QTronic
SIMULATION FOR ENGINEERING

Using Chip Simulation to Optimize Engine Control

Matthias Simons - Daimler AG
Mihai Feier, Jakob Mauss - QTronic GmbH

7th Conference on
Design of Experiments (DoE) in Engine Development
Berlin, 18.–19.06.2013

Outline of the talk

Using Chip Simulation to Optimize Engine Control

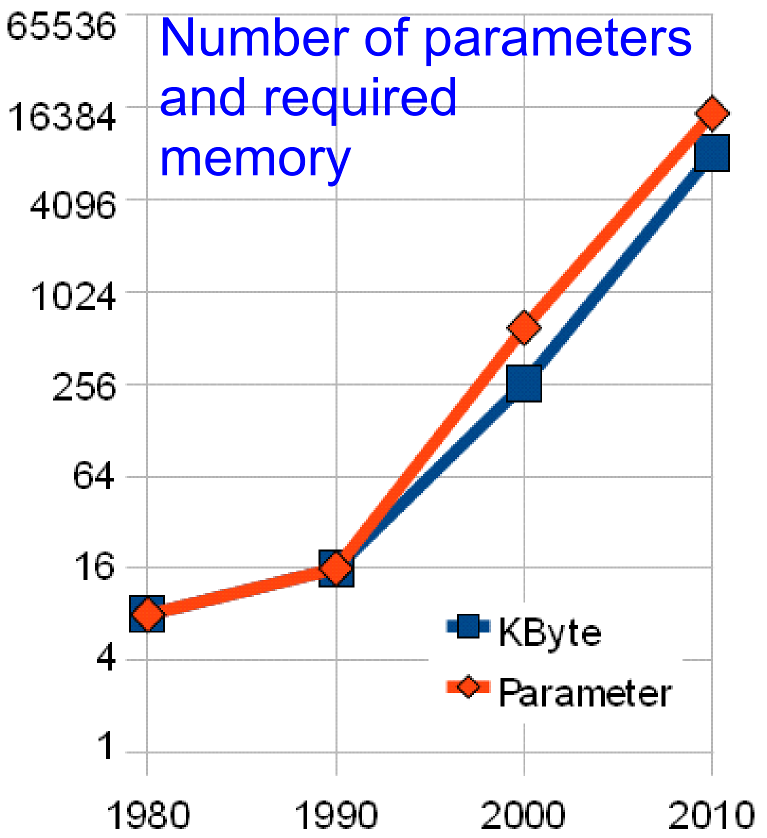
1. Motivation
2. Running ECU functions on PC via chip simulation
3. Coupling with least-squares optimization
4. Conclusion

Motivation

- number of engine control parameters doubles every few years
- budget for engine calibration does not

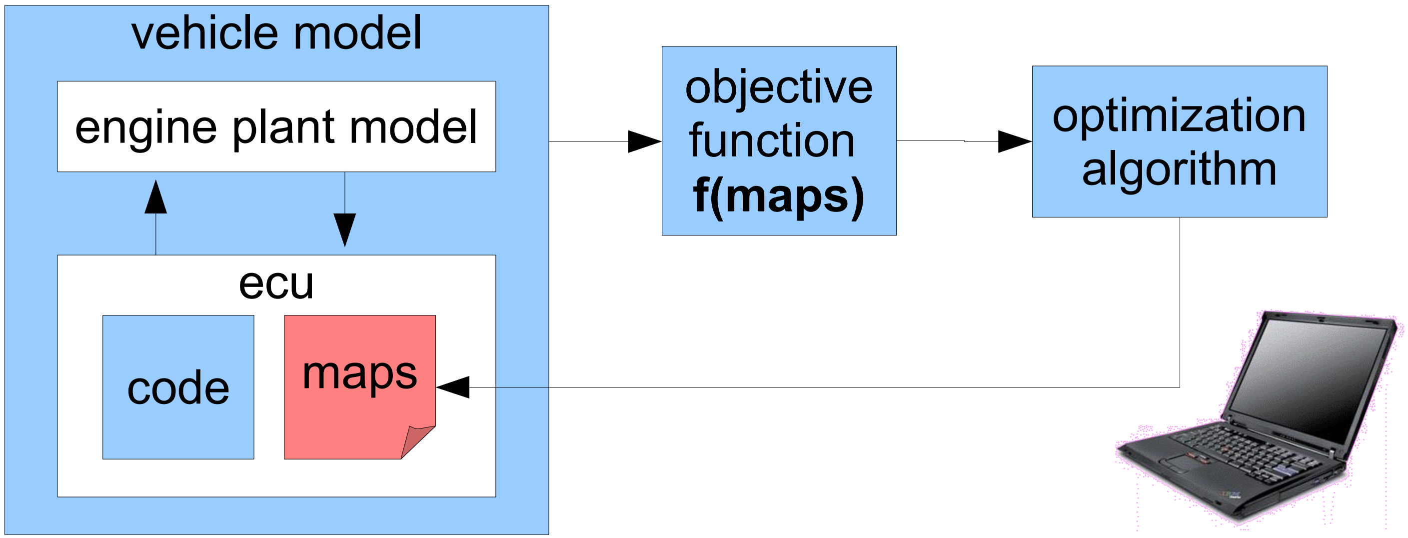
Idea

- increase degree of automation
- move calibration tasks from test rigs to PC and apply mathematical optimization



source: presentation of S. Ullmann (BMW) 5th Conference on DOE, 2009

Motivation: Model-based engine calibration



ECU source (C, Ascet, or Simulink model) typically not available for OEM

Challenge: how to simulate the ECU on PC?

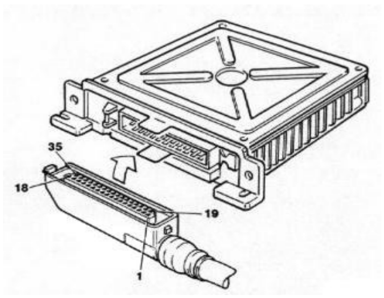
Options:

- **reverse engineer** the ECU function of interest, e. g. with Simulink
  → time consuming, error prone
- **simulate the hex** file of the ECU
  → less work, no modeling error

Simulating a hex file

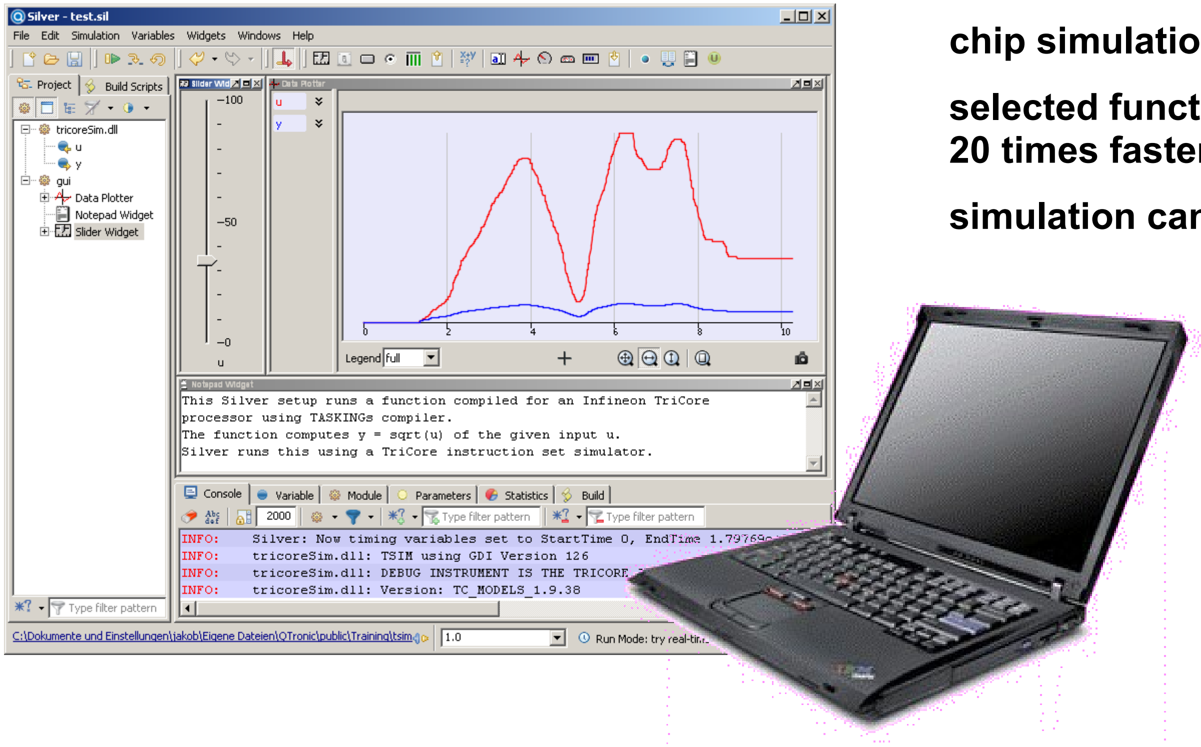
```
:020000040000FA
:0200000480007A
:200000003000000004FD00000000028000FD0080080200801C0200800507AFAFAFAFAFAF62
:20002000AFAFAFAFAFAFAFAFAFAFAFAF02000000D35A4822300402000000080FFFC0080C2
:20004000FECADEFAFEAFFECA0000000074FC00800010000310412007800008003020080C7
:20006000FECADEFAFEAFFECA0000000000000000010000779781BD000000000000000000F
:20008000000000000000000000000000000000000000000000000000000000000000000060
:2000A000000000000000000000000000000000000000000000000000000000000000000040
           00000000000000000000000000000000000000000000000000000000000000020
           0000000000000000000000000000000000000000000000000000000000000000
```

.hex



4 MB

Silver Chip Simulator
for TriCore chip family



chip simulation runs on PC with about 40 MIPS

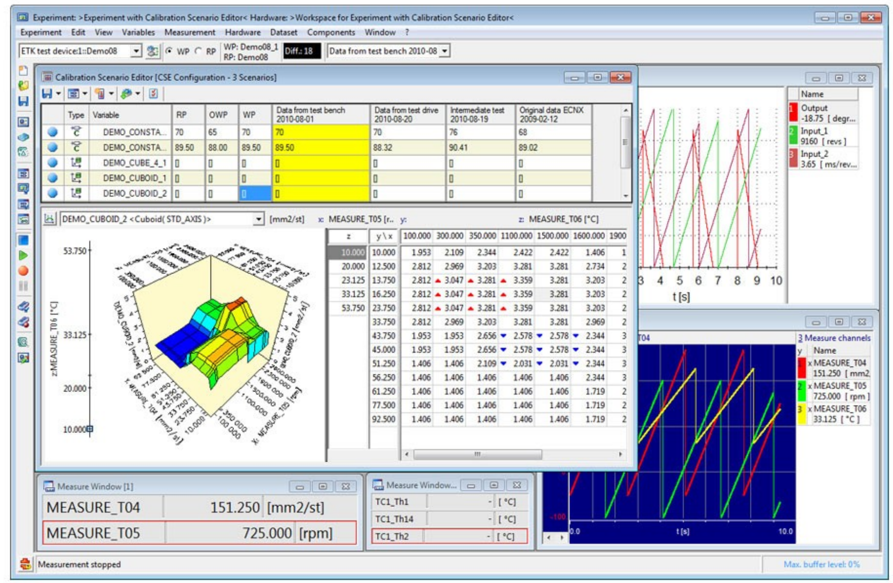
**selected function run e.g.
20 times faster than realtime**

simulation can be exported as SFunction

TriCore Simulation in Silver 2.5

QTronic
SIMULATION FOR ENGINEERING

INCA or CANape for on-line calibration:
measure and tune running simulation



virtual ECU

ECU functions

RTOS emulation

4GB virtual memory

A2L conversion

XCP

TriCore emulation
40 MIPS

TCP/IP

vehicle simulation or measurements

Silver

SIMULINK Enabled
MathWorks Partner

Setting up a TriCore simulation

QTronic
SIMULATION FOR ENGINEERING

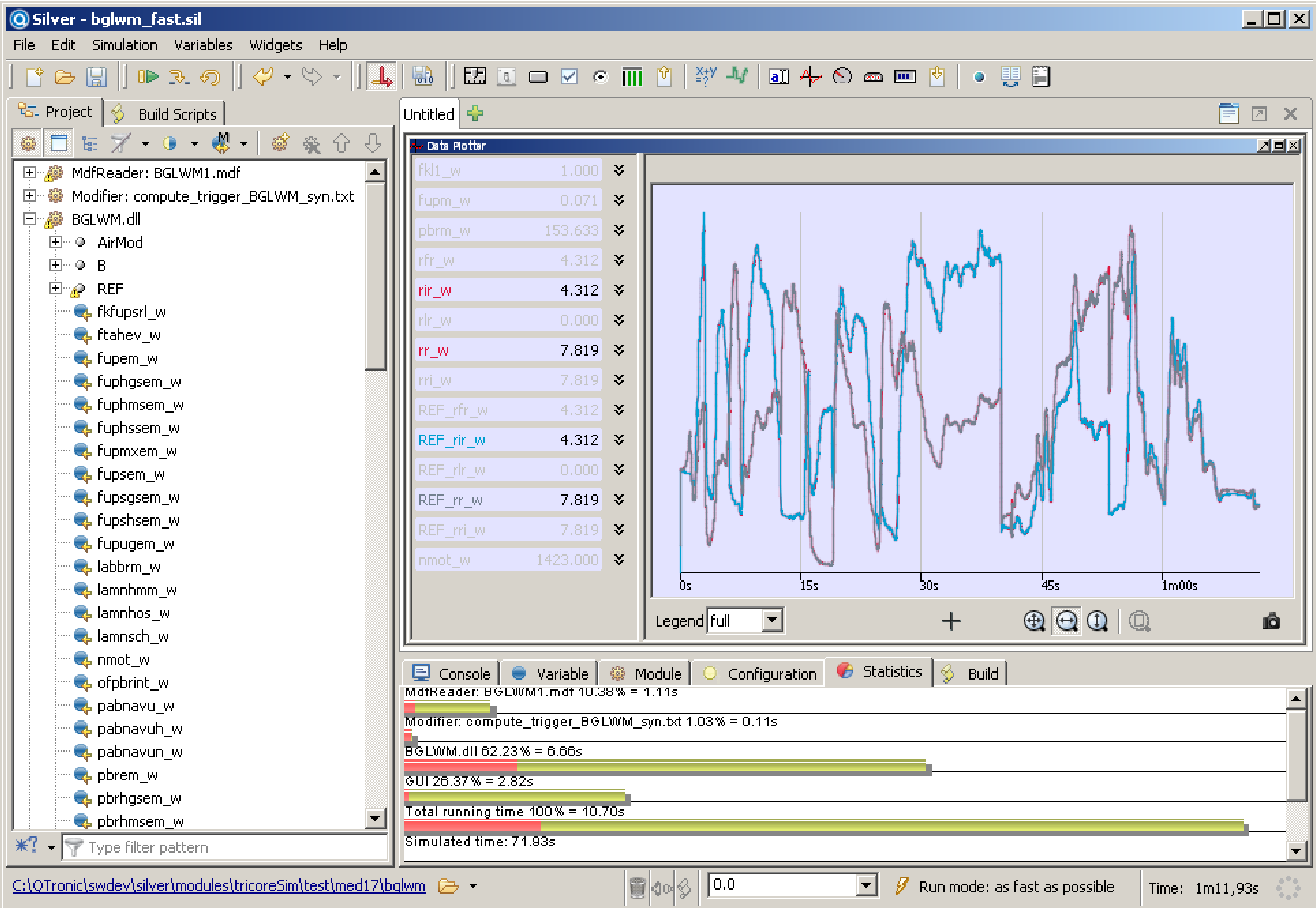
1. write spec.txt to specify what functions to run
2. step and debug the simulation in Silver debug mode
3. generate fast running SFunction or Silver module: runs without a2l and hex

```
# specification of sfunction or Silver module
hex_file(m12345.hex, TriCore_1.3.1)
a2l_file(m12345.a2l)
map_file(m12345.map)       # a TASKING or GNU map file
frame_file(frame.s)        # assembler code to emulate RTOS
frame_set(STEP_SIZE, 10)  # Silver step size in ms
frame_set(TEXT_START, 0xa0000000) # location of frame code

# functions to be simulated, in order of execution
task_initial(ABCDE_ini, 0)
task_initial(ABCDE_inisyn, 0)
task_triggered(ABCDE_syn, trigger_ABCDE_syn)
task_periodic(ABCDE_20ms, 20, 0)
task_periodic(ABCDE_200ms, 200, 0)

# interface of the generated sfunction or Silver module
a2l_function_inputs(ABCDE)
a2l_function_outputs(ABCDE)
a2l_function_parameters_defined(ABCDE)
```

Virtual ECU running in Silver: MED17



generated SFunction in MATLAB/Simulink

QTronic SIMULATION FOR ENGINEERING

spec.txt

.hex .map

.a2l frame.s

tcbuild

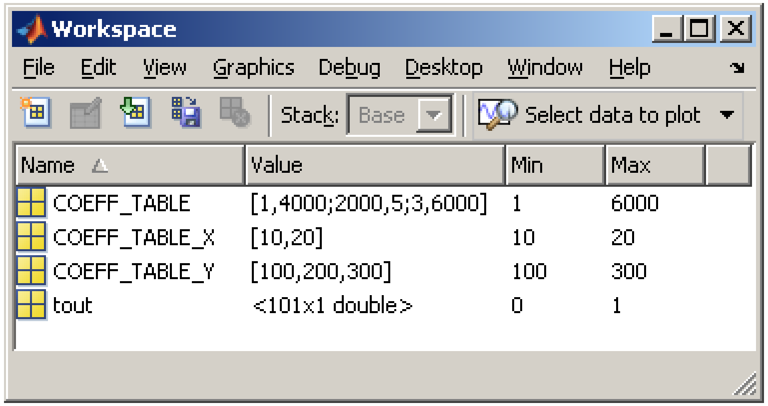
.mexw32

MATLAB/Simulink
S-function
40 MIPS

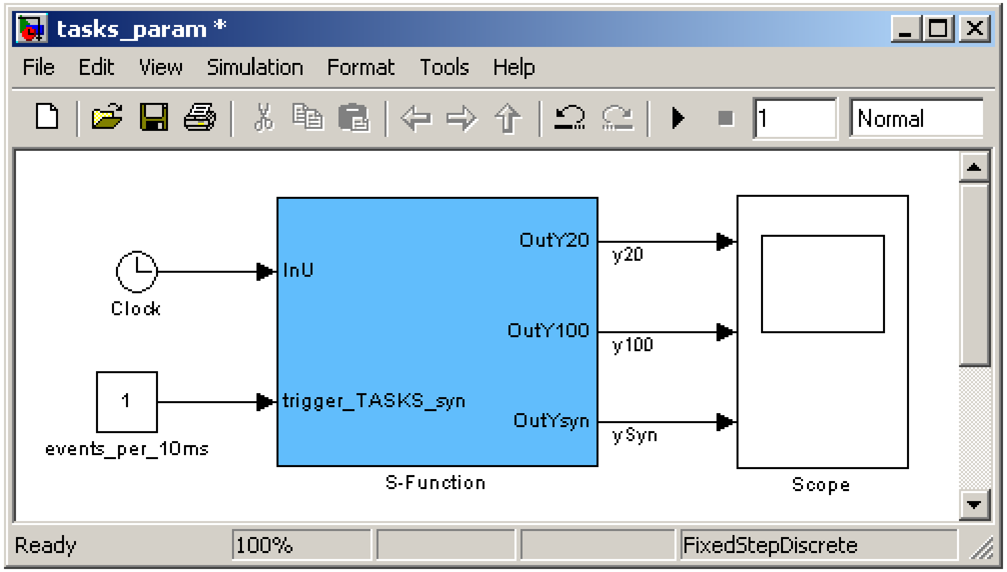
.mask.txt

default values for characteristics from HEX file as m script,

mask for S-function block and similar Simulink snippets



characteristics turned into MATLAB workspace variables
- read by S-function
- may be modified by script



Performance and Limitations

Run complex function for a measured scenario, 3.5 minutes

target	execution time	MIPS
Silver in debug mode	919.15 sec	0.41
generated Silver module or MATLAB/Simulink SFunction	9.30 sec	40.80
MED17 with TC1797, 180 Mhz	210.00 sec	270

Limitations
- instruction accurate, but not cycle accurate
- based on TriCore specification: 'silicon bugs' are not simulated
- PCP, CAN controllers and other on chip devices not modeled

Advantages
- no real-time requirement: simulate faster or slower than real-time
- 4 GB virtual memory available in virtual ECU
- zero-execution time model: simulated task runs infinitely fast
  hence: deterministic simulation without interrupts: easy to analyze

Optimization problem

Engine controller contains steady-state model of the engine

Objective

Tune parameters of the engine model
such that it fits given measurements

Least-squares optimization

Minimize goal function

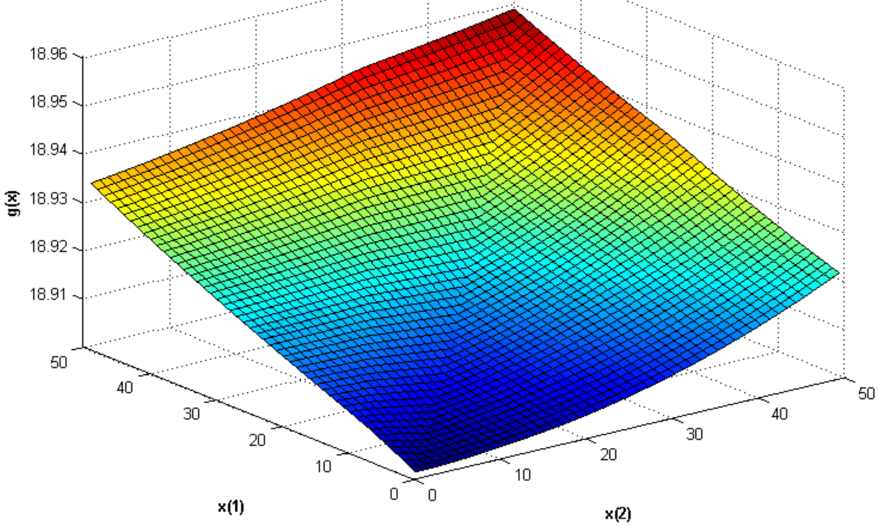
$$g(x) = \sum_{i=1}^{m} f_i^2(x)$$

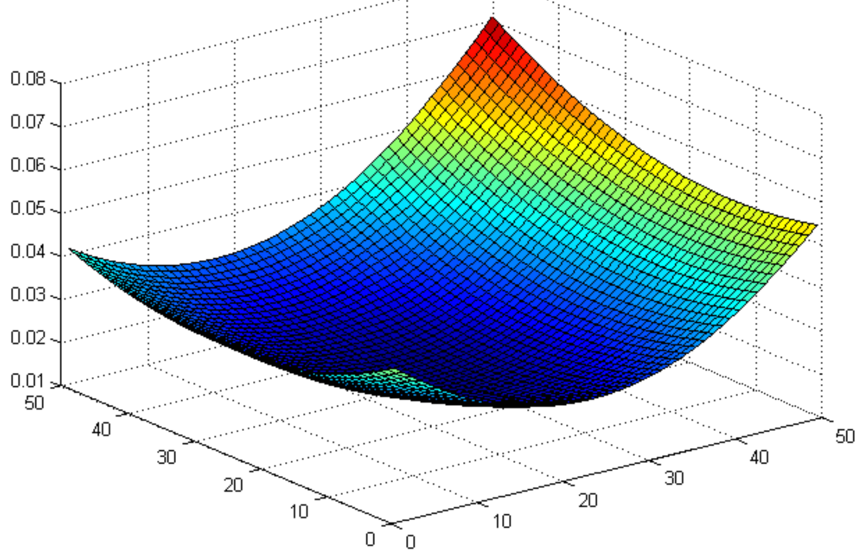
where

- x is a vector of n real parameters
- $f_i(x) = model(x, t_i) - measurement(t_i)$

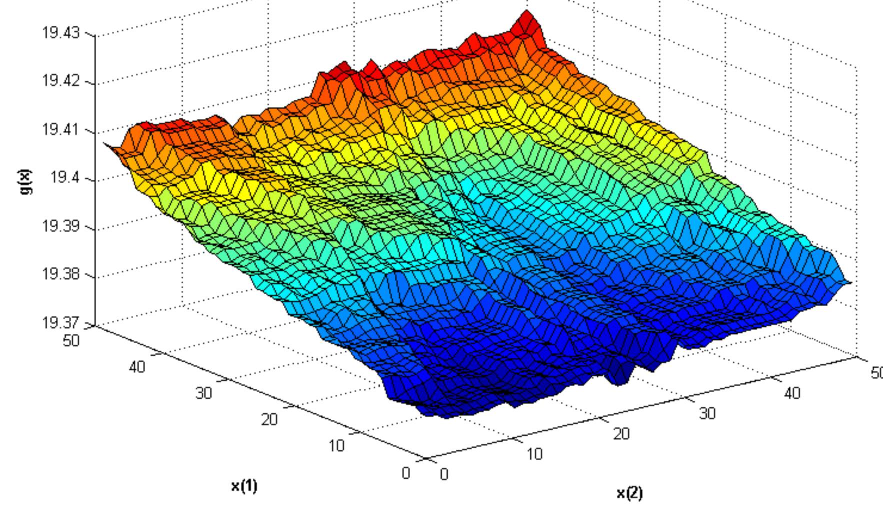
A problem with chip simulation

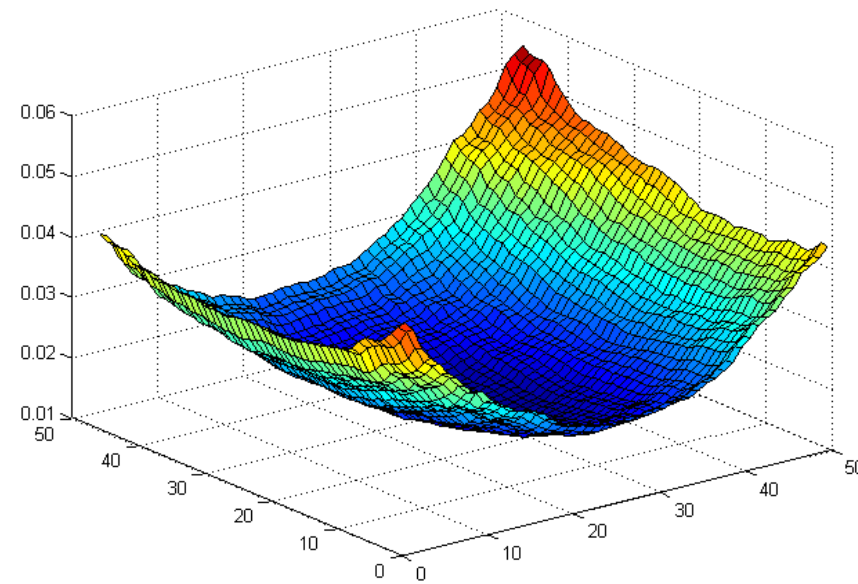
goal function: Simulink model





chip simulation

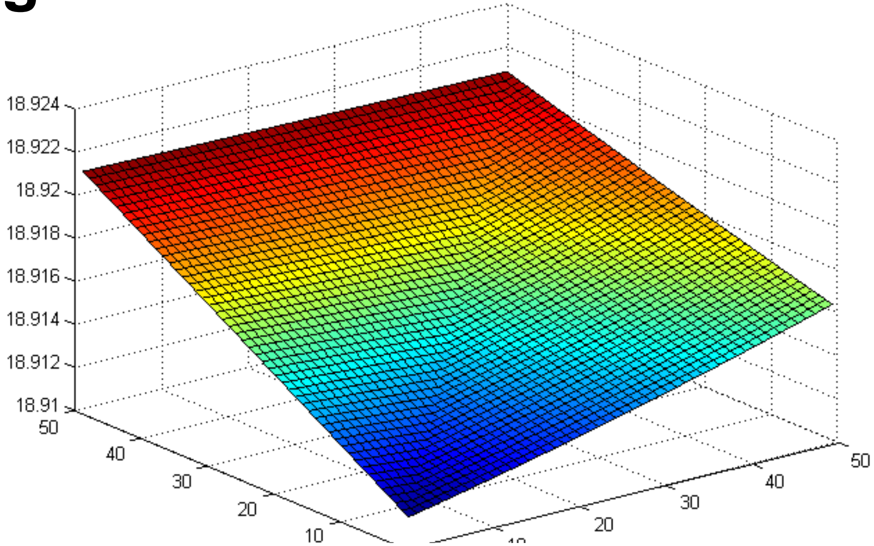


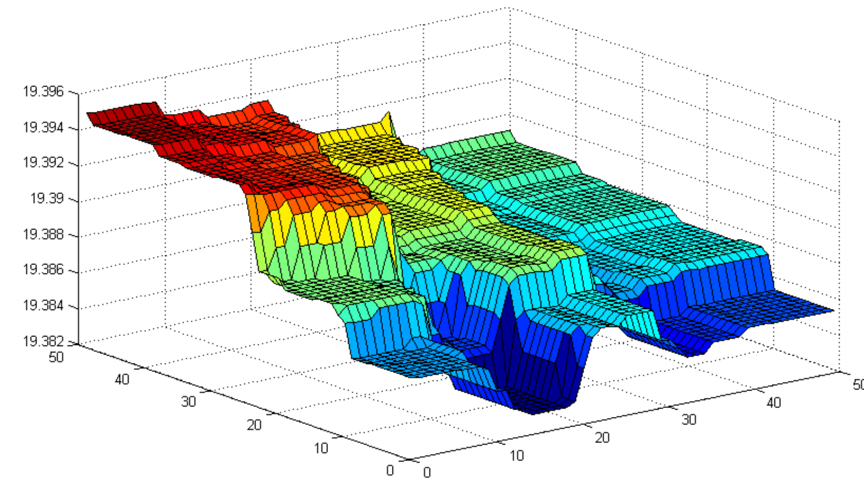


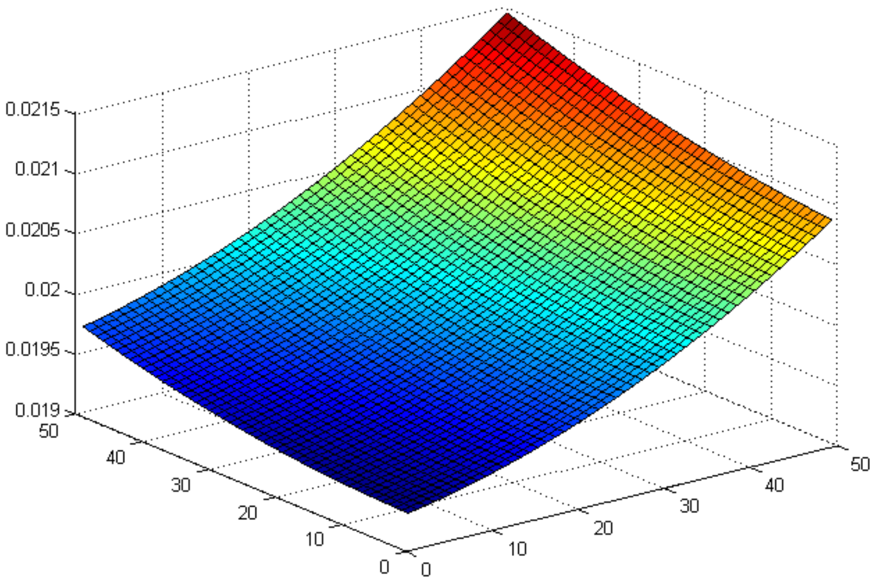
A problem with chip simulation

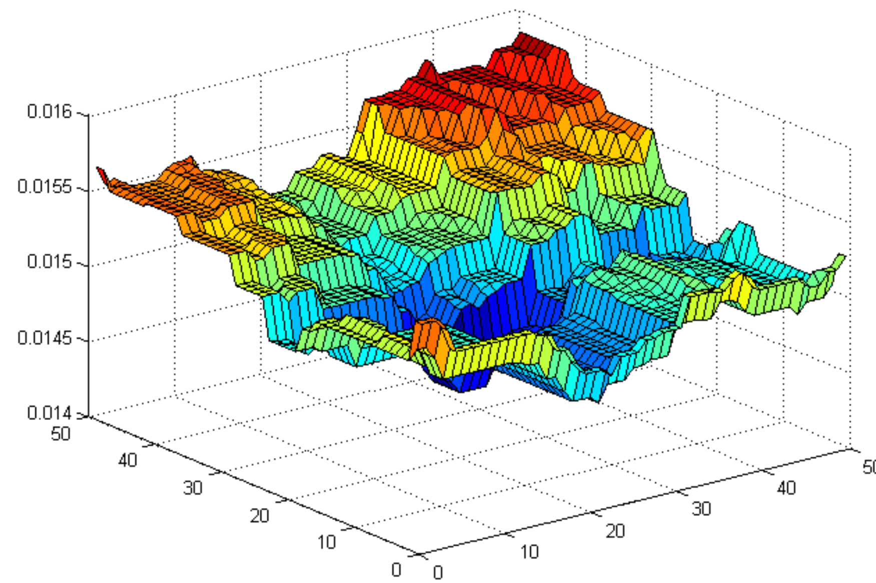
goal function: Simulink model

chip simulation









A problem with chip simulation

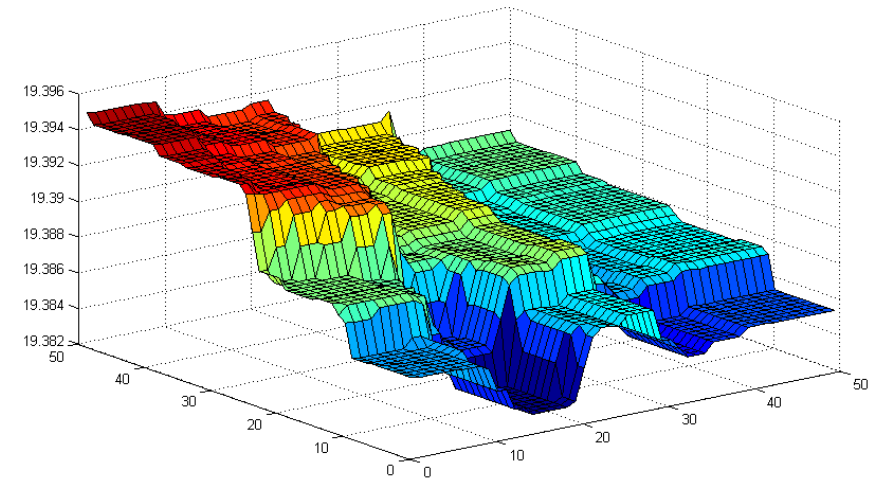
Optimization methods often require **gradients** to guide search

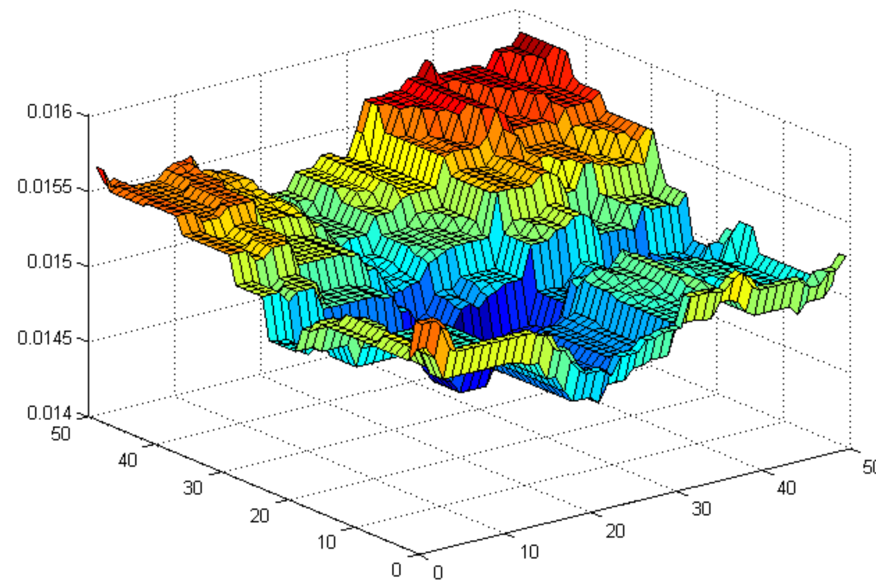
Engine control often implemented using **fixed-point integer** code

→ gradients of the goal function are zero (or undefined)

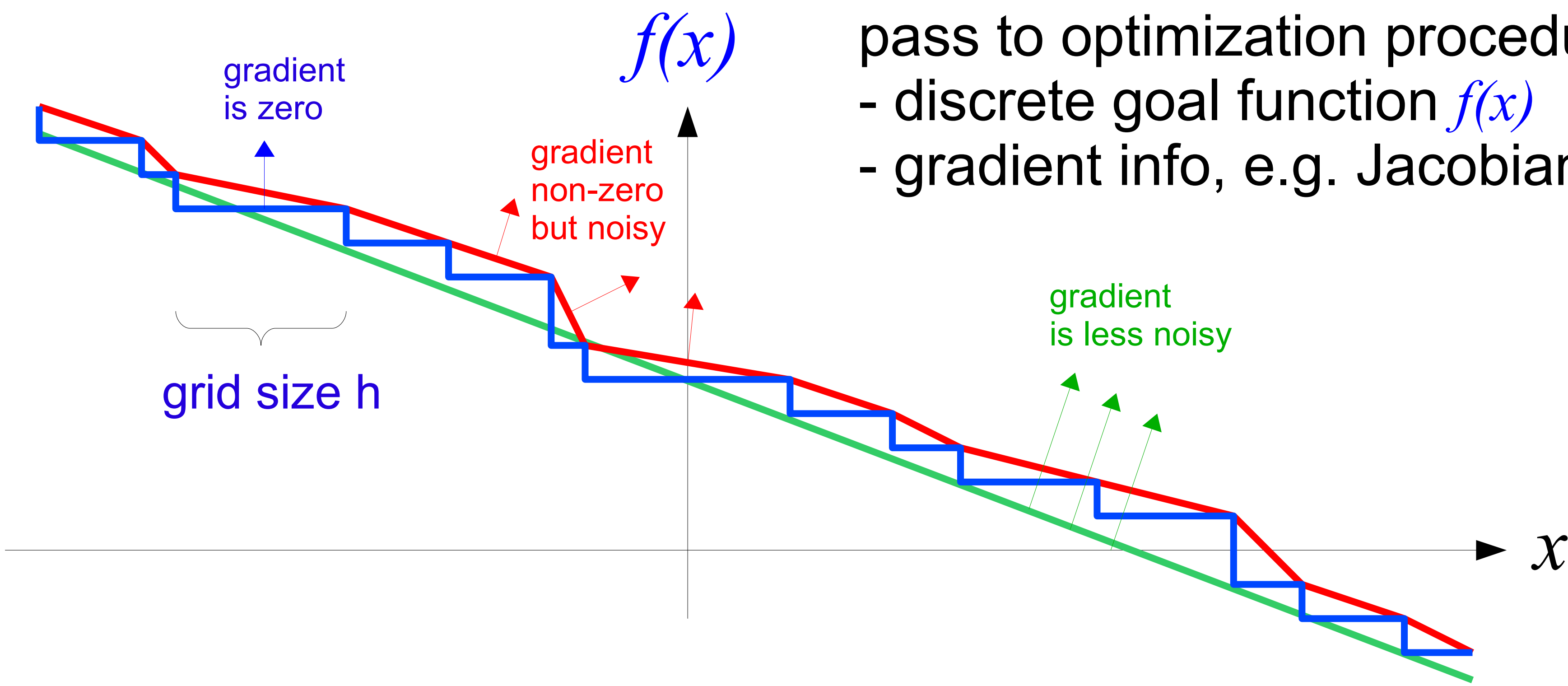
→ no guidance

→ optimization terminates early at local optimum





Idea 1: construct a smooth goal function



pass to optimization procedure
- discrete goal function *f(x)*
- gradient info, e.g. Jacobian *Jij*

f(x) goal function implemented using chip simulation: zero gradient

f(x) use current grid size h to compute gradient

$$\frac{f(x+h)-f(x)}{h}$$

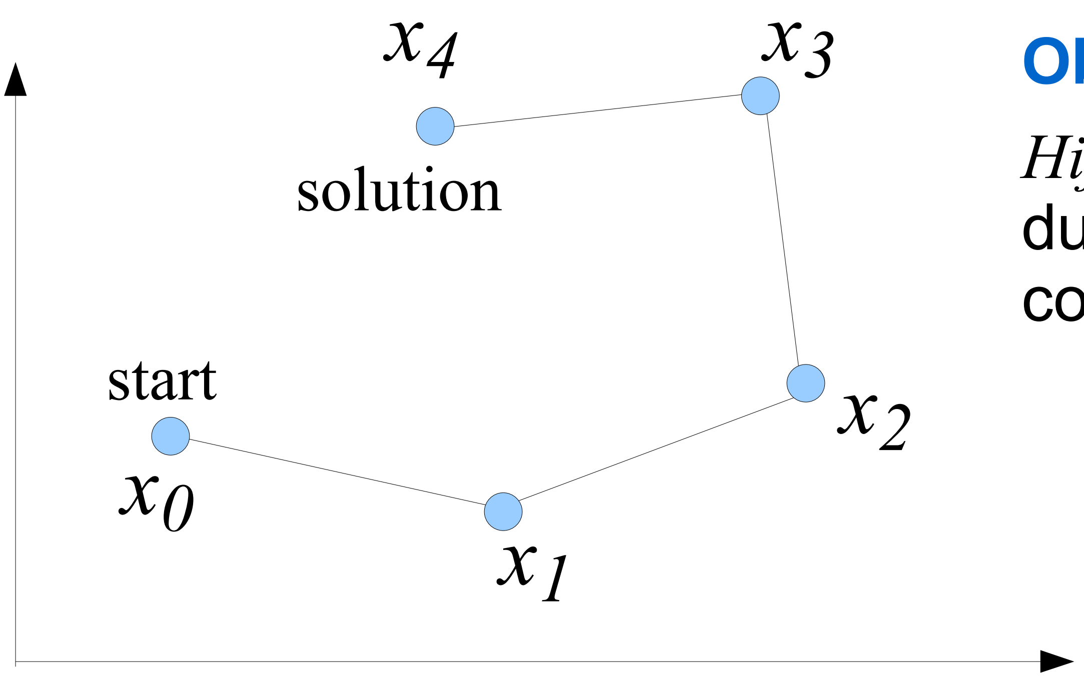
f(x) less noise: use 10 h to compute the gradient

Idea 2: pre-compute grid sizes

m time points, n parameter

→ m x n matrix Hij of grid sizes

→ must be computed at each step $x0$, $x1$, $x2$, ... expensive!



Observation

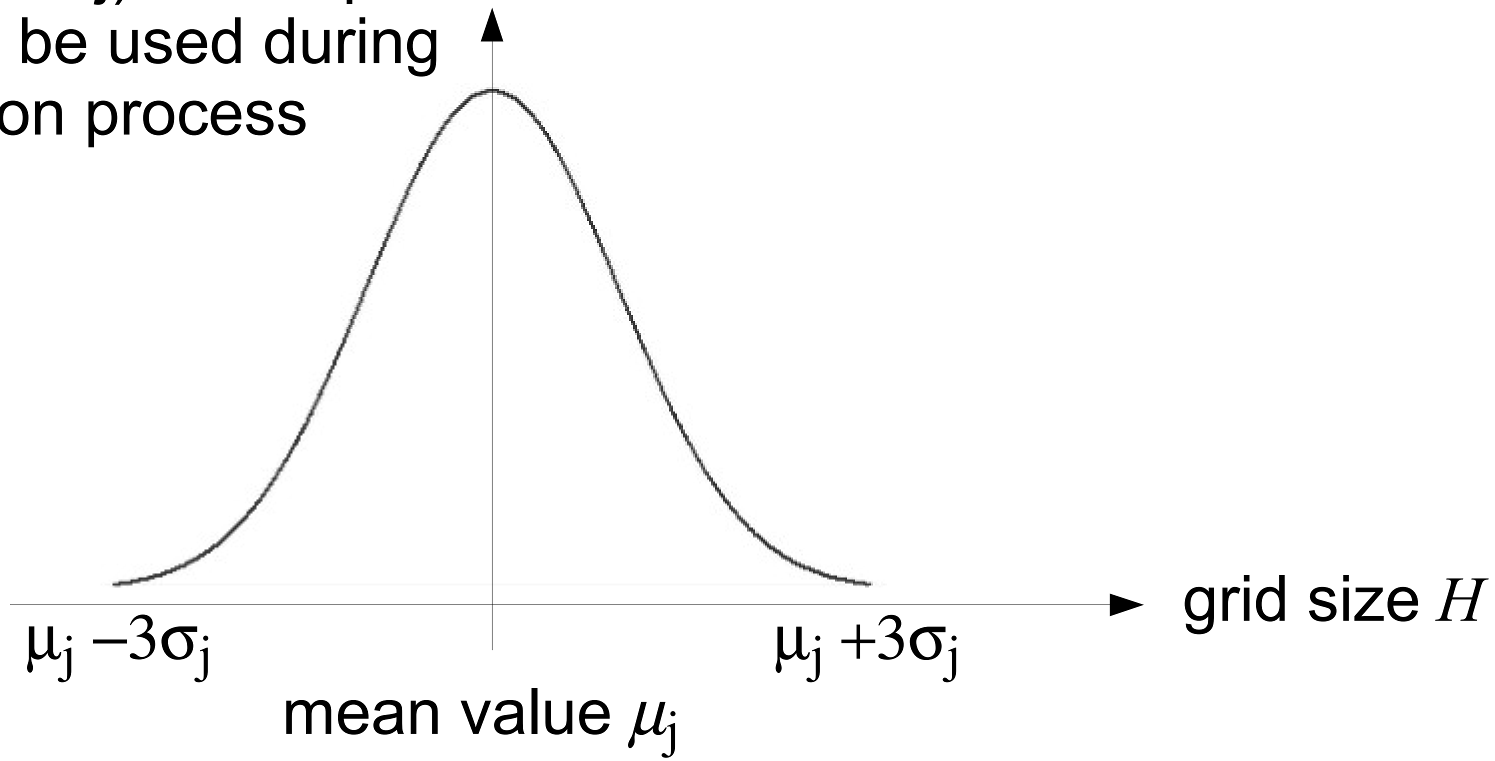
Hij does not change much during the solution process: compute only for $x0$ and reuse

Idea 3: Stochastic model of grid sizes

For large problems, do not compute all elements of matrix *Hij*

Use **stochastic model**: for parameter *xj*

- compute *Hij* for *x0* and some (not all) time points *ti*
- estimate **average** μj and **standard deviation** σj
- use $hj = 10\ (\mu j + 3\sigma j)$ to compute gradient of *xj* to be used during the entire solution process



Example: Tune engine model used by engine controller

- m = 202 measurement time points
- n = 20 parameters
- solver: **lsqnonlin** from MATLAB optimization toolbox
- goal function implemented using chip simulation
  - gradient info passed using option FinDiffRelStep
  - stochastic model of grid sizes
- performance validated against hand-coded smooth Simulink model
  - very **similar solutions** found
  - similar number of function evaluations
  - **factor 2 slower** with chip simulation, to compute grid sizes

Using Chip Simulation to Optimize Engine Control

- chip simulation can be used to port ECU functions to PC
- the resulting model
  - runs much faster than real time
  - can be coupled with optimization procedures to automate engine calibration
- derivative-free optimization: no problem
- otherwise: compute gradient as finite difference with controlled step size