CS 450: Numerical Anlaysis

Lecture 16

Chapter 6 Numerical Optimization

Unconstrained Optimization Algorithms and Nonlinear Least Squares

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

April 20, 2018

Successive Parabolic Interpolation

- Interpolate f with a quadratic function at each step and find its minima:

  Given three points, there is a unique quadratic function interpolating them.

- The convergence rate of the resulting method is roughly 1.324

  By comparison, the convergence of bisection is linear with a constant of 0.618*, while Newton's method converges quadratically.*

Newton's Method for Optimization

- Instead of interpolating points, match derivatives at current approximate point:

  Pick quadratic function $\hat{f}$ as first three terms of Taylor expansion of f about x_k, matching value and first two derivatives of f at x_k.
- The new approximate guess will be given by $x_{k+1} - x_k = -f'(x_k)/f''(x_k)$:

$$f(x_{k+1}-x_k) \approx \hat{f}(x_{k+1}-x_k) = f(x_k)+f'(x_k)(x_{k+1}-x_k)+\frac{1}{2}f''(x_k)(x_{k+1}-x_k)^2$$

since the function is quadratic, we can find its unique critical point to find its minima,

$$\hat{f}'(x_{k+1}-x_k) = f'(x_k) + f''(x_k)(x_{k+1}-x_k) = 0.$$

Safeguarded 1D Optimization

- Safeguarding can be done by bracketing via golden section search:

  Combination of Newton and golden section search achieves quadratic convergence locally and guaranteed convergence provided unimodality of function.

- Backtracking and step-size control:

  As in nonlinear solves, can take small step $x_{k+1} = x_k - \alpha_k f'(x_k)/f''(x_k)$

Multidimensional Optimization

- Direct search methods by simplex (*Nelder-Mead*):

  Form a n-point polytope in n-dimensional space and adjust worst point (highest function value) by moving it along a line passing through the centroid of the remaining points.

- Steepest descent: find the minimizer in the direction of the negative gradient:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

  such that $f(\boldsymbol{x}_{k+1}) = \min_{\alpha_k} f(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k))$, *i.e. perform a line search (solve 1D optimization problem) in the direction of the negative gradient.*

Convergence of Steepest Descent

- Steepest descent converges linearly with a constant that can be arbitrarily close to 1:

  See Michael Heath's notes chapter 6 slide 35 for an example. Convergence is slow locally, in the worst case, and generally depends on the Hessian near the minima. If the gradient is changing quickly, it serves as good approximation only within a small local neighborhood, so the line search may result in arbitrarily small steps.

Newton's Method for Multidimensional Optimization

- Newton's method in n dimensions is given by finding minima of n-dimensional quadratic approximation:

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx \hat{f}(\boldsymbol{x}_k + \boldsymbol{s}) = f(\boldsymbol{x}_k) + \boldsymbol{s}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{s}^T \boldsymbol{H}_f(\boldsymbol{x}_k)\boldsymbol{s}$$

The minima of this function can be determined by identifying critical points

$$\boldsymbol{0} = \nabla \hat{f}(\boldsymbol{x}_k + \boldsymbol{s}) = \nabla f(\boldsymbol{x}_k) + \boldsymbol{H}_f(\boldsymbol{x}_k)\boldsymbol{s},$$

thus to determine s we solve the linear system

$$\boldsymbol{H}_f(\boldsymbol{x}_k)\boldsymbol{s} = -\nabla f(\boldsymbol{x}_k)$$

assuming invertibility of the Hessian, we can write the Newton's method iteration as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \underbrace{\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)}_{\boldsymbol{s}}$$

Convergence of Newton's Method

- Newton's method is quadratically convergent in n-dimensions as well:

$$\begin{aligned}
\boldsymbol{e}_{k+1} &= \boldsymbol{x}_{k+1} - \boldsymbol{x}^* \\
&= \boldsymbol{x}_k - \boldsymbol{x}^* - \boldsymbol{H}_f(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k) \\
&= -\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1}(\boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}^*) - \nabla f(\boldsymbol{x}_k)) \\
&= -\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1}(\boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}^*) - (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^*))), \\
||\boldsymbol{e}_{k+1}|| &\le ||\boldsymbol{H}_f(\boldsymbol{x}_k)^{-1}||\, \underbrace{||\boldsymbol{H}_f(\boldsymbol{x}_k)(\boldsymbol{x}_k - \boldsymbol{x}^*) - (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}^*)||}_{O(||\boldsymbol{x}_k - \boldsymbol{x}^*||^2)} \\
&= C||\boldsymbol{e}_k||^2
\end{aligned}$$

Quasi-Newton Methods

- *Quasi-Newton* methods compute approximations to the Hessian at each step:

  $$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k)$$

  where α_k is a line search parameter. Quasi-Newton methods can be more robust than Newton's method, as the Newton's method step can lead to a direction in which the objective function is strictly increasing.

- The *BFGS* method is a secant update method, similar to Broyden's method:

  At each iteration, perform a rank-1 update to $\boldsymbol{B}_k$. Can update inverse with $O(n^2)$ work, but in practice should update a symmetric indefinite factorization. The BFGS method also preserves symmetry of the Hessian.

Nonlinear Least Squares

- Lets consider a *nonlinear least squares* problem of fitting a nonlinear function $f_{\boldsymbol{x}}(t)$ so that $f_{\boldsymbol{x}}(t_i) \approx y_i$:

  For example, consider fitting $f_{[x_1,x_2]}(t) = x_1 \sin(x_2 t)$.

- We can cast nonlinear least squares as an optimization problem and solve it by Newton's method:

  Define residual vector function $\boldsymbol{r}(\boldsymbol{x})$ so that $r_i(\boldsymbol{x}) = y_i - f_{\boldsymbol{x}}(t_i)$ and minimize

$$\phi(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{r}(\boldsymbol{x})||_2^2 = \frac{1}{2}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x})$$

  Now the gradient is $\nabla\phi(\boldsymbol{x}) = \boldsymbol{J_r}(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x})$ and the Hessian is

$$\boldsymbol{H}_\phi(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x})\boldsymbol{J_r}(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})\boldsymbol{H}_{r_i}(\boldsymbol{x})$$

Gauss-Newton Method

- The Gauss-Newton method is a simplification of Newton's method for the nonlinear least squares problem:

$$\boldsymbol{H}_{\phi}(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^{T}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})\boldsymbol{H}_{r_i}(\boldsymbol{x})$$

  the second term is small when the residual function $\boldsymbol{r}(\boldsymbol{x})$ is small, so approximate

$$\boldsymbol{H}_{\phi}(\boldsymbol{x}) \approx \hat{\boldsymbol{H}}_{\phi}(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{r}}^{T}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x})$$

- The method is then given by

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \hat{\boldsymbol{H}}_{\phi}(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k) \\ &= \boldsymbol{x}_k - (\boldsymbol{J}_{\boldsymbol{r}}^{T}(\boldsymbol{x}_k)\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k))^{-1}\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\boldsymbol{r}(\boldsymbol{x}_k) \end{aligned}$$

  Thus the Gauss-Newton method operates by solving a linear least squares problem $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\boldsymbol{s} \cong \boldsymbol{r}(\boldsymbol{x}_k)$

Levenberg-Marquardt Method

- The *Levenberg-Marquardt* modifies the Gauss-Newton method to use Tykhonov regularization:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{x}_k)\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k) - \mu \boldsymbol{I})^{-1}\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\boldsymbol{r}(\boldsymbol{x}_k)$$

- The scalar μ controls the step size through the least squares problem:

$$\begin{bmatrix} \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k) \\ \sqrt{\mu}\boldsymbol{I} \end{bmatrix} \boldsymbol{s} \cong \begin{bmatrix} \boldsymbol{r}(\boldsymbol{x}_k) \\ \boldsymbol{0} \end{bmatrix}$$

By increasing μ we force the step $\boldsymbol{s}$ to be smaller.