

# Numerical Optimization

## Lecture Notes #6

### Line Search Methods: Step Length Selection

Peter Blomgren,  
(blomgren.peter@gmail.com)

Department of Mathematics and Statistics  
Dynamical Systems Group  
Computational Sciences Research Center  
San Diego State University  
San Diego, CA 92182-7720

<http://terminus.sdsu.edu/>

Fall 2018



## Outline

- 1 Line Search Methods
  - Recap
  - Step Length Selection
- 2 Step Length Selection
  - Interpolation
  - The Initial Step
  - Line Search Satisfying the Strong Wolfe Conditions
- 3 Homework #2
  - Homework #2 — Help & Hints

## Quick Recap: Last Time

Rates of convergence for our different optimization strategies.

We showed that for a simple quadratic model

$f(\bar{\mathbf{x}}) = \frac{1}{2}\bar{\mathbf{x}}^T Q \bar{\mathbf{x}} - \bar{\mathbf{b}}^T \bar{\mathbf{x}}$  the *steepest descent method* is indeed **linearly convergent**.

The result generalizes to general nonlinear objective functions for which  $\nabla f(\bar{\mathbf{x}}^*) = 0$  and  $\nabla^2 f(\bar{\mathbf{x}}^*)$  is positive definite.

We stated the result for **Newton's method** which says that it is locally **quadratically convergent**.

## Quick Recap: Last Time

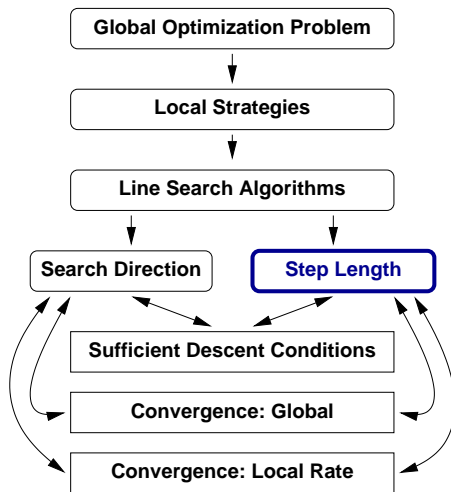
Further, **Quasi-Newton methods**, where the search direction is  $\bar{\mathbf{p}}_k^{\text{QN}} = -B_k^{-1} \nabla f(\bar{\mathbf{x}}_k)$ , exhibit **super-linear convergence** as long as the matrix sequence  $\{B_k\}$  converges to the Hessian  $\nabla^2 f(\bar{\mathbf{x}}^*)$  in the search direction  $\bar{\mathbf{p}}_k$ :

$$\lim_{k \rightarrow \infty} \frac{\|(B_k - \nabla^2 f(\bar{\mathbf{x}}^*))\bar{\mathbf{p}}_k\|}{\|\bar{\mathbf{p}}_k\|} = 0.$$

**Coordinate Descent Methods:** Slower than Steepest descent. Useful if coordinates are decoupled and/or computation of the gradient is not possible or too expensive. — We can potentially leverage multi-threaded computations.



## Unconstrained Optimization — In the Line Search “Universe”



## Lookahead: This Time — A Closer Look at Step Length

We look at techniques for

Best: Finding a minimizer to the 1D-function

$$\Phi(\alpha) = f(\bar{\mathbf{x}}_k + \alpha \bar{\mathbf{p}}_k)$$

OK: Finding a step length  $\alpha_k$  which satisfy a “sufficient decrease condition” such as the Wolfe conditions.

We already have one such algorithm —

## Algorithm: Backtracking Line-search

May Not Satisfy 2nd Wolfe Condition

- [1] Set  $\bar{\alpha} > 0$ ,  $\rho \in (0, 1)$ ,  $c \in (0, 1)$ , set  $\alpha = \bar{\alpha}$
- [2] While  $f(\bar{\mathbf{x}}_k + \alpha \bar{\mathbf{p}}_k) > f(\bar{\mathbf{x}}_k) + c\alpha \bar{\mathbf{p}}_k^T \nabla f(\bar{\mathbf{x}}_k)$
- [3]      $\alpha = \rho\alpha$
- [4] End-While
- [5] Set  $\alpha_k = \alpha$



## Step Length Selection: Assumptions

We must assume that  $\bar{\mathbf{p}}_k$  is a descent direction, *i.e.* that  $\Phi'(0) < 0$  — thus all our steps will be in the positive direction.

When the objective  $f$  is quadratic  $f(\bar{\mathbf{x}}) = \frac{1}{2}\bar{\mathbf{x}}^T Q \bar{\mathbf{x}} + \bar{\mathbf{b}}^T \bar{\mathbf{x}} + c$ , the optimal step can be found explicitly

$$\alpha_k = -\frac{\nabla f(\bar{\mathbf{x}}_k)^T \bar{\mathbf{p}}_k}{\bar{\mathbf{p}}_k^T Q \bar{\mathbf{p}}_k}.$$



For general nonlinear  $f$  we must use an **iterative scheme** to find the step length  $\alpha_k$ .

How the line search is performed impacts the **robustness** and **efficiency** of the overall optimization method.

## Step Length Selection: Classification

It is natural to classify **line search methods** based on how many derivatives they need:

- 0 Methods based on **function values only** tend to be inefficient, since they need to narrow the minimizer to a small interval.
- 1 Gradient information makes it easier to determine if a certain step is good — *i.e.* it satisfies a sufficient reduction condition.
- >1 Methods requiring more than one derivative are quite rare; in order to compute the second derivative the full Hessian  $\nabla^2 f(\bar{\mathbf{x}}_k)$  is needed, this is usually too high a cost.





## Step Length Selection: Our Focus

The best “bang-for-bucks” line search algorithms use the gradient information, hence those will be the focus of our discussion. A line search algorithm roughly breaks down into the following components:

- [1] The initial step length  $\alpha_0$  is selected.
- [2] An interval  $[\alpha_{min}, \alpha_{max}]$  containing acceptable step lengths is identified — **Bracketing phase**.
- [3] The final step length is selected from the acceptable set — **Selection phase**.

Often, [2] and [3] are closely tied together.



## Step Length Selection: Interpolation

1 of 7

First we note that the **Armijo condition** can be written in terms of  $\Phi$  as

$$\Phi(\alpha_k) \leq \Phi(0) + c_1 \alpha_k \Phi'(0),$$

where  $c_1 \sim 10^{-4}$  in practice. This is stronger (but not much stronger) than requiring descent.

⇒ Our new algorithms will be efficient in the sense that the gradient  $\nabla f(\bar{\mathbf{x}}_k)$  is computed **as few times as possible**.

**If the initial step length  $\alpha_0$  satisfies the Armijo condition, then we accept  $\alpha_0$  as the step length and terminate the search.**

— **As we get close to the solution this will happen more and more often (for Newton and quasi-Newton methods with  $\alpha_0 = 1$ .)**

Otherwise, we search for an acceptable step length in  $[0, \alpha_0]$ ...



## Step Length Selection: Interpolation (Quadratic–Cubic)

At this stage we have computed 3 pieces of information:

$$\Phi(0), \Phi'(0), \text{ and } \Phi(\alpha_0),$$

we can use this information to build a **quadratic model**  $\Phi_q(\alpha)$ :

$$\Phi_q(\alpha) = \left[ \frac{\Phi(\alpha_0) - \Phi(0) - \alpha_0 \Phi'(0)}{\alpha_0^2} \right] \alpha^2 + \Phi'(0)\alpha + \Phi(0).$$

Note

$$\Phi_q(0) = \Phi(0), \quad \Phi_q(\alpha_0) = \Phi(\alpha_0), \quad \Phi'_q(0) = \Phi'(0).$$

We set  $\Phi'_q(\alpha) = 0$  to find the minimum of the model — our next  $\alpha$  to try...

$$\Phi'_q(\alpha) = 2\alpha \left[ \frac{\Phi(\alpha_0) - \Phi(0) - \alpha_0 \Phi'(0)}{\alpha_0^2} \right] + \Phi'(0) = 0.$$

## Step Length Selection: Interpolation (Quadratic–Cubic)

3 of 7

Hence

$$\alpha_1 = -\frac{\alpha_0^2 \Phi'(0)}{2[\Phi(\alpha_0) - \Phi(0) - \alpha_0 \Phi'(0)]}.$$

We now **check the Armijo condition**

$$\Phi(\alpha_1) \leq \Phi(0) + c_1 \alpha_1 \Phi'(0).$$

If it fails, then we create a **cubic** function

$$\Phi_c(\alpha) = \mathbf{a}\alpha^3 + \mathbf{b}\alpha^2 + \alpha\Phi'(0) + \Phi(0),$$

which interpolates

$$\Phi(0), \Phi'(0), \Phi(\alpha_0), \text{ and } \Phi(\alpha_1).$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \frac{1}{\alpha_0^2 \alpha_1^2 (\alpha_1 - \alpha_0)} \begin{bmatrix} \alpha_0^2 & -\alpha_1^2 \\ -\alpha_0^3 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \Phi(\alpha_1) - \Phi(0) - \alpha_1 \Phi'(0) \\ \Phi(\alpha_0) - \Phi(0) - \alpha_0 \Phi'(0) \end{bmatrix}$$



## Step Length Selection: Interpolation (Quadratic–Cubic)

4 of 7

The next iterate ( $\alpha_2$ ) is now the minimizer of  $\Phi_c(\alpha)$  which lies in  $[0, \alpha_1]$ , it is given as one of the roots of the quadratic equation

$$\Phi'_c(\alpha) = 3a\alpha^2 + 2b\alpha + \Phi'(0) = 0,$$

it is...

$$\alpha_2 = \frac{-b + \sqrt{b^2 - 3a\Phi'(0)}}{3a}.$$

In the extremely rare cases that  $\alpha_2$  does not satisfy the Armijo condition

$$\Phi(\alpha_2) \leq \Phi(0) + c_1\alpha_2\Phi'(0),$$

we create a new cubic model interpolating

$$\Phi(0), \Phi'(0), \Phi(\alpha_1), \text{ and } \Phi(\alpha_2)$$

*i.e.*  $\Phi(0)$ ,  $\Phi'(0)$  and the two most recent  $\alpha$ 's.

## Step Length Selection: Interpolation (Cubic–Hermite Based)

5 of 7

At this point we must introduce to following **safeguards** to guarantee that we make sufficient progress:

**If**  $|\alpha_{k+1} - \alpha_k| < \epsilon_1$  **or**  $|\alpha_{k+1}| < \epsilon_2$   
**then**  $\alpha_{k+1} = \alpha_k/2$ .

The algorithm described **assumes** that computing the derivative is significantly more expensive than computing function values.

However it is often, but not always, possible to compute the directional derivative (or a good estimate thereof) with minimal extra cost.

In those cases we build the cubic interpolant so that it interpolates

$$\Phi(\alpha_k), \Phi'(\alpha_k), \Phi(\alpha_{k-1}), \text{ and } \Phi'(\alpha_{k-1})$$

this is a **Hermite Polynomial of degree 3** (see Math 541 [R.I.P.]<sub>1</sub>)



## Step Length Selection: Interpolation

6 of 7

The cubic Hermite polynomial satisfying

$$\begin{aligned}H_3(\alpha_{k-1}) &= \Phi(\alpha_{k-1}), & H'_3(\alpha_{k-1}) &= \Phi'(\alpha_{k-1}) \\H_3(\alpha_k) &= \Phi(\alpha_k), & H'_3(\alpha_k) &= \Phi'(\alpha_k).\end{aligned}$$

can be written explicitly as

$$\begin{aligned}H_3(\alpha) &= \left[1 + \frac{\alpha - \alpha_{k-1}}{\alpha_k - \alpha_{k-1}}\right] \left[\frac{\alpha_k - \alpha}{\alpha_k - \alpha_{k-1}}\right]^2 \Phi(\alpha_{k-1}) \\&+ \left[1 + 2\frac{\alpha_k - \alpha}{\alpha_k - \alpha_{k-1}}\right] \left[\frac{\alpha - \alpha_{k-1}}{\alpha_k - \alpha_{k-1}}\right]^2 \Phi(\alpha_k) \\&+ (\alpha - \alpha_{k-1}) \left[\frac{\alpha_k - \alpha}{\alpha_k - \alpha_{k-1}}\right]^2 \Phi'(\alpha_{k-1}) \\&+ (\alpha - \alpha_k) \left[\frac{\alpha - \alpha_{k-1}}{\alpha_k - \alpha_{k-1}}\right]^2 \Phi'(\alpha_k).\end{aligned}$$



Charles Hermite,  
French mathematician  
(1822–1901).  
© Public Domain.

(Straight from Math 541 [R.I.P.]...)

## Step Length Selection: Interpolation

7 of 7

The minimizer of  $H_3(\alpha)$  in  $[\alpha_{k-1}, \alpha_k]$  is either at one of the end points, or else in the interior (given by setting  $H'_3(\alpha) = 0$ ).

The interior point is

$$\alpha_{k+1} = \alpha_k - (\alpha_k - \alpha_{k-1}) \left[ \frac{\Phi'(\alpha_k) + d_2 - d_1}{\Phi'(\alpha_k) - \Phi'(\alpha_{k-1}) + 2d_2} \right]$$

where

$$d_1 = \Phi'(\alpha_{k-1}) + \Phi'(\alpha_k) - 3 \left[ \frac{\Phi(\alpha_{k-1}) - \Phi(\alpha_k)}{\alpha_{k-1} - \alpha_k} \right]$$
$$d_2 = \text{sign}(\alpha_k - \alpha_{k-1}) \sqrt{d_1^2 - \Phi'(\alpha_{k-1})\Phi'(\alpha_k)}$$

Either  $\alpha_{k+1}$  is accepted as the step length, or the search process continues...

Cubic interpolation gives **quadratic convergence** in the step length selection algorithm.





## Step Length Selection: The Initial Step

1 of 2

For Newton and quasi-Newton methods, the search vector  $\bar{\mathbf{p}}_k$  contains an intrinsic sense of **scale** (being formed from the local descent, and curvature information), hence the initial trial step length should always be  $\alpha_0 = 1$ , otherwise we break the quadratic respective super-linear convergence properties.

For other search directions, such as **steepest descent** and **conjugate gradient** (to be described later) directions which do not have a sense of scale, other methods must be used to select a good first trial step:

**Strategy #1:** Assume that the rate of change in the current iteration will be the same as in the previous iteration, select  $\alpha_0$ :

$$\alpha_0^{[k]} = \alpha^{[k-1]} \frac{\bar{\mathbf{p}}_{k-1}^T \nabla f(\bar{\mathbf{x}}_{k-1})}{\bar{\mathbf{p}}_k^T \nabla f(\bar{\mathbf{x}}_k)}.$$

## Step Length Selection: The Initial Step

2 of 2

**Strategy #2:** Use the minimizer of the quadratic interpolant to  $f(\bar{\mathbf{x}}_{k-1})$ ,  $f(\bar{\mathbf{x}}_k)$ , and  $\phi'(0) = \bar{\mathbf{p}}_k^T \nabla f(\bar{\mathbf{x}}_k)$  as the initial  $\alpha$ :

$$\alpha_0^{[k]} = \frac{2[f(\bar{\mathbf{x}}_k) - f(\bar{\mathbf{x}}_{k-1})]}{\bar{\mathbf{p}}_k^T \nabla f(\bar{\mathbf{x}}_k)}$$

If this strategy is used with a quadratically or super-linearly convergent algorithm, the choice of  $\alpha_0$  must be modified slightly to preserve the convergence properties:

$$\alpha_{0,\text{new}}^{[k]} = \min(1, 1.01\alpha_0^{[k]})$$

this ensures that the step length  $\alpha_0 = 1$  will eventually always be tried.



## Line Search for the Strong Wolfe Conditions

1 of 6

## Algorithm: LS/Strong Wolfe Conditions

```
01. Set  $\alpha_0 = 0$ , choose  $\alpha_1 > 0$ ,  $\alpha_{\max}$ ,  $c_1$ , and  $c_2$ ,  $i = 1$ 
02. while( TRUE )
03.   Compute  $\Phi(\alpha_i)$ 
04.   if  $(\Phi(\alpha_i) > \Phi(0) + c_1\alpha_i\Phi'(0))$  or  $(\Phi(\alpha_i) \geq \Phi(\alpha_{i-1})$  and  $i > 1)$ 
05.      $\alpha_* = \text{zoom}(\alpha_{i-1}, \alpha_i)$ , and terminate search
05.5  end::if-04
06.   Compute  $\Phi'(\alpha_i)$ 
07.   if  $|\Phi'(\alpha_i)| \leq -c_2\Phi'(0)$ 
08.      $\alpha_* = \alpha_i$ , and terminate search
08.5  end::if-07
09.   if  $\Phi'(\alpha_i) \geq 0$ 
10.      $\alpha_* = \text{zoom}(\alpha_i, \alpha_{i-1})$ , and terminate search
10.5  end::if-09
11.   Choose  $\alpha_{i+1} \in [\alpha_i, \alpha_{\max}]$ 
12.    $i = i + 1$ 
13. end::while
```



## Line Search for the Strong Wolfe Conditions

2 of 6

In the **first stage** of the algorithm, either an acceptable step length, or a range  $[\alpha_i, \alpha_{i+1}]$  containing an acceptable step length is identified — none of the conditions 04, 07, 09 are satisfied so the step length is increased 11.

If in the first stage we identified a range, the **second stage** invokes a function `zoom` which will identify an acceptable step from the interval.

**Note:** 04 establishes that  $\alpha_i$  is too long a step, thus  $\alpha_*$  must be in the range  $[\alpha_{i-1}, \alpha_i]$ .

**Note:** if 07 holds, then both the strong Wolfe conditions hold (since `not(04)` must also hold.)

**Note:** Finally, if 09 holds then the step is too large (since we are going uphill at this point.)



## Line Search for the Strong Wolfe Conditions

3 of 6

The zoom function takes two arguments:  $\text{zoom}(\alpha_{\text{low}}, \alpha_{\text{high}})$  satisfying the following:

- [1] The interval bounded by  $\alpha_{\text{low}}$ , and  $\alpha_{\text{high}}$  contains step lengths which satisfy the strong Wolfe conditions.
- [2]  $\alpha_{\text{low}}$  is the  $\alpha$  corresponding to the lower **function value**, *i.e.*  $\Phi(\alpha_{\text{low}}) < \Phi(\alpha_{\text{high}})$ .
- [3]  $\alpha_{\text{low}}$  and  $\alpha_{\text{high}}$  satisfy:  $\Phi'(\alpha_{\text{low}})(\alpha_{\text{high}} - \alpha_{\text{low}}) < 0$ .

See the figure on slide 23.

## Line Search for the Strong Wolfe Conditions

4 of 6

## Algorithm: zoom function

```
01. while( TRUE )
02.   Interpolate to find  $\alpha_j$  between  $\alpha_{low}$  and  $\alpha_{high}$ 
03.   Compute  $\Phi(\alpha_j)$ 
04.   if  $(\Phi(\alpha_j) > \Phi(0) + c_1\alpha_j\Phi'(0))$  or  $(\Phi(\alpha_j) \geq \Phi(\alpha_{low}))$ 
05.      $\alpha_{high} = \alpha_j$ 
06.   else
07.     Compute  $\Phi'(\alpha_j)$ 
08.     if  $|\Phi'(\alpha_j)| \leq -c_2\Phi'(0)$ 
09.        $\alpha_* = \alpha_j$ , and return( $\alpha_*$ )
09.5    end::if-08
10.    if  $\Phi'(\alpha_j)(\alpha_{high} - \alpha_{low}) \geq 0$ 
11.       $\alpha_{high} = \alpha_{low}$ 
11.5    end::if-10
12.     $\alpha_{low} = \alpha_j$ 
12.5  end::if-else-04-06
13.  end::while
```

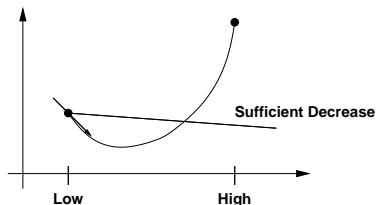


## Line Search for the Strong Wolfe Conditions

5 of 6

In practical applications ( $c_1 = 10^{-4}$  and  $c_2 = 0.9$ ), enforcing strong Wolfe conditions require a similar amount of work compared with the Wolfe conditions.

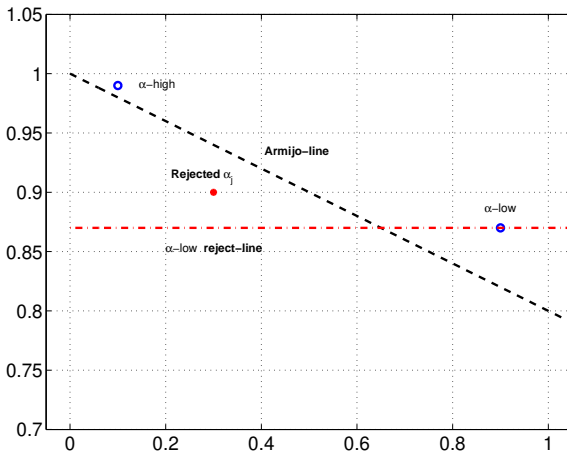
The advantage of the strong conditions is that by decreasing  $c_2$  we can force the accepted step lengths to be closer to the local minima of  $\Phi(\cdot)$ , this is particularly helpful in applications of steepest descent or conjugate gradient methods.



**Figure:** A possible scenario for zoom — since  $\alpha_{\text{low}} < \alpha_{\text{high}}$  we must have negative slope at  $\alpha_{\text{low}}$ .

# Line Search for the Strong Wolfe Conditions

6 of 6



**Figure:** Illustrating the 04-condition; since we know we can push the objective down to  $\Phi(\alpha_{low})$ , we reject  $\alpha_j$  even though it satisfies the Armijo condition.



## Homework #2 — Due at 12:00pm, Friday October 5, 2018

Re-do Homework #1, replacing the backtracking line search with the algorithm discussed in this lecture.

Do not forget the safe-guards.

Note that (some of) the interpolation formulas are anchored at 0 on the left; but neither  $\alpha_{\text{low}}$  nor  $\alpha_{\text{high}}$  is guaranteed to be 0.

**Compare** the performance for both the Newton and Steepest Descent algorithms; is there a significant difference?

Help and hints on the next slide...

## Homework #2 — Help &amp; Hints

- Modularize your code — Have separate `zoom`, and `interpolate` functions, and a “driver” which directs “traffic.”
- Implement `zoom` first. Debug using a simple version of `interpolate(a_low, a_high) = (a_low + a_high) / 2`.
- Once `zoom` works, replace the interpolation step by *either*
  - [easier] Hermite-based cubic interpolation
  - [harder] Quadratic-Cubic interpolation
    - In order to debug the interpolation, it is useful to plot the interpolation function in the  $(a_{low}, a_{high})$  interval, and verify that the value selected for the next  $\alpha$  indeed corresponds to the minimum.



## Functions :: symbolic, “anonymous”

```
1  %(Rosenbrock Function) :: usage f(1.2,1.3)
2  f = @(x,y) 100*(y-x.^2).^2+(1-x).^2;
3
4  %(Use symbolic toolbox to compute derivatives)
5  syms x y
6  df_dx = diff(f(x,y),x)
7  df_dy = diff(f(x,y),y)
8  df_dxx = diff(f(x,y),x,x)
9  df_dxy = diff(f(x,y),x,y)
10 df_dyy = diff(f(x,y),y,y)
11
12 %(Make "callable" non-symbolic functions)
13 f_dx = matlabFunction(df_dx, 'Vars',[x y])
14 f_dy = matlabFunction(df_dy, 'Vars',[x y])
15 f_dxx = matlabFunction(df_dxx, 'Vars',[x y])
16 f_dxy = matlabFunction(df_dxy, 'Vars',[x y])
17 f_dyy = matlabFunction(df_dyy, 'Vars',[x y])
18
19 %(Gradient and Hessian functions)
20 f_grad = @(x,y) [f_dx(x,y) ; f_dy(x,y)]
21 f_hess = @(x,y) [f_dxx(x,y) f_dxy(x,y) ; f_dxy(x,y) f_dyy(x,y)]
22
23 %(Function, gradient, and hessian with vector arguments)
24 vf      = @(x) f(x(1),x(2))
25 vf_grad = @(x) f_grad(x(1),x(2))
26 vf_hess = @(x) f_hess(x(1),x(2))
```



## Functions :: symbolic, “anonymous”

```
28  %(Steepest descent, and Newton directions)
29  sd = @(x) -vf_grad(x)/norm(vf_grad(x))
30  nd = @(x) -vf_hess(x)\vf_grad(x)
31
32  %Linear model --- Notice :: functions as arguments!
33  lmod = @(a,pk,xk,vf,vf_grad) vf(xk) + a*pk'*vf_grad(xk)
34
35  %Quadratic model
36  qmod = @(a,pk,xk,vf,vf_grad,vf_hess) ...
37         vf(xk) + a*pk'*vf_grad(xk) + 1/2*a^2*pk'*vf_hess(xk)*pk
38
39  %(Armijo condition check)
40  armijo = @(a,c1,xk,pk,f,vf_grad) ...
41          (f(xk+a*pk) <= f(xk) + c1*a*pk'*vf_grad(xk))
42
43  c1    = 10^(-4)
44  x0    = [1.2 ; 1.2]
45  sd0   = sd(x0)
46  nd0   = nd(x0)
47  alpha = 1
```



## Functions :: symbolic, “anonymous”

```
49 if armijo(alpha,c1,x0,sd0,vf,vf_grad)
50     fprintf('SD can take full step from x0 = [%g,%g]\n',...
51           x0(1),x0(2))
52 else
53     fprintf('SD can NOT take full step from x0 = [%g,%g]\n',...
54           x0(1),x0(2))
55 end
56
57 if armijo(alpha,c1,x0,nd0,vf,vf_grad)
58     fprintf('Newton can take full step from x0 = [%g,%g]\n',...
59           x0(1),x0(2))
60 else
61     fprintf('Newton can NOT take full step from x0 = [%g,%g]\n',...
62           x0(1),x0(2))
63 end
```



# Index

- algorithm
  - line search
    - "zoom" function, 22
    - strong Wolfe conditions, 19
- roadmap
  - unconstrained optimization, 5
- step length
  - interpolation
    - safeguards, 14
    - strategies, 10
  - selection
    - classification, 8
    - cubic model, 12
    - quadratic model, 11
- unconstrained optimization
  - roadmap, 5