1D Problems

- For Root Finding, Newton’s Method requires \( g'(t) \), but the Secant Method replaces this derivative with a secant line though two prior iterates.
- One may also use Finite Differences to approximate the derivative if the perturbation \( h \) is small enough, since Finite Differences only use values of the function \( g(t) \).
- Alternatively, Automatic Differentiation may be used to find the value of a derivative at a particular point, e.g. \( g'(t^k) \), if/when the appropriate code exists (e.g. backprop code).
- For 1D Optimization, one can pursue a similar strategy using secant lines for first derivatives, interpolating parabolas for second derivatives, or Finite Differences to replace both first and second derivatives (alternatively, Automatic Differentiation may be leveraged as well…).
- Overall, since 1D methods can rely on mixed approaches that robustly utilize Bisection and Golden Section Search, replacing derivatives with estimates based on function evaluations is a tenable (albeit, sometimes computationally expensive) strategy (even when those approximations/estimates are poor).

Nonlinear Systems

- Solving a nonlinear system requires solving \( J(\hat{c}^k) \hat{\Delta}c = \hat{\Delta}F \) for \( \hat{\Delta}c \), whether line search is used or not.
- In the case of line search, using \( \hat{\Delta}c \) in \( \hat{c}(t) = c^k + t \hat{\Delta}c \), one looks for roots/minima as discussed above attempting to avoid/approximate/estimate derivatives.
- Regardless, the Jacobian matrix of first derivatives \( J(\hat{c}^k) \) needs to be evaluated.
- One could use small perturbations (small \( h \)’s) of \( \hat{c}^k \) in the various coordinate directions to approximate the partial derivatives in the Jacobian via Finite Differences.
- Quasi-Newton Methods get more cavalier with the idea of a search direction \( \hat{\Delta}c \), and as such make some aggressive approximations to \( J(\hat{c}^k) \) realizing that the resulting search directions may be widely/wildly perturbed (thus, relying on a more careful consideration of the 1D line search sub-problem in order to “make progress” towards solutions).
- One of the more popular strategies is Broyden’s Method where an initial guess for the Jacobian (sometimes just the identity matrix) is continuously corrected with rank one updates (similar in spirit to a secant approach).
- For example, \( J^0 = I \) and \( J^{k+1} = J^k + \frac{1}{(\hat{\Delta}c^k) \hat{\Delta}c^k} \left( \hat{F}(\hat{c}^{k+1}) - \hat{F}(\hat{c}^k) - J^k \hat{\Delta}c^k \right) (\hat{\Delta}c^k)^T \).
- Here, \( J^{k+1} \hat{\Delta}c^k = \hat{F}(\hat{c}^{k+1}) - \hat{F}(\hat{c}^k) \) or \( J^{k+1} (\hat{c}^{k+1} - \hat{c}^k) = \hat{F}(\hat{c}^{k+1}) - \hat{F}(\hat{c}^k) \) illustrates that \( J^{k+1} \) satisfies a secant type equation of the form \( \frac{\Delta F}{\Delta c} \) (abusing notation).
Optimization

- The cost function $F(\vec{c})$ is differentiated to obtain a nonlinear system of equations $\vec{j}(\vec{c}) = \vec{0}$ where $\vec{j}(\vec{c})$ is the Jacobian of $F(\vec{c})$

- Then, the linear system $\vec{H}(\vec{c}^k) \Delta \vec{c} = -\vec{j}(\vec{c}^k)$ is solved to find the search direction $\Delta \vec{c}$ for $\vec{c}(t) = \vec{c}^k + t \Delta \vec{c}$, where $\vec{H}(\vec{c}^k)$ is the Hessian of $F(\vec{c})$

- *All three* aforementioned options for how to proceed in optimization problems require solving $\vec{H}(\vec{c}^k) \Delta \vec{c} = -\vec{j}(\vec{c}^k)$ to find the line search direction!

- Although Finite Differences may sometimes be used successfully for Jacobians, the Hessian matrix of second partial derivatives is harder to approximate reasonably.

- Automatic Differentiation methods (such as Back Propagation) aren’t popular for Hessians, since they have $O(n^2)$ terms as opposed to the $O(n)$ terms in an optimization Jacobian, creating both storage and evaluation issues.

Quasi-Newton Methods (for Optimization)

- Similar to solving nonlinear systems of equations, the main idea is to aggressively approximate the coefficient matrix, the Hessian in this case

- Often one has many variables in $\vec{c}$, so computing, storing, and inverting the $O(n^2)$ Hessian is not desirable.

  Instead, these methods often approximate the inverse Hessian for $\Delta \vec{c} = -\vec{H}^{-1}(\vec{c}^k) \vec{j}(\vec{c}^k)$, or better yet compute the action of the inverse Hessian on a vector, i.e. $\vec{H}^{-1} \vec{v}$

- Similar to Broyden’s Method, we desire $\vec{H}^{-1}(\vec{c}^{k+1} - \vec{c}^k) = \vec{j}(\vec{c}^{k+1}) - \vec{j}(\vec{c}^k)$ to mimic a secant approach.

- Some of the more popular approaches include BFGS, Symmetric Rank One (SR1), and Broyden-type methods.

- This table from Wikipedia has the Hessian in the first column and its inverse in the second column (note there are notational differences from our notes):

<table>
<thead>
<tr>
<th>Method</th>
<th>$B_{k+1} =$</th>
<th>$H_{k+1} = B_{k+1}^{-1} =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFGS</td>
<td>$B_{k} + \frac{y_k y_k^T}{y_k^T \Delta x_k} - \frac{B_{k} \Delta x_k (B_{k} \Delta x_k)^T}{\Delta x_k^T B_{k} \Delta x_k}$</td>
<td>\left( I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k} \right) H_k \left( I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k} \right) + \frac{\Delta x_k \Delta x_k^T}{y_k^T \Delta x_k}$</td>
</tr>
<tr>
<td>Broyden</td>
<td>$B_{k} + \frac{y_k - B_{k} \Delta x_k}{\Delta x_k^T \Delta x_k} \Delta x_k^T \Delta x_k$</td>
<td>$H_k + \frac{\Delta x_k - H_k y_k) \Delta x_k^T H_k}{\Delta x_k^T H_k y_k}$</td>
</tr>
<tr>
<td>Broyden family</td>
<td>$(1 - \varphi_k) B_{k+1}^{BFGS} + \varphi_k B_{k+1}^{DFP}, \quad \varphi \in [0, 1]$</td>
<td></td>
</tr>
<tr>
<td>DFP</td>
<td>\left( I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k} \right) B_{k} \left( I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k} \right) + \frac{y_k y_k^T}{y_k^T \Delta x_k} \Delta x_k^T \Delta x_k$</td>
<td>$H_k + \frac{\Delta x_k \Delta x_k^T y_k^T}{\Delta x_k^T y_k^T H_k y_k - \Delta x_k^T H_k y_k}$</td>
</tr>
<tr>
<td>SR1</td>
<td>$B_{k} + \frac{(y_k - B_{k} \Delta x_k)(y_k - B_{k} \Delta x_k)^T}{(y_k - B_{k} \Delta x_k)^T \Delta x_k}$</td>
<td>$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^T}{(\Delta x_k - H_k y_k)^T y_k}$</td>
</tr>
</tbody>
</table>

- $y_k = \nabla \vec{j}(x_{k+1}) - \nabla \vec{j}(x_k)$

- L-BFGS is a limited memory version of BFGS that estimates the inverse Hessian using only a few vectors instead of a dense nxn matrix (often < 10 vectors), which makes it quite popular for machine learning (e.g. see Andrew Ng et al. ICML 2011, 600+ citations).
Nonlinear Least Squares

- A *special cost function* is chosen of the form $F(\boldsymbol{c}) = \frac{1}{2} f(\boldsymbol{c})^T f(\boldsymbol{c})$ where $f(\boldsymbol{c})$ is a vector valued function
- Recall from Unit 13 that such a cost function emanated from trying to match a function with parameters $\boldsymbol{c}$ to training data
- $F(\boldsymbol{c}) = \frac{1}{2} \sum_j (f_j(\boldsymbol{c}))^2$ leads to the nonlinear system $\hat{f}(\boldsymbol{c}) = \sum_j f_j(\boldsymbol{c}) \frac{\partial f_j}{\partial \boldsymbol{c}} = J_f(\boldsymbol{c}) f(\boldsymbol{c}) = 0$, where $J_f(\boldsymbol{c})$ is the Jacobian of $f(\boldsymbol{c})$ as opposed to the Jacobian of $F(\boldsymbol{c})$
- Using the Taylor expansion $f(\boldsymbol{c}) \approx f(\hat{\boldsymbol{c}}^k) + \frac{\partial f}{\partial \boldsymbol{c}} (\hat{\boldsymbol{c}}^k - \boldsymbol{c}) + \cdots$ to substitute for the second term results in an *equivalent* set of equations: $\hat{f}(\boldsymbol{c}) = J_f(\hat{\boldsymbol{c}}) (f(\hat{\boldsymbol{c}}^k) + J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} + \cdots) = 0$
- Gauss-Newton ignores the higher order terms in the Taylor expansion and evaluates the leading $J_f^T(\hat{\boldsymbol{c}})$ at $\hat{\boldsymbol{c}}^k$ to obtain $J_f^T(\hat{\boldsymbol{c}}^k) J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -J_f^T(\hat{\boldsymbol{c}}^k) f(\hat{\boldsymbol{c}}^k)$
- The right hand side is exactly $-\hat{f}(\hat{\boldsymbol{c}}^k)$, and thus Gauss-Newton can be seen as approximating the Hessian $\hat{H}(\hat{\boldsymbol{c}}^k)$ with $J_f^T(\hat{\boldsymbol{c}}^k) J_f(\hat{\boldsymbol{c}}^k)$
- By now, one should realize that the solution of $J_f^T(\hat{\boldsymbol{c}}^k) J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -J_f^T(\hat{\boldsymbol{c}}^k) f(\hat{\boldsymbol{c}}^k)$ can be obtained instead by carefully considering only $J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -f(\hat{\boldsymbol{c}}^k)$ via a least squares (QR) and minimal norm approach
- Moreover, one could just set the second factor in $J_f^T(\hat{\boldsymbol{c}}) (f(\hat{\boldsymbol{c}}^k) + J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} + \cdots) = 0$ to zero, again ignoring higher order terms, and obtain $J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -f(\hat{\boldsymbol{c}}^k)$ directly
- Recall that row scaling changes the importance of the equations, and thus also changes the (so-called unique) least squares solution for the overdetermined degrees of freedom
- Weighted Gauss-Newton intentionally does this by multiplying by a diagonal weight matrix $D$ before multiplication by $J_f^T$, that is $J_f^T(\hat{\boldsymbol{c}}^k) D J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -J_f^T(\hat{\boldsymbol{c}}^k) D f(\hat{\boldsymbol{c}}^k)$
- Again, a more savvy equivalent approach considers $D^{1/2} J_f(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -D^{1/2} f(\hat{\boldsymbol{c}}^k)$ where the square root of the diagonal entries makes these equations “equivalent” to their normal equations counterpart when considering residual minimization in the L2 norm
- Levenberg-Marquardt adds $\epsilon I = 0$ to these equations (along the lines of the regularization discussed in Unit 12A) to obtain $(J_f^T(\hat{\boldsymbol{c}}^k) J_f(\hat{\boldsymbol{c}}^k) + \epsilon I) \Delta \boldsymbol{c} = -J_f^T(\hat{\boldsymbol{c}}^k) f(\hat{\boldsymbol{c}}^k)$

Coordinate Descent

- The cost function $F(\boldsymbol{c})$ is minimized one degree of freedom, i.e. one $c_i$, at a time freezing all the other entries in $\boldsymbol{c}$ to their current $\hat{\boldsymbol{c}}^k$ estimates
- I.e. the line search directions are the coordinate directions
- $\Delta \boldsymbol{c} = (0, \ldots, 0, 1, 0, \ldots, 0)^T = \hat{e}_i$ where the $\hat{e}_i$ are the standard basis functions, and so the 1D line search uses $\hat{c}(t) = \hat{c}^k + t \hat{e}_i$ without needing to invert the Hessian

Gradient/Steepest Descent

- Set $\hat{H}(\hat{\boldsymbol{c}}^k) = I$, so that $\hat{H}(\hat{\boldsymbol{c}}^k) \Delta \boldsymbol{c} = -\hat{f}(\hat{\boldsymbol{c}}^k)$ becomes $\Delta \boldsymbol{c} = -\hat{f}(\hat{\boldsymbol{c}}^k) = -\nabla F(\hat{\boldsymbol{c}}^k)$, which makes the line search direction the steepest descent direction
- Again, no Hessian inversion is required