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

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**.

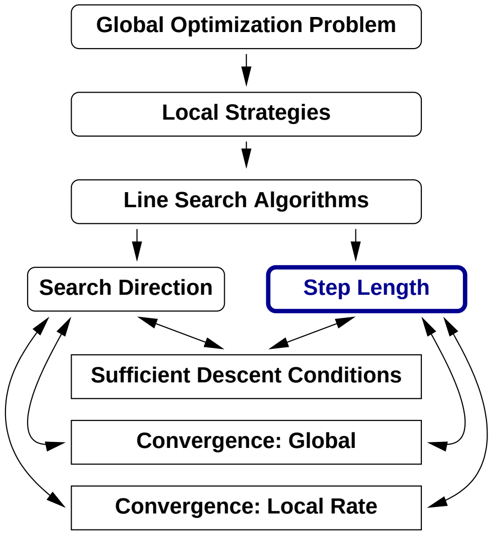
Quick Recap: Last Time

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 $\bar{\mathbf{p}}_k$:

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

Coordinate Descent Methods: Slower than Steepest descent. Useful of 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 α_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 $\overline{\alpha} > 0$, $\rho \in (0,1)$, $c \in (0,1)$, set $\alpha = \overline{\alpha}$
[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] $\quad \alpha = \rho\alpha$
[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 α_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 derivate 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 α_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

First we note that the **Armijo condition** can be written in terms of Φ 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) that requiring descent.

$\Rightarrow$ 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 α_0 satisfies the Armijo condition, then we accept α_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 α 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)

Hence

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

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)

The next iterate (α_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 α_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 α's.

Step Length Selection: Interpolation (Cubic – Hermite Based)

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.].)

Step Length Selection: Interpolation

The cubic Hermite polynomial satisfying

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

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

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

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

Either α_{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

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 α_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

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 α:

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

If this strategy is used with a quadratically or super-linearly convergent algorithm, the choice of α_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

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. $\alpha_* = \textbf{zoom}(\alpha_{i-1}, \alpha_i)$, 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. $\alpha_* = \textbf{zoom}(\alpha_i, \alpha_{i-1})$, 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

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 α_i is too long a step, thus α_* 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

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

[1] The interval bounded by α_{low}, and α_{high} contains step lengths which satisfy the strong Wolfe conditions.

[2] α_{low} is the α corresponding to the lower **function value**, *i.e.* $\Phi(\alpha_{\text{low}}) < \Phi(\alpha_{\text{high}})$.

[3] α_{low} and α_{high} satisfy: $\Phi'(\alpha_{\text{low}})(\alpha_{\text{high}} - \alpha_{\text{low}}) < 0$.

See the figure on slide 23.

SAN DIEGO STATE UNIVERSITY

Line Search for the Strong Wolfe Conditions

Algorithm: zoom function

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

SAN DIEGO STATE UNIVERSITY

Line Search for the Strong Wolfe Conditions

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.

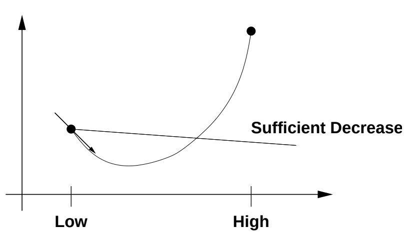


Figure: A possible scenario for zoom — since $\alpha_{\text{low}} < \alpha_{\text{high}}$ we must have negative slope at α_{low}.

SAN DIEGO STATE UNIVERSITY

Line Search for the Strong Wolfe Conditions

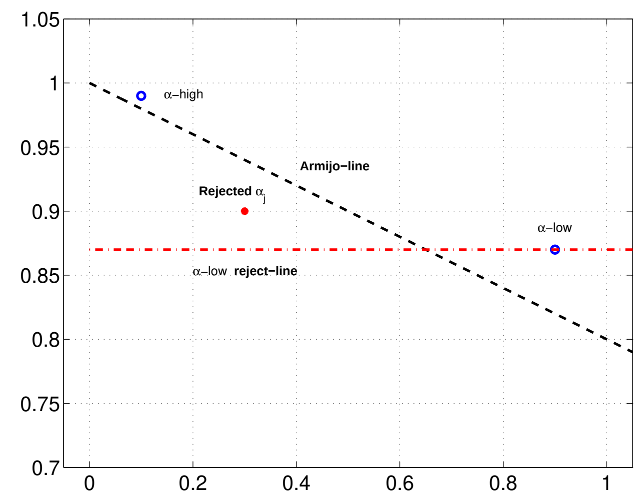


Figure: Illustrating the 04-condition; since we know we can push the objective down to $\Phi(\alpha_{\text{low}})$, we reject α_j even though it satisfies the Armijo condition.

SAN DIEGO STATE UNIVERSITY

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 α_{low} nor α_{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 & 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(alow,ahigh) = (alow+ahigh)/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 (alow,ahigh) interval, and verify that the value selected for the next alpha indeed corresponds to the minimum.

Functions :: symbolic, "anonymous"

```
%(Rosenbrock Function) :: usage f(1.2,1.3)
f = @(x,y) 100*(y-x.^2).^2+(1-x).^2;

%(Use symbolic toolbox to compute derivatives)
syms x y
df_dx  = diff(f(x,y),x)
df_dy  = diff(f(x,y),y)
df_dxx = diff(f(x,y),x,x)
df_dxy = diff(f(x,y),x,y)
df_dyy = diff(f(x,y),y,y)

%(Make "callable" non-symbolic functions)
f_dx  = matlabFunction(df_dx, 'Vars',[x y])
f_dy  = matlabFunction(df_dy, 'Vars',[x y])
f_dxx = matlabFunction(df_dxx,'Vars',[x y])
f_dxy = matlabFunction(df_dxy,'Vars',[x y])
f_dyy = matlabFunction(df_dyy,'Vars',[x y])

%(Gradient and Hessian functions)
f_grad = @(x,y) [f_dx(x,y) ; f_dy(x,y)]
f_hess = @(x,y) [f_dxx(x,y) f_dxy(x,y) ; f_dxy(x,y) f_dyy(x,y)]

%(Function, gradient, and hessian with vector arguments)
vf      = @(x) f(x(1),x(2))
vf_grad = @(x) f_grad(x(1),x(2))
vf_hess = @(x) f_hess(x(1),x(2))
```


Functions :: symbolic, "anonymous"

```
%(Steepest descent, and Newton directions)
sd = @(x) -vf_grad(x)/norm(vf_grad(x))
nd = @(x) -vf_hess(x)\vf_grad(x)

%Linear model --- Notice :: functions as arguments!
lmod = @(a,pk,xk,vf,vf_grad) vf(xk) + a*pk'*vf_grad(xk)

%Quadradic model
qmod = @(a,pk,xk,vf,vf_grad,vf_hess) ...
       vf(xk) + a*pk'*vf_grad(xk) + 1/2*a^2*pk'*vf_hess(xk)*pk

%(Armijo condition check)
armijo = @(a,c1,xk,pk,f,vf_grad) ...
         (f(xk+a*pk) <= f(xk) + c1*a*pk'*vf_grad(xk))

c1    = 10^(-4)
x0    = [1.2 ; 1.2]
sd0   = sd(x0)
nd0   = nd(x0)
alpha = 1
```

Functions :: symbolic, “anonymous”

```
if armijo(alpha,c1,x0,sd0,vf,vf_grad)
    fprintf('SD can take full step from x0 = [%g,%g]\n',...
            x0(1),x0(2))
else
    fprintf('SD can NOT take full step from x0 = [%g,%g]\n',...
            x0(1),x0(2))
end

if armijo(alpha,c1,x0,nd0,vf,vf_grad)
    fprintf('Newton can take full step from x0 = [%g,%g]\n',...
            x0(1),x0(2))
else
    fprintf('Newton can NOT take full step from x0 = [%g,%g]\n',...
            x0(1),x0(2))
end
```


Index