

## Mapping with Occam

The Occam inversion method is specifically designed for nonlinear inversion of geophysical data. In the original paper the method was introduced using 1D resistivity and 1D magnetotelluric (MT) data and models; was quickly extended to 2DMT models; and has been applied to many other problems since. While the original Occam paper was the first example I am aware of that used Tikhonov regularization for nonlinear geophysical inversion, the basic idea of regularized inversion was not new. The essence of the Occam method is not so much the use of regularization, but the assignment of a target misfit  $\chi_*^2$  and then the use of a line search over the Lagrange multiplier  $\mu$  at each iteration in order to achieve the target misfit  $\chi_*^2$ . Other methods simply set  $\mu$  and then minimize the resulting combination of misfit and model penalty.

Going back to the original paper, we introduced the methodology using a linear forward problem, which can always be cast as

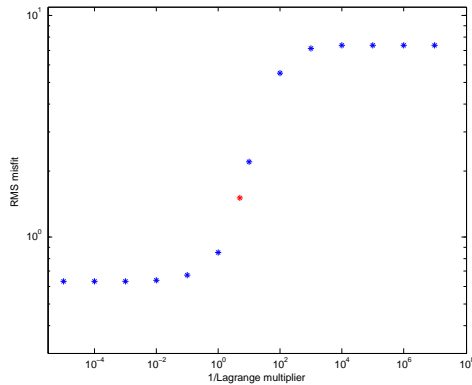
$$\hat{\mathbf{d}} = \mathbf{G}\mathbf{m}$$

where  $\hat{\mathbf{d}}$  are the predicted data values,  $\mathbf{G}$  a matrix of weights describing the linear forward problem, and  $\mathbf{m}$  is a vector of model parameters. Note that although  $\mathbf{m}$  is always a column vector, it can represent a 2D surface by taking sequential columns from a 2D matrix and stacking them together (and similarly 3D volumes, or even 4D problems). We then showed that given an appropriate choice of  $\mu$ , the regularized model is given by

$$\mathbf{m} = [\mu \mathbf{R}^T \mathbf{R} + (\mathbf{W}\mathbf{G})^T \mathbf{W}\mathbf{G}]^{-1} (\mathbf{W}\mathbf{G})^T \mathbf{W}\mathbf{d}$$

where  $\mathbf{R}$  is a matrix generating some penalty on the model, usually a roughness measure obtained by taking first differences of adjacent parameters, and  $\mathbf{W}$  is a diagonal matrix of reciprocal data errors.  $\mathbf{d}$  of course are the real data.

As we know,  $\mu$  trades off roughness against data misfit. One expects that for linear problems, misfit can be driven to zero as long as there are more model parameters than data. However, if there are multiple, inconsistent, data at a given location, or at least close enough that they are influenced by a single model parameter, then this may not be the case.

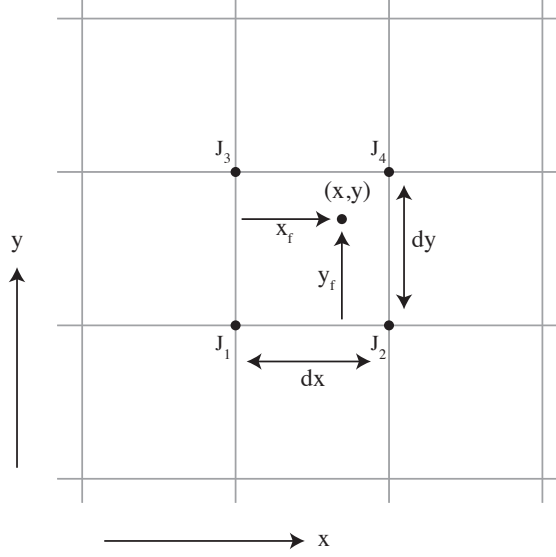


In practice, no matter how rough you make the model (small  $\mu$ ) there will be some minimum misfit associated with incompatible data. Making the model smooth increases misfit, but only so much: a maximally smooth model (all parameters equal) cannot be made smoother by increasing  $\mu$  and the misfit saturates to a large value. The plot to the left is a typical misfit versus Lagrange multiplier tradeoff for the problems we discuss below. The red point is the chosen value to achieve an RMS misfit of 1.50. Here the smallest misfit possible is RMS 0.63.

The mapping problem is simply a linear forward problem where the forward model is just the values of the data themselves. But, this presents some challenges if the data are not uniformly distributed, or have multiple values at a given location. Also, most of the existing mapping schemes weight the data equally.

Occam presents a useful solution to this problem. With a first derivative roughness, the map will go flat where there are no data constraints. The  $\mathbf{W}$  matrix can have unique values for each data point, and one can use the data errors and misfit value to choose just how smooth the map should be. The model should not depend on the level of discretization (although the computer time does – something like the 5th power of the number of model parameters along one axis!), and we can build a model on a uniform grid that will work with plotting programs after the fact.

Let's do it...



First we need to set up a model of a map, by taking a 2D array of points on a plane. In `OccamMap.m` we provide minimum  $x$ , maximum  $x$ , and number of nodes  $n_x$  (similarly for  $y$ ).

The way `OccamMap.m` works, we find the position of  $(x, y)$  for each data point in map coordinates (i.e. the fractional indices of the mesh nodes  $(1 : n_x)$  and  $(1 : n_y)$  where  $n_x$  and  $n_y$  are the number of model parameters on each axis). The lower left corner of the box,  $J_1$ , is simply the integer value of this number (`floor()`) and the fractional part is the distance into the box in the two directions  $(x_f, y_f)$  from  $J_1$ .

Note that inside Occam, the model is not a matrix but a vector, taken columnwise (i.e. along the  $y$  directions) across  $x$ . Physically, we interpret  $y$  as positive *up*, but in the matrix it increases *down* the columns.

For mapping Occam, we can then find the distances  $w_i$  to the corners  $J_i$ :

$$w_1 = \sqrt{x_f^2 + y_f^2}$$

$$w_2 = \sqrt{(1 - x_f)^2 + y_f^2}$$

$$w_3 = \sqrt{x_f^2 + (1 - y_f)^2}$$

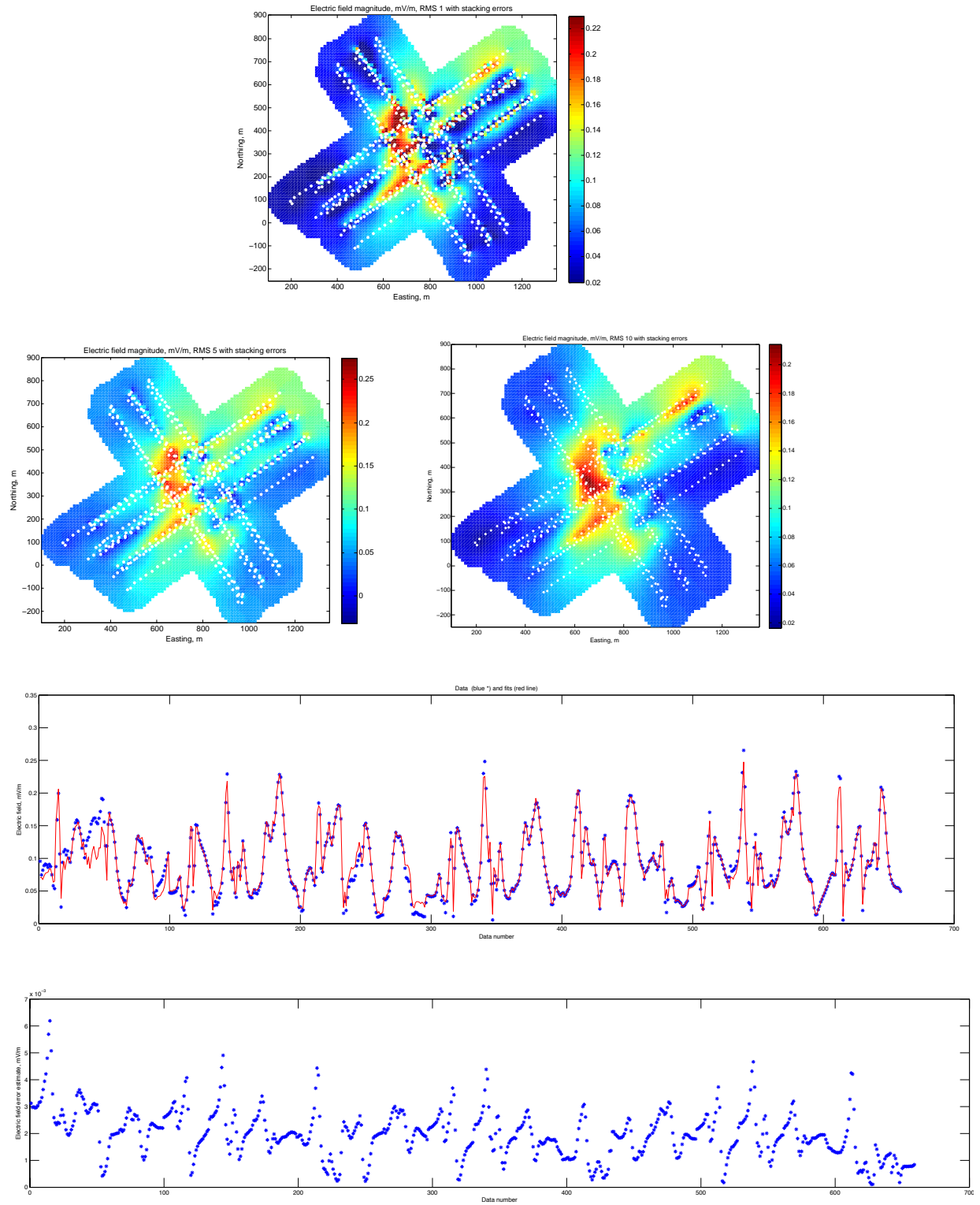
$$w_4 = \sqrt{(1 - x_f)^2 + (1 - y_f)^2}$$

One prediction for the value at  $(x, y)$  is a weighted average of the values at  $J_i$ , the weights being proportional to how close the values are to the nodes. In the code I use  $\sqrt{2} - w_i$ , because the maximum value  $w_i$  can be is  $\sqrt{2}$ . I normalize the  $\sqrt{2} - w_i$  weights for each data point by the sum  $s$  of the weights, which can vary from 2.4 to 3.4, so that they add to one.

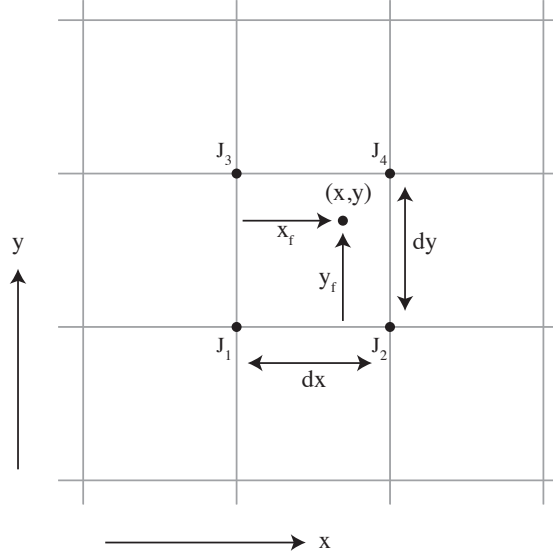
The  $G(j, i)$  matrix entries for the row representing  $d_j$  are then  $(\sqrt{2} - w_i)/s$  at the model column matrix entries representing  $J_i$ .

The following plots show fits to electric field magnitudes for RMS 1, 5 and 10, along with the data fits for the RMS 5 model. I used the data errors from the stacking process (also shown), which are largely a measure of how much the electric field is changing over the 30 second averaging window. The errors are less than 5  $\mu/m$ , and not large enough to

capture electrode drift and navigation errors, so I would prefer the misfits greater than RMS 1.0. Remarkably, Occam is able to fit these data to RMS 0.001, or about a nV/m.



## Self Potential Modeling using Occam



Now, for the self potential case we need approximations to  $-dV/dx$  and  $-dV/dy$ . Within the box we have two approximations to each of these, one from each edge:

$$-\frac{dV}{dx} \approx \frac{J_1 - J_2}{dx} \quad \text{and} \quad -\frac{dV}{dx} \approx \frac{J_3 - J_4}{dx}$$

and

$$-\frac{dV}{dy} \approx \frac{J_1 - J_3}{dy} \quad \text{and} \quad -\frac{dV}{dy} \approx \frac{J_2 - J_4}{dy}$$

We can average these, but better yet weight the averages by how close the data point is to the edges of the box from which we are approximating the gradients:

$$-\frac{dV}{dx} \approx (1 - y_f) \frac{J_1 - J_2}{dx} \quad \text{and} \quad -\frac{dV}{dx} \approx y_f \frac{J_3 - J_4}{dx}$$

and

$$-\frac{dV}{dy} \approx (1 - x_f) \frac{J_1 - J_3}{dy} \quad \text{and} \quad -\frac{dV}{dy} \approx x_f \frac{J_2 - J_4}{dy}$$

Summing the weighted terms:

$$-\frac{dV}{dx} \approx \frac{(1 - y_f)}{dx} J_1 - \frac{(1 - y_f)}{dx} J_2 + \frac{y_f}{dx} J_3 - \frac{y_f}{dx} J_4$$

and

$$-\frac{dV}{dy} \approx \frac{(1 - x_f)}{dy} J_1 + \frac{x_f}{dy} J_2 - \frac{(1 - x_f)}{dy} J_3 - \frac{x_f}{dy} J_4$$

The data estimates for  $-dV/dx$  and  $-dV/dy$  are the  $x$  and  $y$  components of the measured electric fields, turned into one data vector by adding the  $y$  component after the  $x$  component. The coefficients for the  $j$ th data point are just the coefficients above for the  $J_i$  at the appropriate model entries, first for  $x$  and then for  $y$ .

The following plots show the self potential computed in this way for RMS 5 and RMS 10. They are encouragingly similar, except for a difference in peak amplitude. The minimum RMS in this case is about 4.0. If you try to extract the electric fields from the slope of these models, you get the right sort of numbers, but this is a very noisy process.

