Root finding: Newton's method

- You have no doubt seen this method somewhere, but we will analyze it in a bit more depth
- We seek $f(p) = 0$ for $x = p$.
- We want to use Taylor's theorem to linearize the problem near p.
- If we Taylor expand about x near p, we obtain

$$f(p) = f(x) + \frac{f'(x)}{1!}(p - x) + \frac{f''(\xi(p))}{2!}(p - x)^2$$

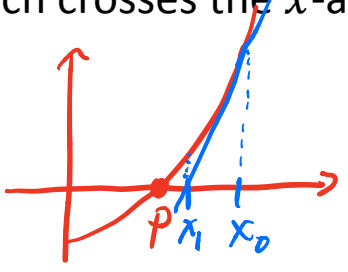
- The number $\xi(p)$ makes the formula exact.
- To solve the problem approximately, we neglect the quadratic term, which may be expected to work if $|p - x| \ll 1$

Root finding: Newton's method

- Also use $f(p) = 0$ to obtain

$$0 \approx f(x) + \frac{f'(x)}{1!}(p - x)$$

- This is the equation for a line tangent at x, which crosses the x-axis near p, but not at it (if things work right)



- Solving for p,

$$p \approx x - \frac{f(x)}{f'(x)}$$

- Because we aren't at the root, we turn this into an iteration. The x we expanded about becomes x_0; the approximate root into x_1:

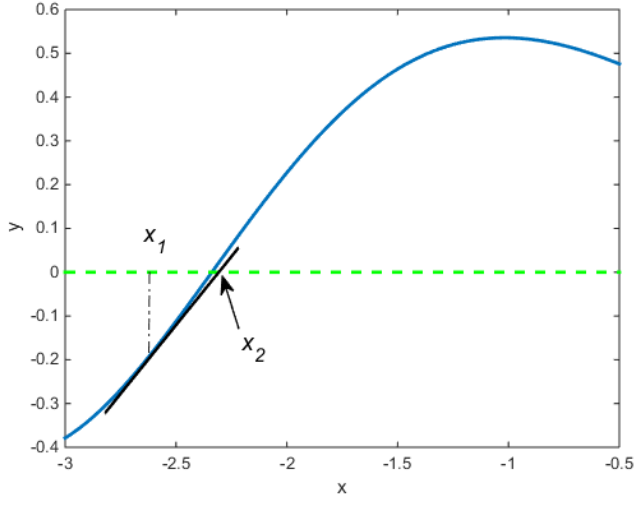
$$x_1 \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

Root finding: Newton's method

- x_1 is an approximation as well. Using it on the right side, we can generate a new approximation.
- Using x_1 as input, we compute a new $f(x_1)$ and $f'(x_1)$, and find the new approximation x_2
- We can repeat this and make it into an iteration:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0,1, \ldots$$

- Will it work? If so, how long to do this?



Newton's method: convergence analysis

- Call the root r; we study what happens to the error $e_k = r - x_k$ for n big enough
- We assume that we can make e_n as small as we like
- Using $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0,1, \ldots$
- Subtract r from both sides, and eliminate x_k:

$$e_{k+1} = e_k - \frac{f(r - e_k)}{f'(r - e_k)}, \qquad n = 0,1, \ldots$$

- The arguments of f and f' are small; Taylor expand them

$$e_{k+1} = e_k + \frac{f(r) - e_k f'(r) + \frac{1}{2} e_k^2 f''(r) + O(e_k^3)}{f'(r) - e_k f''(r) + O(e_k^2)}$$

Newton's method: convergence analysis

- Now use f(r)=0 in

$$e_{k+1} = e_k + \frac{f(r) - e_k f'(r) + \frac{1}{2}e_k^2 f''(r) + O(e_k^3)}{f'(r) - e_k f''(r) + O(e_k^2)}$$

- Then factor out f'(r); result in denominator can be written as geometric series:

$$e_{k+1} = e_k - e_k \left[1 - \frac{1}{2}\frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \left[1 - \frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right]^{-1}$$

- Multiply out the last two terms:

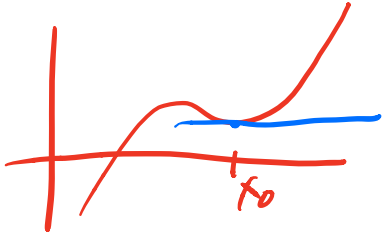
$$\begin{aligned} e_{k+1} &= e_k - e_k \left[1 - \frac{1}{2}\frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \left[1 + \frac{f''(r)}{f'(r)}e_k + O(e_k^2)\right] \\ &= -\frac{1}{2}\frac{f''(r)}{f'(r)}e_k^2 + O(e_k^3). \end{aligned}$$

Newton's method: convergence analysis

- This means $|e_{k+1}| \approx C|e_k|^2$, with the approximation getting better as k increases
- This is "quadratic convergence" or "quadratic rate of convergence"
- Number of correct digits doubles with each iteration
- If we take the log of both sides, we get $\log|e_{k+1}| \approx 2\log|e_k| + K$
- This gives us an empirical way to detect rate of convergence
- Consider two sequences:
- $a_k = 2^{-k}, k = 0,1, \ldots$ each term in the sequence is half of previous
- $b_k = 2^{-2^k}, k = 0,1, \ldots$ this time, the exponent doubles every time

Newton's method: observations and advice

- If mistakes in df/dx or if multiple roots, then rate of convergence falls to only a linear rate of convergence
- If you see linear convergence, check your functions for mistakes, or multiple roots
- The guess must be close enough to the root to avoid zero slope in the function and to converge quadratically



Newton's method for systems

- We want to turn the previous approach into something for nonlinear systems.
- We want to solve a system of the form for the x_i where

$$f_1(x_1, x_2) = 0, \qquad f_2(x_1, x_2) = 0$$

- We need both f_i to be zero at the same locations $\boldsymbol{x} = \boldsymbol{r}$, with

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \boldsymbol{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

- Let

$$\boldsymbol{F} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix}$$

$F: \mathbb{R}^2 \to \mathbb{R}^2$

- At the roots we have

$$\boldsymbol{F} = \begin{bmatrix} f_1(\boldsymbol{p}) \\ f_2(\boldsymbol{p}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \boldsymbol{0}$$

$F(x) = 0$

Newton's method for systems

- Taylor expand through the linear terms for each function
- Expand $\boldsymbol{F}$ about $\boldsymbol{x}^{(0)} = \begin{bmatrix} x_1^{(0)} & x_2^{(0)} \end{bmatrix}^T$ for $\boldsymbol{F}(\boldsymbol{p})$ with $||\boldsymbol{p} - \boldsymbol{x}^{(0)}|| \ll 1$
- Then

$$f_1(\boldsymbol{p}) = f_1\left(\boldsymbol{x}^{(0)}\right) + f_{1,x_1}\left(\boldsymbol{x}^{(0)}\right)\left(p_1 - x_1^{(0)}\right) + f_{1,x_2}\left(\boldsymbol{x}^{(0)}\right)\left(p_2 - x_2^{(0)}\right)$$

$$f_2(\boldsymbol{p}) = f_2(\boldsymbol{x}^{(0)}) + f_{2,x_1}\left(\boldsymbol{x}^{(0)}\right)\left(p_1 - x_1^{(0)}\right) + f_{2,x_2}\left(\boldsymbol{x}^{(0)}\right)\left(p_2 - x_2^{(0)}\right)$$

- We have truncate the expansion, and $f_i(\boldsymbol{p}) = 0$ by definition. Then

$$\boldsymbol{F}(\boldsymbol{p}) = \boldsymbol{0} \approx \boldsymbol{F}\left(\boldsymbol{x}^{(0)}\right) + \boldsymbol{J}(\boldsymbol{x}^{(0)})(\boldsymbol{p} - \boldsymbol{x}^{(0)})$$

- Here $\boldsymbol{J}(\boldsymbol{x}^{(0)})$ is the Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{x}^{(0)}) = \begin{bmatrix} f_{1,x_1}\left(\boldsymbol{x}^{(0)}\right) & f_{1,x_2}\left(\boldsymbol{x}^{(0)}\right) \\ f_{2,x_1}\left(\boldsymbol{x}^{(0)}\right) & f_{2,x_2}\left(\boldsymbol{x}^{(0)}\right) \end{bmatrix}$$

Newton's method for systems

- Here $\boldsymbol{J}(\boldsymbol{x}^{(0)})$ is the Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{x}^{(0)}) = \begin{bmatrix} f_{1,x_1}(\boldsymbol{x}^{(0)}) & f_{1,x_2}(\boldsymbol{x}^{(0)}) \\ f_{2,x_1}(\boldsymbol{x}^{(0)}) & f_{2,x_2}(\boldsymbol{x}^{(0)}) \end{bmatrix}$$

- The elements are the partial derivatives of the f_i with respect to x_i and evaluated at $\boldsymbol{x}^{(0)}$
- This linearized system only approximates $\boldsymbol{p}$

$$\boldsymbol{0} \approx \boldsymbol{F}(\boldsymbol{x}^{(0)}) + \boldsymbol{J}(\boldsymbol{x}^{(0)})(\boldsymbol{p} - \boldsymbol{x}^{(0)})$$

$F(p) = 0$

- Solving gives

$$\boldsymbol{p} \approx \boldsymbol{x}^{(0)} - \boldsymbol{J}^{-1}(\boldsymbol{x}^{(0)})\boldsymbol{F}(\boldsymbol{x}^{(0)})$$

- Analogous with scalar version

$$p \approx x - \frac{f(x)}{f'(x)}$$

Newton's method for systems

- Turn this into an iteration; theoretically

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{J}^{-1}(\boldsymbol{x}^{(k)})\boldsymbol{F}\left(\boldsymbol{x}^{(k)}\right), k = 0,1, \ldots$$

- Analogous with scalar version

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 0,1, \ldots$$

- The Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{x}^{(k)}) = \begin{bmatrix} f_{1,x_1}\left(\boldsymbol{x}^{(k)}\right) & f_{1,x_2}\left(\boldsymbol{x}^{(k)}\right) \\ f_{2,x_1}\left(\boldsymbol{x}^{(k)}\right) & f_{2,x_2}\left(\boldsymbol{x}^{(k)}\right) \end{bmatrix}$$

- The function $\boldsymbol{F}$ must also be updated every iteration

$$\boldsymbol{F}\left(\boldsymbol{x}^{(k)}\right) = \begin{bmatrix} f_1(\boldsymbol{x}^{(k)}) \\ f_2(\boldsymbol{x}^{(k)}) \end{bmatrix}$$

Newton's method for systems

- This is a ***theoretical*** iteration; we ***don't compute*** with this:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{J}^{-1}(\boldsymbol{x}^{(k)})\boldsymbol{F}\left(\boldsymbol{x}^{(k)}\right), k = 0,1, \ldots$$

- We instead solve the linear system and then update the iterate.
- Define

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)}$$

- Then, rewrite the top equation as

$$\boldsymbol{J}\left(\boldsymbol{x}^{(k)}\right)\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right) = -\boldsymbol{F}\left(\boldsymbol{x}^{(k)}\right)$$

- Or,

$$\boldsymbol{J}\left(\boldsymbol{x}^{(k)}\right)\left(\Delta\boldsymbol{x}^{(k)}\right) = -\boldsymbol{F}\left(\boldsymbol{x}^{(k)}\right)$$

linear system

- This system is solved each iteration, then compute the updated iterate from $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}^{(k)}$

Newton's method for systems

- So, the approach is as follows:
- Write the equations as $f_i(x_1, \dots x_n) = 0, \quad i = 1,2, \dots, n.$
- Create

$$\boldsymbol{F}(\boldsymbol{x}_k) = \begin{bmatrix} f_1(\boldsymbol{x}_k) \\ \vdots \\ f_n(\boldsymbol{x}_k) \end{bmatrix}$$

- Solve the system

$$\boldsymbol{J}(\boldsymbol{x}_k)(\Delta \boldsymbol{x}_k) = -\boldsymbol{F}(\boldsymbol{x}_k)$$

- Then compute the updated iterate from

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k$$

- Repeat until $\|\Delta \boldsymbol{x}_{k+1}\|$ and $\|\boldsymbol{F}(\boldsymbol{x}_{k+1})\|$ are small enough

Nonlinear least squares fitting

- Overdetermined *nonlinear* systems to find fits to m data pts for n parameters, with $m > n$
- Before, we sought linear combos of $f_i(t)$, finding unknown c_i for

$$f(t) = c_1 f_1(t) + \cdots + c_n f_n(t)$$

- But now, we have

$$\boldsymbol{f}(\boldsymbol{t}, \boldsymbol{y}; \boldsymbol{c}) \approx \boldsymbol{0}$$

- Now we don't have a linear function, but we put each of the data $\boldsymbol{t}, \boldsymbol{y} \in \mathbb{R}^m$ and relatively few parameters $\boldsymbol{c} \in \mathbb{R}^n$ into the function f so that we have the vector output $\boldsymbol{f} \in \mathbb{R}^m$

Nonlinear least squares fits

$F: \mathbb{R}^n \to \mathbb{R}^m$

- For convenience think of the problem as finding the parameters $\boldsymbol{x} \in \mathbb{R}^n$ in the vector function $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^m$ so that we minimize the residual:

$$\text{Find } \mathrm{x} \in \mathbb{R}^n \text{ such that } \|\mathrm{f}(\mathrm{x})\|_2 \text{ is minimized.}$$

- We can also think of this as minimizing $\boldsymbol{f}^{\boldsymbol{T}}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$
- To solve the problem, we proceed by linearization again
- Define the linearization about $\boldsymbol{x}_k$ as

$$\boldsymbol{q}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k)$$

- For convenience, define

$$\boldsymbol{h}_k = \boldsymbol{x} - \boldsymbol{x}_k, \qquad \boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{x}_k), \qquad \boldsymbol{J}_k = \boldsymbol{J}(\boldsymbol{x}_k)$$

Nonlinear least squares fits

- The problem from the linearized version of $\boldsymbol{f}$ then becomes solving for the iterate update $\boldsymbol{h}$ such that

$$\min_{\boldsymbol{h}} \|\boldsymbol{f}_k + \boldsymbol{J}_k \boldsymbol{h}\|_2$$

- This is minimized if $\boldsymbol{J}_k \boldsymbol{h} = -\boldsymbol{f}_k$
- This linear rectangular system is solved
- Each time we update via $\boldsymbol{x_{k+1}} = \boldsymbol{x}_k + \boldsymbol{h}$
- We then update $\boldsymbol{f}_k$ and $\boldsymbol{J}_k$, and solve the system again
- This is exactly what we did for Newton's method, and this time it is called Gauss-Newton iteration

Nonlinear least squares fits – Gauss-Newton

- So, here is our recipe:

1. Begin with initial guess
2. Evaluate $\boldsymbol{f}_k$ and $\boldsymbol{J}_k$
3. ~~Solve $\boldsymbol{J}_k\boldsymbol{h} = -\boldsymbol{f}_k$~~ for $\boldsymbol{h}$, $k = 1,2, \ldots$
4. Each time we update via $\boldsymbol{x_{k+1}} = \boldsymbol{x}_k + \boldsymbol{h}$
5. Repeat previous three steps until $\boldsymbol{h}$ is small enough

$\min_h \|f_k + J_k h\|_2$

- How to solve the linear least squares problem for $\boldsymbol{h}$?

Nonlinear least squares fits – Gauss-Newton

- How to solve the linear least squares problem for $\boldsymbol{h}$?
- We can use Matlab's backslash; it will find the least squares solution automatically
- Because of this, we can use `newtonsys.m`!!!
- It takes $\boldsymbol{f}$ and $\boldsymbol{J}$ as input, and returns the result of iterating on the linearized system!
- We usually have to relax the tolerances since we likely can't make the residual equal zero

Quasi-Newton methods

- In many problems, it can be difficult to implement an exact Jacobian matrix for the problem.
- We want to find an approach like the secant method where we don't need the derivatives
- For the secant method, we replaced $f'(x_k)$ with

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

- In the limit as the denominator tends to zero, we get the derivative by definition, but we don't take the limit in numerical methods
- How to use this for the Jacobian?

Quasi-Newton methods

- $\boldsymbol{J}(\boldsymbol{x}^{(0)})$ is the Jacobian matrix,

$$\boldsymbol{J}(\boldsymbol{x}^{(0)}) = \begin{bmatrix} f_{1,x_1}(\boldsymbol{x}^{(0)}) & f_{1,x_2}(\boldsymbol{x}^{(0)}) \\ f_{2,x_1}(\boldsymbol{x}^{(0)}) & f_{2,x_2}(\boldsymbol{x}^{(0)}) \end{bmatrix}$$

in the 2 by 2 case

- More generally, we can write one column of the Jacobian as at right ($\boldsymbol{e}_j$ is the j-th column of $I_{n\times n}$)
- We can use the finite difference approximation for each element in the Jacobian
- The columns are:

$$\mathbf{J}(\mathbf{x})\mathbf{e}_j = \begin{bmatrix} \frac{\partial f_1}{\partial x_j}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_j}(\mathbf{x}) \\ \vdots \\ \frac{\partial f_m}{\partial x_j}(\mathbf{x}) \end{bmatrix}$$

$$\mathbf{J}(\mathbf{x})\mathbf{e}_j \approx \frac{\mathbf{f}(\mathbf{x}+\delta\mathbf{e}_j) - \mathbf{f}(\mathbf{x})}{\delta}, \qquad j = 1, \ldots, n.$$

Quasi-Newton methods

- The approximate Jacobian

$$\mathbf{J}(\mathbf{x})\mathbf{e}_j \approx \frac{\mathbf{f}(\mathbf{x}+\delta\mathbf{e}_j)-\mathbf{f}(\mathbf{x})}{\delta}, \qquad j=1,\ldots,n.$$

- We have the function values $\boldsymbol{f}(\boldsymbol{x})$ already, but we need to evaluate the first term $\boldsymbol{f}(\boldsymbol{x}+\delta\boldsymbol{e}_j)$ to get the Jacobian
- If we expect a noise level of ϵ, then pick $\delta = \sqrt{\epsilon}$
- If only roundoff is around to pollute the computation, then $\delta = \sqrt{\epsilon_M}$ where ϵ_M=`eps`
- Call approximated Jacobian $\tilde{\boldsymbol{J}}(\boldsymbol{x})$

Quasi-Newton methods

- We could use the Newton method for system with the approximate Jacobian
- Solve the system

$$\tilde{\boldsymbol{J}}(\boldsymbol{x}_k)(\Delta \boldsymbol{x}_k) = -\boldsymbol{F}(\boldsymbol{x}_k)$$

- Then compute the updated iterate from

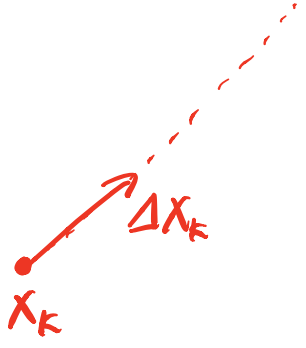
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k$$

- Repeat until $\|\Delta \boldsymbol{x}_{k+1}\|$ and $\|\boldsymbol{F}(\boldsymbol{x}_{k+1})\|$ are small enough
- But there are additional factors to consider
- Sometimes (often?) the Newton's method gets more sensitive and additional fixes needed

Quasi-Newton methods

- One fix is using *damped iteration* or *line search*
- Instead of the Newton update

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\Delta\boldsymbol{x}_k$$



- We can use

$$\boldsymbol{p}(t)=\boldsymbol{x}_k+t\Delta\boldsymbol{x}_k$$

- For damped Newton iteration, it is often the case that a fixed 0<t<1 is used in the update, and the result is called $\boldsymbol{x}_{k+1}$
- We can also vary t and find a value that results in

$$\|\mathbf{f}(\mathbf{p}(t))\| < \|\mathbf{f}(\mathbf{x}_k)\|$$

- This is line search

Quasi-Newton methods

- We can use

$$\boldsymbol{p}(t)=\boldsymbol{x}_k + t\Delta\boldsymbol{x}_k$$

- Use $t = 1$ if it results in $\|\mathbf{f}(\mathbf{p}(t))\| < \|\mathbf{f}(\mathbf{x}_k)\|$
- If it the norm of the residual doesn't decrease, cut t in half and check again.
- Repeat until $\|\mathbf{f}(\mathbf{p}(t))\| < \|\mathbf{f}(\mathbf{x}_k)\|$ is satisfied or until t is too small and the method fails

Quasi-Newton: Levenberg's method

- It may also be advantageous to adjust the iteration process by modifying the linear solves and update direction
- This was the Newton's method equation for the update:

$$\tilde{\boldsymbol{J}}(\boldsymbol{x}_k)(\Delta \boldsymbol{x}_k) = -\boldsymbol{F}(\boldsymbol{x}_k)$$

- A different way to try decrease $\|\boldsymbol{f}(\boldsymbol{x})\|_2$ is to use steepest descent
- In this approach , consider $r = (\|\boldsymbol{f}(\boldsymbol{x})\|_2)^2 = \boldsymbol{f}^{\mathrm{T}}\boldsymbol{f}$
- Recall that from calculus, that the negative of the gradient is the direction of steepest descent of a function
- Then, $\nabla r(\boldsymbol{x}) = \nabla\left(\boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})\right) = 2\boldsymbol{J}^{T}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$
- We could try to figure out a step from $\boldsymbol{v}(\boldsymbol{x}) = -s\boldsymbol{J}^{T}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$ where s is a scalar that must be found (like t in line search)

Quasi-Newton: Levenberg's method

- The steepest descent method can slow down depending on the direction's relation to the minimum
- Newton's method may be difficult to start but converges quickly near the answer
- Get the best of both by combining them: Levenberg
- In this method, one solves:

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})\boldsymbol{v} = -\boldsymbol{J}^T\boldsymbol{f}$$

- For $\lambda = 0$, we get back to Newton's method
- For $\lambda \to \infty$, we get close to steepest descent
- We implement a method that varies λ so that it starts large and gets reduced as needed

Quasi-Newton methods

- Levenberg's method (pub'd 1944) was rediscovered and improved a little bit by Donald Marquardt in 1963 while working at DuPont (Wikipdedia names three others that rediscovered it in '58 to '60)
- Levenberg-Marquardt method:

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda \mathrm{diag}(\boldsymbol{J}^T\boldsymbol{J}))\boldsymbol{v} = -\boldsymbol{J}^T\boldsymbol{f}$$

- The improvement works a bit better at large λ for some problems
- We implement a method that varies λ so that it starts large and gets reduced as needed
- This method can be chosen as an option in lsqnonlin in Matlab (optimization toolbox)