Chapter 3
Numerical Methods

Part 2

3.2 Systems of Equations

3.3 Nonlinear and Constrained Optimization

Carnegie Mellon
THE ROBOTICS INSTITUTE

Outline

- 3.2 Systems of Equations
- 3.3 Nonlinear and Constrained Optimization
- Summary

Outline

- 3.2 Systems of Equations
  - 3.2.1 Linear Systems
  - 3.2.2 Nonlinear Systems
- 3.3 Nonlinear and Constrained Optimization
- Summary

3.2.1.1 Square Systems

- You know this one already…
- Suppose H is square and:

$$\underline{z} = H\underline{x}$$

- The “solution” is:

$$\underline{x} = H^{-1}\underline{z}$$

- Use MATLAB and you’re done.
- But how do you invert a matrix yourself?
- → Row operations do not change the solution of the linear system.

Gaussian Elimination

- Consider three equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = y_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = y_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = y_3$$

- Multiply 2nd equation by a31/a21

$$a_{31}x_1 + \left(\frac{a_{31}}{a_{21}}\right)a_{22}x_2 + \left(\frac{a_{31}}{a_{21}}\right)a_{23}x_3 = \left(\frac{a_{31}}{a_{21}}\right)y_2$$

- Subtract this from 3rd equation:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = y_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = y_2$$
$$a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 = y_3^{(1)}$$

Gaussian Elimination

- Multiply 1st equation by a21/a11 and eliminate x1 from second equation:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 &= y_2^{(1)} \\
a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 &= y_3^{(1)}
\end{aligned}
$$

- Use same process to (new) 2nd and 3rd equations to eliminate x2:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 &= y_2^{(1)} \\
a_{33}^{(2)}x_3 &= y_3^{(2)}
\end{aligned}
$$

Gaussian Elimination

- Now we have:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\
a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 &= y_2^{(1)} \\
a_{33}^{(2)}x_3 &= y_3^{(2)}
\end{aligned}
$$

- Solve 3rd equation for x3, then 2nd equation for x2 etc.
- Notice:
  – Process generalizes to larger systems.
  – Process works for arbitrary matrices.

3.2.1.2 Left Pseudoinverse

- Consider again $\underline{z} = H\underline{x}$ where H is m X n, m>n. Called an overdetermined system.
- Define the residual vector:

$$\underline{r}(\underline{x}) = \underline{z} - H\underline{x}$$

- Define a cost function as its magnitude:

$$f(\underline{x}) = \frac{1}{2}\underline{r}^T(\underline{x})\underline{r}(\underline{x})$$

- Substitute the definition of residual:

$$f(\underline{x}) = \frac{1}{2}(\underline{z} - H\underline{x})^T(\underline{z} - H\underline{x})$$

3.2.1.2 Left Pseudoinverse

- Use the product rule to differentiate $f(\underline{x}) = \frac{1}{2}(\underline{z} - H\underline{x})^T(\underline{z} - H\underline{x})$ x:

$$f_{\underline{x}} = (\underline{z} - H\underline{x})^T H$$

- This will vanish at any local minimum:

$$(\underline{z} - H\underline{x}^*)^T H = 0$$

- Requires residual at minimizer to be orthogonal to the column space of H.
  - Hence, known as the “normal equations”.
- The value of H$\underline{x}$*:
  - Is in the column space of H
  - Has a residual (z-H$\underline{x}$*) of minimum length.

3.2.1.2 Left Pseudoinverse

- Move z to other side and solve:

$$H^T H \underline{x}^* = H^T \underline{z}$$

$$\boxed{\underline{x}^* = (H^T H)^{-1} H^T \underline{z}}$$

- The matrix:

$$\boxed{H^+ = (H^T H)^{-1} H^T}$$

- … is called the Left Pseudoinverse of H because…

$$H^+ H = (H^T H)^{-1} H^T H = I_n \qquad \mathrm{m > n}$$

3.2.1.3 Right Pseudoinverse

- Consider again $\underline{z} = H\underline{x}$ where H is m X n, m<n. Called an underdetermined system.
- There are potentially an infinite number of solutions.
- Simple technique is to introduce a regularizer (cost function) to rank all solutions and pick the best.
- Define a cost function as the (squared) magnitude of x:

$$f(\underline{x}) = \frac{1}{2}\underline{x}^T\underline{x}$$

3.2.1.3 Right Pseudoinverse

- Now form a constrained optimization problem:

$$\text{optimize:}_{\underline{x}} \quad f(\underline{x}) \qquad \underline{x} \in \Re^n$$

$$\text{subject to:} \quad \underline{c}(\underline{x}) = \underline{z} - H\underline{x} = \underline{0} \qquad \underline{h} \in \Re^m$$

- Form the Lagrangian:

$$l(\underline{x}, \underline{\lambda}) = \frac{1}{2}\underline{x}^T\underline{x} + \underline{\lambda}^T(\underline{z} - H\underline{x})$$

- First necessary condition is:

$$l_{\underline{x}}(\underline{x}, \underline{\lambda})^T = \underline{x} - H^T\underline{\lambda} = \underline{0} \Rightarrow \underline{x} = H^T\underline{\lambda}$$

3.2.1.3 Right Pseudoinverse

- Substitute into the second necessary condition (constraints):

$$l_{\underline{\lambda}}(\underline{x}, \underline{\lambda})^T = \underline{z} - H\underline{x} = \underline{0}$$

$$\underline{z} - HH^T\underline{\lambda} = \underline{0}$$

$$HH^T\underline{\lambda} = \underline{z}$$

- The solution for the multipliers is:

$$\underline{\lambda} = (HH^T)^{-1}\underline{z}$$

3.2.1.3 Right Pseudoinverse

- Substitute back into the first equation:

$$\underline{x} = H^T\underline{\lambda} = H^T(HH^T)^{-1}\underline{z}$$

- The matrix:

$$H^+ = H^T(HH^T)^{-1}$$

- … is known as the right pseudoinverse because …

$$HH^+ = HH^T(HH^T)^{-1} = I_m \qquad \mathrm{m} < \mathrm{n}$$

3.2.1.4 About The Pseudoinverse

- Both LPI and RPI
  - reduce to the regular inverse when the matrix is square.
  - require H to be of full rank
  - invert a matrix whose dimension is the smaller of m and n
- It is possible to define “weighted” pseudoinverses (easy to re-derive). For example:

$$f(\underline{x}) = \frac{1}{2}\underline{r}^T(\underline{x})W\underline{r}(\underline{x})$$

Produces the Weighted LPI

Outline

- 3.2 Systems of Equations
  - 3.2.1 Linear Systems
  - 3.2.2 Nonlinear Systems
- 3.3 Nonlinear and Constrained Optimization
- Summary

Standard Form

- The problem of solving: $\underline{g}(\underline{x}) = \underline{b}$
- Is equivalent to solving: $\underline{c}(\underline{x}) = \underline{g}(\underline{x}) - \underline{b} = 0$
- Often, x̲ is really an unknown vector of parameters denoted as p̲.
- Note that:

$$\underline{c}_{\underline{x}} = \frac{\partial \underline{c}(\underline{x})}{\partial \underline{x}} = \frac{\partial \underline{g}(\underline{x})}{\partial \underline{x}} = \underline{g}_{\underline{x}}$$

3.2.2.1 Newton's Method

- Basic trick of numerical methods.....
- Linearize the constraints about a nonfeasible point

$$\underline{c}(\underline{x}+\Delta\underline{x}) = \underline{c}(\underline{x}) + \underline{c}_{\underline{x}}\Delta\underline{x} + \ldots$$

- Require feasibility after perturbation:

$$\underline{c}(\underline{x}+\Delta\underline{x}) = \underline{0}$$

- Leads to:

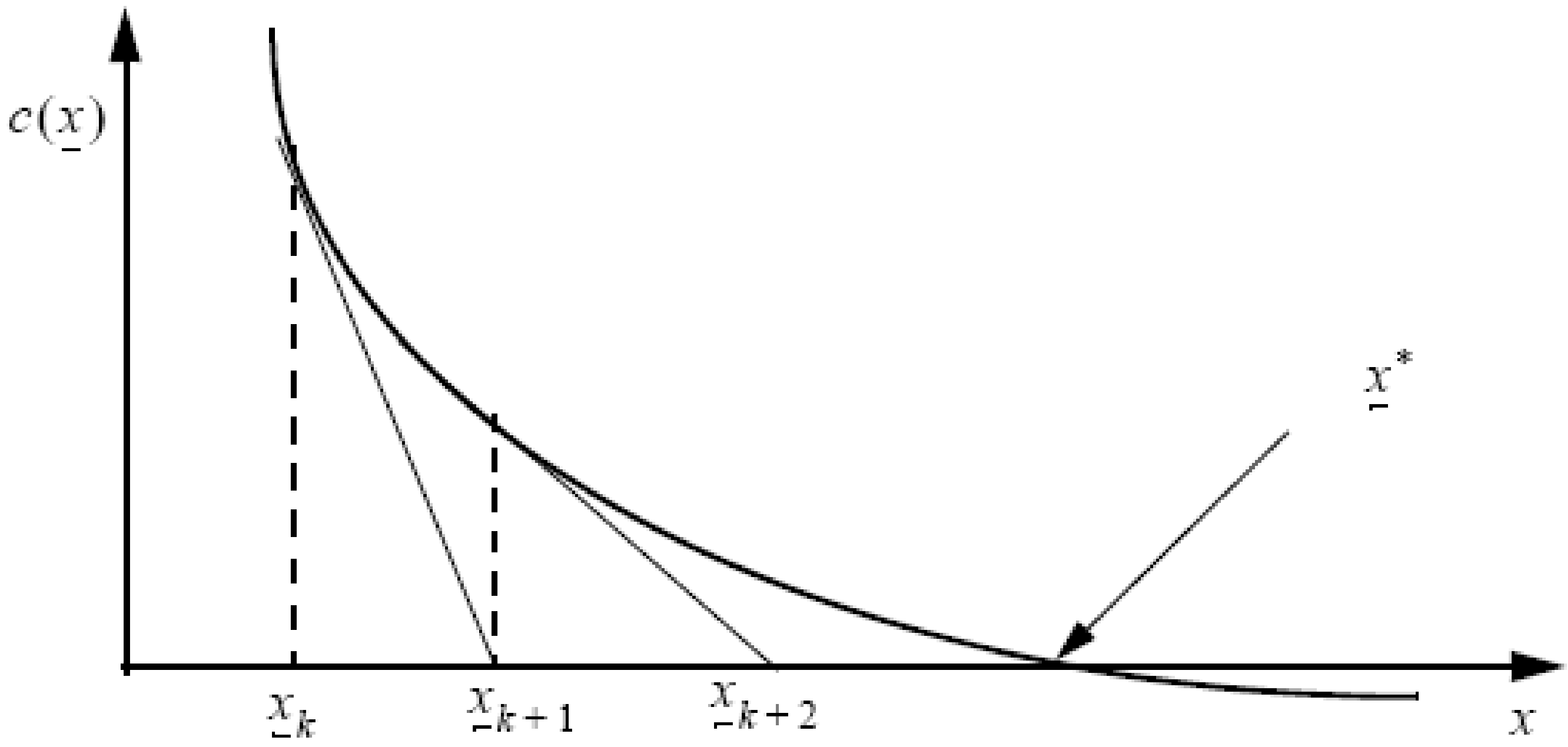
$$\underline{c}_{\underline{x}}\Delta\underline{x} = -\underline{c}(\underline{x})$$

- Basic iteration is:

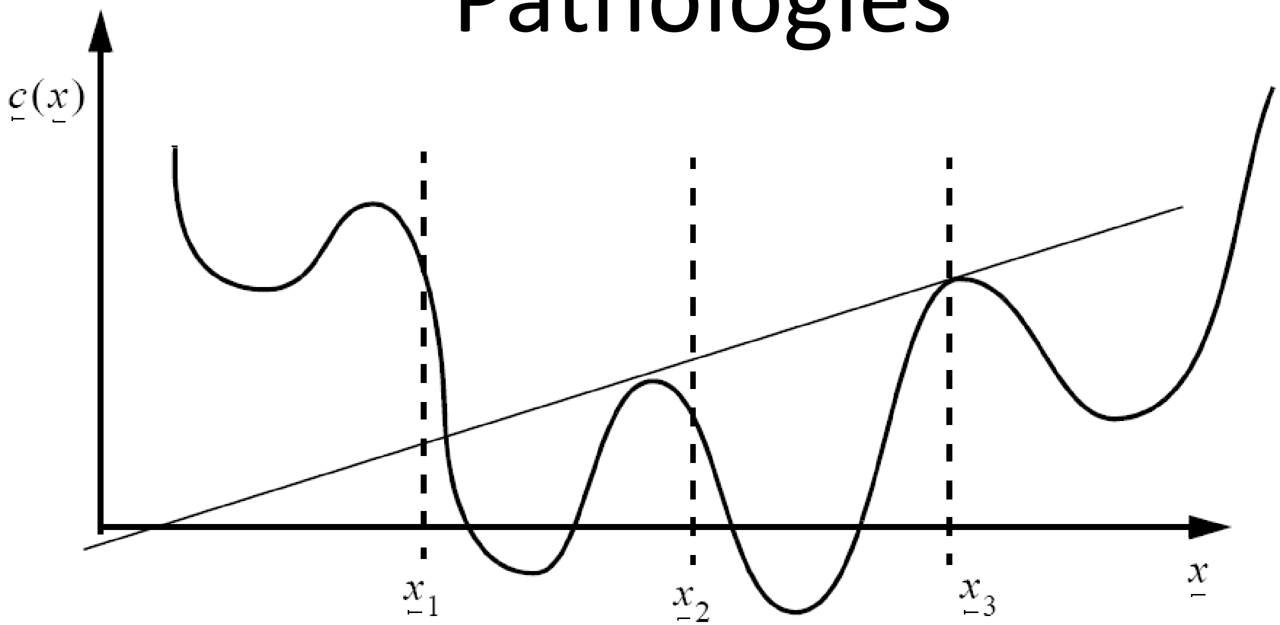
The precise change which, when added to x, will produce a root of (the linearization of) c(x)

$$\Delta\underline{x} = -\underline{c}_{\underline{x}}^{-1}\underline{c}(\underline{x}) = -\underline{c}_{\underline{x}}^{-1}[\underline{g}(\underline{x}) - \underline{b}]$$

Visualizing Newton's Method



Pathologies



- Nonlinear functions can have several roots – each with its own radius of convergence.
- At an extremum (not at a root) the Jacobian is not invertible.
- Near an extremum, huge jumps to a different root are possible.

3.2.2.3 Numerical Derivatives

- Often its simpler, less error prone, and less computation to differentiate numerically.
- Compute the constraint vector one additional time at a perturbed location:

$$\frac{\partial \underline{c}}{\partial x_i} = \frac{\underline{c}(\underline{x} + \Delta \underline{x}_i) - \underline{c}(\underline{x})}{\Delta \underline{x}_i} \qquad \Delta \underline{x}_i = \begin{bmatrix} 0 & 0 & \ldots & \Delta x_i & \ldots & 0 & 0 \end{bmatrix}$$

i-th position

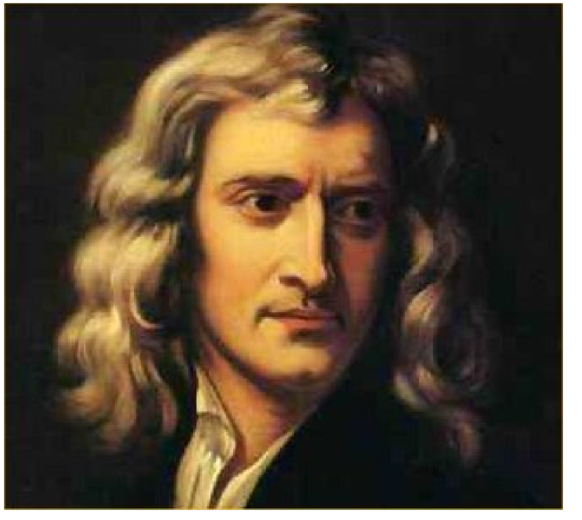
- This gives a numerical approximation for the i-th column of the Jacobian.
- Collect them all to get $\underline{C}_{\underline{x}}$

Outline

- 3.2 Systems of Equations
- 3.3 Nonlinear and Constrained Optimization
  - 3.3.1 Nonlinear Optimization
  - 3.3.3 Constrained Optimization
- Summary

Isaac Newton

- English mathematician and scientist.
- Perhaps the greatest analytic thinker in human history.
- Graduated Trinity College Cambridge in 1665.
- Then came the Great Plague.
  - University shut down for 2 years.
- Worked at home on calculus, gravitation, and optics.
  - Figured them all out!
- We will use his calculus to solve nonlinear equations
  - and a few other things !!!



Isaac Newton
1643 -1727

3.3.1 Nonlinear Optimization

- The general nonlinear optimization problem:

$$\text{minimize:}_{\underline{x}} \quad f(\underline{x}) \qquad \underline{x} \in \Re^n$$

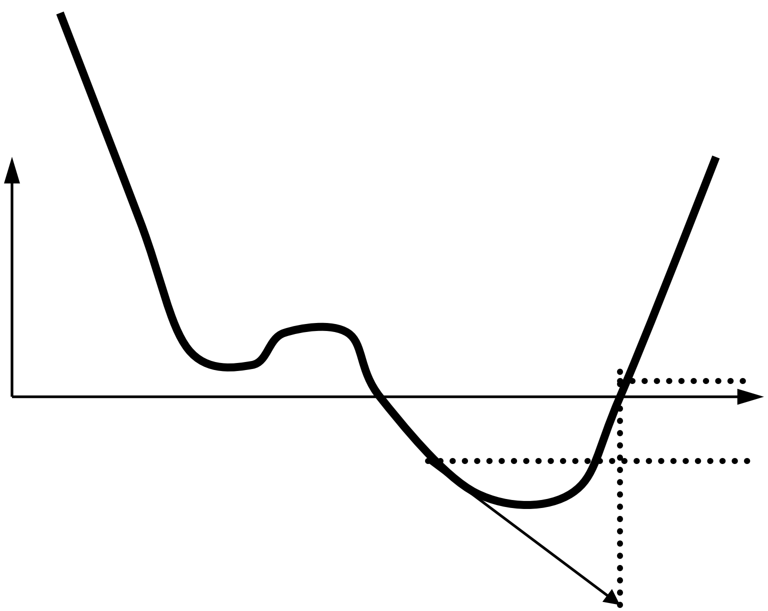
- Numerical techniques produce a sequence of estimates such that:

$$f(\underline{x}_{k+1}) < f(\underline{x}_k)$$

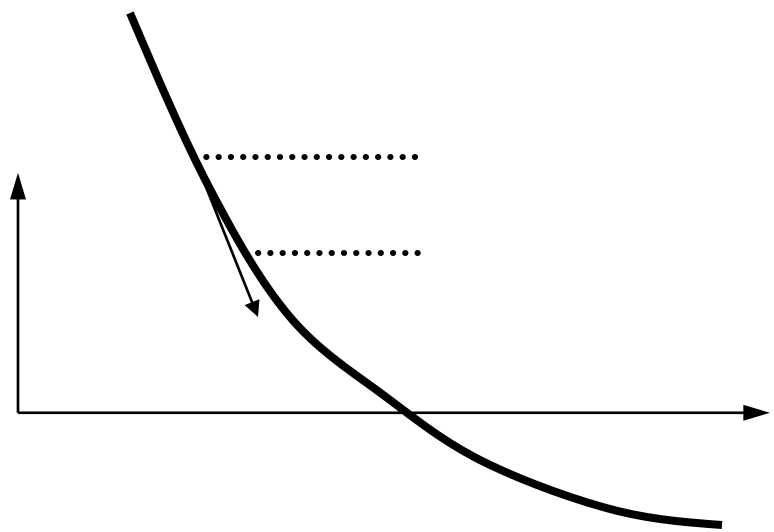
- …by controlling both the length and the direction of the steps.
- Two basic techniques:
  - 1) Line Search - adjusts length after choosing direction.
  - 2) Trust Region – adjusts direction after choosing length.

3.3.1.1 Line Search

- Often need to search the descent direction and that's expensive.
- Consider ways to be smart about this.....



Step Too Large



Step Too Small

3.3.1.1 Line Search

- Given a descent direction d, converts to a 1D problem:

$$\text{minimize:}_{\alpha} f(\underline{x} + \alpha \underline{d}) \qquad \alpha \in \Re^1$$

- Define the linearization of the scalar function:

$$\hat{f}(\alpha \underline{d}) = f(\underline{x}) + f_{\underline{x}}(\alpha \underline{d})$$

- Convergence is guaranteed if every iteration achieves sufficient decrease (relative to linear approximation)

$$\eta = \frac{f(\underline{x}) - f(\underline{x} + \alpha \underline{d})}{\hat{f}(\underline{0}) - \hat{f}(\alpha \underline{d})} > \eta_{min} \qquad \eta_{min}: \quad 0 < \eta_{min} < 1$$

- For efficiency, try large steps and backtrack if necessary with:

$$\alpha_{k+1} = (2^{-i})\alpha_k \quad : \quad i = 0, 1, 2, \ldots$$

Line Search Algorithm

algorithm *lineSearch* ( )
  $\underline{x} \leftarrow \underline{x}_0$ // initial guess
  $\eta_{min} \leftarrow const \in [0, 1/4]$
  $\alpha_{last} \leftarrow \alpha_0$
  while (**true**)
    $\underline{d} \leftarrow findDirection(\underline{x})$
    $\alpha \leftarrow \alpha_{last} \times 4$ // or $\alpha \leftarrow 1$ for Newton step
    while (**true**)
      $\eta \leftarrow$ (see Equation 3.45)
      if ($\eta > \eta_{min}$) **break** — Accept step
      $\alpha \leftarrow \alpha \div 2$ — Reduce stepsize
    endwhile
    $\underline{x} \leftarrow \underline{x} + \alpha\underline{d}$; $\alpha_{last} \leftarrow \alpha$ — Move to new estimate
    if(*finished*()) **break**
  endwhile
  return

Algorithm 3.1: Line Search in a Descent Direction with Backtracking. The algorithm searches repeatedly in a descent direction.

3.3.1.1.2 Descent Direction: Gradient Descent

- Also called steepest descent.
- Consider, approximating the objective by degree 1 Taylor polynomial…

$$f(\underline{x} + \Delta\underline{x}) \approx f(\underline{x}) + f_{\underline{x}}\Delta\underline{x}$$

- Hence the increase in the objective is the projection of Δx onto the gradient $f_{\underline{x}}$.
- Choose the negative gradient for max decrease:

$$\underline{d}^T = -f_{\underline{x}}$$

3.3.1.1.3 Descent Direction: Newton Step

- Of course the gradient vanishes at a local minimum.
- Write Taylor series for the gradient.

$$f_{\underline{x}}(\underline{x} + \Delta\underline{x}) \approx f_{\underline{x}}(\underline{x}) + f_{\underline{xx}}\Delta\underline{x} = \underline{0}^T$$

- Hence the step is given by:

$$f_{\underline{xx}}\Delta\underline{x} = -f_{\underline{x}}^T \Rightarrow \boxed{\Delta\underline{x} = -f_{\underline{xx}}^{-1} f_{\underline{x}}^T}$$

Sometimes Called Newton-Raphson Method.

- This is equivalent to fitting a parabola to f and computing the minimum of the parabola.

3.3.1.2 Descent Direction: Trust Region

- Solve this auxiliary constrained optimization problem:

$$\text{optimize:}_{\Delta \underline{x}} \quad \hat{f}(\Delta \underline{x}) = f(\underline{x}) + f_{\underline{x}} \Delta \underline{x} + f_{\underline{x}\underline{x}} \frac{\Delta \underline{x}^2}{2} \qquad \Delta \underline{x} \in \Re^n$$

$$\text{subject to:} \quad \underline{g}(\Delta \underline{x}) = \Delta \underline{x}^T \Delta \underline{x} \le \rho_k^2$$

Inequality constraint (stay in a circle)

- The solution is also a solution of:

$$(f_{\underline{x}\underline{x}} + \mu I)\Delta \underline{x}^* = -f_{\underline{x}}^T \qquad \mu \ge 0$$

- When objective is locally quadratic μ is small.
- Otherwise μ is large and algorithm is reduced to gradient descent.
- Trust region is adapted based on ratio of actual and predicted reduction: $\eta = \dfrac{f(\underline{x}) - f(\underline{x} + \Delta \underline{x})}{\hat{f}(\underline{0}) - \hat{f}(\Delta \underline{x})}$

Levenberg-Marquardt Algorithm

algorithm *Levenberg-Marquardt* ( )
$\underline{x} \leftarrow \underline{x}_0$ // initial guess
$\Delta\underline{x}_{max} \leftarrow const$
$\rho_{max} \leftarrow const$
$\rho \leftarrow \rho_0 \in [0, \rho_{max}]$
$\eta_{min} \leftarrow const \in [0, 1/4]$
while (**true**)
  solve Equation 3.51 for $\Delta\underline{x}$
  compute η **using** Equation 3.53
  if ($\eta < \eta_{min}$) **then** $\Delta\underline{x} \leftarrow \underline{0}$ Reject step
  $\underline{x} \leftarrow \underline{x} + \Delta\underline{x}$ // step to new point
  if($\eta < 1/4$) $\rho \leftarrow \rho/4$ // decrease trust   Reduce Trust
  else if($\eta > 3/4$ **and** $|\Delta\underline{x}| = \rho$)
    $\rho \leftarrow min(2\rho, \rho_{max})$ // increase trust   Increase Trust
  endif
  if(*finished*()) **break**
endwhile
return

Algorithm 3.2: Levenberg-Marquardt. This is a popular optimization algorithm based on the trust region technique.

Carl Friedrich Gauss

- German mathematician and scientist.
- Some say greatest mathematician in history.
- Famous for doing math in his head.
- Major contributions to number theory.
- “Proved” fundamental theorem of algebra.
- Invented method of least squares to predict orbital phenomena.

Carl Friedrich Gauss
1777 -855

3.3.1.3 Nonlinear Least Squares

- Consider nonlinear observations z of x:

$$\underline{z} = \underline{h}(\underline{x}) \quad \underline{z} \in \Re^m, \underline{x} \in \Re^n, m > n$$

Usually, not Satisfied exactly

- Define a residual and cost function:

$$\underset{\cdot}{r}(\underline{x}) = \underline{z} - \underline{h}(\underline{x})$$

$$f(\underline{x}) = \frac{1}{2}\underset{\cdot}{r}^T(\underline{x})W\underset{\cdot}{r}(\underline{x})$$

Assume a symmetric W

- The weights can come from the inverse of the covariance:

$$W = R^{-1} = Exp(\underline{z}\underline{z}^T)^{-1}$$

3.3.1.3.1 Derivatives

$$f(\underline{x}) = \frac{1}{2}\underline{r}^T(\underline{x})W\underline{r}(\underline{x})$$

- Nonlinear → must be solved by iterative methods.
- Gradient:

Row Vector $\quad f_{\underline{x}} = \underline{r}^T(\underline{x})W\underline{r}_{\underline{x}} \quad$ Jacobian Matrix

- Also:

$$\underline{r}_{\underline{x}} = -\underline{h}_{\underline{x}}$$

Tensor

- Hessian:

Matrix $\quad f_{\underline{x}\underline{x}} = \underline{r}_{\underline{x}}^T W \underline{r}_{\underline{x}} + \underline{r}_{\underline{x}\underline{x}} W \underline{r}(\underline{x})$

- Also:

$$\underline{r}_{\underline{x}\underline{x}} = -\underline{h}_{\underline{x}\underline{x}}$$

- Give these to any minimization algorithm (like Levenberg-Marquardt). Recall the Newton step:

$$\Delta\underline{x} = -f_{\underline{x}\underline{x}}^{-1} f_{\underline{x}}^T$$

3.3.1.3.2 Gauss Newton Algorithm

- From last slide: $\underline{f}_{\underline{x}\underline{x}} = \underline{r}_{\underline{x}}^T W \underline{r}_{\underline{x}} + \underline{r}_{\underline{x}\underline{x}} W \underline{r}(\underline{x})$
- Residuals are often small since they are caused solely by noise.
- In that case $\underline{r}(\underline{x})$ can be neglected to give:

Gauss Newton Approximation to The Hessian

$$f_{\underline{x}\underline{x}} \approx \underline{r}_{\underline{x}}^T W \underline{r}_{\underline{x}}$$

- This is a very cheap 2nd derivative computed from a 1st derivative (which you would need anyway).
- The Newton step becomes:

$$\Delta\underline{x} = -f_{\underline{x}\underline{x}}^{-1} f_{\underline{x}}^T = -\underline{r}_{\underline{x}}^T W \underline{r}_{\underline{x}} \underline{r}_{\underline{x}}^T W \underline{r}(\underline{x}) \qquad \textbf{Eqn A}$$

3.3.1.3.3 Rootfinding to a Minimum?

- The objective nearly vanishes at a minimum.
- Linearize observations and solve for the "root" of the gradient:

$$r_{\underline{x}} \Delta \underline{x} = -\underline{r}(\underline{x})$$

Overdetermined System

- Solve iteratively with left pseudoinverse:

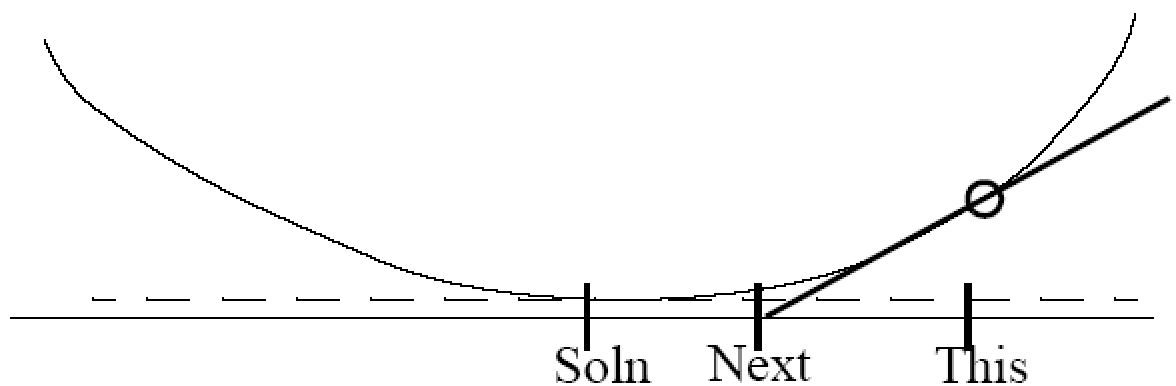
$$\Delta \underline{x} = -[r_{\underline{x}}^T r_{\underline{x}}]^{-1} r_{\underline{x}}^T r(\underline{x})$$

Same as Eqn A for W=I
This is a valid approach

- To be safe, use this as a descent direction and use line search.
- Gauss Newton nonlinear least squares is equivalent to (gradient) rootfinding for small residuals.

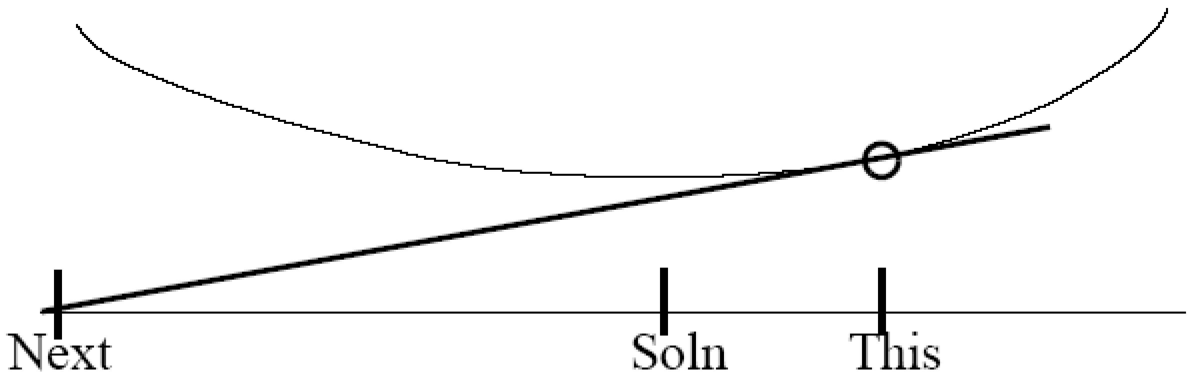
Small Residuals

- Everything is fine as long as the minimum residual is small relative to the present one.



Large Residuals

- When the present residual is close to the minimum, the slope is near zero.
  - Eventually the update actually increases the residual.



Gauss-Newton does not work for large residuals.

Outline

- 3.2 Systems of Equations
- 3.3 Nonlinear and Constrained Optimization
  - 3.3.1 Nonlinear Optimization
  - 3.3.3 Constrained Optimization
- Summary

Constrained Optimization

- Problem Statement:

$$\text{optimize:}_{\underline{x}} \quad f(\underline{x}) \qquad \underline{x} \in \Re^n$$
$$\text{subject to:} \quad \underline{c}(\underline{x}) = \underline{0} \qquad \underline{c} \in \Re^m$$

- Recall the necessary conditions:

$$f_{\underline{x}}^T + \underline{c}_{\underline{x}}^T \underline{\lambda} = \underline{0} \qquad \text{n eqns}$$
$$\underline{c}(\underline{x}) = \underline{0} \qquad \text{m eqns}$$

- These are n+m (generally nonlinear) equations in the n+m unknowns (x^*, λ^*).

Compact Necessary Conditions

- Define the Lagrangian:

$$l(\underline{x}, \underline{\lambda}) = f(\underline{x}) + \underline{\lambda}^T \underline{c}(\underline{x})$$

- Then, the necessary conditions become:

$$l_{\underline{x}}^T = \underline{0} \qquad \text{n eqns}$$

$$l_{\underline{\lambda}}^T = \underline{0} \qquad \text{m eqns}$$

- These are (the same) n+m (generally nonlinear) equations in the n+m unknowns (x^*, λ^*).

Constrained Newton Method

- Linearize of course!

$$\begin{bmatrix} \underline{l}_{\underline{x}\underline{x}} & \underline{c}_{\underline{x}}^{T} \\ \underline{c}_{\underline{x}} & 0 \end{bmatrix} \begin{bmatrix} \Delta \underline{x} \\ \Delta \underline{\lambda} \end{bmatrix} = - \begin{bmatrix} l_{\underline{x}}^{T} \\ \underline{c}(\underline{x}) \end{bmatrix}$$

- Where:

$$l_{\underline{x}} = f_{\underline{x}} + \underline{\lambda}^{T} \underline{c}_{\underline{x}}$$

$$l_{\underline{x}\underline{x}} = f_{\underline{x}\underline{x}} + \underline{\lambda}^{T} \underline{c}_{\underline{x}\underline{x}}$$

- Efficient ways to invert this matrix were covered in the math section.
- Solution gives a descent direction for line search or trust region algorithm.

Initial Lagrange Multipliers

- An initial estimate of x is doable.
- What about λ?
- One way is to solve the first (n) first order conditions for the (m) multipliers.

$$\underline{f}_{\underline{x}}^{T} + \underline{c}_{\underline{x}}^{T}\underline{\lambda} = \underline{0}$$

- They overdetermine λ so the solution is a left pseudoinverse (of $\underline{c}_{\underline{x}}^{T}$).

$$\underline{\lambda}_0 = -[\underline{c}_{\underline{x}}\underline{c}_{\underline{x}}^{T}]^{-1}\underline{c}_{\underline{x}}f_{\underline{x}}^{T}$$

Constrained Gauss-Newton

- Consider the constrained nonlinear least squares problem:

$$\text{minimize:} \quad f(\underline{x}) = \frac{1}{2}\underline{r}(\underline{x})^T \underline{r}(\underline{x})$$
$$\text{subject to:} \quad \underline{g}(\underline{x}) = \underline{b}$$

- The 1st and 2nd derivatives are:

$$l_{\underline{xx}} = f_{\underline{xx}} + \underline{\lambda}^T \underline{g}_{\underline{xx}} = \underline{r}_{\underline{x}}^T \underline{r}_{\underline{x}} + \underline{\lambda}^T \underline{g}_{\underline{xx}}$$
$$l_{\underline{x}} = f_{\underline{x}} + \underline{\lambda}^T \underline{g}_{\underline{x}} = \underline{r}^T(\underline{x})\underline{r}_{\underline{x}} + \underline{\lambda}^T \underline{g}_{\underline{x}}$$

Small residuals Assumed here

- Now go back and use the constrained Newton method on these to find a descent direction.

Penalty Function Approach

- Consider the following unconstrained problem:

$$f_k(\underline{x}) = f(\underline{x}) + \frac{1}{2} w_k \underline{c}(\underline{x})^T \underline{c}(\underline{x})$$

- Solve this for progressively increasing values of the weight w_k.
- Why do this?
  - Many constraints are soft and can be traded-off against the objective.
  - This has only n dof rather than n+m.
  - Can be used to get a good initial estimate for a constrained approach.

Outline

- 3.2 Systems of Equations
- 3.3 Nonlinear and Constrained Optimization
  - 3.3.1 Nonlinear Optimization
  - 3.3.3 Constrained Optimization
- Summary

Summary

- The inverse of a nonsquare matrix can be defined based on minimization of some suitable objective.
- The roots of nonlinear functions can be found by linearization and iteration. Newton's method converges quadratically.
- Minimization problems are very different from rootfinding.
  - Though they are easy to confuse when doing least squares.
  - Small residuals is a key assumption. Know when you are making it.

Summary

- Numerical methods for optimization either search for roots of the gradient or for local minima. Two techniques are:
  - Line search
  - Trust region
- Protected steps (line search) and backtracking are key ways to achieve robustness.