System Identification and Parameter Estimation

Wb 2301

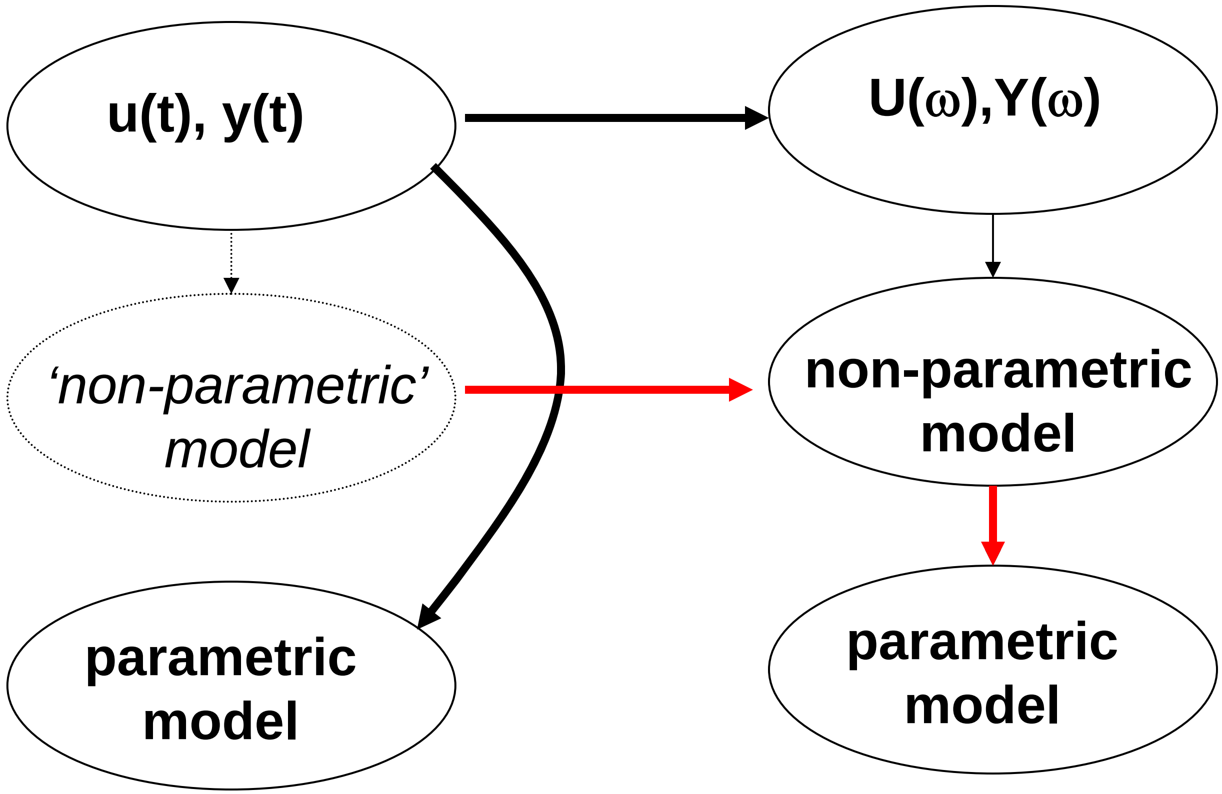
Frans van der Helm

Lecture 8

Optimization methods

Identification:
time-domain vs. frequency-domain

ARX
ARMA
Etc.



Frequency
Response
Function
(FRF)

Contents
parameter estimation

- Parameter estimation in time-domain:
  - ‘Non-parametric’ models: ARMA, OE, etc.
  - Models with <u>physical</u> parameters
    - Input – output data
    - model structure & model parameters
      - Linear and non-linear models
    - Simulation of model structure
    - optimization algorithms: Adapt model parameters for best fit to simulation
- Parameter estimation in frequency domain:
  - Non-parametric models: Phase and amplitude
    - Can be derived from non-parametric time-domain models
  - Models with <u>physical</u> parameters
    - Results non-parametric model: phase and amplitude
    - model structure & model parameters
      - Linear models
    - optimization algorithms:
Adapt model parameters for best fit in frequency domain

Contents
parameter estimation

- Optimization algorithms:
    - Grid search
    - Gradient search
        - Steepest descent (Newton)
        - Quasi-Newton
        - Levenberg-Marquardt
    - Random search
        - Bremermann optimizer
    - Genetic algorithm

Contents
parameter estimation

- Special model structures:
  - Neural networks
  - (Expert systems and fuzzy sets)

Parameter estimation in time-domain

- Parameter estimation in time-domain:
  - ‘Non-parametric’ models: ARMA, OE, etc.
    - Parameters are <u>not physically interpretable</u>
    - No physical parameter fitting afterwards
    - Only for control purposes
    - By <u>transition to frequency domain</u>: Parameter estimation
  - Parametric models
    - Input – output data
    - model structure & model parameters
      - <u>Linear and non-linear models</u>
    - Simulation of model structure, e.g. in Matlab/Simulink
    - Criterion function: Model predictions vs. recorded data
    - optimization algorithms: Adapt model parameters for best fit to simulation

Linear and non-linear models

- Parameter estimation by iterative search
  - Static systems
  - Dynamic systems
- Criterion function
- Optimization procedure
  - Grid search
  - Gradient search
  - Random search
  - Genetic algorithms
- Validation

Static and dynamic systems

- $y(k) = f(\theta,u(k)) + n(k)$
  - θ: parameter vector
  - k = 1 .. N datapoints / time samples
  - $Z^N = [y(k)\ u(k)]$
- error definition:
  - $e(k) = y(k) - f(\theta,u(k))$
- criterion function (least squares):
  - $J(Z^N, \theta) = 0.5*\sum e(k)^2$
    - summation over k realizations/time instants
    - subject to constraints:
      - linear / non-linear
      - equality constraints / inequality constraints
    - Constraints define **'feasible region'** for parameters
- find minimum of $J(Z^n, \theta)$

Dynamic systems

- Non-linear function $y(t) = f(x(t),u(t),\theta,t)$
- correct model structure
- Known (measured) input $u(t)$
- initial guess of parameter vector θ
- **simulation:** $\hat{y}(t) = f(x(t),u(t),\theta,t)$
- error function: $e = y(t) - \hat{y}(t)$
- iterative search requires many simulations!!

Direct parameter fit

$$M.\ddot{y} + B.\dot{y} + K.(y - y0) = u$$

$$y = (\frac{u}{K} - \frac{M}{K}.\ddot{y} - \frac{B}{K}.\dot{y}).$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} u_1 & -\ddot{y}_1 & -\dot{y}_1 \\ \vdots & \vdots & \vdots \\ u_n & -\ddot{y}_n & -\dot{y}_n \end{bmatrix} . \begin{bmatrix} 1/K \\ M/K \\ B/K \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix}$$

$$Y = A.\theta + E$$

$$\theta = (A^T.A)^{-1}.A^T.Y$$

Parameter fit using simulations

- Minimize $V = e^T e;\ e = y - \hat{y} = y - sim(H(\theta, t), u(t))$

Grid search

- Systematically search the parameter space and find the minimum
  - Very elaborate
  - Depending on resolution of grid
  - Likely to find 'global' minimum

Gradient search

- Starting point θ_i in feasible region
- Optimal parameter vector θ^* is defined at minimum of $J(Z^n, \theta)$. Then:

$$\frac{\partial J(Z^N, \theta^*)}{\partial \theta} = 0$$

- Iterative search:

$$\theta_{i+1} = \theta_i + \alpha . f^i$$

  - α: step size
  - f: search direction
    - Newton algorithms:

$$f^i = -\left[\frac{\partial^2 J(Z^N, \theta_i)}{\partial \theta_i^{\,2}} \right]^{-1} . \frac{\partial J(Z^N, \theta_i)}{\partial \theta_i}$$

First and second gradient (least squares criterion)

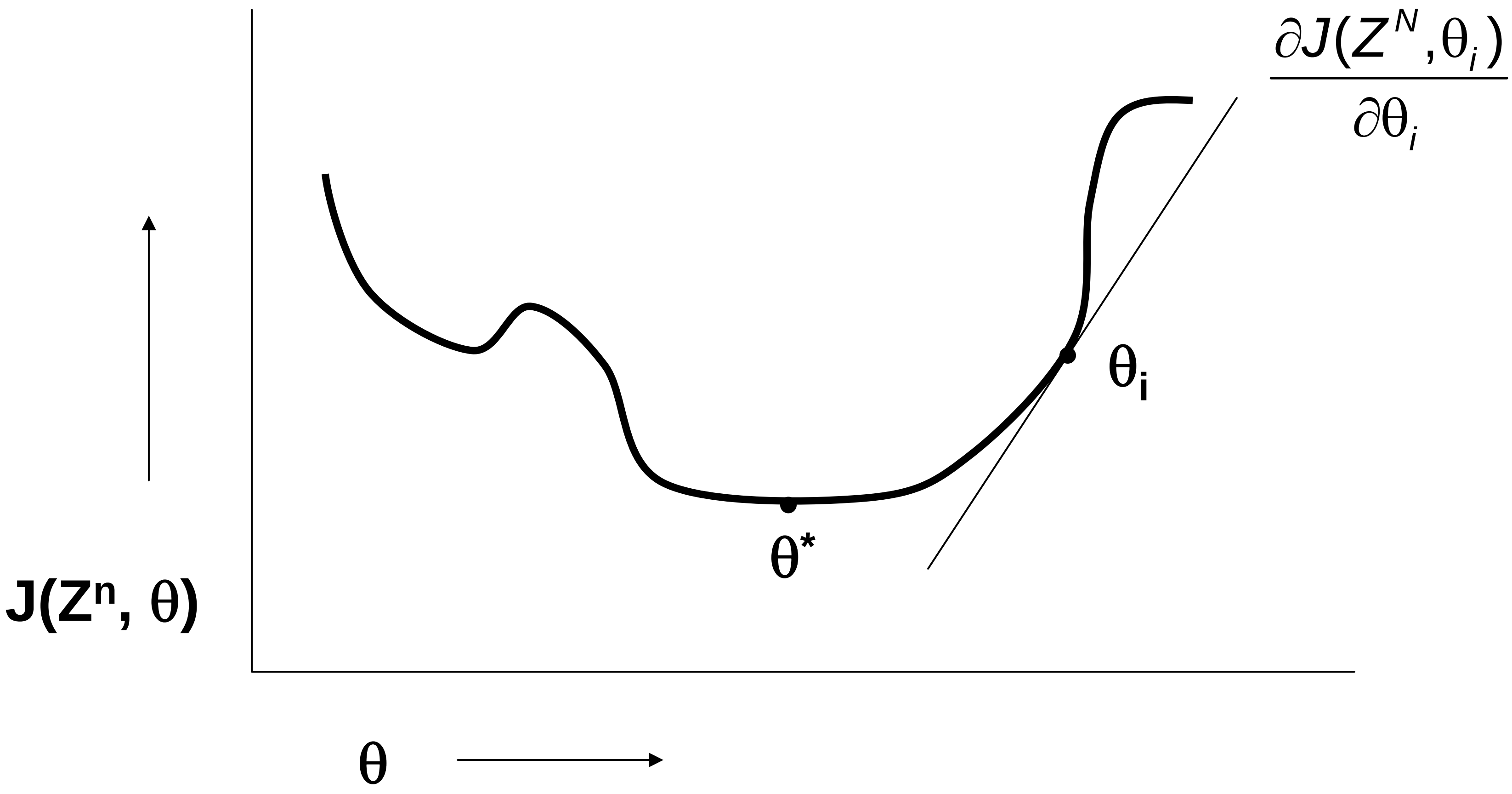
First derivative:

$$\frac{\partial J(Z^N,\theta^*)}{\partial\theta} = \frac{\partial e(Z^N,\theta^*)}{\partial\theta}.e(Z^N,\theta^*) = \varphi(Z^N,\theta^*).e(Z^N,\theta^*)$$

Second derivative:

$$\frac{\partial^2 J(Z^N,\theta^*)}{\partial\theta^2} = \varphi(Z^N,\theta^*).\varphi(Z^N,\theta^*)^T + \frac{\partial^2\varphi(Z^N,\theta^*)}{\partial\theta^2}$$

Gradient search



Gradient search

- Steepest descent: Search direction depends only on first derivative
  - Slow close to minimum
- Search direction depends on first and second derivative (Hessian matrix)
  - Takes dependency between parameters into account
  - Fast close to minimum
  - Expensive to calculate Hessian
- Quasi-Newton: Uses approximation of Hessian, e.g.

'Gauss-Newton': $$\frac{\partial^2 J(Z^N,\theta^*)}{\partial\theta^2} = \varphi(Z^N,\theta^*).\varphi(Z^N,\theta^*)^T + \frac{\partial^2 \varphi(Z^N,\theta^*)}{\partial\theta^2}$$

Vanishes near optimum

Gradient Search

- Levenberg - Marquardt algorithm:

$$\frac{\partial^2 J(Z^N,\theta^*)}{\partial\theta^2} = \varphi(Z^N,\theta^*).\varphi(Z^N,\theta^*)^T + \delta I$$

  - strengthens diagonal of Hessian
  - decrease of interaction between parameters (e.g. when model is overparameterized or badly parameterized)
  - Better convergence, more robust

Incorporation of constraints

- Criterion $J(Z^N,\theta) = 0.5*\sum e(k)^2$
- Subject to
  - Linear equality constraints: $A.\theta - B = 0$
  - Linear inequality constraints: $A.\theta - B < 0$
  - Non-linear equality constraints: $f(\theta) - C = 0$
  - Non-linear inequality constraints: $f(\theta) - C < 0$
- Equality constraints incorporated into criterion:
  - $J^*(Z^N,\theta) = J(Z^N,\theta) + \lambda_1.(A.\theta - B) + \lambda_2.(f(\theta) - C)$
  - λ_1, λ_2 : Lagrange multiplier, adaptive weight factor
  - $\partial J^*/ \partial\lambda_1 = 0 \rightarrow A.\theta - B = 0$
  - $\partial J^*/ \partial\lambda_2 = 0 \rightarrow f(\theta) - C = 0$
- Inequality constraints incorporated into criterion:
  - $J^*(Z^N,\theta) = J(Z^N,\theta) + \lambda_3.(A.\theta - B - s_1) + \lambda_4.(f(\theta) - C - s_2)$
  - s_1,s_2: slack variable, $s_1,s_2 > 0$

Gradient methods

- Very costly in calculating derivatives
  - “much information about only one point in parameter space”
- Algorithms are tuned to converge (if possible)
- Sensitive to local minima
- Result might depend on initial parameter guess
- Most often used !!

Random search methods

- Random search direction in parameter space:

$$\theta_{i+1} = \theta_i + \alpha.\boldsymbol{f}^i$$

- Calculate n criterion values along search direction
- Fit (n-1)th order polynome through criterion value
- Calculate minimum of polynome
- check if minimum is lower than previous minimum
- determine new search direction

Genetic algorithms

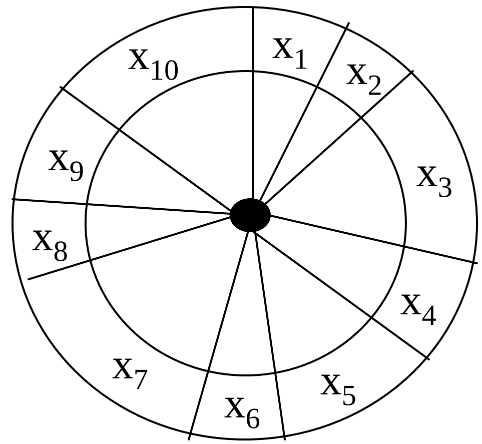
- Mimic nature in the search for the optimum:
  "Survival of the fittest"
- Start with a population of vectors x
  - Calculate criterion value J(x)
  - Select the best solutions of J(x): Maximal values
  - Generate children (new population vectors x):
    - Combine vectors x: Cross-over of part of vector x (intermediate solutions: local)
    - Mutation of vector x (scatter solutions over workspace: global)

Select initial population

- Generate m random vectors between lower and upper bounds: $x_1 .. x_m$
- Covers parameter space

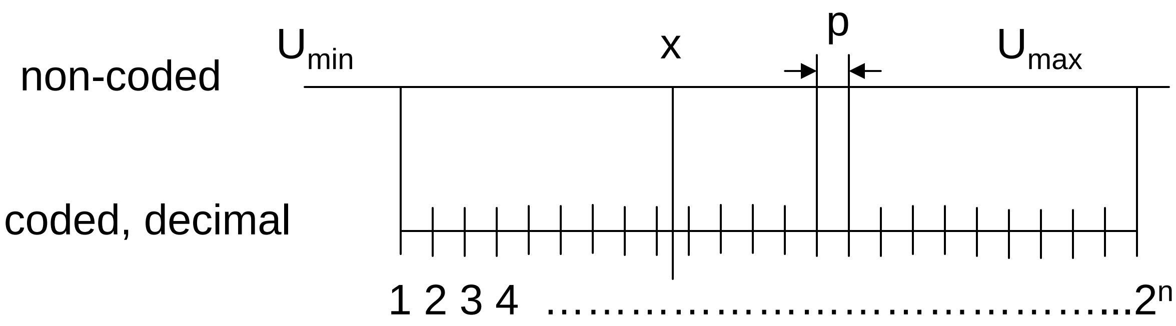
Select the best solution

- Calculate the criterion value J(x) for $x = x_1 .. x_m$



- Attribute probabilities according to criterion value (‘fitness’): $p(x_1) = J(x_1) / \Sigma J(x_1 .. x_m)$, etc.
- Draw m parameter vectors based on probability: “The greater the fitness, the greater the chance for reproduction”

Encoding from decimal in binary representation



$$p = \frac{u_{\max} - u_{\min}}{2^n - 1} \qquad x_{coded} = (round(\frac{x - u_{\min}}{p}))_{base 2}$$

- Round: turned into integer
- Decimal values from 1 to 2^n
- Decimal to binary transition
- Binary string representation: e.g. [0 1 1 0]
- All parameters in sequence of binary strings

Generate children

- Cross-over at random string intersection:

{0 1 1 | 0 1 0 | 1 1 0} ⇒ {0 1 1 | 1 0 0 | 1 1 1}

{1 0 1 | 1 0 0 | 1 1 1} ⇒ {1 0 1 | 0 1 0 | 1 1 0}

- Mutation:

{1 0 1 | 0 1 0 | 1 1 0} ⇒ {1 0 1 | 1 0 0 | 1 1 0}

- Decode

Conclusions
Genetic algorithms

- Advantage:
  - Less sensitive to local minima
  - No gradient calculations
  - Able to handle discontinuous parameter space and constraints on parameters
  - Can be combined with gradient algorithm near optimal solution
  - Suitable for parallel computers
- Disadvantage:
  - Slow, slow, slow
  - Especially near optimal solution

Optimization algorithms
Matlab

- lsqnonlin:
  - Gradient search, least squares criterion function assumed
  - Output error function: **vector**
  - Upper and lower boundaries on parameters
  - No constraints
- Fminunc
  - Gradient search, any criterion function
  - Output error function: criterion value
  - Upper and lower boundaries on parameters
  - No constraints

Optimization algorithms Matlab

- fminsearch:
  - Nelder-Mead simplex (direct search) method, any criterion function
  - Output error function: criterion value
  - Upper and lower boundaries on parameters
  - No constraints
- Fmincon
  - Gradient search, any criterion function
  - Output error function: criterion value
  - Upper and lower boundaries on parameters
  - Linear and non-linear, equality and inequality constraints

Optimization algorithms
Non-Matlab

- Levmar.m:
  - Gradient search, least squares criterion
  - Output error function: ERROR vector
  - Levenberg-Marquardt search: very robust against interaction between parameters
  - Turbo-parameters for steepest descent search
  - No upper and lower boundaries on parameters
  - No constraints