

Normalization of Design Variables and Convergence Criteria

AE 6310: Optimization for the Design of Engineered Systems

Spring 2017

Dr. Glenn Lightsey

Lecture Notes Developed By Dr. Brian German



Normalization of Design Variables

It is typically considered to be good practice to normalize the design variables in some way before solving the optimization problem.

Normalization may improve the performance of optimization algorithms in several ways:

- ❖ Make it easier to bracket the minimum in line searches
- ❖ Achieve more consistent convergence
- ❖ Reduce the number of iterations required to reach the minimum
- ❖ Reduce issues associated with numerical round-off error



Normalization of Design Variables

There are many possible ways to normalize the design variables. We will discuss three:

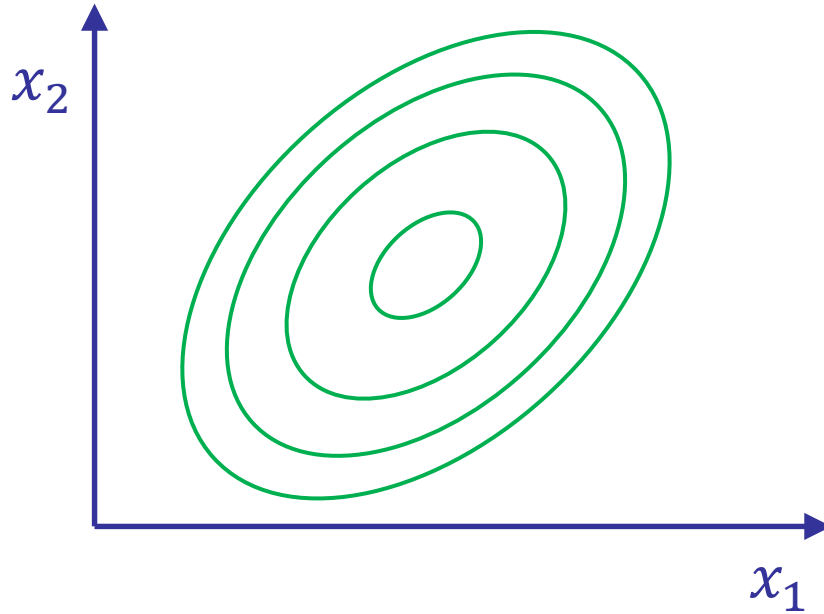
- ❖ Normalization based on the Hessian
- ❖ Normalization based on a reference value for each variable
- ❖ Normalization based on a reference interval for each variable

There are advantages and disadvantages to these (and all) normalization methods. It is up to the user to determine which method is best or most practicable for a given problem.



Hessian-Based Normalization

Presume that we have a n -D quadratic function with quadratic terms given by $\mathbf{x}^T H \mathbf{x}$. The contours of the function may look like this:



Skewed contours whose principal axes are not aligned with the coordinate axes are more difficult for algorithms such as univariate search and steepest descent.



Hessian-Based Normalization

Hessian-based normalization attempts to correct this problem by altering the contours.

To determine the normalization, we first, we apply a transformation of the form,

$$\mathbf{x} = R\mathbf{y}$$

to obtain,

$$\mathbf{x}^T H \mathbf{x} = \mathbf{y}^T R^T H R \mathbf{y}$$

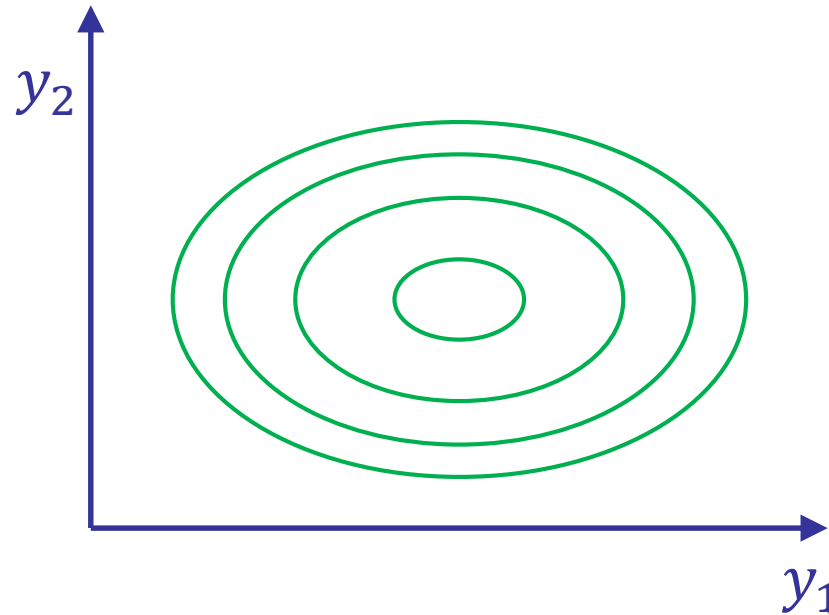


Hessian-Based Normalization

Next, we choose the columns of matrix R to be the eigenvectors of the Hessian, H . This will produce a diagonal Hessian matrix, \bar{H} , in the transformed variables, \mathbf{y} :

$$\bar{H} = R^T H R$$

This transformation rotates the contours to align with the \mathbf{y} coordinate axes.



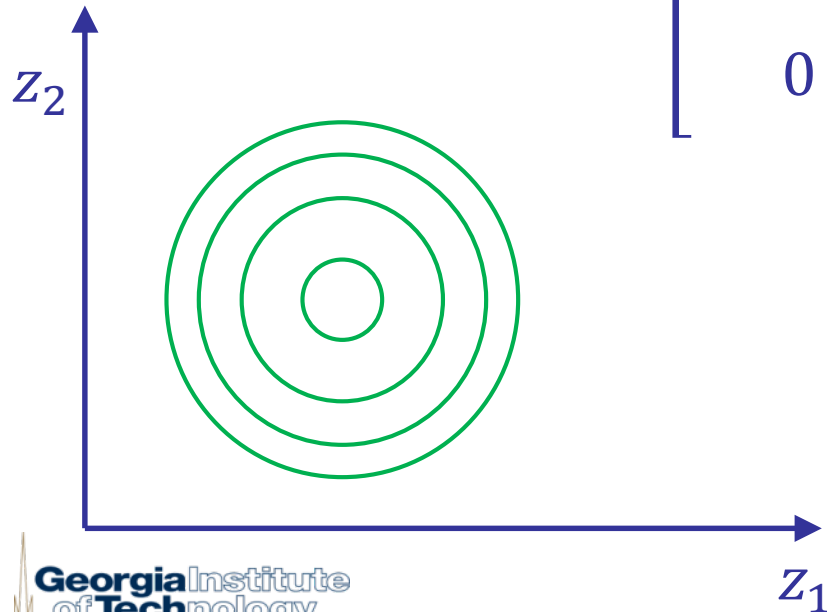
Hessian-Based Normalization

Finally, we scale the \mathbf{y} coordinates to produce new coordinates

$$\mathbf{z} = S\mathbf{y}$$

by choosing S as,

$$S = \begin{bmatrix} 1/\sqrt{\bar{H}_{11}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\sqrt{\bar{H}_{nn}} \end{bmatrix}$$



This choice of S makes the axes the same scale in each coordinate direction



Hessian-Based Normalization

Advantages:

- ❖ Methods converge quickly when the problem is posed in terms of the scaled design variables, e.g. steepest descent should converge in 1 step for a quadratic function

Disadvantages:

- ❖ Hessian is costly to compute
- ❖ For non-quadratic function, may need to re-normalize variables every few iterations



Normalization by a Reference Value

To normalize by a reference value, we select a “typical” value of each design variable somewhere in the middle of the range of expected variability and divide by its magnitude:

$$\mathbf{z} = \begin{bmatrix} 1/|x_{1,ref}| & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/|x_{n,ref}| \end{bmatrix} \mathbf{x}$$

The reference values can be selected based on the maximum or nominal values of the design variables in a given search vector.



Normalization by a Reference Value

Advantages:

- ❖ Implies that the approximately same % change in each variable is achieved with the same line search step size
- ❖ Useful for unconstrained search, when bounds on the variables are truly unknown but we have a sense of a “typical” value

Disadvantages:

- ❖ Method will not work well if the reference value is close to zero
- ❖ Does not work very well if the natural “unit change” in the design variable is not linear



Normalization by a Reference Interval

We can also normalize by a reference interval defined by upper and lower bounds on each design variable:

$$\mathbf{z} = \begin{bmatrix} \frac{x_1 - x_{1,L}}{x_{1,U} - x_{1,L}} \\ \vdots \\ \frac{x_n - x_{n,L}}{x_{n,U} - x_{n,L}} \end{bmatrix}$$

Results in each variable ranging from 0 to 1 within the interval.
Especially useful if a “natural” interval is available, e.g. side constraints.

It is also possible to normalize to a -1 to 1 interval. Try it!



Convergence Criteria

Convergence criteria are metrics that we compute and check during optimization in order to determine when to stop the search.

There are several types of convergence criteria:

- ❖ Maximum number of iterations (line searches) or function calls
- ❖ Absolute or relative change in the objective function
- ❖ Absolute or relative change in the design variable vector
- ❖ First-order necessary conditions for optimality

Most algorithms determine when to stop based on several (or even all) of these criteria.



Maximum Number of Iterations

Stop when the number of iterations, q , reaches a specified maximum, q_{\max} :

while $q \leq q_{\max}$

...

end



Absolute Change in the Objective Function

Stop when

$$|f_k - f_{k-1}| \leq \varepsilon_A \quad \text{💬}$$

where ε_A is either a specified constant value, e.g. $\varepsilon_A = 0.0001$ or a fraction of the objective function value at the first iteration, e.g. $\varepsilon_A = 0.001|f_0|$:

```
while abs(f(k)-f(k-1)) > epsA
```

```
...
```


```
end
```

Similar absolute change criteria can be formulated for elements of the design variable vector, \mathbf{x}



Relative Change in Objective Function

Stop when

$$\frac{|f_k - f_{k-1}|}{\max[|f_k|, 10^{-5}]} \leq \varepsilon_R$$


where ε_R is a specified fractional change in the objective function, e.g. $\varepsilon_R = 0.001$. The denominator is formulated to make sure that we avoid dividing by zero if $f_k \rightarrow 0$. (We could choose a small value other than 10^{-5}).

Similar relative change criteria can be formulated for elements of the design variable vector, \mathbf{x}



First-Order Necessary Conditions

Stop when

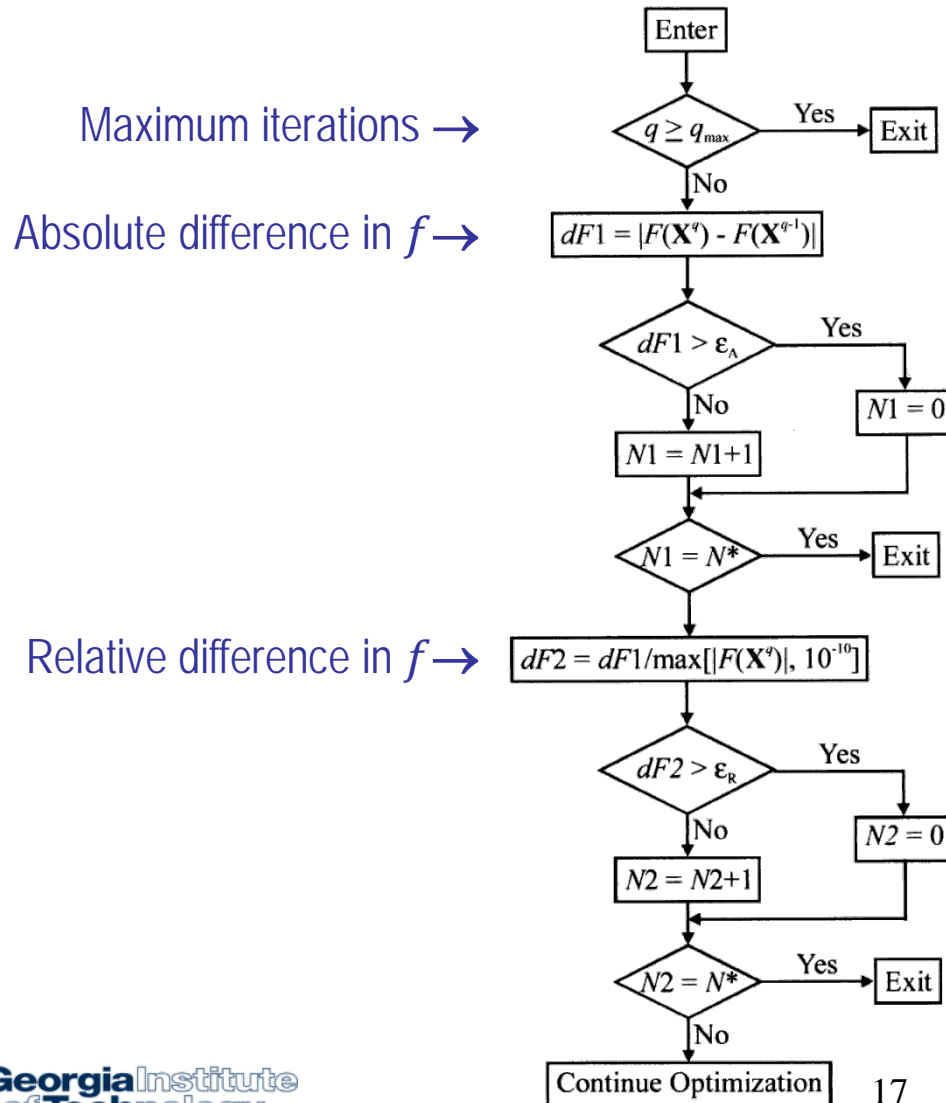
$$\|\nabla f_k\|^p = \left(\sum_i \left| \frac{\partial f}{\partial x_i} \right|^p \right)^{1/p} \leq \varepsilon_G$$

where p defines a type of norm. Typically, $p = 1$, $p = 2$, or $p \rightarrow \infty$.

Or, we could check that all elements $\left| \frac{\partial f}{\partial x_i} \right|$ of the gradient are less than or equal ε_G individually.



A Typical Convergence Checking Algorithm



Some criteria are allowed to be violated for a few iterations of the algorithm to be sure that the search is not prematurely stopped

Vanderplaats, Fig. 3.12



Convergence Criteria from MATLAB optimset

| | | |
|--------------------|------------------|---|
| MaxFunEvals | positive integer | Maximum number of function evaluations allowed. |
| MaxIter | positive integer | Maximum number of iterations allowed. |
| TolFun | positive scalar | Termination tolerance on the function value. |
| TolX | positive scalar | Termination tolerance on x. |

