Avoiding Derivatives
Part II Roadmap

• Part I – Linear Algebra (units 1-12) \( Ac = b \)

• Part II – Optimization (units 13-20)
  • (units 13-16) Optimization -> Nonlinear Equations -> 1D roots/minima
  • (units 17-18) Computing/Avoiding Derivatives
  • (unit 19) Hack 1.0: “I give up” \( H = I \) and \( J \) is mostly 0 (descent methods)
  • (unit 20) Hack 2.0: ”It’s an ODE!?“ (adaptive learning rate and momentum)
1D Root Finding (see Unit 15)

- Newton’s method requires $g'$, as do mixed methods using Newton
- **Secant** method replaces $g'$ with a secant line through two prior iterates
- **Finite differencing** (unit 17) may be used to approximate this derivative as well, although one needs to determine the size of the perturbation $h$
- **Automatic differentiation** (unit 17) may be used to find the value of $g'$ at a particular point, if/when “backprop” code exists, even when $g$ and $g'$ are not known in closed form
- Convergence is only guaranteed under certain conditions, emphasizing the importance of safe set methods (such as mixed methods with bisection)
- Safe set methods also help to guard against errors in derivative approximations
1D Optimization (see Unit 16)

• Root finding approaches search for critical points as the roots of $g'$
  • All root finding methods use the function itself ($g'$ here)
  • Newton (and mixed methods using Newton) require the derivative of the function ($g''$ here)
• Can use secant lines for $g'$ and interpolating parabolas for $g''$, using either prior iterates (unit 16) or finite differences (unit 17)
• Automatic differentiation (unit 17) may be leveraged as well
  • Although, not (typically) for approaches that require $g''$
• Safe set methods (such as mixed methods with bisection or golden section search) help to guard against errors in the approximation of various derivatives
Nonlinear Systems (see Unit 14)

- \( J_F(c^q) \Delta c^q = -F(c^q) \) is solved to find the search direction \( \Delta c^q \)
  - Then, line search utilizes various 1D approaches (unit 15/16)
- The Jacobian matrix of first derivatives \( J_F(c^q) \) needs to be evaluated (given \( c^q \))
- Each entry \( \frac{\partial F_i}{\partial c_k}(c^q) \) can be approximated via \textit{finite differences} (unit 17) or \textit{automatic differentiation} (unit 17)
- Quasi-Newton approaches make various aggressive approximations to the Jacobian \( J_F(c^q) \)
- Quasi-Newton can wildly perturb the search direction, so \textit{robust/safe set} approaches to the 1D line search become quite important to making “progress” towards solutions
Broyden’s Method

• An initial guess for the Jacobian is repeatedly corrected with rank one updates, similar in spirit to a secant approach

• Let $J^0 = I$

• Solve $J^q \Delta c^q = -F(c^q)$ to find search direction $\Delta c^q$
  • Use 1D line search to find $c^{q+1}$ and thus $F(c^{q+1})$; then, update $\Delta c^q = c^{q+1} - c^q$

• Update $J^{q+1} = J^q + \frac{1}{(\Delta c^q)^T \Delta c^q} (F(c^{q+1}) - F(c^q) - J^q \Delta c^q)(\Delta c^q)^T$

• Note: $J^{q+1}(c^{q+1} - c^q) = F(c^{q+1}) - F(c^q)$
• That is, $J^{q+1}$ satisfies a secant type equation $J \Delta c = \Delta F$
Optimization (see Unit 13)

• Scalar cost function $\hat{f}(c)$ has critical points where $J^T_{\hat{f}}(c) = 0$ (unit 13)

• $H^T_{\hat{f}}(c^q)\Delta c^q = -J^T_{\hat{f}}(c^q)$ is solved to find a search direction $\Delta c^q$ (unit 14)

• Then, line search utilizes various 1D approaches (unit 15/16)

• The Hessian matrix of second derivatives $H^T_{\hat{f}}(c^q)$ and the Jacobian vector of first derivatives $J^T_{\hat{f}}(c^q)$ both need to be evaluated (given $c^q$)

• The various entries can be evaluated via finite differences (unit 17) or automatic differentiation (unit 17)

• These approaches can struggle on the Hessian matrix of second partial derivatives

• This makes Quasi-Newton approaches quite popular for optimization
  • When $c$ is large, the $O(n^2)$ Hessian $H^T_{\hat{f}}$ is unwieldy/intractable, so some approaches instead approximate the action of $H^{-T}_{\hat{f}}$ on a vector (i.e., on the right hand side)
Broyden’s Method (for Optimization)

- **Same** formulation as for nonlinear systems

- Solve for the search direction, and use 1D line search to find $c^{q+1}$ and $J_f^T(c^{q+1})$

- Update $\Delta c^q = c^{q+1} - c^q$ and $\Delta J_f^T = J_f^T(c^{q+1}) - J_f^T(c^q)$

- Update $(H_f^T)^{q+1} = (H_f^T)^q + \frac{1}{(\Delta c^q)^T \Delta c^q} \left( \Delta J_f^T - (H_f^T)^q \Delta c^q \right) (\Delta c^q)^T$

- So that $(H_f^T)^{q+1} \Delta c^q = \Delta J_f^T$
Broyden’s Method (for Optimization)

- For the inverse, using $\Delta c^q = c^{q+1} - c^q$ and $\Delta J_f^T = J_f^T(c^{q+1}) - J_f^T(c^q)$
- Update $(H_f^{-T})^{q+1} = (H_f^{-T})^q + \frac{(\Delta c^q - (H_f^{-T})^q \Delta J_f^T)(\Delta c^q)^T(H_f^{-T})^q}{(\Delta c^q)^T(H_f^{-T})^q \Delta J_f^T}$
- So that $(H_f^{-T})^{q+1} \Delta J_f^T = \Delta c^q$
- Solving $H_f^T(c^{q+1})\Delta c^{q+1} = -J_f^T(c^{q+1})$ is replaced with defining the search direction by $\Delta c^{q+1} = -(H_f^{-T})^{q+1}J_f^T(c^{q+1})$
SR1 (Symmetric Rank 1)

• For the inverse, using $\Delta c^q = c^{q+1} - c^q$ and $\Delta J_{\hat{f}}^T = J_{\hat{f}}^T (c^{q+1}) - J_{\hat{f}}^T (c^q)$

• Update $(H_{\hat{f}}^{-T})^{q+1} = (H_{\hat{f}}^{-T})^q + \frac{(\Delta c^q - (H_{\hat{f}}^{-T})^q \Delta J_{\hat{f}}^T)(\Delta c^q - (H_{\hat{f}}^{-T})^q \Delta J_{\hat{f}}^T)^T}{\Delta c^q - (H_{\hat{f}}^{-T})^q \Delta J_{\hat{f}}^T \Delta J_{\hat{f}}^T}$

• So that $(H_{\hat{f}}^{-T})^{q+1} \Delta J_{\hat{f}}^T = \Delta c^q$

• Solving $H_{\hat{f}}^T (c^{q+1}) \Delta c^{q+1} = -J_{\hat{f}}^T (c^{q+1})$ is replaced with defining the search direction by $\Delta c^{q+1} = -(H_{\hat{f}}^{-T})^{q+1} J_{\hat{f}}^T (c^{q+1})$
DFP (Davidon-Fletcher-Powell)

- For the inverse, using $\Delta c^q = c^{q+1} - c^q$ and $\Delta J_f^T = J_f^T(c^{q+1}) - J_f^T(c^q)$

- Update $(H_f^{-T})^{q+1} = (H_f^{-T})^q - \frac{(H_f^{-T})^q \Delta J_f^T \Delta J_f (H_f^{-T})^q}{\Delta J_f (H_f^{-T})^q \Delta J_f^T} + \frac{\Delta c^q (\Delta c^q)^T}{(\Delta c^q)^T \Delta J_f^T}$

- So that $(H_f^{-T})^{q+1} \Delta J_f^T = \Delta c^q$

- Solving $H_f^T(c^{q+1})\Delta c^{q+1} = -J_f^T(c^{q+1})$ is replaced with defining the search direction by $\Delta c^{q+1} = -(H_f^{-T})^{q+1} J_f^T(c^{q+1})$
BFGS (Broyden-Fletcher-Goldfarb-Shanno)

- For the inverse, using $\Delta c^q = c^{q+1} - c^q$ and $\Delta J^T_{f^\hat{}} = J^T_{f^\hat{}}(c^{q+1}) - J^T_{f^\hat{}}(c^q)$
- Update $(H^T_{f^\hat{}})^{q+1} = \left(I - \frac{\Delta c^q \Delta J^T_{f^\hat{}}}{(\Delta c^q)^T \Delta J^T_{f^\hat{}}}ight) (H^T_{f^\hat{}})^q \left(I - \frac{\Delta J^T_{f^\hat{}}(\Delta c^q)^T}{(\Delta c^q)^T \Delta J^T_{f^\hat{}}}ight) + \frac{\Delta c^q (\Delta c^q)^T}{(\Delta c^q)^T \Delta J^T_{f^\hat{}}}$
- So that $(H^T_{f^\hat{}})^{q+1} \Delta J^T_{f^\hat{}} = \Delta c^q$

- Solving $H^T_{f^\hat{}}(c^{q+1}) \Delta c^{q+1} = -J^T_{f^\hat{}}(c^{q+1})$ is replaced with defining the search direction by $\Delta c^{q+1} = -(H^T_{f^\hat{}})^{q+1} J^T_{f^\hat{}}(c^{q+1})$
L-BFGS (Limited Memory BFGS)

- These methods store an \( n \times n \) approximation to the inverse Hessian
  - This can become unwieldy for large problems
  - Smarter storage can be accomplished by storing the vectors that describe the outer products; however, the number of vectors grows with \( q \)
- L-BFGS estimates the inverse Hessian using only a few of the prior vectors
  - often less than 10 vectors (vectors, vector spaces, not matrices)
- This makes it quite popular for machine learning

- On optimization methods for deep learning, Andrew Ng et al., ICML 2011
  - “we show that more sophisticated off-the-shelf optimization methods such as Limited memory BFGS (L-BFGS) and Conjugate gradient (CG) with line search can significantly simplify and speed up the process of pretraining deep algorithms”
Gradient/Steepest Descent

• Approximate $H^T_{\hat{f}}$ very crudely with the identity matrix
  • which is the first step of all the aforementioned methods

• That is, $H^T_{\hat{f}}(c^q)\Delta c^q = -J^T_{\hat{f}}(c^q)$ becomes $I\Delta c^q = -J^T_{\hat{f}}(c^q)$

• So, the search direction is $\Delta c^q = -J^T_{\hat{f}}(c^q) = -\nabla \hat{f}(c^q)$
  • This is the steepest descent direction

• See unit 19
Coordinate Descent

- Coordinate Descent ignores $H_f^T(c^q)\Delta c^q = -J_f^T(c^q)$ completely.
- Instead, $\Delta c^q$ is set to the various coordinate directions $\hat{e}_k$. 

Nonlinear Least Squares (ML relevancy)

• Minimize a cost function of the form: \( \hat{f}(c) = \frac{1}{2} \tilde{f}^T(c)\tilde{f}(c) \)

• Recall from Unit 13:
  • Determine parameters \( c \) that make \( f(x, y, c) = 0 \) best fit the training data, i.e. that make \( \|f(x_i, y_i, c)\|_2^2 = f(x_i, y_i, c)^T f(x_i, y_i, c) \) close to zero for all \( i \)
  • Combining all \( (x_i, y_i) \), minimize \( \hat{f}(c) = \frac{1}{2} \sum_i f(x_i, y_i, c)^T f(x_i, y_i, c) \)

• Let \( m \) be the number of data points and \( \hat{m} \) be the output size of \( f(x, y, c) \)

• Define \( \tilde{f}(c) \) by stacking the \( \hat{m} \) outputs of \( f(x, y, c) \) consecutively \( m \) times, so that the vector valued output of \( \tilde{f}(c) \) is length \( m \times \hat{m} \)

• Then, \( \hat{f}(c) = \frac{1}{2} \sum_i f(x_i, y_i, c)^T f(x_i, y_i, c) = \frac{1}{2} \tilde{f}^T(c)\tilde{f}(c) \)
Nonlinear Least Squares (Critical Points)

- Minimize \( \hat{f}(c) = \frac{1}{2} \tilde{f}^T(c) \tilde{f}(c) \)

- Jacobian matrix of \( \tilde{f} \) is \( J_{\tilde{f}}(c) = \begin{pmatrix} \frac{\partial \tilde{f}}{\partial c_1}(c) & \frac{\partial \tilde{f}}{\partial c_2}(c) & \cdots & \frac{\partial \tilde{f}}{\partial c_n}(c) \end{pmatrix} \)

- Critical points of \( \hat{f}(c) \) have \( J_{\tilde{f}}^T(c) \hat{f}(c) = 0 \)
Gauss Newton

- \( J_f^T(c)\bar{f}(c) = 0 \) becomes \( J_f^T(c)(\bar{f}(c^q) + J_f(c^q)\Delta c^q + \cdots) = 0 \)
- Using the Taylor series: \( \bar{f}(c) = \bar{f}(c^q) + J_f(c^q)\Delta c^q + \cdots \)
- Eliminating high order terms: \( J_f^T(c)(\bar{f}(c^q) + J_f(c^q)\Delta c^q) \approx 0 \)
- Evaluating \( J_f^T \) at \( c^q \) gives \( J_f^T(c^q)J_f(c^q)\Delta c^q \approx -J_f^T(c^q)\bar{f}(c^q) \)
- Compare to \( H_f^T(c^q)\Delta c^q = -J_f^T(c^q) \) and note that \( J_f^T(c) = J_f^T(c)\bar{f}(c) \)
- Thus, Gauss Newton uses the estimate: \( H_f^T(c^q) \approx J_f^T(c^q)J_f(c^q) \)
Gauss Newton \((QR\ \text{approach})\)

• The Gauss Newton equations
  \[ J_f^T(c^q)J_f(c^q)\Delta c^q = -J_f^T(c^q)\tilde{f}(c^q) \]
  are the normal equations for
  \[ J_f(c^q)\Delta c^q = -\tilde{f}(c^q) \]

• Thus, (instead) solve
  \[ J_f(c^q)\Delta c^q = -\tilde{f}(c^q) \]
  via any least squares \((QR)\) and minimum norm approach

• Note: setting the second factor in
  \[ J_f^T(c)(\tilde{f}(c^q) + J_f(c^q)\Delta c^q) \approx 0 \]
  to zero also leads to
  \[ J_f(c^q)\Delta c^q = -\tilde{f}(c^q) \]

• This is a linearization of the nonlinear system \(\tilde{f}(c) = 0\), aiming to minimize
  \[ \hat{f}(c) = \frac{1}{2}\tilde{f}^T(c)\tilde{f}(c) \]
Weighted Gauss Newton

- Given a diagonal matrix $D$ indicating the importance of various equations:
  
  $DJ_{\tilde{f}}(c^q)\Delta c^q = -D\tilde{f}(c^q)$
  
  $J_{\tilde{f}}^T(c^q)D^2J_{\tilde{f}}(c^q)\Delta c^q = -J_{\tilde{f}}^T(c^q)D^2\tilde{f}(c^q)$

- Recall: Row scaling changes the importance of the equations
- Recall: Thus, it also changes the (unique) least squares solution for any overdetermined degrees of freedom
Regularized Gauss Newton

• When concerned about small singular values in $J_{\tilde{f}}(c^q)\Delta c^q = -\tilde{f}(c^q)$, one can add $\epsilon I = 0$ as extra equations (unit 12 regularization)

• This results in $\left( J_{\tilde{f}}^T(c^q)J_{\tilde{f}}(c^q) + \epsilon^2 I \right)\Delta c^q = -J_{\tilde{f}}^T(c^q)\tilde{f}(c^q)$

• This is often called **Levenberg-Marquardt** or **Damped (Nonlinear) Least Squares**