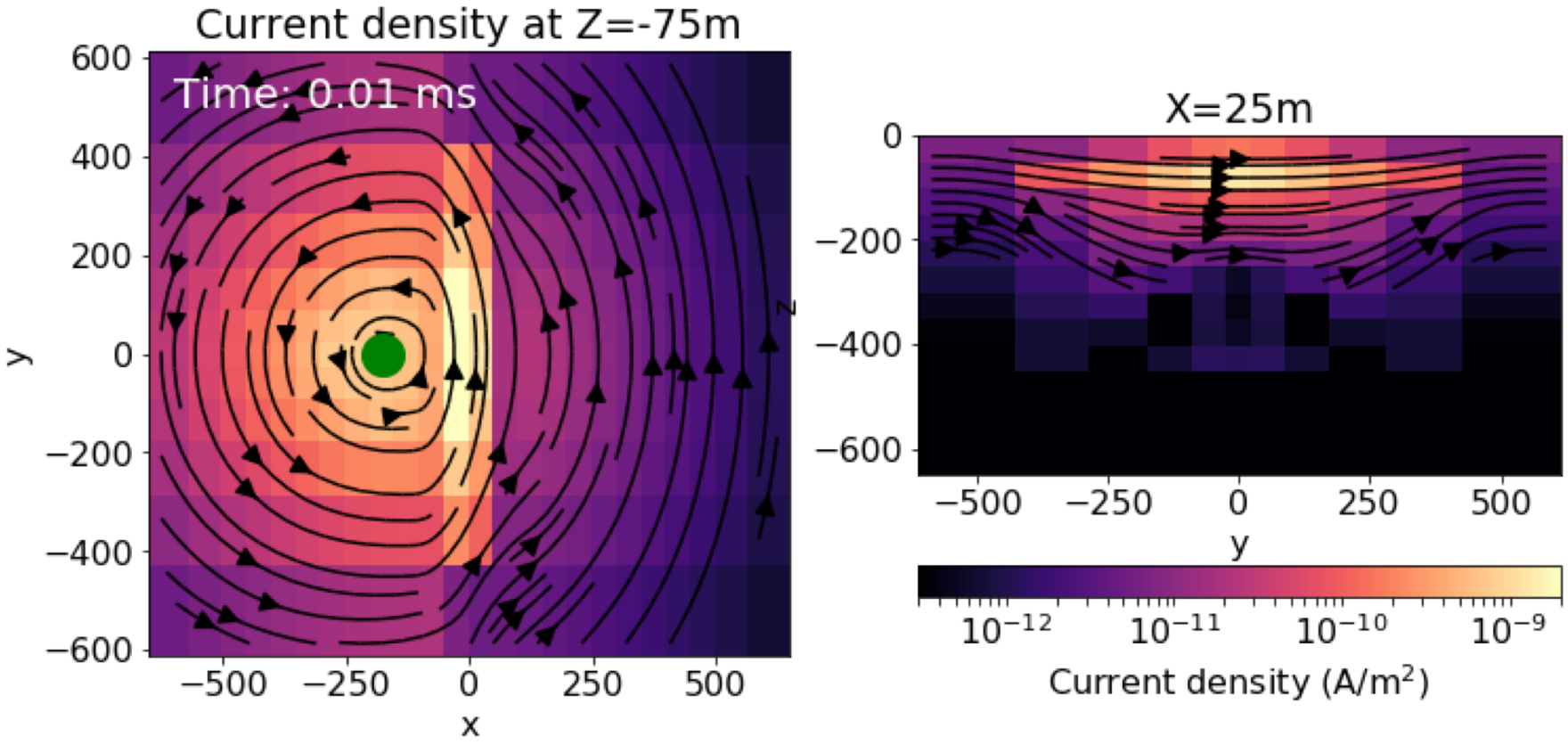
(Cockett et al., 2015; Heagy et al., 2017)

Conductive plate in a halfspace

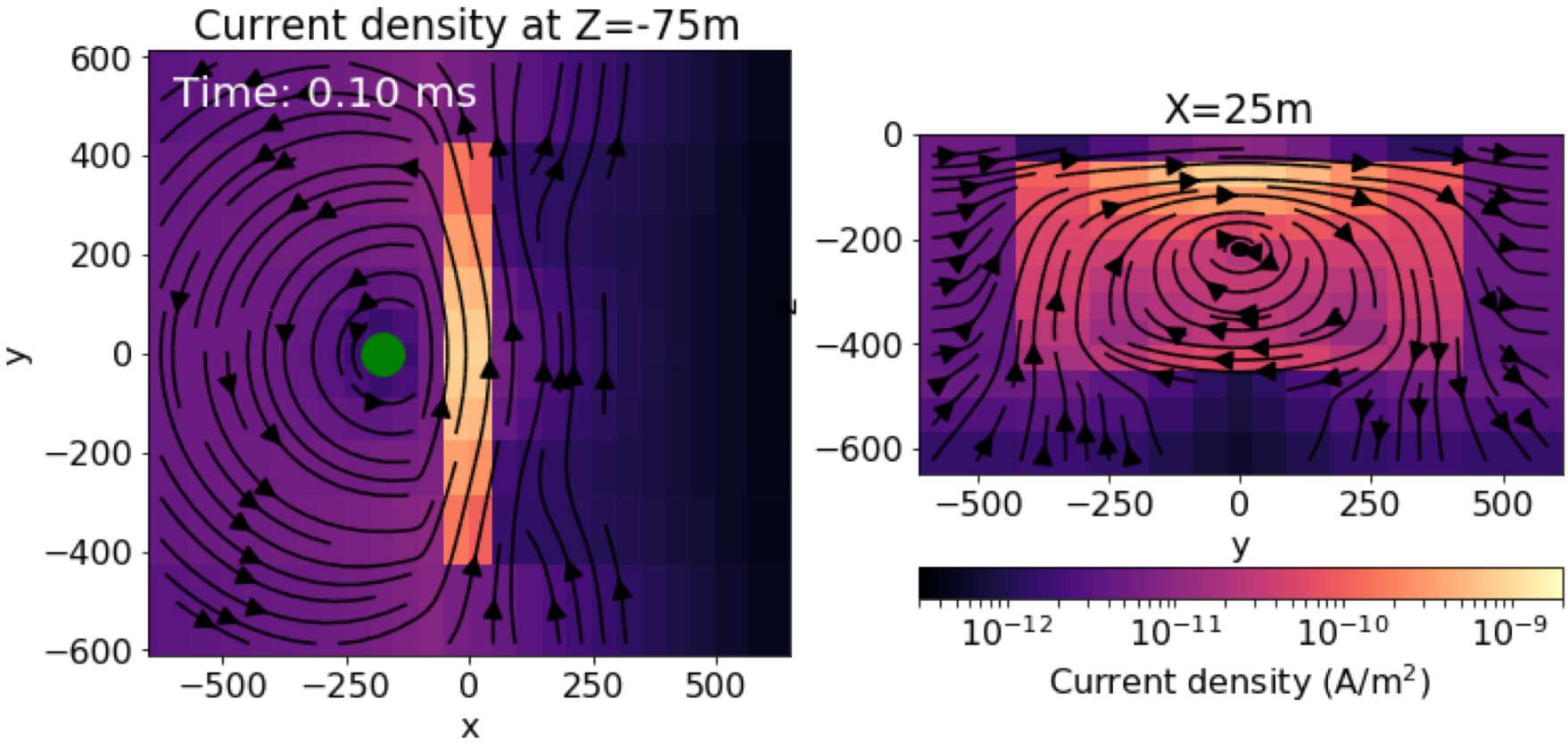
- Survey design:
  - Excitation of the target
  - Which fields to measure



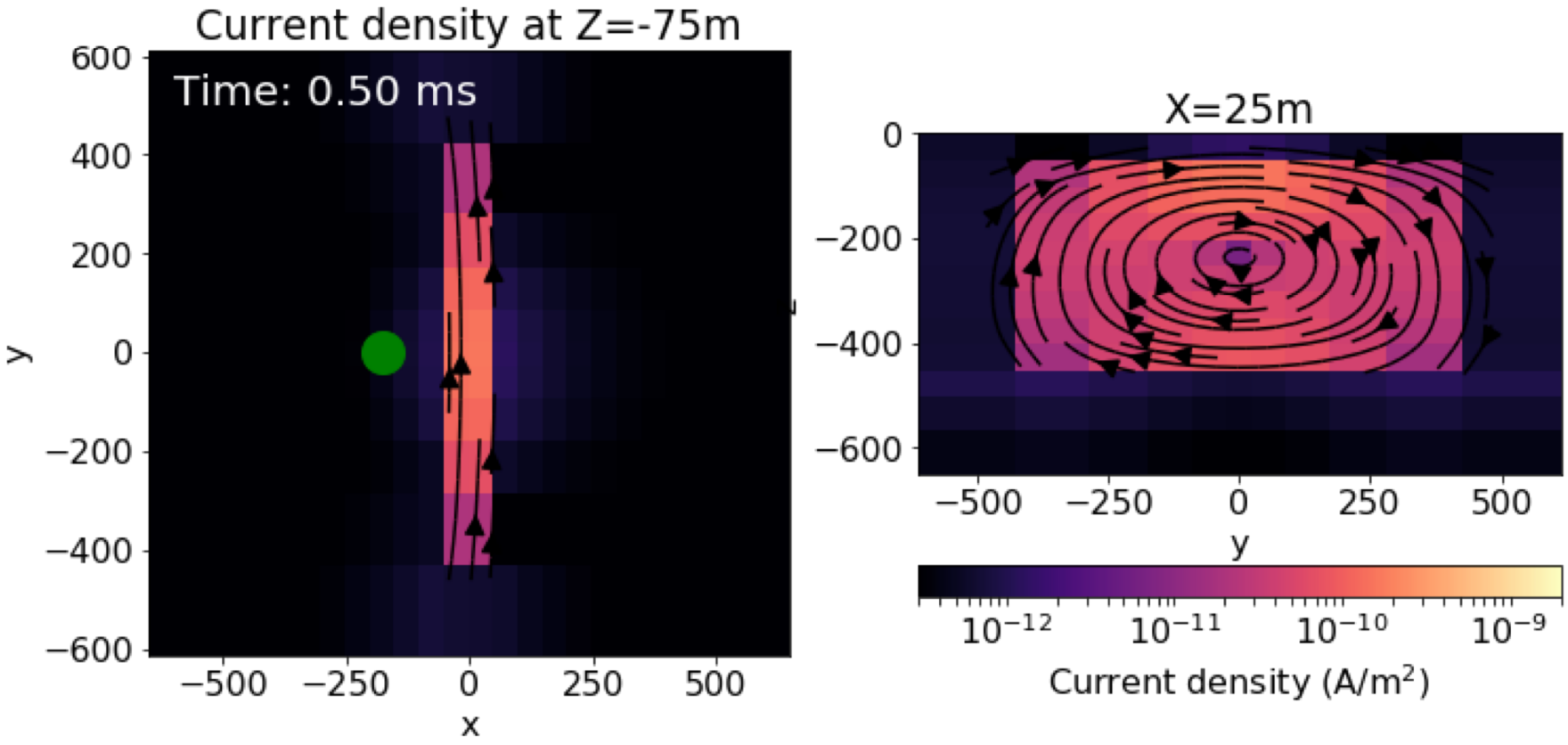
Conductive plate in a halfspace



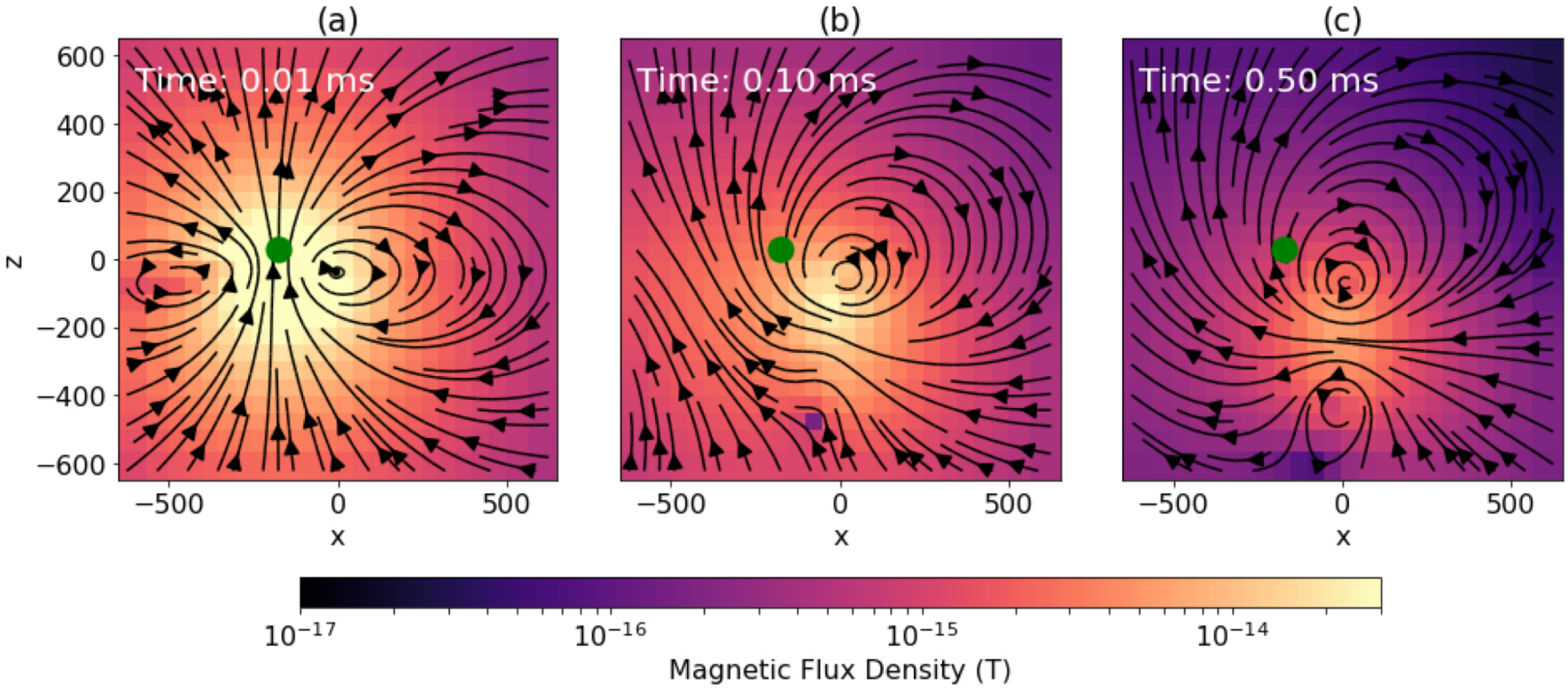
Conductive plate in a halfspace



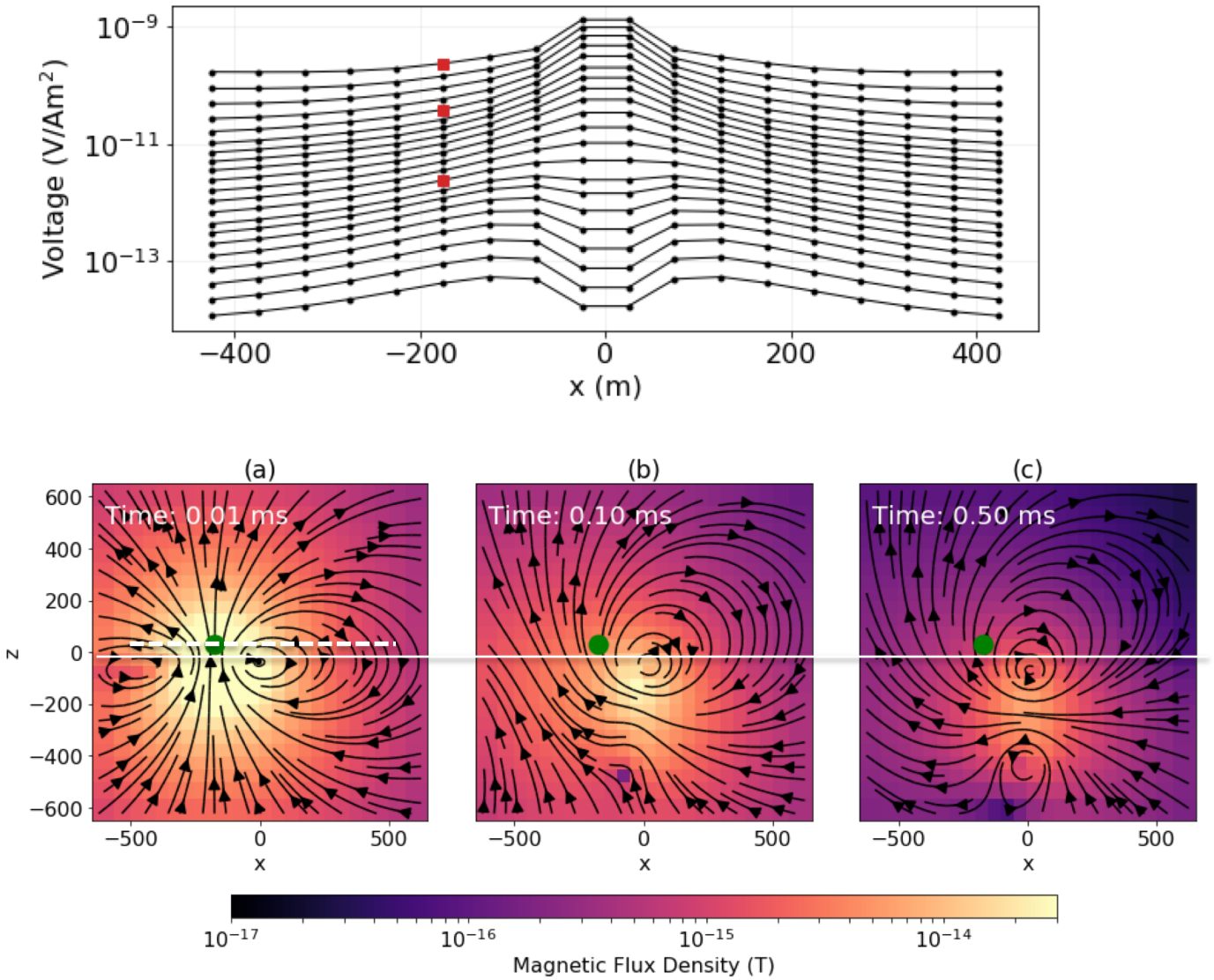
Conductive plate in a halfspace



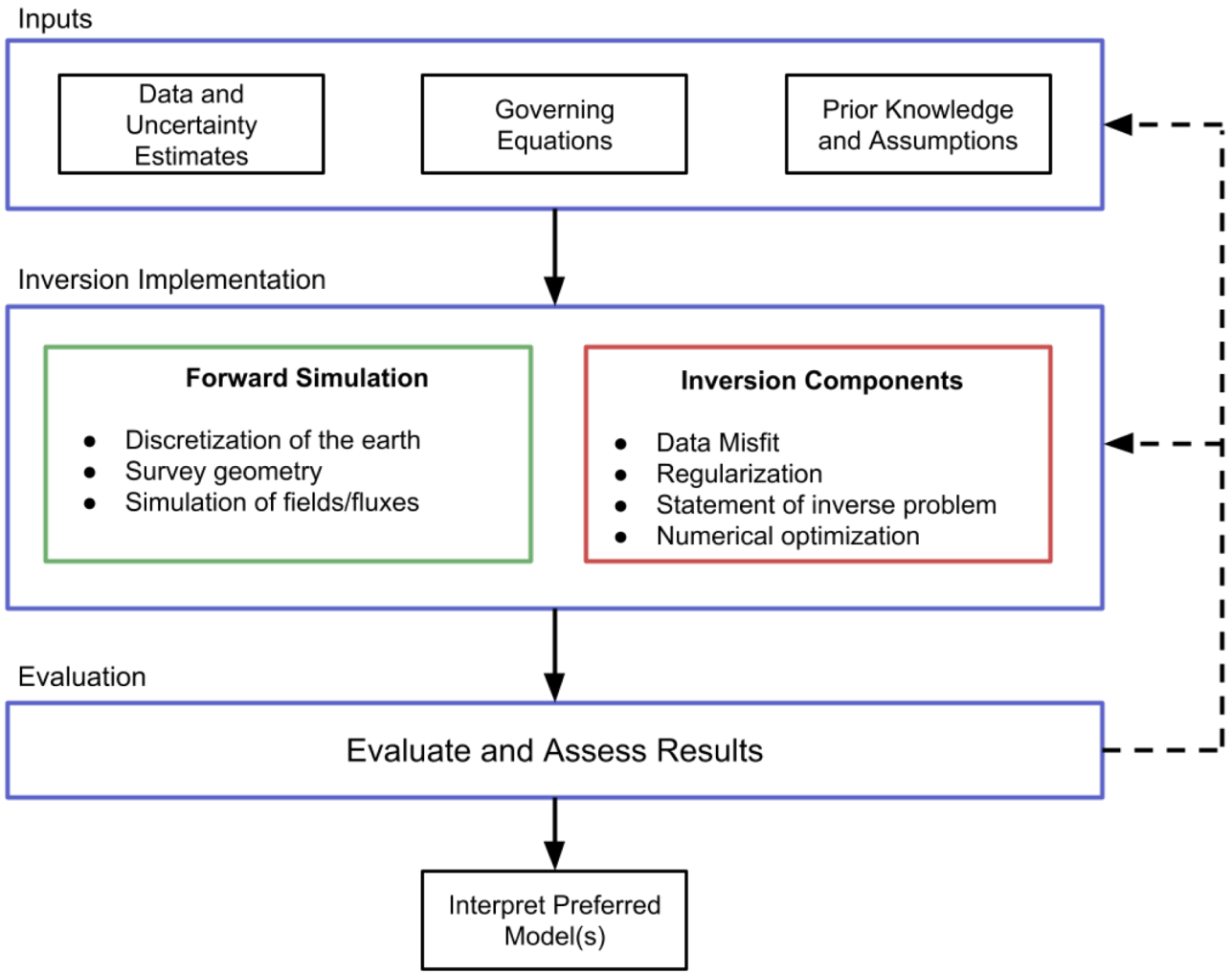
Conductive plate in a halfspace



Conductive plate in a halfspace



Inversion flow chart



Inverse problem

- Minimize

$$\phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$\text{subject to} \qquad \mathbf{m}_{lower} < \mathbf{m} < \mathbf{m}_{upper}$$

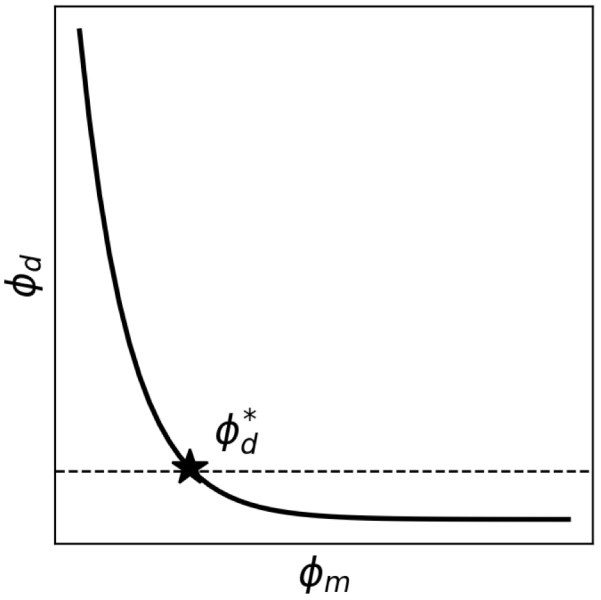
Data misfit

$$\phi_d(\mathbf{m}) = \frac{1}{2}\|\mathbf{W}_\mathrm{d}(F[\mathbf{m}] - \mathbf{d}_\mathrm{obs})\|_2^2.$$

Regularization

$$\phi_m(\mathbf{m}) = \frac{1}{2}\|\mathbf{W}_\mathrm{m}(\mathbf{m} - \mathbf{m}_\mathrm{ref})\|_2^2.$$

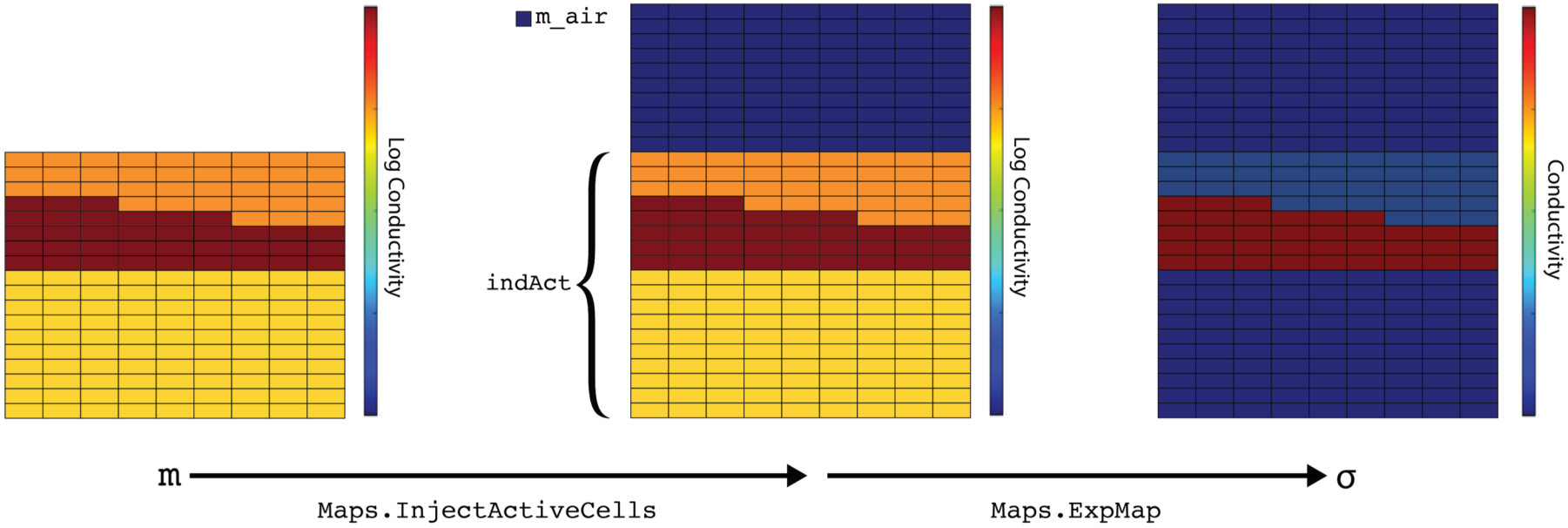
Tikhonov curve



What is your model?

- Subsurface log conductivity
- 1D, 2D, 3D voxel model
- Parametric model
- …

- Need to map to forward simulation mesh
- Keep track of derivatives for inversion



Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

- Taylor expand: Gauss Newton equation

$$(\mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d \mathbf{J} + \beta \mathbf{W}_m^\top \mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$$

Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

- Taylor expand: Gauss Newton equation

$$(\mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d \mathbf{J} + \beta \mathbf{W}_m^\top \mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$$

- Sensitivity $\mathbf{J}$
  - Requires forward modelling
  - Large, dense matrix
    - For large problems: calculate action on a vector $\mathbf{J}\mathbf{v}, \mathbf{J}^\top\mathbf{v}$

Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

- Taylor expand: Gauss Newton equation

$$(\mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d \mathbf{J} + \beta \mathbf{W}_m^\top \mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$$

- Sensitivity $\mathbf{J}$

$$\mathbf{J}\mathbf{v} = \mathbf{P}\mathbf{A}^{-1}\mathbf{G}\mathbf{v}$$

$$\mathbf{J}^\top\mathbf{v} = \mathbf{G}^\top\mathbf{A}^{-\top}\mathbf{P}^\top\mathbf{v}$$

$\mathbf{A}$: system matrix

$\mathbf{P}$: computes data at receivers

$\mathbf{G} = \nabla_{\mathbf{m}}(\mathbf{A}(\mathbf{m})\mathbf{u})$: derivative of the product of system matrix and solution (fixed $\mathbf{u}$)

Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

- Taylor expand: Gauss Newton equation

$$(\mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d \mathbf{J} + \beta \mathbf{W}_m^\top \mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$$

- Use inexact PCG to solve for model update (N_{CG} iterations)

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \delta\mathbf{m}$$

Gauss-Newton approach

- Inverse problem

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

- Gradient

$$\mathbf{g}(\mathbf{m}) = \mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}^{obs}) + \beta \mathbf{W}_m^\top \mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})$$

- Taylor expand: Gauss Newton equation

$$(\mathbf{J}^\top \mathbf{W}_d^\top \mathbf{W}_d \mathbf{J} + \beta \mathbf{W}_m^\top \mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$$

- Use inexact PCG to solve for model update (N_{CG} iterations)

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \delta\mathbf{m}$$

Number of forward modellings: $2(N_{CG} + 1) \sim 20$

Gauss-Newton approach

$$\min_{\mathbf{m}} \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m})$$

$$= \frac{1}{2}\|\mathbf{W}_d(F[\mathbf{m}]) - \mathbf{d}^{obs}\|^2 + \frac{\beta}{2}\|\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref})\|^2$$

Choose β_0, $\mathbf{m}_{ref}$

Evaluate $\boldsymbol{\phi}(\mathbf{m}_{ref})$, $\mathbf{g}(\mathbf{m}_{ref})$, matrices $\mathbf{W}_d$, $\mathbf{W}$...

for i in range([0, max_beta_iter]):

    for k in range([0, max_inner_iterations]):

- IPCG to solve $(\mathbf{J}^\top\mathbf{W}_d^\top\mathbf{W}_d\mathbf{J} + \beta\mathbf{W}_m^\top\mathbf{W}_m)\delta\mathbf{m} = -\mathbf{g}(\mathbf{m})$
- line search for step length α
- Update model $\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha\delta\mathbf{m}$
- Exit if $\phi < \phi_d^*$ or $\dfrac{\|\mathbf{g}(\mathbf{m}_{k+1})\|}{\|\mathbf{g}(\mathbf{m}_k)\|} < \text{tol}$

    Reduce β

Tally up the computations

Number of transmitters	1000
Number of time steps	50
Solving a GN step	20
Number of GN iterations	20

Tally up the computations

Number of transmitters	1000
Number of time steps	50
Solving a GN step	20
Number of GN iterations	20

- Total number of Maxwell solutions is 20,000,000

Tally up the computations

Number of transmitters	1000
Number of time steps	50
Solving a GN step	20
Number of GN iterations	20

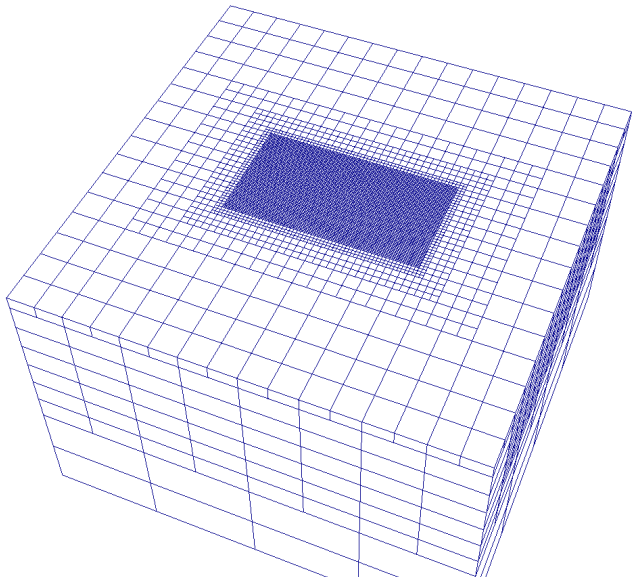
- Total number of Maxwell solutions is 20,000,000
- Suppose: t_{factor}=1 sec
  - 100 processors: 55 hours
  - 1000 processors 5.5 hours

Need:
- Fast forward modelling
- Multiple cpu

Some important components

- Trade off (accuracy vs computation)
- Consider a 3D airborne EM simulation (1000 sources)

Octree mesh



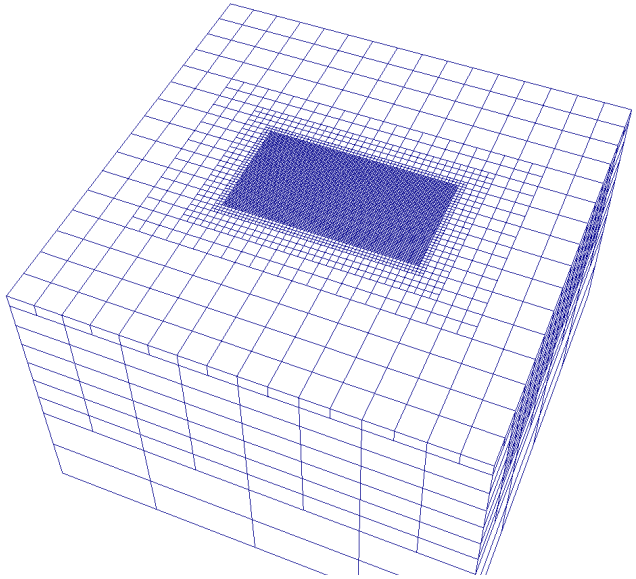
> 1,000,000 cells (this is big!)

How do we tackle this?

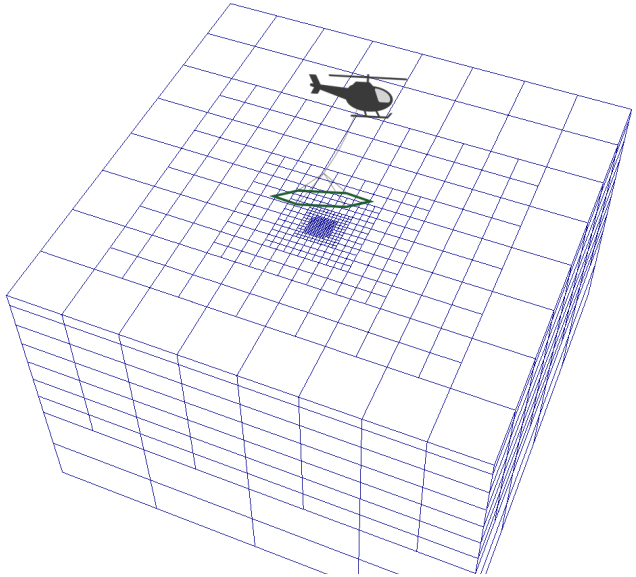
Some important components

- Trade off (accuracy vs computation)
- Separate forward modelling mesh for each transmitter

Global mesh



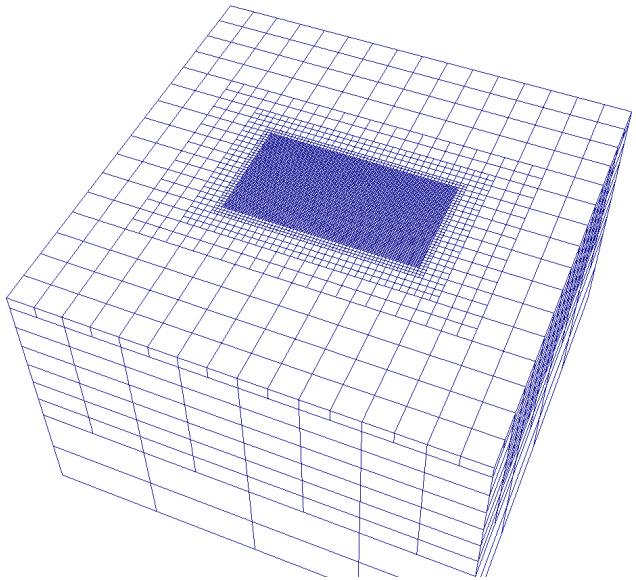
Local mesh



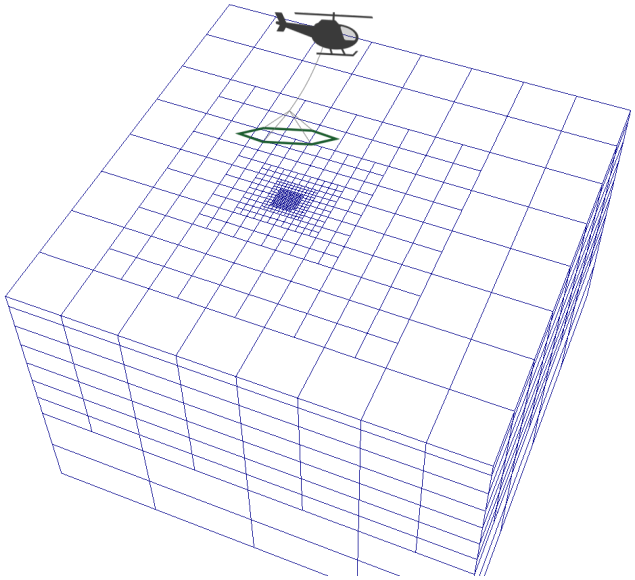
Some important components

- Trade off (accuracy vs computation)
- Separate forward modelling mesh for each transmitter

Global mesh



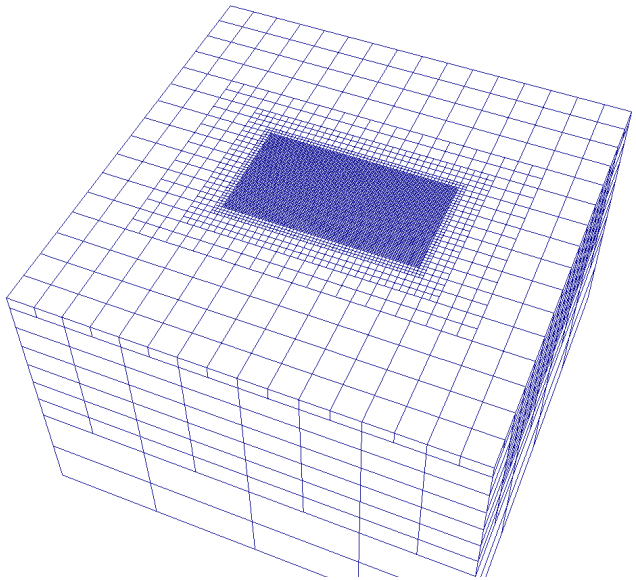
Local mesh



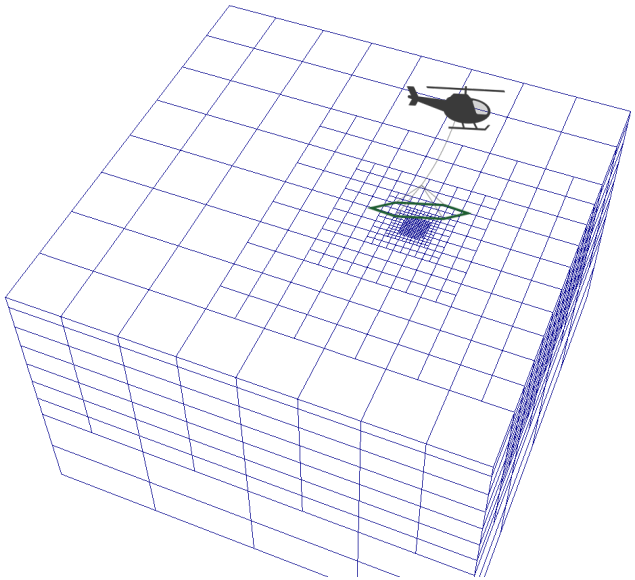
Some important components

- Trade off (accuracy vs computation)
- Separate forward modelling mesh for each transmitter

Global mesh



Local mesh

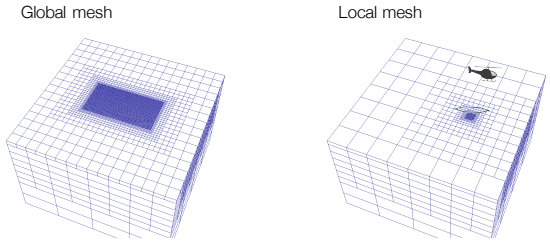


Computations: Summary

Airborne, and other EM problems, are hard!

Advances:

- Direct solvers (factor Maxwell operator)
- Semi-structured meshes (OcTree, reduce the # of variables)
- Separating forward and inverse meshes
- Handling the sensitivity matrix
- Access to multi-cores



Generating quality codes for research and processing is too challenging for one person or small group.

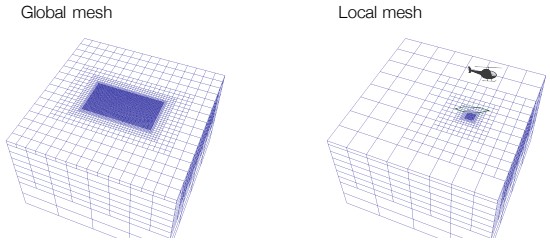
What is the path forward?

Computations: Summary

Airborne, and other EM problems, are hard!

Advances:

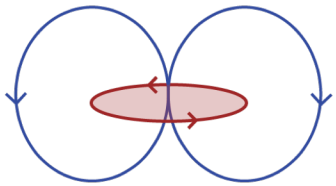
- Direct solvers (factor Maxwell operator)
- Semi-structured meshes (OcTree, reduce the # of variables)
- Separating forward and inverse meshes
- Handling the sensitivity matrix
- Access to multi-cores



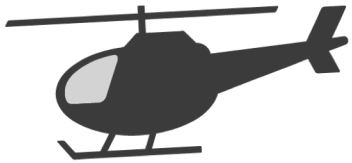
Generating quality codes for research and processing is too challenging for one person or small group.

What is the path forward?

We need a community!



EM Induction



Field Examples

Computation

Open Source Software