Applications of the Levenberg-Marquardt Algorithm to the Inverse Problem

Mark Bun

October 19, 2009

Abstract

This paper continues the exploration of numerical recovery algorithms that George Tucker, Sam Whittle and Ting-You Wang presented in [4]. We investigate several implementations and applications of the Levenberg-Marquardt nonlinear least-squares optimization algorithm, studying its effectiveness on various electrical networks.

Contents

1 Introduction

Let $G = (V, E)$ be a connected graph with boundary, and let $\gamma : E \to \mathbb{R}^+$ be a conductivity function on E. We take $\Gamma(G, \gamma)$ to be an electrical network. We begin with some standard definitions, and refer the reader to [1] and [4] for further discussion:

Definition 1.1 (Kirchhoff Matrix)**.** Suppose the vertex set V is partitioned as $V = \{v_0, v_1, \ldots, v_m, v_{m+1}, \ldots v_{m+l}\}$ where $\{v_0, \ldots, v_m\}$ are boundary nodes and $\{v_{m+1}, \ldots v_{m+l}\}$ are interior nodes. Then the Kirchhoff matrix K is the $(m+l) \times (m+l)$ matrix with entries given by:

$$K_{ij} = \begin{cases} -\gamma_{ij} & \text{if } i \neq j \text{ and there is an edge between } v_i \text{ and } v_j \\ \sum_k \gamma_{ik} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

We note that by our partitioning of V, we can naturally express the Kirchhoff matrix in block form as

$$K = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix},$$

where A represents boundary-to-boundary connections, B and B^T represent interior-to-boundary connections and C represents interior-interior connections.

Definition 1.2 (Response Matrix)**.** The response matrix Λ is the $m \times m$ map taking boundary voltages to boundary currents, and is expressed as the Schur complement of C in K (i.e. $K \backslash C$):

$$\Lambda = A - BC^{-1}B^T.$$

Note that the response matrix is also a Kirchhoff matrix (for a different electrical network).

Suppose we are given a graph G and a response matrix Λ_0. The inverse problem is to use this information to recover the conductivity function γ.

Let $(e_0, e_1, \ldots, e_{n-1})$ be a consistent ordering of the edge set E. We diverge from the notation in [4] by letting $\mathbf{x} = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1}) \in \mathbb{R}^n$, recasting the recovery problem into an optimization problem over $\mathbb{R}^n$. We define the following functions:

$$\mathbf{F}(\mathbf{x}) = \text{vec}(\Lambda(\mathbf{x}) - \Lambda_0) \in \mathbb{R}^{m^2},$$

$$f(\mathbf{x}) = \frac{1}{2}|\mathbf{F}(\mathbf{x})|^2.$$

Here, $\Lambda(\mathbf{x})$ is the response matrix computed using the components of $\mathbf{x}$ as conductivities. We note that $\mathbf{x}$ is a solution to the inverse problem if and only if

$f(\mathbf{x}) = 0$. Since $f \geq 0$, a solution will be a global minimizer for f. Therefore, assuming Λ_0 is a valid response matrix, we can reformulate the electrical conductivity inverse problem as

$$\arg\min_{\mathbf{x}\in\mathbb{R}_+{}^n} f(\mathbf{x}). \tag{1}$$

That is, we want to find $\{\mathbf{x} : f(\mathbf{x}) \text{ is minimized}\}$. If the inverse problem is well posed, we will have a unique solution. In this paper, we explore various iterative methods for solving (1) and similar formulations.

1.1 Notation

We denote the gradient of a scalar-valued function $f(\mathbf{x})$ by $\nabla f(\mathbf{x})$ and the Hessian by $H_f(\mathbf{x})$. The Jacobian (matrix) of a vector-valued function $\mathbf{F}(\mathbf{x})$ will be denoted by $J_{\mathbf{F}}(\mathbf{x})$. We note that $H_f = J_{\nabla f}$. If it is convenient and clear from context, we may drop the subscripts of Jacobians and Hessians.

To minimize confusion, I will do my best to display scalars and matrices in regular typeset, and vectors in bold.

2 Implementation Notes

Whereas the computations in [4] were run primarily in C and Matlab, all of the algorithms in this paper were implemented and tested in Sage 4.0.2. There were several reasons for this choice:

- Inputting and manipulating graphs and matrices is extremely straightforward.
- Python (the programming language Sage is based on) is almost as intuitive to read and write as pseudocode, which I hope will inspire further examination and enhancement of my code.
- Sage is very quick with symbolic manipulations.
- Sage allows easy access to functions in Python libraries like NumPy and SciPy.
- It's free, so I can use it on my laptop.

However, the Sage implementation results in some limitations as well. For instance, since Python is interpreted, the speed at which we can make computations is severely restricted.

All of the computations in this paper were done using double-precision floating point numbers.

Definition 2.1 (Machine epsilon)**.** Machine epsilon is the difference between 1 and the smallest representable floating point number greater than 1.

Double precision numbers have 53 bits of mantissa, so we would expect Sage's machine epsilon to be 2^{-52}. The value of machine epsilon I computed was 2.22044604925031e-16, as anticipated, and I used this as ϵ in most of my calculations.

3 Levenberg-Marquardt

The Levenberg-Marquardt method is an iterative algorithm for solving nonlinear least squares problems. The algorithm is similar to the several variable Newton's method, which the reader is probably familiar with. Instead of directly finding an $\mathbf{x}$ such that $f(\mathbf{x}) = 0$, we attempt to find a local minimum of f, which is necessarily a stationary point. That is, we want to find $\mathbf{x}$ such that $\nabla f(\mathbf{x}) = \mathbf{0}$.

3.1 Introduction

The best way to understand how Levenberg-Marquardt works is to first look at the methods of gradient descent and Gauss-Newton iteration. The idea of the former method is to start with an initial guess and descend in the opposite direction of ∇f until $\nabla f = 0$. We present the algorithm in [4] here. Further discussion and several excellent convergence theorems for all of these methods can be found in [3].

Algorithm 1 Gradient (Steepest) Descent

```
initialize x ∈ R^n
initialize α ∈ R {step size}
initialize ε
while iterations < maxIterations do
  x = x − α∇f(x)
  if f(x) < ε then
    return x
  end if
end while
return x
```

The authors of [4] took the step size α to be constant, but more refined algorithms often perform a line search to determine a suitable choice for α at each iteration.

Although gradient descent is attractive for its simplicity and serves as a good starting point for optimization algorithms, convergence near solutions tends to be very slow. We can do better if we have some information about the second derivatives of f. This is where Newton's method comes in.

Algorithm 2 Newton's Method

```
initialize x ∈ R^n
initialize α ∈ R {step size}
initialize ε
while iterations < maxIterations do
  solve for d in H_f(x)d = −∇f(x)
  x = x + αd
  if f(x) < ε then
    return x
  end if
end while
return x
```

Newton's method is extremely powerful not only because it gives global convergence for convex functions, but because the speed of convergence is quadratic. The downside is that computing the Hessian of f is impractically time-consuming. To get around this, we make the following observations (given in [4]) about the functions we defined in the introduction:

$$\nabla f = J_{\mathbf{F}}^T \mathbf{F}.$$

$$H_f = J^T J + \sum_i F_i H_{F_i}.$$

We note that in our least squares problem, we have $F_i \to 0$ near a solution, so we can use $J^T J$ as a first-order approximation to the Hessian. Replacing H_f in Newton's method with $J^T J$ gives rise to the Gauss-Newton method.

The Levenberg-Marquardt algorithm can be seen as an interpolation between gradient descent and Gauss-Newton. It augments the Hessian approximant in Gauss-Newton with a diagonal damping term, resulting in a descent equation that looks like

$$(J^T J + \mu \text{diag}[v_1, v_2, \cdots, v_n])\mathbf{d} = -\nabla f = -J^T \mathbf{F}.$$

For $\mu = 0$, this is exactly the Gauss-Newton method. If we take the diagonal matrix to be the identity matrix I_n, then for $\mu >> 0$, we have $\mathbf{d} \to -\nabla f/\mu$ which is the steepest descent direction. We observe that Levenberg-Marquardt is more robust than Gauss-Newton because a sufficiently large value of μ ensures that the matrix on the left-hand side is positive definite. That said, we want to avoid taking μ to be unnecessarily large, lest we experience the inefficiency of gradient descent. Thus in our algorithm, we want to adjust μ at each iteration to take advantage of gradient descent when we are far from a solution, but then apply what is essentially Gauss-Newton to pinpoint it exactly.

What follows is the implementation presented in [4].

Algorithm 3 George, Sam and Ting's Levenberg-Marquardt Method

initialize $\mathbf{x} \in \mathbb{R}^n$
initialize $\mu \in \mathbb{R}$ {damping parameter}
while iterations < maxIterations **do**
  solve for $\mathbf{d}$ in $(J^T J + \mu I_n)\mathbf{d} = -J^T \mathbf{F}$
  if $f(\mathbf{x} + \mathbf{d}) < f(\mathbf{x})$ **then**
    $\mathbf{x} = \mathbf{x} + \mathbf{d}$
    decrease μ
    iterations++
  else
    increase μ
  end if
end while
return $\mathbf{x}$

3.2 Reimplementation

I made a few changes in my reimplementation of their algorithm, which I will explain in notes to follow.

1. At line 8, I replaced the identity matrix I_n with the diagonal of our Hessian approximant. For systems where the diagonal entries of H vary significantly, this allows us to get away with using smaller values of μ and still have a positive definite matrix.

2. At line 9, I allowed the program to terminate early when f became sufficiently small. This is practical in scientific calculations since we only need computational precision to be commensurate with measurement precision.

3. One of the challenges in [4] was to efficiently handle μ. Recall that we want to use relatively large values of μ when we are far from a solution, and then decrease it signficantly when we are near one. I decided that an easy way to do this was to simply set μ to the value of f at each iteration, so $\mu \to 0$ as we approach the solution, but increases if we stray from it.

4. At line 15, instead of immediately incrementing μ, I chose to do a very crude line search to continue with the descent direction the algorithm found. This avoids making μ unnecessarily large.

5. At line 23, the algorithm checks each component of $\mathbf{x}$ and bounces values that are too small or too large into a region of feasible conductivities. The authors of [4] noted that conductivities sometimes attained negative values, but had little effect on the convergence of their algorithm. Unfortunately, my implementation generated problems. The components of $\mathbf{F}$ are rational functions, and often have terms look like

$$\frac{1}{\sum_i x_i}.$$

Algorithm 4 My Levenberg-Marquardt Method

initialize $\mathbf{x} \in \mathbb{R}^n$
initialize $\mu \in \mathbb{R}$ {damping parameter}
initialize $\epsilon > 0$ {tolerance}
initialize $\delta > 0$ {search tolerance}
initialize $\tau >> 0$ {upper bound}
while iterations < maxIterations **do**
  $H = J^T J$
  solve for $\mathbf{d}$ in $(H + \mu \text{ diag } H)\mathbf{d} = -J^T\mathbf{F}$
  if $f(\mathbf{x} + \mathbf{d}) < \epsilon$ **then**
    return $\mathbf{x}$
  else if $f(\mathbf{x} + \mathbf{d}) < f(\mathbf{x})$ **then**
    $\mathbf{x} = \mathbf{x} + \mathbf{d}$
    $\mu = f(\mathbf{x})$
    iterations++
  else
    initialize $\alpha > 0$ {step size}
    while $\alpha > \delta$ **do**
      $\alpha/ = 2$
      test $f(\mathbf{x} + \alpha\mathbf{d})$ as above
    end while
    $\mu = f(\mathbf{x} + \alpha\mathbf{d})$
  end if
  for $i \in (0, \cdots, n-1)$ **do**
    if $x_i < \epsilon$ **then**
      $x_i = \sqrt{\epsilon}$
    end if
    $x_i = \min\{x_i, \tau\}$
  end for
end while
return $\mathbf{x}$

If the values of x_i vary in sign or are all very small, the algorithm experiences stability issues and possible zero division. To combat this, my algorithm ensures that the conductivites have some positive lower bound at each iteration, and are also kept from heading to infinity.

3.3 Linear Systems

The single-line expression for Λ involves taking the inverse of a principal submatrix of the Kirchhoff matrix:

$$\Lambda = A - BC^{-1}B^T.$$

However, in computing $X = C^{-1}B^T$, it is much easier to solve the linear system

$$CX = B^T.$$

The built-in Sage algorithm solves such a system through standard Gaussian elimination. The authors of [4] recommend taking advantage of the positive definiteness of C by using its Cholesky decomposition to more efficiently solve this system. In practice, it turns out that Sage's built-in decomposition algorithm is unstable for large networks. Instead, I attempted to speed up the solution of this linear system by using the Gauss-Seidel iterative algorithm for each column of B^T.

Algorithm 5 Gauss-Seidel

```
initialize x ∈ R^n
initialize ε > 0
while |Ax − b| > ε do
  for i ∈ (0, ..., n − 1) do
    σ = 0
    for j ∈ (0, ..., i − 1, i + 1, ..., n − 1) do
      σ+ = A_ij x_j
    end for
    x_i = (b_i − σ)/a_ii
  end for
end while
return x
```

Since C is positive definite, the Gauss-Seidel algorithm converges to the solution of the linear system. Unfortunately, it turned out that my implementation of the algorithm was incredibly slow, making it impossible to recover even 3×3 lattices. The final version of Levenberg-Marquardt still employs Sage's Gaussian elimination routine, though I would very much like to see the application of a stable Cholesky decomposition (or even LU-decomposition).

3.4 Computing the Jacobian

The authors of [4] provide the neat speed-up

$$\partial_i \Lambda(\mathbf{x}) = D^T \partial_i K(\mathbf{x}) D, \qquad D = \begin{pmatrix} I_m \\ -C^{-1}B^T \end{pmatrix}.$$

We note that $\operatorname{vec} \partial_i \Lambda(\mathbf{x})$ gives the ith column of $J_{\mathbf{F}}$. Differentiating the Kirchhoff matrix is incredibly simple, as each component of $\mathbf{x}$ appears in it exactly 4 times:

$$K = \begin{pmatrix} \ddots & & & & \\ & x_i + \sum & \dots & -x_i & \\ & \vdots & \ddots & \vdots & \\ & -x_i & \dots & x_i + \sum & \\ & & & & \ddots \end{pmatrix},$$

so the derivative is given by

$$\partial_i K = \begin{pmatrix} \ddots & & & & \\ & 1 & \dots & -1 & \\ & \vdots & \ddots & \vdots & \\ & -1 & \dots & 1 & \\ & & & & \ddots \end{pmatrix}.$$

3.5 Results

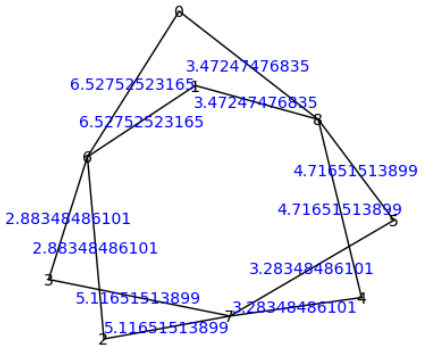
I ran the algorithm on $n \times n$ square lattice networks with constant conductivity 1 and an initial guess of all 2s (As George, Sam and Ting did). It's fairly obvious that this method does not scale very well.

n	iterations	CPU time (s)
1	4	0.06
2	5	0.41
3	6	1.81
4	8	7.64
5	8	23.52
6	12	90.05
7	10	143.21
8	16	472.31
9	26	1657.82
10	29	2886.30

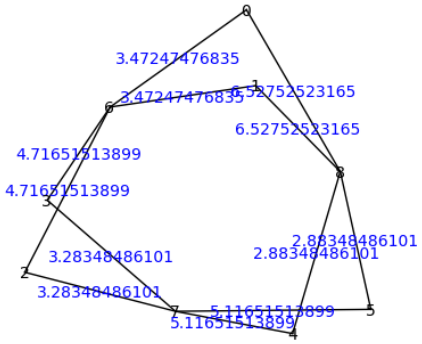
The algorithm works well on a wide variety of graphs. An interesting case to look at is a two-to-one electrical network with response matrix

$$\begin{pmatrix} 7 & -3 & -1 & -1 & -1 & -1 \\ -3 & 7 & -1 & -1 & -1 & -1 \\ -1 & -1 & 6 & -2 & -1 & -1 \\ -1 & -1 & -2 & 6 & -1 & -1 \\ -1 & -1 & -1 & -1 & 6 & -2 \\ -1 & -1 & -1 & -1 & -2 & 6 \end{pmatrix}.$$

With an initial guess of all .01, the algorithm converged to the following solution:



Whereas an initial guess of all 1's gave the solution



3.6 Condition Number of the Damped Hessian Approximant

Below is a table with the condition number of $(H + \mu \text{ diag } H)$ at each iteration (as well as after each adjustment of μ), computed on an 8×8 lattice.

iterations	Condition number
1	37957.6551305
2	3.89679085225e+18
3	3.91087691539e+25
4	2.14935271079e+25
5	8.21004143849e+21
6	9.68127027747e+21
7	1.15863105438e+22
8	4.09919757851e+20
9	7.9414851257e+21
10	6.81069566976e+21
11	8.87062188588e+20
12	1.18654876563e+21
13	9.86841420403e+20
14	1726675233.96
15	69830744.9929
16	78793765.0769
17	297092440.155
18	263394054.973
19	275440186.735
20	275823371.428

I currently have no explanation for why these numbers are phenomenally large, but they suggest that it might be prudent to perform a decomposition on the Hessian approximant in order to improve the stability of the algorithm.

4 Regularization

Suppose we are electrical engineers and want to recover electrical conductivities in a physical network. We can construct a network response matrix by putting voltages at various boundary nodes and measuring the resulting currents. But there's a problem. Unless our instruments and the test conditions are perfect, the result isn't really a response matrix (as it is mathematically defined) and the techniques we have established probably won't work. However, if we know something about the network, it may still be possible to "solve" the "inverse problem".

In [4], the authors introduce the idea of using a regularization term to handle noisy data. Suppose we know that the distribution of conductivites in our network is relatively smooth or piecewise constant. Then instead of minimizing just the functional, we can obtain more accurate results by also minimizing an aggregation of the jumps between conductivites.

In the continuous case, there are two commonly used regularizers:

$$\int_\Omega |\nabla u|^2, \tag{2}$$

$$||\nabla u||_{TV} \sim \int_\Omega |\nabla u|. \tag{3}$$

Regularizer (2) is generally appropriate for smooth distributions, whereas (3) is more appropriate for piecewise constant functions, or in our case, clumps of resistors. It is often easier to deal with expression (2) since we can apply least-squares methods naturally. The discretizations we will use, respectively, are

$$S(\gamma) = \sum_{i \in \text{int } G} \sum_{j \sim i, k \sim i} (\gamma_{ij} - \gamma_{ik})^2,$$

$$C(\gamma) = \sum_{i \in \text{int } G} \sum_{j \sim i, k \sim i} |\gamma_{ij} - \gamma_{ik}|.$$

To be consistent with prior notation, we will henceforth refer to these as $S(\mathbf{x})$ and $C(\mathbf{x})$. When working with regularizers, the authors of [4] used a Lagrangian approach, minimizing the function

$$L(\mathbf{x}) = f(\mathbf{x}) + \beta S(\mathbf{x}).$$

As one would expect, they found that their choice of β had a considerable impact on their results. They further noted that obtaining a handle on the measurement error in Λ_0 could lead to a suitable choice for β. This is the motivation for the following algorithm.

4.1 Augmented Lagrangian Method

Construct the response matrix for a physical system as follows: for $i \neq j$, apply unit voltage at boundary node i, measure the resulting current at boundary node j, and insert this value in the ij position of Λ_0. In each diagonal position ii, take the negative average of the sum of entries in row i and the sum of entries in column i.

Suppose our measurement error for each entry of the response matrix has a Gaussian distribution with zero mean and standard deviation σ, and suppose our measurments are uncorrelated. We want to determine a reasonable upper bound on our functional

$$f(\mathbf{x}) = \frac{1}{2} \sum_{i=0}^{m^2-1} (\Lambda(\mathbf{x})_i - \Lambda_{0,i})^2$$

so that each entry of the generated response matrix $\Lambda(\mathbf{x})$ and the corresponding entry of the target response matrix (that is, the one without noise that we

don't actually have access to) lie in some small ball around the corresponding entry in the measured "response" matrix. Since about 95% of measurements in a Gaussian distribution lie within 2 standard deviations of the mean, it is probably reasonable to take

$$\begin{aligned} f(\mathbf{x}) &\leq \frac{1}{2} \sum_{i=0}^{m^2-1} (2\sigma)^2 \\ &= 2m^2\sigma^2. \end{aligned}$$

Any $\mathbf{x}$ within this range gives a decent approximation to the solution of our inverse problem (I invite the interested reader to experiment with different upper bounds). We can use a regularization term to pick out the solution in this set that is "best." Suppose we have an appropriate regularizer $R(\mathbf{x})$. Then we can effectively approximate the solution of the inverse problem by solving the constrained optimization problem

$$\begin{aligned} &\arg\min_{\mathbf{x}\in{\mathbb{R}_+}^n} R(\mathbf{x}) \\ &\text{s.t. } f(\mathbf{x}) \leq 2m^2\sigma^2. \end{aligned}$$

One way to solve this problem would be to minimize the standard Lagrangian function as the authors of [4] did. Alternatively, Ernie Esser suggested using an augmented Lagrangian approach, in which we add a quadratic penalty term to the Lagrangian function, giving

$$P(\mathbf{x},\lambda,\zeta) = R(\mathbf{x}) + \begin{cases} \lambda(f(\mathbf{x}) - 2m^2\sigma^2) + \frac{1}{2}\zeta(f(\mathbf{x}) - 2m^2\sigma^2)^2 & \text{if} f(\mathbf{x}) - 2m^2\sigma^2 > -\frac{\lambda}{\zeta} \\ -\frac{1}{2}\frac{\lambda^2}{\zeta} & \text{otherwise.} \end{cases}$$

The penalty term measures the extent to which our constraint equation is violated (and along with the Lagrangian term, is conspicuously missing when the constraint is satisfied), so minimizing it helps our solutions move quickly into the desired range. Further, the algorithm I will present allows us to iteratively determine suitable choices for λ and ζ, eliminating any guesswork related to these parameters. Since we are only dealing with one constraint equation, the algorithm is fairly simple; a good exposition of the general case is given in [3].

Algorithm 6 Augmented Lagrangian Method

initialize $\mathbf{x} \in \mathbb{R}^n$
initialize $\lambda, \zeta > 0$
while iterations $<$ maxIterations **do**
  $\tilde{\mathbf{x}} =$ approximation to $\underset{\mathbf{x}}{\operatorname{argmin}}\, P(\mathbf{x}, \lambda, \zeta)$ {Found using an early-terminating application of Levenberg-Marquardt}
  $\lambda = \max\{\lambda + \zeta(f(\tilde{\mathbf{x}}) - 2m^2\sigma^2), 0\}$
  $\zeta = \max\{10\zeta, \text{iterations}^2\}$ if $|f(\tilde{\mathbf{x}}) - 2m^2\sigma^2| > .25 \cdot |f(\mathbf{x}) - 2m^2\sigma^2|$
  $\mathbf{x} = \tilde{\mathbf{x}}$
end while
return $\mathbf{x}$

Although somewhat slow due to our repeated use of Levenberg-Marqurdt, this algorithm gave surprisingly favorable results. Whereas straight-up application of Levenberg-Marquardt frequently gave conductivites that bounced out of the acceptable region during intermediate calculations, this method generally gives intermediate conductivities that tend very uniformly to the desired result.

It turned out that the differentiation algorithm we used to compute the Jacobian (presented in the following section) gave $\nabla C(\curvearrowleft) = 0$ due to the fact that $\operatorname{Im}|F| = 0$ for all $z \in \mathbb{C}$. Despite this blunder, many of the following results were obtained faster, and in some cases, a "correct" differentiator failed to give convergence at all (though when it did, the results were usually a bit more accurate). In further work, we will explore ways of using $C(\curvearrowleft)$ without having to differentiate functions that are not smooth everywhere.

Here, we tackle the problem presented in [4] of recovering a 5×5 network with a clump of 100 Ω resistors in the center and 1 Ω resistors elsewhere. Using the regularizer $S(\mathbf{x})$ as they did, it was possible to recover the interface, but the conductivities were considerably off. With a standard deviation $\sigma = .005$, I obtained the conductivity list

(1.02231638805, 0.97058890618, 1.04732624153, 1.01159368132, 0.934163146072,
0.983610843515, 0.967278644389, 0.940665371582, 0.967375355672, 0.936811614118,
0.959573701358, 0.935639396694, 0.951669509006, 0.978159835805, 0.942684656395,
1.03402164492, 1.01401997016, 0.956812282243, 1.00031926378, 1.02573029112,
0.971410094332, 1.05382374743, 1.30571467939, 1.29922438046, 1.25546613045,
1.42498346171, 1.00397523083, 1.29174802461, 1.03363429116, 1.33203171672,
1.27760436899, 1.59324559451, 1.59962345322, 1.58214347977, 1.70009420544,
1.30205028038, 1.59064344086, 1.30202822119, 1.49614549778, 1.25776010968,
1.73541425222, 1.59611543196, 1.72291171701, 1.71445063989, 1.46020581007,
1.60861020226, 1.26043559279, 1.3507458121, 1.0853046252, 1.61410976144, 1.32564588718,
1.61652757442, 1.45977576446, 1.33141398438, 1.35043687146, 1.10962297145,
1.11749361746, 1.32410570672, 1.29394061023, 1.01852212477).

The regularizer $C(\mathbf{x})$ turned out to be much more effective for standard deviations. With $\sigma = .000005$, the algorithm gave

(1.00002215942, 1.00001653091, 0.999979967239, 1.00001969461, 1.000008175, 1.00005083333, 1.00000623973, 0.999964247814, 1.00023387487, 1.00028159564, 1.00018315127, 1.00012136971, 1.00001458215, 1.00000559845, 0.999999886387, 0.999941012494, 0.999983297131, 0.999967256332, 1.00003435096, 0.999955744984, 0.999897346255, 0.999968086124, 0.9992537917, 1.00035955531, 0.999109466841, 0.999336087694, 1.00001945543, 1.00052890657, 1.00002410158, 1.00043657391, 0.999694993924, 99.5442579525, 102.164699639, 102.528077772, 89.2977325985, 1.00005504994, 103.246733884, 0.999418612659, 0.999377194263, 0.999528608136, 103.005505828, 100.924886113, 100.429322545, 100.836907875, 0.999312927073, 101.97078432, 0.999305231948, 1.00036209349, 0.999935841256, 96.1423472662, 1.00044713496, 100.442032881, 0.999459741864, 1.00011652912, 1.000249821, 0.999792738877, 1.00017875663, 0.999873147073, 0.999799145305, 1.00012665371).

As the error grew, the algorithm deteriorated. For $\sigma = .00005$, it produced

(0.999996032051, 0.999752590918, 1.00051524149, 0.999865614827, 1.00051393702, 1.00081316485, 1.0003013867, 0.999952291627, 1.00317692332, 1.00190909238, 1.00338841435, 1.0028209843, 0.999901136974, 0.999685601802, 1.00023598765, 0.999759535125, 1.00007925703, 0.999828183293, 0.999983625846, 1.00000137095, 1.00127407379, 0.999571818832, 0.998182715454, 1.00287847722, 0.997618065869, 0.985797805312, 1.00128059689, 1.00323110941, 0.999249833203, 1.00353203465, 0.993840165154, 69.7775515638, 78.9209154557, 69.6530563147, 292.573568982, 1.00300017086, 81.6883338729, 0.994294895505, 0.982430720267, 0.993172097894, 442.472266954, 79.1853952418, 472.211668408, 314.369837559, 0.981858227609, 82.2731083351, 0.996948180848, 1.00369287186, 0.999583438497, 82.1266879999, 1.00283964097, 65.1521899031, 0.986039160893, 1.00093969372, 1.00467468262, 1.00038339564, 1.00039471554, 0.996595595441, 0.996060844583, 1.00083898898).

For $\sigma = .005$, the interface was barely recoverable:

(1.02210892888, 1.05359431427, 1.03076820911, 1.01848761578, 1.00685408568, 0.931912505035, 0.960180176577, 0.987083804414, 1.01622145417, 1.05276087345, 0.981566122779, 1.12722929317, 1.04018164012, 0.968086474157, 0.996739177521, 0.980067433646, 0.999298218399, 0.97782677978, 0.956886940048, 0.98931899352, 1.20181830088, 1.02399659216, 1.00776942835, 1.17705845386, 0.931143708115, 0.588618661502, 1.0173386639, 1.37838578143, 0.950425956242, 0.867915349463, 0.969197641858, 514.742453957, 10000.0, 78.8113480023, 2248.1400897, 0.969268877007, 1.49011611938e-08, 0.813891264663, 0.834733818306, 1.0994217114, 1.49011611938e-08, 3.82658480115, 1.49011611938e-08, 1.49011611938e-08, 1.00672447191, 43.6420910734, 0.829305209955, 0.892220575976, 0.952047715186, 10000.0, 1.1534744237, 6.05957695454,

1.06522668363, 1.06918761181, 1.21745338104, 0.94376670042, 0.962917249603, 0.667412701551, 0.936751269199, 1.01791481719).

4.2 Numerical Differentiation

The authors of [4] found that it was frequently much quicker to compute derivatives numerically with finite differences rather than evaluate them symbolically. In their implementation of gradient descent, they used the following approximation:

$$\partial_i \mathbf{F}(\mathbf{x}) \approx \frac{\mathbf{F}(\mathbf{x} + h\mathbf{e_i}) - \mathbf{F}(\mathbf{x})}{h},$$

where $\mathbf{e_i}$ is the standard basis vector in $\mathbb{R}^n$ with a 1 in the ith row and zeroes elsewhere, and h is small. For many purposes, this approximation is sufficiently stable. However, for h on the order of machine epsilon, the subtraction in the numerator can cause problems with roundoff error. To get around this, I took advantage of Sage's support for complex arithmetic by applying the approximation

$$\partial_i \mathbf{F}(\mathbf{x}) \approx \frac{\operatorname{Im} \mathbf{F}(\mathbf{x} + ih\mathbf{e_i})}{h}.$$

A good discussion of this method is given in [2]. To see why this approximation is valid, we give a single variable theorem.

Theorem 4.1. *Suppose $f(z)$ is analytic in a ball $|z - x_0| < r$ for $x_0 \in \mathbb{R}$ and $r > 0$. Suppose further that $f : (x_0 - r, x_0 + r) \to \mathbb{R}$. Then*

$$f'(x_0) = \lim_{h \to 0} \frac{\operatorname{Im} f(x_0 + ih)}{h}.$$

Proof. Rewrite $\operatorname{Im} f(x_0 + ih)$ as

$$\frac{f(x_0 + ih) - \overline{f(x_0 + ih)}}{2i}.$$

By the Schwarz reflection principle

$$f(x_0 + ih) = \overline{f(x_0 - ih)},$$

so

$$\frac{\operatorname{Im} f(x_0 + ih)}{h} = \frac{f(x_0 + ih) - f(x_0 - ih)}{2ih}.$$

Since f is analytic in a neighborhood of x_0, it possesses a power series

$$f(x_0 + k) = f(x_0) + f'(x_0)(k) + O(k^2).$$

Therefore,

$$\begin{aligned}\lim_{h\to 0}\frac{\operatorname{Im} f(x_0+ih)}{h} &= \lim_{h\to 0}\frac{f(x_0+ih)-f(x_0-ih)}{2ih}\\ &= \lim_{h\to 0}\frac{f(x_0)+f'(x_0)(ih)-(f(x_0)+f'(x_0)(-ih))+O(h^2)}{2ih}\\ &= \lim_{h\to 0}\frac{2ihf'(x_0)+O(h^2)}{2ih} = f'(x_0).\end{aligned}$$

□

The theorem above becomes messier in full generality, since it deals with several complex variables. But the approximation is still valid: since the standard regularization term is a simple polynomial, it is jointly analytic in all of its variables and real on the real axis. Although this approximation works very well (taking $h = x\sqrt{\epsilon}+\epsilon$), it turns out that since $S(\mathbf{x})$ is so simple, it is actually faster to differentiate it symbolically. On the other hand, since $C(\mathbf{x})$ fails to be globally differentiable, we must use a numerical subroutine like this.

5 Convexity

Practitioners of optimization love convex functions because we can apply almost all of our techniques worry-free to them.

Definition 5.1. A function $f : S \to \mathbb{R}$ is convex if S is a convex set and if for any $\mathbf{x_0}, \mathbf{x_1} \in S$ and for all $t \in (0, 1)$:

$$f(t\mathbf{x_0} + (1-t)\mathbf{x_1}) \leq tf(\mathbf{x_0}) + (1-t)f(\mathbf{x_1}).$$

If the inequality above is strict, then f is said to be strictly convex on S.

Since we are dealing with smooth rational functions, the following observation will be helpful.

Lemma 5.2. *Consider $f \in C^2(S)$ for some convex set S. Then f is convex if and only if $H_f(\mathbf{x})$ is positive semidefinite for all $\mathbf{x} \in S$. If H_f is positive definite for all $\mathbf{x} \in S$, then f is strictly convex (the converse does not hold).*

Finally, the following theorem tells us why we should care about convexity.

Lemma 5.3. *Suppose f is convex on S. If $\mathbf{x} \in S$ is a local minimizer for f, then it is a global minimizer. Further,* $\operatorname{argmin} f(\mathbf{x})$ *forms a convex set. If f is strictly convex and $\mathbf{x}$ is a local minimizer for f, then* $\{\mathbf{x}\} = \operatorname{argmin} f(\mathbf{x})$.

Therefore, if our functional f is convex, a local minimizer that we obtain through our iterative methods is a solution to the inverse problem.

5.1 Sufficient conditions

Unfortunately, it seems that with regard to the convexity of f, there are more counterexamples than theorems. Here are a few minor results, for which we consider $\mathbf{x} \in \mathbb{R}^n_+$.

Lemma 5.4. *Suppose that F_i^2 is convex for $0 \leq i \leq m^2 - 1$. Then f is convex.*

Proof. The sum of convex functions is convex, and convexity is unaffected by positive scalar multiplication. □

Lemma 5.5. *Suppose $f : S \in \mathbb{R}^n \to \mathbb{R}$ and $g : f[S] \to \mathbb{R}$ are convex and g is nondecreasing. Then $g \circ f$ is convex.*

Proof. This falls straight out of the definition. From the convexity of f, we have $f(t\mathbf{x_0} + (1-t)\mathbf{x_1}) \leq tf(\mathbf{x_0}) + (1-t)f(\mathbf{x_1})$. Therefore

$$\begin{aligned} g(f(t\mathbf{x_0} + (1-t)\mathbf{x_1})) &\leq g(tf(\mathbf{x_0}) + (1-t)f(\mathbf{x_1})) && \text{since } g \text{ is nondecreasing} \\ &\leq tg(f(\mathbf{x_0})) + (1-t)g(f(\mathbf{x_1})) && \text{since } g \text{ is convex.} \end{aligned}$$

□

Theorem 5.6. *Suppose we have an electrical network Γ consisting entirely of boundary nodes. Then Γ admits a convex functional f.*

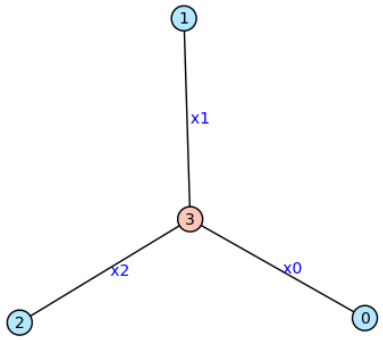
Proof. The response matrix $\Lambda(x)$ for Γ is just the Kirchhoff matrix. Hence, each entry is either zero, a negative component $-x_i$ or a sum $\sum x_i$. The components of $\mathbf{F}$ are then zero or of the form $-x_i - \Lambda_{0,i}$ or $\sum x_i - \Lambda_{0,i}$. Squaring gives zero, $x_i^2 + 2\Lambda_{0,i}x_i + \Lambda_{0,i}^2$ or $(\sum x_i)^2 - 2\Lambda_{0,i}\sum x_i + \Lambda_{0,i}^2$. The middle expression is the sum of convex functions, so it is convex. The quadratic term in the third expression is the composition of two convex functions, and y^2 is nondecreasing for $y > 0$, so this function is convex as well by Lemma 5.5. By Lemma 5.4, f is convex. □

Lemma 5.7. *Suppose each F_i is either convex and nonnegative or concave and nonpositive. Then f is convex.*

Proof. If $y \geq 0$, then y^2 is nondecreasing, so if F_i is convex and nonnegative, F_i^2 is convex by Lemma 5.5. If F_i is concave and nonpositive, then $-F_i$ is convex and nonnegative, so $(-F_i)^2 = F_i^2$ is convex. □

5.2 A Counterexample

The counterexample to most of the results that we would like to have is very simple: the Y (wye) network pictured below.



The Kirchhoff matrix for this network is

$$K(\mathbf{x}) = \begin{pmatrix} x_0 & 0 & 0 & -x_0 \\ 0 & x_1 & 0 & -x_1 \\ 0 & 0 & x_2 & -x_2 \\ -x_0 & -x_1 & -x_2 & x_0 + x_1 + x_2 \end{pmatrix}.$$

Schur-complementing gives the response matrix

$$\Lambda(\mathbf{x}) = \frac{1}{x_0 + x_1 + x_2} \begin{pmatrix} x_0x_1 + x_0x_2 & -x_0x_1 & -x_0x_2 \\ -x_0x_1 & x_0x_1 + x_1x_2 & -x_1x_2 \\ -x_0x_2 & -x_1x_2 & x_0x_2 + x_1x_2 \end{pmatrix}.$$

Observe that

$$\begin{aligned} \left| \frac{x_i x_j}{x_1 + x_2 + x_3} \right| &\leq \left| \frac{x_i x_j}{x_i} \right| \\ &= |x_j| \qquad\qquad \to 0 \text{ as } |\mathbf{x}| \to 0. \end{aligned}$$

Yet if we fix $x_k = 1, k \neq i, j$ and let $x_i = x_j \to \infty$, we have

$$\begin{aligned} \left| \frac{x_i x_j}{x_1 + x_2 + x_3} \right| &= \left| \frac{x_i x_j}{x_i + x_j + 1} \right| \\ &= \left| \frac{x_i^2}{2x_i + 1} \right| \\ &\to \frac{x_i}{2} \to \infty. \end{aligned}$$

Hence, if any entry of the response matrix is nontrivial, the corresponding F_i is neither nonpositive nor nonnegative on all of $\mathbf{R}^n_+$. Therefore, the conditions of Lemma 5.7 are not satisfied and we are more or less stuck.

Another important note is that if $|\mathbf{x}| \to \infty$, we don't necessarily have that the entries of the response matrix $\Lambda(\mathbf{x})_{ij} \to \infty$. For instance, fix $x_0 = x_1 = 1$. Then we have

$$\Lambda(\mathbf{x}) = \frac{1}{2+x_2}\begin{pmatrix} 1+x_2 & -1 & -x_2 \\ -1 & 1+x_2 & -x_2 \\ -x_2 & -x_2 & 2x_2 \end{pmatrix}.$$

Letting $x_2 \to \infty$ gives the response matrix

$$\Lambda = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{pmatrix}.$$

This is not unexpected, since the conductivity of an edge heading toward infinity is analogous to identifying the nodes at the ends of that edge. However, it does mean that $f(\mathbf{x})$ can be bounded as $|\mathbf{x}| \to \infty$, which is not particularly pleasant when it comes to optimization.

In studying convexity, I came across the following interesting result that may or may not be helpful in understanding the Hessian of f.

Theorem 5.8. *Let A be an $n \times n$ symmetric matrix with real entries, and consider $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. The quadratic form $g(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ has $2A$ as its Hessian.*

Proof. By the product rule

$$\begin{aligned} \partial_j(\mathbf{x}^T A\mathbf{x}) &= (\partial_i \mathbf{x}^T)A\mathbf{x} + \mathbf{x}^T(\partial_i A)(\mathbf{x} + \mathbf{x}^T A)\partial_i \mathbf{x} \\ &= \mathbf{e}_j^T A\mathbf{x} + \mathbf{x}^T A\mathbf{e}_j \\ &= \sum_{i=1}^n A_{ji}x_i + \sum_{i=1}^n A_{ij}x_i \\ &= 2\sum_{i=1}^n A_{ij}x_i. \end{aligned}$$

Therefore, $\partial_i\partial_j \mathbf{x}^T A\mathbf{x} = 2A_{ij}$.

□

Corollary 5.9. *If A is positive semidefinite, then $g(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ is convex. If A is positive definite, g is strictly convex.*

6 Closing Remarks

The field of numerical methods as they apply to the inverse problem is ripe with additional venues to explore. Here are some suggestions I have for future work.

1. Can we effectively extend our techniques to solving the inverse problem with mixed maps instead of response matrices? How about for other networks, such as random walk networks, heat networks, or Schroedinger networks?

2. Is it possible to use properties of graphs to determine information about the behavior of $\mathbf{F}$ and f? Are there any nontrivial graphs that admit convex functionals?

3. Can we solve the inverse problem numerically without ever solving the forward problem during intermediate stages?

6.1 Acknowledgements

I am most indebeted to Professor Jim Morrow and Ernie Esser for their wealth of knowledge and ideas related to the problems in this paper. I would also like to thank Tom Boothby for his expertise in Sage and computational methods, Chad Klumb for providing some of my niftiest examples and Igor Tolkov for helping me recover some code I thought was lost in cyberspace.

References

[1] E. Curtis, J. Morrow, *Inverse Problems for Electrical Networks,* Series on Applied Mathematics, World Scientific, Singapore, 2000.

[2] W. Squire, G. Trapp, Using Complex Variables to Estimate Derivatives of Real Valued Functions. *SIAM Review,* 40(1):110-112, Mar., 1998.

[3] W. Sun, Y. Yuan, *Optimization Theory and Methods: Nonlinear Programming,* Springer Optimization and Its Applications, Springer, New York, 2006.

[4] G. Tucker, S. Whittle, T. Wang, *On Numerical Recovery Methods for the Inverse Problem,* 2006.