Approximations and Derivatives

Practical Approaches to Engineering Optimization
Introduction

- Having seen methods to optimize, now how are these programs connected to the analysis programs?
- We need to be able to get the objective function and constraints from the analysis program, and feed them to the optimizer.
- Note that we may have to scale design variables and constraints before using them.
# Introduction

- **Standard Optimization Process:**

<table>
<thead>
<tr>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify variables, parameters</td>
</tr>
<tr>
<td>Run analysis using the variables available</td>
</tr>
<tr>
<td>Extract results and scale if needed</td>
</tr>
<tr>
<td>Run Optimizer and obtain new set of variables</td>
</tr>
<tr>
<td>Have you converged?</td>
</tr>
</tbody>
</table>

- **Flowchart:**

- **Decision Points:**
  - NO
  - YES
  - STOP
Approximations

Are used because there are too many:

- Constraints ➔ Temporarily do not consider constraints that are not critical
- Design variables ➔ use shape functions, variable linking
- Computationally costly analyses ➔ use formal approximation techniques
Consequences

- Errors in the evaluation of the objective and constraints due to the approximation, rounding and truncation
- Possibility of loss of convergence due to errors
- However, much lower cost of function evaluations
Elimination of constraints

Motivation:
- Many constraints are far from critical
  - they can be ignored for the time being
- Gradient computations can be costly
  - we should not evaluate unnecessary gradients
- This is what good designers should do anyway
Constraint deletion

- Based on the constraint value
  - Delete non-critical constraints
- Example: Delete all \( g < -0.3 \)
Constraint deletion

- Also based on regionalization
  - Retain only a limited number of constraints in each region of the structure
  - Example: Delete all $g < -0.3$, retain max 2 per region

![Graph showing constraint deletion in different regions.](image)
Reduction of variables

- Design Variables linking
  - Impose symmetry conditions
  - Aid manufacturability
  - Use experience
  - Scale existing designs
  - Reduce the number of independent variables
Direct linking

- Group areas (1,2,3,4), (5,6), (7, 8, 9, 10) so that each group is controlled by one design variable.
- This reduces the problem to 3 design variables out of an original 10 design variables.
Direct linking

\[
\begin{align*}
\begin{bmatrix}
A_1 \\
A_2 \\
A_3 \\
A_4 \\
A_5 \\
A_6 \\
A_7 \\
A_8 \\
A_9 \\
A_{10}
\end{bmatrix} &=
\begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & 0 \\
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 1 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2 \\
X_3
\end{bmatrix}
\end{align*}
\]
Design variable linking

- Reduced basis method
- Shape variables
Example simplification of variables

Design variables
Example simplification of variables
## Optimization Process with Approximations

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify variables, parameters</td>
<td>Run analysis using the variables available</td>
</tr>
<tr>
<td>Extract results and scale if needed – Build approximation</td>
<td>Run Optimizer and obtain new set of variables</td>
</tr>
<tr>
<td>Run approximation</td>
<td>Have you converged on the approximation?</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Overall convergence?</td>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>STOP</td>
<td></td>
</tr>
</tbody>
</table>

11/30/2005

Dr. G. Fadel - Clemson University
Types of Approximations

- Three types of Approximations:
  - Local, in the neighborhood of the current design point
    - Linear Taylor series
    - Reciprocal
    - Conservative or Hybrid
  - Mid Range
    - Multi point approximation in a region around the design point
  - Global
    - Response surface
    - Kriging Methods
    - Neural Networks
Calculation of Derivatives

Since some approximations may need derivatives, how are these obtained?

- Explicitly
- Using Finite differences
  - Forward
  - Central
- Symbolically
- Automatic Differentiation
Calculation of derivatives

- Explicit: The code returns the value of the derivatives of the objective and constraint functions at a specific location in the design space (for $x=x_i$)

- Using Finite differences – These have to be computed with respect to every design variable
Considering a function defined at only discrete points (typically progressively computed in optimization), how can one find approximations of the derivatives at these discrete points?

We seek the derivatives of the function at nodes \((x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots\)
Computation of derivatives

- The derivatives are obtained by fitting the given node points with an appropriate polynomial and then finding the derivatives of this expression.

- The formulas are typically applied to 3, 4 or 5 points, and if more than 5 points are specified, the formulas are applied successively to groups of 3, 4 or 5 adjacent points to find the derivatives.
Three point formula

- In the figure, 3 points are shown, a, b, and c, separated by an equal increment h in the x direction.
- We fit a parabola through the three points and evaluate the equation at every discrete points we have.
Three point formula

Using Lagrangian Interpolation formula

\[ y(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} y_2 \]

\[ y(x) = \frac{(x-h)(x-2h)}{(-h)(-2h)} y_0 + \frac{(x)(x-2h)}{(h)(-h)} y_1 + \frac{(x)(x-h)}{(2h)(h)} y_2 \]

\[ y(x) = \frac{x^2}{2h^2}(y_0 - 2y_1 + y_2) + \frac{x}{2h}(-3y_0 + 4y_1 - y_2) + y_0 \]
Computing the derivatives

\[ y(x) = \frac{x^2}{2h^2}(y_0 - 2y_1 + y_2) + \frac{x}{2h}(-3y_0 + 4y_1 - y_2) + y_0 \]

The first derivative of the function \( y(x) \) is as follows:

\[ y'(x) = \frac{x}{h^2}(y_0 - 2y_1 + y_2) + \frac{1}{2h}(-3y_0 + 4y_1 - y_2) \]

At \( x = x_0 = 0 \), \( y'(x_0) = \frac{1}{2h}(-3y_0 + 4y_1 - y_2) \)

At \( x = x_1 = h \), \( y'(x_1) = \frac{1}{2h}(-y_0 + y_2) \)

At \( x = x_2 = 2h \), \( y'(x_2) = \frac{1}{2h}(y_0 - 4y_1 + 3y_2) \)
Computing the derivatives

\[ y'(x) = \frac{x}{h^2} (y_0 - 2y_1 + y_2) + \frac{1}{2h} (-3y_0 + 4y_1 - y_2) \]

The second derivative of the function \( y(x) \) is as follows:

\[ y''(x) = \frac{1}{h^2} (y_0 - 2y_1 + y_2) \]

It is constant along the function since we are using a second order parabola.

\[ y''(x_0) = y''(x_1) = y''(x_2) = \frac{1}{h^2} (y_0 - 2y_1 + y_2) \]

Similar derivations can be carried out for higher degree functions.
Increasing the accuracy
Forward difference formulas

- By using more points, the accuracy of the approximation can be improved.

<table>
<thead>
<tr>
<th></th>
<th>$f_i$</th>
<th>$f_{i+1}$</th>
<th>$f_{i+2}$</th>
<th>$f_{i+3}$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$hf'(x_i)$=</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td>$0(h)$</td>
</tr>
<tr>
<td>$h^2f''(x_i)$=</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td></td>
<td>$0(h)$</td>
</tr>
<tr>
<td>$2hf'(x_i)$</td>
<td>-3</td>
<td>4</td>
<td>-1</td>
<td></td>
<td>$0(h^2)$</td>
</tr>
<tr>
<td>$12h^2f''(x_i)$</td>
<td>2</td>
<td>-5</td>
<td>4</td>
<td>-1</td>
<td>$0(h^2)$</td>
</tr>
</tbody>
</table>
Central difference formulas

- You need points on both sides of the current point

<table>
<thead>
<tr>
<th>$2hf''(x_i)$ =</th>
<th>$f_{i-2}$</th>
<th>$f_{i-1}$</th>
<th>$f_i$</th>
<th>$f_{i+1}$</th>
<th>$f_{i+2}$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h^2f'''(x_i)$ =</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0($h^2$)</td>
</tr>
<tr>
<td>$12hf''(x_i)$ =</td>
<td>1</td>
<td>-8</td>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>0($h^4$)</td>
</tr>
<tr>
<td>$12h^2f'''(x_i)$ =</td>
<td>-1</td>
<td>16</td>
<td>-30</td>
<td>16</td>
<td>-1</td>
<td>0($h^4$)</td>
</tr>
</tbody>
</table>
Generalizing to $n$ dimensions

- For every variable or dimension, the finite difference must be evaluated.
- Typically, first order derivatives are obtained using forward differencing, i.e. the value at the function(s) at the current point is needed, and the value of the function at a point at a distance $h$ from the original point in each dimension.
- The order of error of this approximation is $O(h)$, i.e. the points should be very close to each other to have a small error.
- For a 5 design variables problem, the function(s) must be evaluated 6 times at and around a point to get the derivatives with respect to each variable.
Symbolic computations

- Several computer programs allow you to symbolically write the derivative of the function.
- Maple, Matlab symbolic extensions are two such programs.
- If the evaluation of the objective or constraints is through a computer code, automatic differentiation should be used.
Automatic Differentiation

- Developed at ANL by George Corliss and Andreas Griewank (1991), it is a computer program that interprets computer code, and using the chain rule, inserts lines of code to evaluate the derivatives accurately at the same time as the function is evaluated at a point.

- The code is not available, but you can submit your code to be preprocessed and the derivatives code to be inserted.
Automatic Differentiation

references

Automatic Differentiation software

- **Fortran 77**
  - ADIFOR: Automatic Differentiation of Fortran
  - TAMC: Adjoint Model Complier
  - DAFOR: Differential Algebraic Extension of Fortran
  - GRESS: Gradient-Enhanced Software System
  - Odyssée: Automatic Differentiation of Fortran Programs
  - PADRE2: Partial Derivatives and Rounding Errors up to 2nd order

- **Fortran 90**
  - AD01: A Fortran 90 Package for Automatic Differentiation
  - ADOL-F: Automatic Differentiation of FORTRAN Codes
  - IMAS: Integrated Modeling and Analysis System
  - OPTIMA90: Automatic Differentiation Tools
  - Fortran Calculus from Optimal Designs, Inc.

- **ANSI-C and C++**
  - ADIC: Automatic Differentiation of C Programs
  - ADOL-C: A Package for the Automatic Differentiation of C/C++

- **Modeling Languages**
  - AD with AMPL, a Modeling Language for Mathematical Programming
  - COSY
  - An Implementation of the Forward and Reverse Mode in Maple

- **Support Libraries**
  - SparsLinC: Sparse Linear Combination Library

- **Commercial Systems**
  - AD Model Builder from Otter Research.
Local Approximations

- **Linear** Approximation
- Based on Taylor Series

\[ g(x) = g(x_0) + \sum_i (x_i - x_{0i}) \frac{\partial g(x_0)}{\partial x_i} \]

- Very popular method, still very heavily used.
- Requires move limits (range of validity of the design variables (typically 5-20% depending on non linearity of functions)
Local Approximations

- **Reciprocal** Approximation
- Take \( y_i = 1/x_i \)

\[
g(y) = g(y_0) + \sum_i (y_i - y_{0i}) \frac{\partial g(y_0)}{\partial y_i}
\]

\[
g(x) = g(x_0) + \sum_i (x_i - x_{0i}) \frac{x_{0i}}{x_i} \frac{\partial g(x_0)}{\partial x_i}
\]

- Very popular method, used mostly for stress constraints. (problem if \( x_i = 0 \), subsequently modified by adding constant term)
- Requires also move limits
Local Approximations

- **Hybrid Approximation**

  \[ g(x) = g(x_0) + \sum_i b_i (x_i - x_{0i}) \frac{\partial g(x_0)}{\partial x_i} \]

  Where \( b_i = \begin{cases} 
  \frac{x_{0i}}{x_i} & \text{if } x_{0i} \frac{\partial g(x_0)}{\partial x_i} \geq 0 \\
  x_i & \text{if } x_{0i} \frac{\partial g(x_0)}{\partial x_i} < 0 
  \end{cases} \)

- Requires also move limits
Mid-range Approximations

- Uses information from previous points explored to improve the approximation
- Two point exponential approximation was introduced in early 1990’s
- Extension to Taylor series adjusted by matching the derivatives at the previous design point
Two-point exponential approximation

Substitute $x^{p_i}$ for $x$ in the Taylor series:

$$g(X^p) = g(X_0^p) + \sum (x_i^{p_i} - x_{0i}^{p_i}) \frac{\partial g}{\partial x_i^p}$$

After resubstitution, one can write:

$$g(X) = g(X_0) + \sum \left( \frac{x_i}{x_{0i}} \right)^{p_i} - 1 \frac{x_{0i}}{p} \frac{\partial g}{\partial x_i} (X_0)$$

with

$$p_i = 1 + \frac{\log \left( \frac{\partial g}{\partial x_i} (X_i) \right) - \log \left( \frac{\partial g}{\partial x_i} (X_0) \right)}{\log(x_{i1}) - \log(x_{i0})} = 1 + \frac{\ln \left\{ \frac{\partial g}{\partial x_i} (X_i) \right\}}{\ln \left\{ \frac{x_{i1}}{x_{0i}} \right\}}$$

the point $X_1$ corresponds to the design point at the previous iteration and $X_0$ refers to the current design point where the approximation is carried out.
Mid-range approximations

Other variations on the two point approximation exist.

I introduced a method to compute move limits based on the value of the exponents which represent some measure of non-linearity.
Global Approximations

- Typically 2\textsuperscript{nd} order approximation over the whole design space. This is called a Response surface:
- The approximation functions are obtained by the least-squares method

\[ \hat{y} = b_0 + \sum_{i=1}^{k} b_i x_i + \sum_{i=1}^{k} b_{ii} x_i^2 + \sum_{i} \sum_{j \neq i} b_{ij} x_i x_j \]

\[ i < j \]
Response Surface

- Sample points must be selected carefully to obtain a representative distribution
- Use Design of Experiments to sample
  - Random methods
  - Diagonal design
  - Full grid
  - Central Composite
  - Box-Behnken Designs
  - Factorial Designs
  - Latin Hypercube
  - Orthogonal Arrays
  - ...

11/30/2005
Random Designs

- In this type of design, a number of random sample points are generated. The values of the parameters of these sample points are calculated by mapping the result of a random number generator in the range defined by the minimum and maximum values of the input parameters.

- Random designs are usually used when the input parameters have wide ranges and a large number of sample points are acceptable.
Gau-Beta Random Design

- The Gau-Beta random design is a special version of random design where the probability for the evaluations follows a normal distribution.
- The Gau-β distribution is used since experimental errors are often normal distributed to a good approximation.
- Furthermore, the assumption of a normal distribution simplifies many theorems and methods of data analysis.
Diagonal Design

- For the Diagonal design the n dimensional hypercube is cut by a diagonal line where all the sample points are placed.
- Two of the design points are the two corners of the hypercube and the rest of the m sample points have equidistant spacing.
Full Grid Design

- In the Full grid design each of the $n$ parameter ranges is divided in $k$ equidistant sections.
- On each of these sections a design point is placed. This results in an enormous number of sample points.
- This makes the design only usable for a small number of parameters.
Classical Experimental Designs

- Classical experimental designs are so named because they have been developed for what are considered to be more “classical” applications of response surface meta-modeling: physical experiments which suffer from variability and random error.

- Among these designs, the central composite and Box-Behnkehn designs are well known and easily generated.
Central Composite Designs

- A central composite design (CCD) is a combination of 2k factorial points, 2k star points, and a center point for k factors.
- CCD’s are the most widely used experimental design for fitting second order response surfaces.
- Different CCD’s are formed by varying the distance from the center of the design space to the star points.
Central Composite Designs

- **Ordinary central composite design (CCD)** – star points are placed a distance of \( (a > 1) \) from the center with the cube points placed at \( \sqrt{a} \) from the center. The perpendicular distance from the center of the cube to the face is 1 unit.

- **Face centered central composite design (CCF)** – star points are positioned on the faces of the cube.

- **Inscribed central composite design (CCI)** – star points are positioned at \( \sqrt{a} \) from the center with the cube points placed at.
Central Composite Designs for 3 Factors
Screening Analysis for 3 factors
Box-Behnkehn designs

![Diagram of Box-Behnkehn designs showing points in a three-dimensional space with axes X1, X2, and X3.](image)
Full-Factorial Design
3 Factors at 2 Levels
Latin hypercubes

- A Latin hypercube is a matrix of \( n \) rows and \( k \) columns where \( n \) is the number of levels being examined and \( k \) is the number of design variables. Each column contains the levels 1, 2, ..., \( n \), randomly permuted, and the \( k \) columns are matched at random to form the Latin hypercube.
Latin hypercubes
Latin Hypercubes

- Latin hypercubes are quite easy to generate, because they require only a random permutation of $n$ levels in each column of the design matrix.
- The big advantage of Latin hypercube designs is that they ensure stratified sampling, i.e., each of the input variables is sampled at $n$ levels.
- Thus, when a Latin hypercube is projected or collapsed into a single dimension, $n$ distinct levels are obtained.
- This is extremely beneficial for deterministic computer experiments since Latin hypercube points do not overlap thus minimizing any information loss.
Orthogonal Arrays

- An Orthogonal Array (OA) is a special type of a Latin square and is a matrix of $n$ rows and $k$ columns.

- An orthogonal array has an associated strength $r$ depending on the number of combinations of “$\pi$” levels appearing in any of the $r$ columns of the OA.
Orthogonal Arrays

- A specific type of matrix that allows you to reduce the number of experiments needed to find the effects of different parameters that affect a process (Design of experiments (DOE))
Orthogonal Arrays

- Classical Approach:

  - 3 parameters, 2 levels

<table>
<thead>
<tr>
<th></th>
<th>L</th>
<th>L</th>
<th>L</th>
<th>L</th>
<th>H</th>
<th>H</th>
<th>H</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>

- Full Factorial: $2^3 = 8$ experiments
Orthogonal Arrays

- Developed by S. R. Fisher in England to control the error of an experiment.
- Adapted by Taguchi to measure not only the effect of a factor under study on the average result, but also to determine the variation from the average result.
- Orthogonal means balanced, not mixed => statistically independent.
Orthogonal Arrays

- Each level has an equal number of occurrences within each column
- Also, for each level within one column, each level within any other column will occur an equal number of times

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 2 \\
1 & 2 & 1 \\
1 & 2 & 2 \\
\end{array}
\]
Orthogonal Array for 2 factors
Orthogonal Arrays

Since each column is orthogonal to the others, if the results associated with one level of a specific factor are much different at another level, it is because changing that factor from one level to the next has a strong impact on the quality characteristic being measured.
Orthogonal Arrays

- Analysis: Combine data associated with each level for each factor or interaction (column). The difference in the average results for each level is the measure of the affect of that factor. Those factors with the greatest effect or difference are the ones that can be used to improve the process.
Orthogonal Arrays

- Considering 15 parameters, a full factorial requires for 2 levels $2^{15} = 32768$ experiments. Orthogonal arrays only require 16 runs!

- Not cure all. Selection of parameters very important to get meaningful results

- $L_A B^C$ A = N experiments, B = levels C = columns

- $L_4 2^3$, $L_8 2^7$, $L_{12} 2^{11}$ (no interactions), $L_{16} 2^{15}$

- $L_{54}(2^1 x 3^{25})$
Conclusion Design Space Sampling

- Many approaches available
- Consider the problem, and the most representative method to sample the space. Issues of computational cost and hypervolume coverage are drivers
- Once sampled, fit response surface or other global approximation method.
Kriging Method

- Combination of Surface response plus deviation from it (trigonometric function)
- Called also DACE Design and Analysis of Computer Experiments, and Spatial Correlation Metamodels (models of models)
- Main signal plus noise superimposed.
- Computationally expensive to compute, but better accuracy, especially for highly non-linear functions.
Kriging Method

- Interpolation technique originally developed by D. G. Krige, a South African engineer.
- In the 1950s he devised this method to determine true ore-grades, based on samples.
- He further improved this method in cooperation with G. Matheron, a French mathematician at the ‘Ecole des Mines’.
Kriging Method

- Based on the assumption that the parameter being interpolated can be treated as a regionalized variable.
- A regionalized variable is intermediate between a truly random variable and a completely deterministic variable in that it varies in a continuous manner from one location to the next and therefore points that are near each other have a certain degree of spatial correlation, but points that are widely separated are statistically independent.
Mathematics of Kriging

- Two-component model
- The first component consists of a general linear or a quadratic model
- The second (or lack of fit) is treated as the realization of a stationary Gaussian random function.
- Plainly speaking, the predictive model in Kriging has both local and global components, as opposed to response surface models, which are global in nature.
Mathematics of Kriging

- More specifically, Kriging postulates a combination of a global model and departures of the following form

\[ y(x) = f(x) + z(x) \]

- \( y(x) \) is the unknown function of interest, \( f(x) \) is a known function of \( x \) (linear or quadratic response surface) and \( z(x) \) is the realization of a stochastic process with mean zero and variance \( \sigma^2 \)
Comparison of methods

Plot of \( f(x) \)

15 Point Constant Kriging

Kriging with constant model

Response Surface Model of \( f(x) \)

15 Point Quadratic Kriging

Kriging with quadratic model
Artificial Neural Networks

- Artificial neural networks are biologically inspired. They attempt to mimic the human brain, but naturally fall far short of its capabilities. Yet, they are useful for capturing the behavior of a function.
- Typically, in an average biological system, $10^{11}$ neurons participate in about $10^{15}$ interconnections, the resemblance to real systems is at best superficial.
- They consist of learning, generalizing and abstracting principles.
- They take advantage of parallel computation.
- Can be overtrained and useless.
ANN Simplification
Artificial Neural Networks

\[ \eta = F(Z) = \sum_{i} W_i x_i + \beta \] is the activation function

Requires training and knowledge of ANN and problem to set up
ANN

- A connected set of such neurons comprises an artificial neural network
- You can have a flat network that consists of two layers, input and output layers
- You can have one or more hidden layers that are in between the input and output layers.
ANN

- Given a set of input-output data (x-y pairs), obtain the values of the interconnection weights $w_{ij}$ and activation function parameters to map this data.
- This part is the training of the ANN
- Next, given some x-vector that is not part of the training set, the network should produce an approximation to Y.
ANN – Back-propagation overview of training

- Let $w_{ij}^k$ be the weight between nodes $i$ and $j$ between the $k-1$ and $k$ layer.
- Initialize the weights randomly.
- Compute the weighted sum of inputs to a neuron $j$:
  \[ Z_j = \sum w_{ij}^k x_i \]
- Process $Z$ through the neuron activation function (Sigmoid function):
  \[ Y = F(Z) = \frac{1}{1 + \exp(-\frac{Z+T}{T_0})} \]
ANN Back propagation training

- Compute error in output:
  \[ E_i = (T_i - Y_i) \]

- Modify the weights of the output layer ‘k’ as:
  \[ \Delta W_{pi,k} = \nu \delta_{i,k} Y_{p,i} \]

Where

\[ \delta_{i,k} = \frac{\partial Y_{i,k}}{\partial Z} E_i \]

- To improve the training characteristics, a momentum term is often added to the weight update formula.

\[ \Delta W_{pi,k}^{t+1} = \nu \delta_{i,k} Y_{p,j} + \alpha \Delta W_{pi,k}^{t} \]
ANN Comments

- Multilayer networks can approximate arbitrarily well given continuous functions, provided that a large number of units is available- (Weierstrass property)
- Feedforward networks are equivalent to a parametric approximating function $f(W,x)$ where $W$ represent the set of weights in the network
- Minimize the number of hidden layers, more hidden layers means more weights to acquire through training
- Overtraining can lead to very noisy functions that are only accurate at the trained points. (similar to fitting a high order polynomial through a small set of points)
Conclusions

- Approximations must be carefully selected to obtain the desired solution.
- Global approximations are used to model the whole problem.
- Local approximations do not give much feedback on the overall problem behavior, but are used to progress locally towards an optimum.