Software Development in the Laboratory

Igor Grešovnik, December 2010

Center odličnosti za biosenzoriko, instrumentacijo in procesno kontrolo
Laboratorij za krmilne sisteme
www.cobik.si

www.ung.si/si/raziskave/vecfazni-procesi/

This is presentation of ideas for organization of software development within the Laboratory for Multiphase Processes (University of Nova Gorica) and Laboratory for Control Systems (COBIK).

My Responsibilities

Optimization methods

Control shell for numerical models

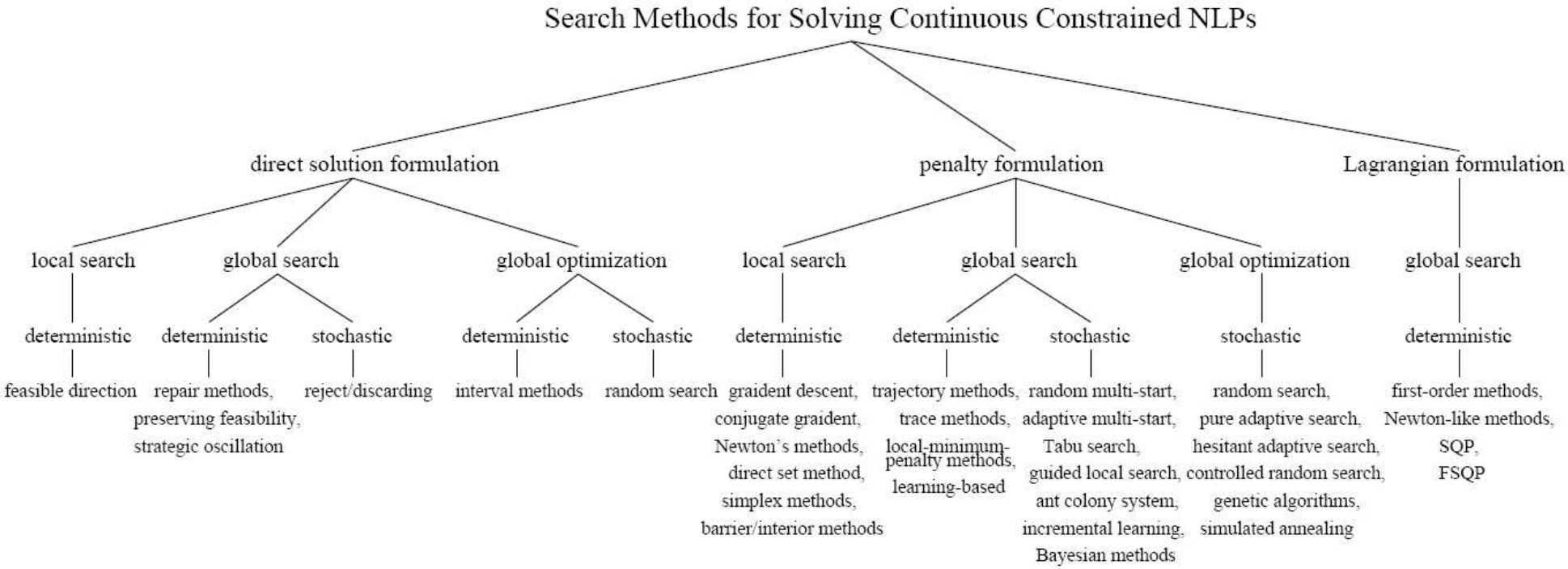
Coordination: Integrated Optimization Platform

- **Integration of optimization & simulation software for use in academic & industrial enfironment**
  - COBIK: production of fullerenes
  - University: Steel production (continuous casting, rolling, etc.), project with Štore Steel

Optimization Problems – Formulation & Algorithms

minimise	$f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n$
subject to	$c_i(\mathbf{x}) \le 0, \ i \in I$
and	$c_j(\mathbf{x}) = 0, \ j \in E,$
where	$l_k \le x_k \le u_k, \ k = 1, 2, ..., n$.

- **Classification of optimization algorithms:**



Inverse Identification of Model Parameters

Concept:

- **Perform laboratory & industrial measurements**
- **Prepare numerical models of these tests, some parameters unspecified**
- **Minimization of discrepancies between measurements and model results -> parameter estimates**

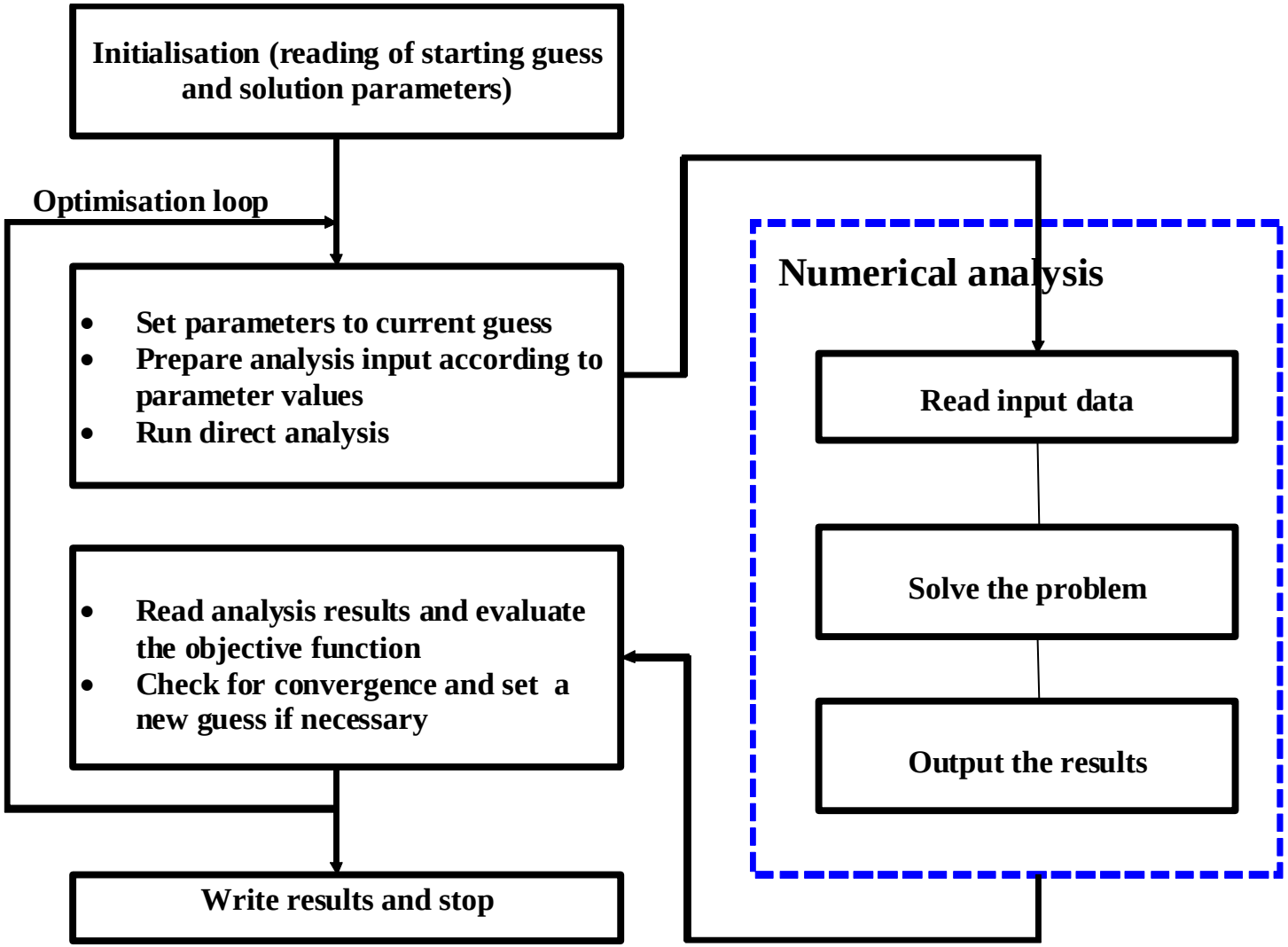
Numerical difficulties:

- **Long computational times**
- **Noise in model results**
- **Non-availability of derivatives w.r. parameters**

Engineering problems:

- **Complexity of industrial systems** (variability of process conditions, complex interactions)
- **Limited set of affordable measurements**
    - Insufficiency of data for solution of identification problem

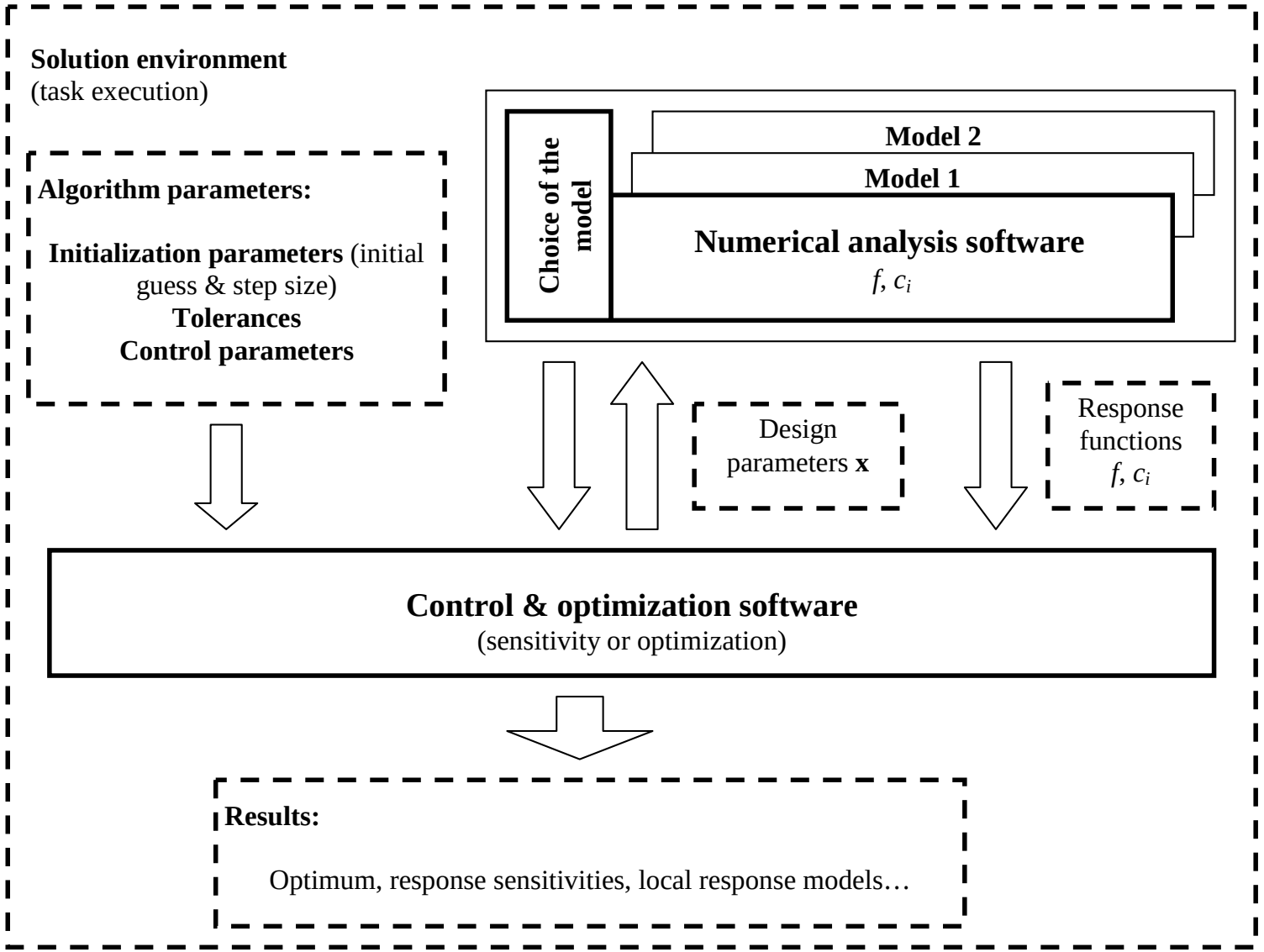
Optimization Problems – Solution Scheme



Optimization Problems – Solution Scheme

1. Take current optimization parameters
2. Prepare numerical model according to parameters
3. Run numerical simulation of the process
4. Extract the relevant quantities from simulation results
5. From measured data
   - Read result file
   - Extract relevant data
6. Calculate the response functions and eventually their gradients (in our case the discrepancy function *f*)
7. Store the response functions in output arguments and return

Integrated Optimization Platform



File Format for Data Exchange

Analysis input:

```
{ { p1, p2, … }, { reqcalcobj, reqcalcconstr, reqcalcgradobj,
      reqcalcgradconstr }, cd }
```

Analysis output file:

```
{
  { p1, p2 ... },
  {
    calcobj, obj,
    calcconstr, { constr1, constr2, ... },
    calcgradobj, { dobjdp1, dobjdp2, ... },
    calcgradconstr,
    {
      { dconstr1dp1, dconstr1dp2, ... },
      { dconstr2dp1, dconstr2dp2, ... },
      ...
    },
    errorcode
  },
  { reqcalcobj, reqcalcconstr, reqcalcgradobj, reqcalcgradconstr }
  < , { ind1, ind2, ... }, { coef1, coef2, ... }, defdata >
}
```

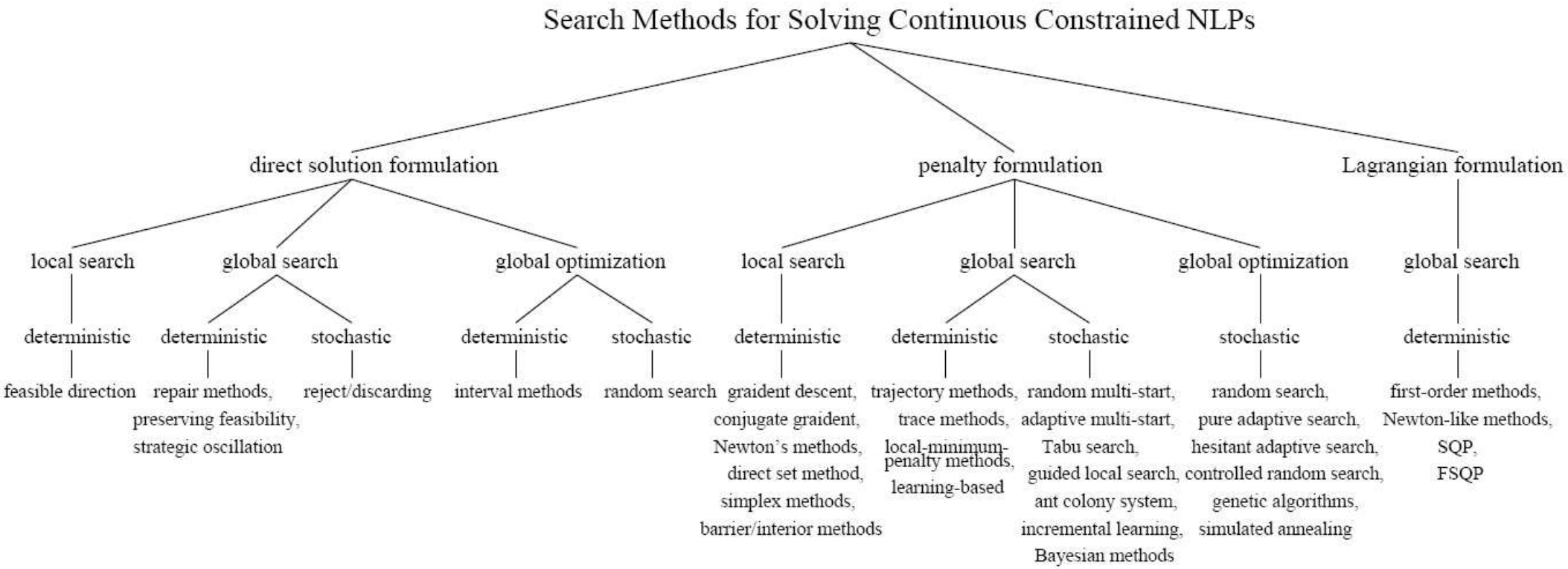
Popular Optimization Algorithms: Constrained

- **Transformation to unconstrained problems**
  - Penalty methods
  - Lagrangean methods (eliminate constraints by Lagrange multipliers)
  - Projection methods (feasible set)
- **SQP**
  - Newton method for 1st order conditions for a local minimum
  - QP subproblem
  - Feasible set method
  - Sequentially solve unconstrained problems by BFGS
  - Superilinear convergence

Popular Optimization Algorithms: Global Optimisation

- **Predominantly stochastic approaches**
  - Simulated annealing, genetic algorithms, particle swarm method, differential evolution
- **Robust in terms of response requirements**
  - Perform on discontinuous, multimodal functions
- **Inefficient in terms of required number of function evaluations**
- **Notion of "global algorithm" is asymptotic**
  - When number of iterations goes to **infinity**, the **probability** of "missing" the neighbourhood of a global optimum tends to 0
- **Even if not "global" in practical sense, these methods less easily get stuck in a local minimum**
- **Notion of local convergence is difficult to define**
- **Incorporate heuristics often taken from nature**
  - Biological evolution, swarm intelligence, statistical mechanics, self-organization of dynamical systems
- **Performance typically increased by tuning a number of control parameters (case dependent)**
  - The "No free lunch theorem"

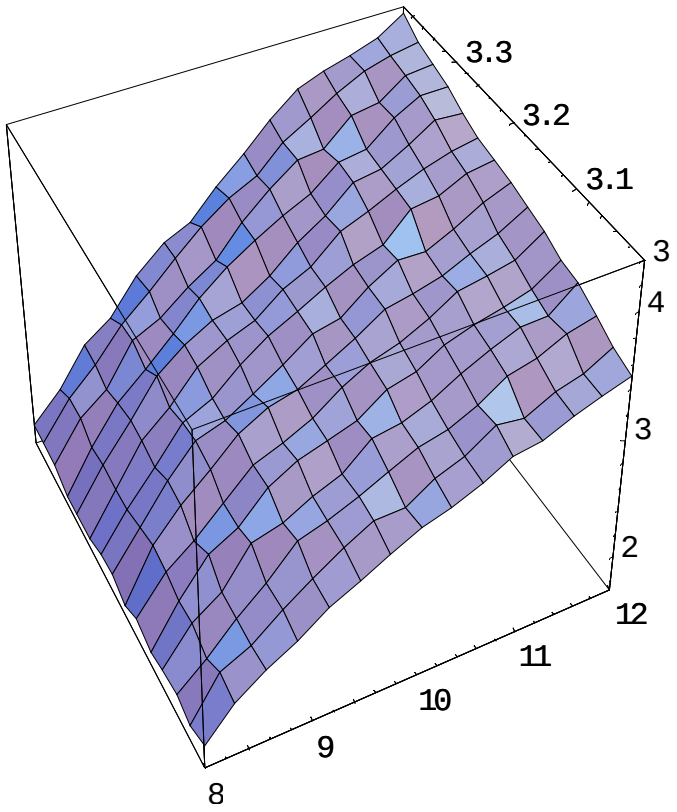
Classification of Optimization Algorithms



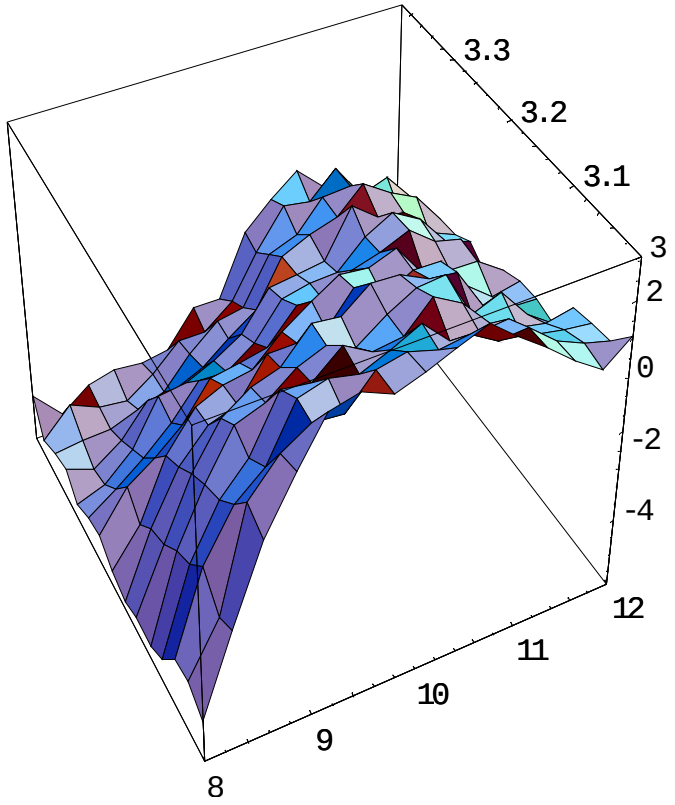
Response functions with noise

a) objective function

b) constraint function

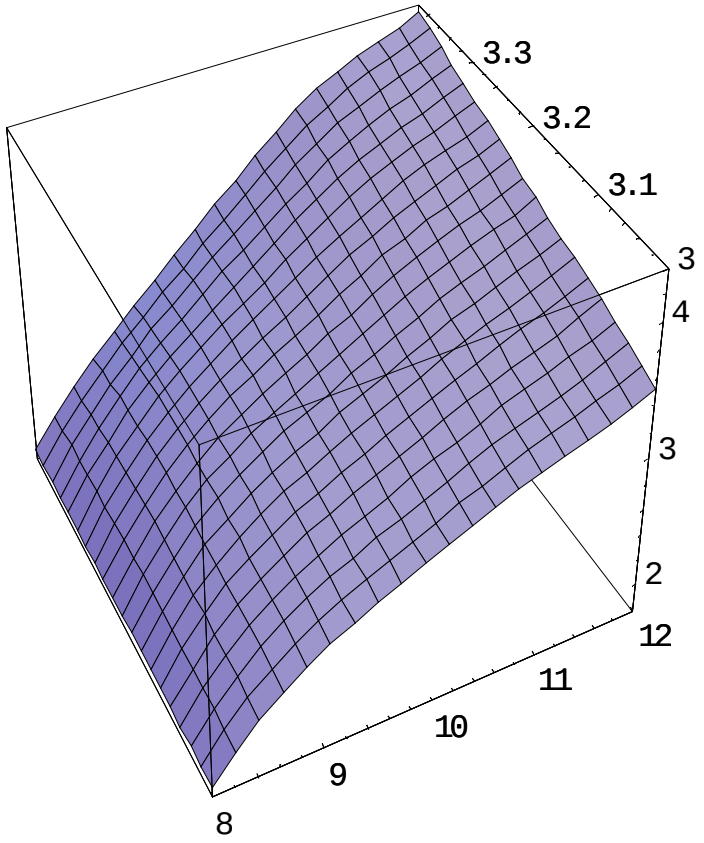


a)

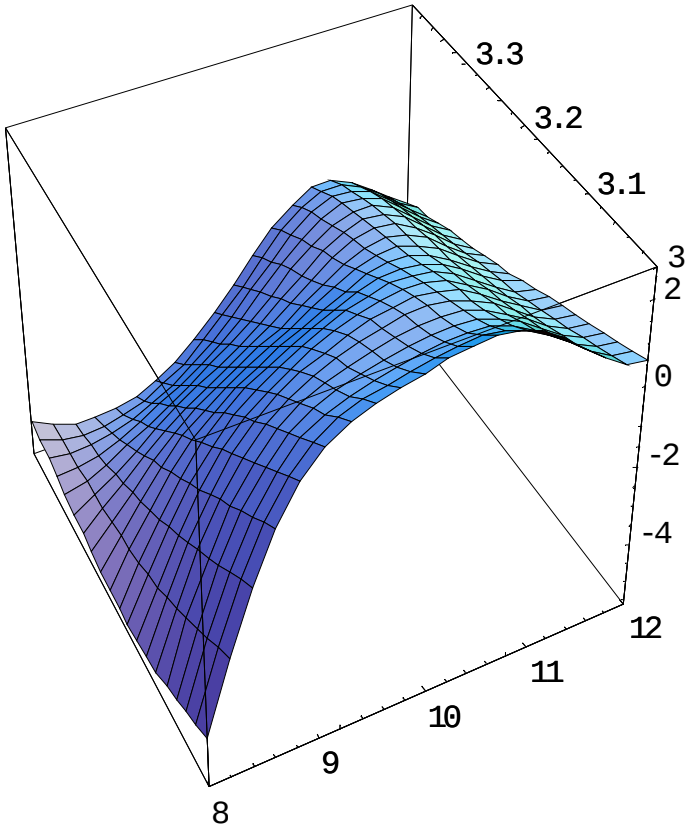


b)

Response Smoothing (Global, MLS Approximation)

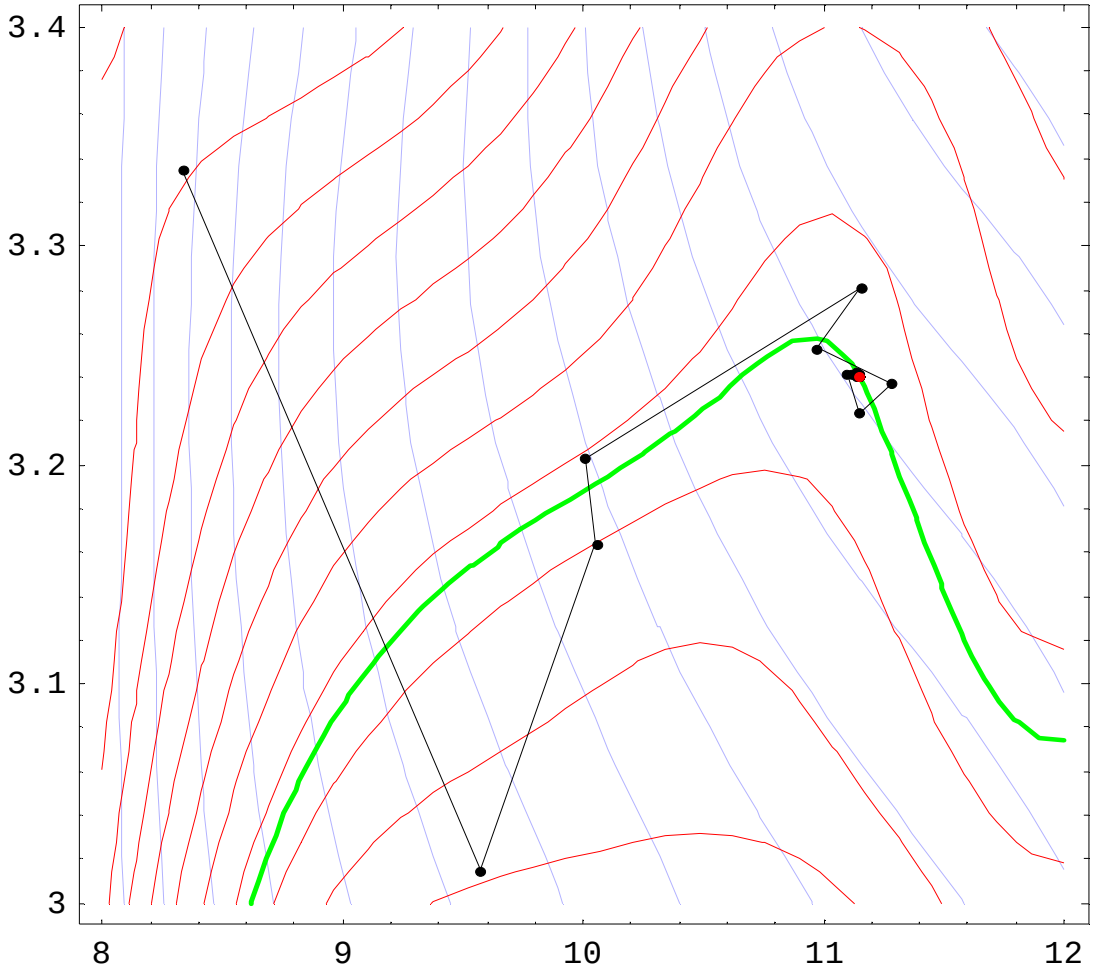


a)

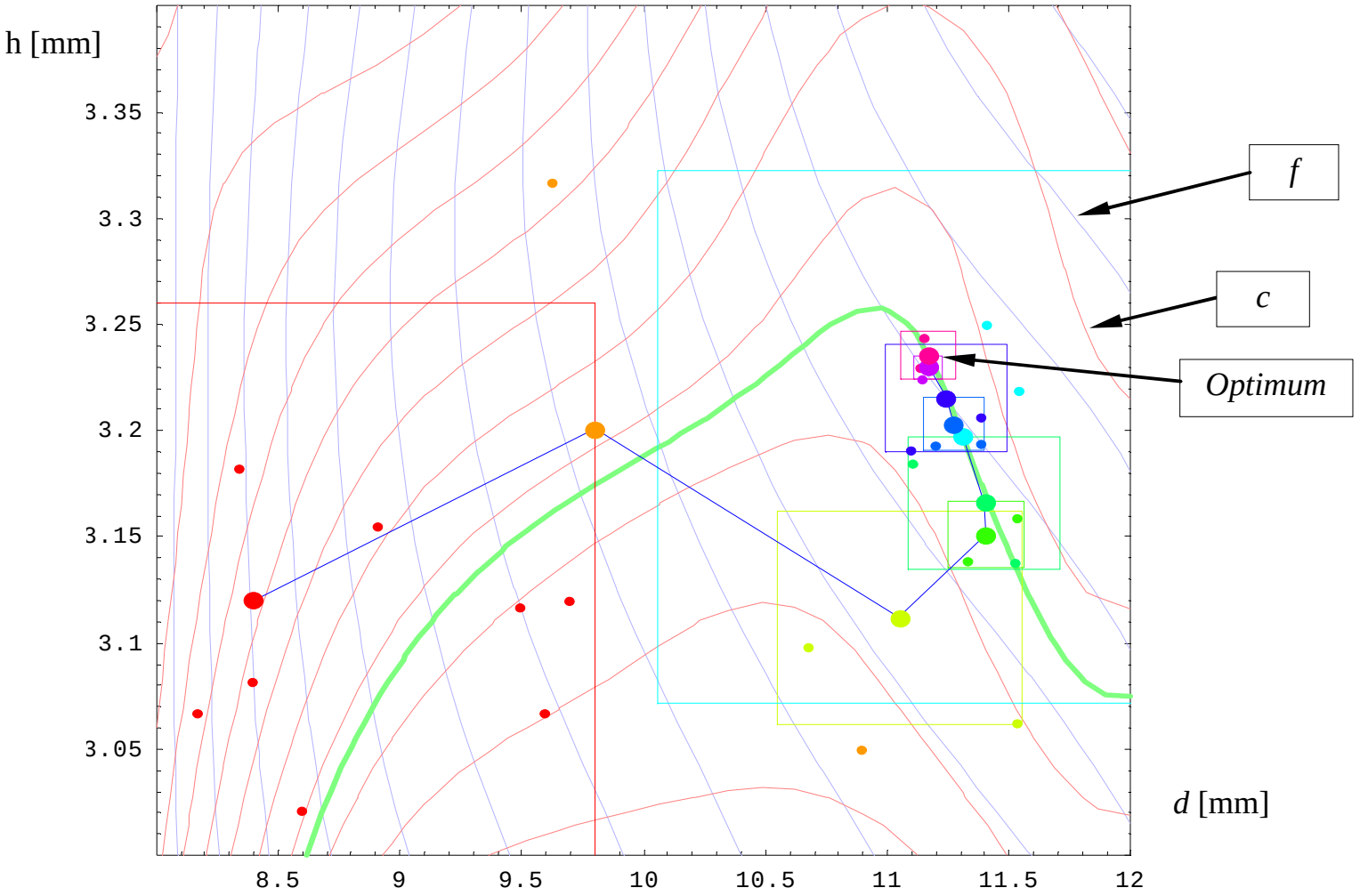


b)

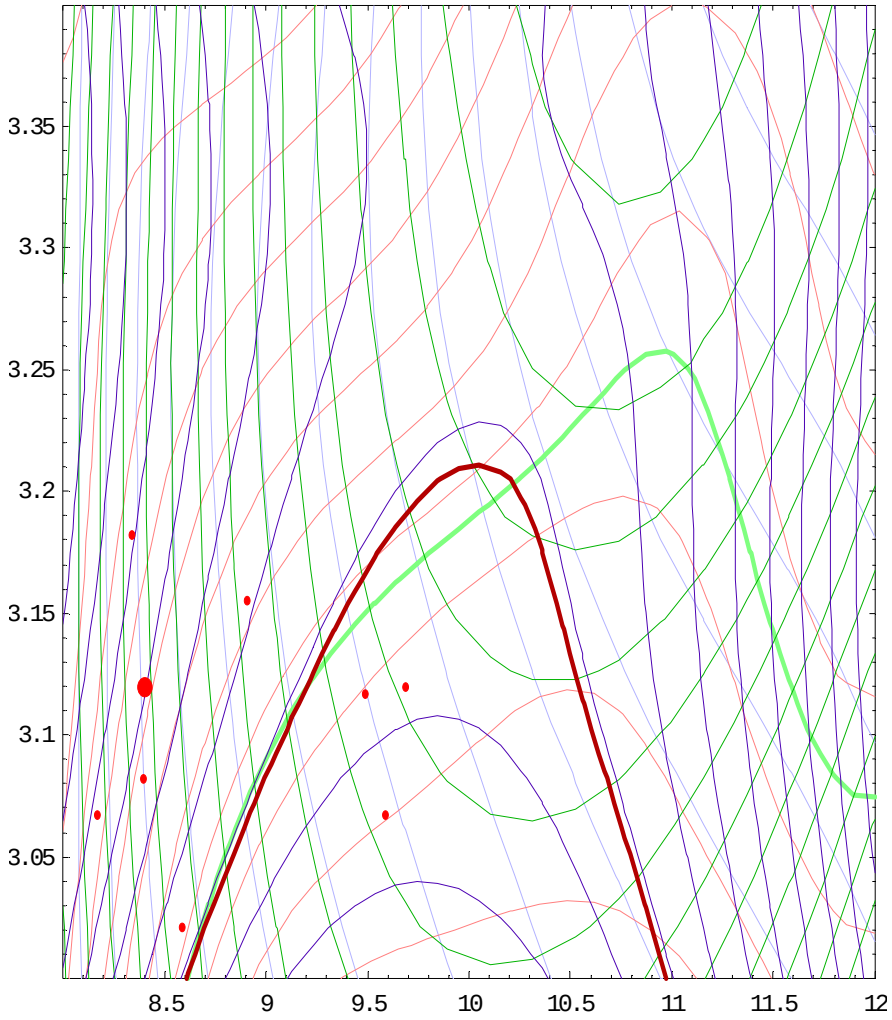
Optimization of Approximated Response (SQP)



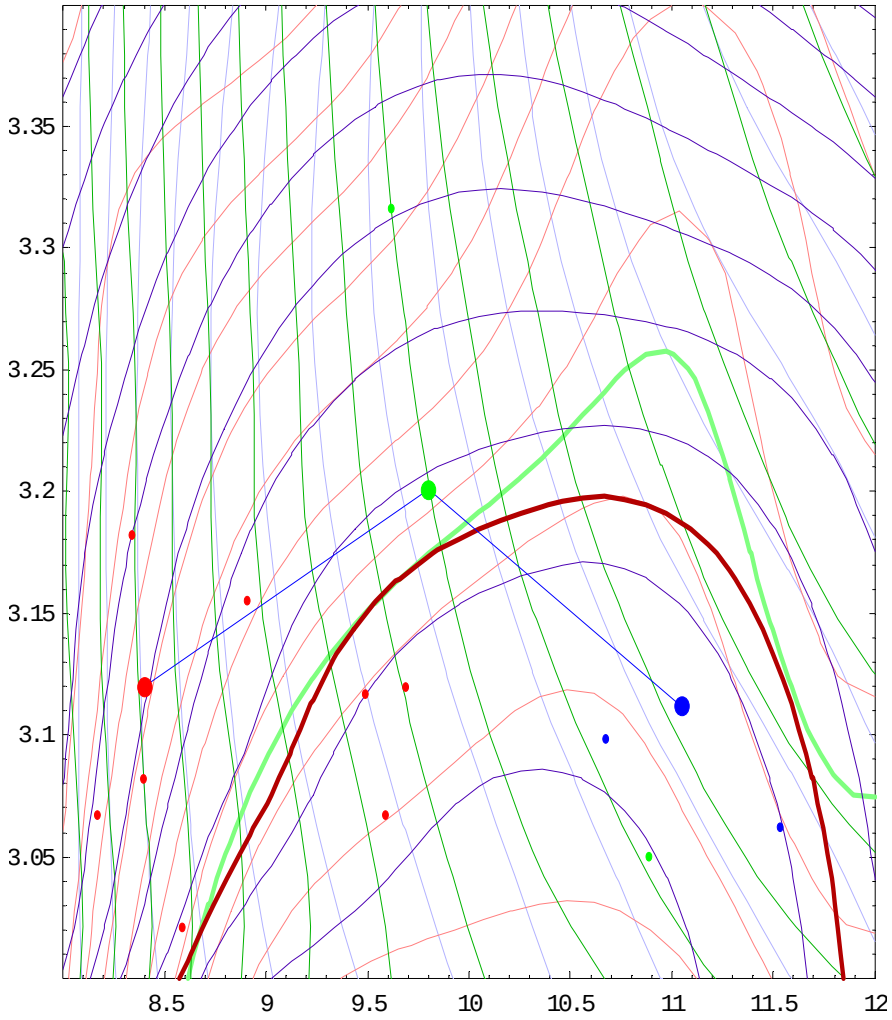
Feasible Alternative: Optimization Based on Successive MLS Approximations of Response Functions with Restricted Step Approach



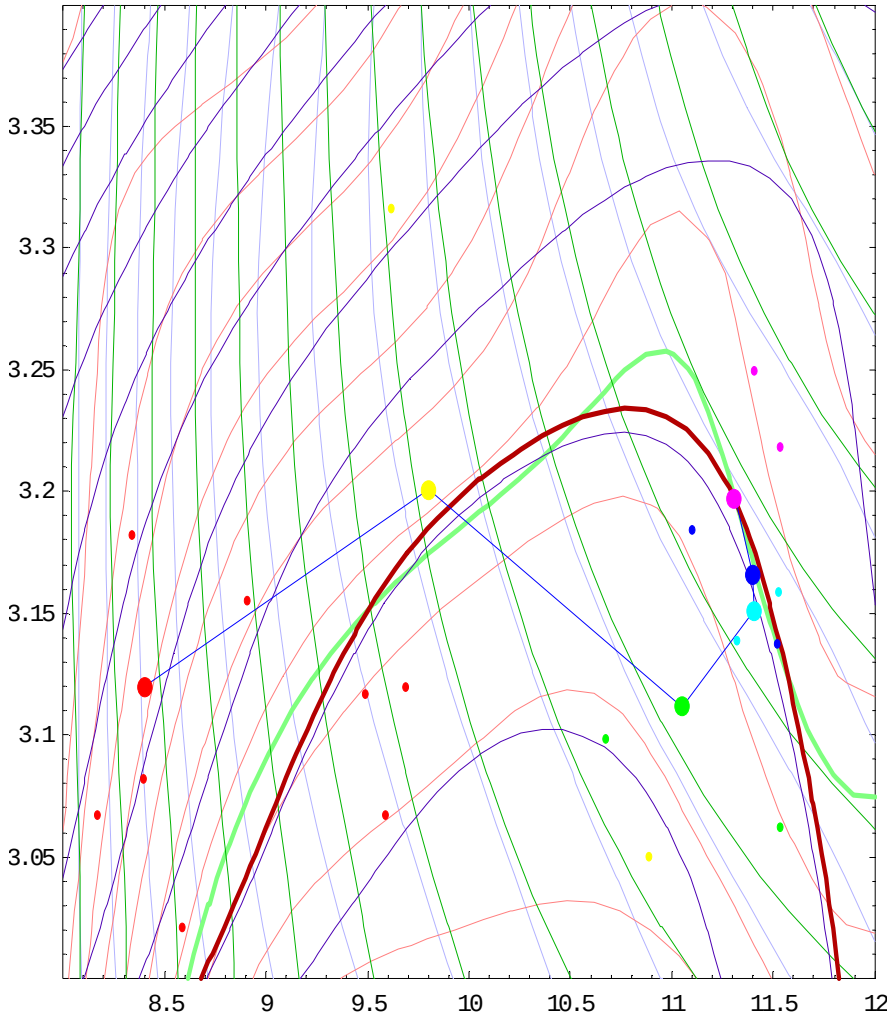
Approximated response for iteration 1



Approximated response for iteration 3

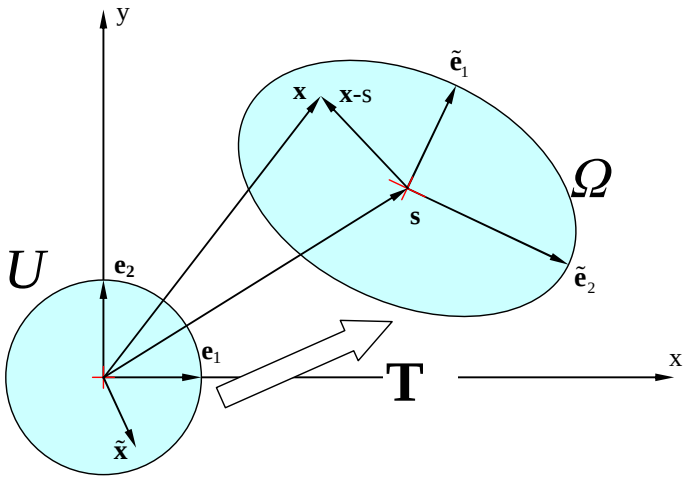


Approximated response for iteration 6

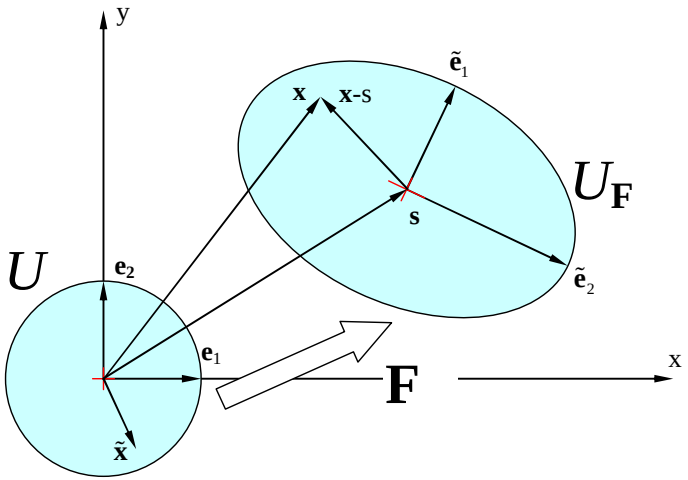


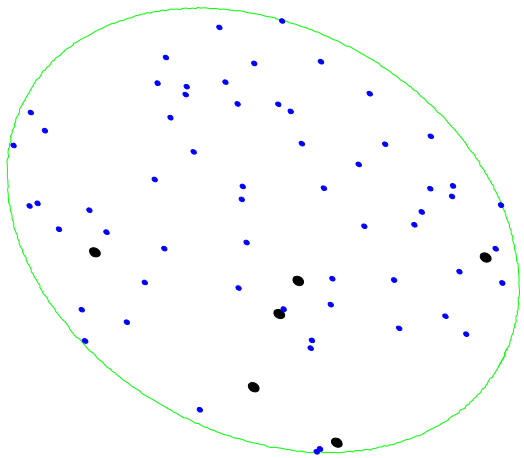
Building Blocks: Restricted Step Constraint

- **Restricts the feasible region to a neighborhood of the current guess**
- **Unit ball constraint:**
  - $c_U(\tilde{\mathbf{x}}) = \|\mathbf{x}\|_2 - 1 \le 0$
- **General form:**
  - Obtained from unit ball constraint by affine transformation of co-ordinates:
  - $c_{rs}(\mathbf{x}) = \|\tilde{\mathbf{A}}^{-1}(\mathbf{x} - \mathbf{x}_0)\|_2 - 1 \le 0$

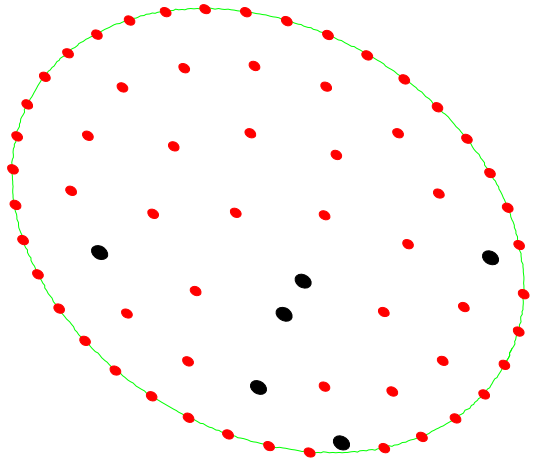


Design of Experiments





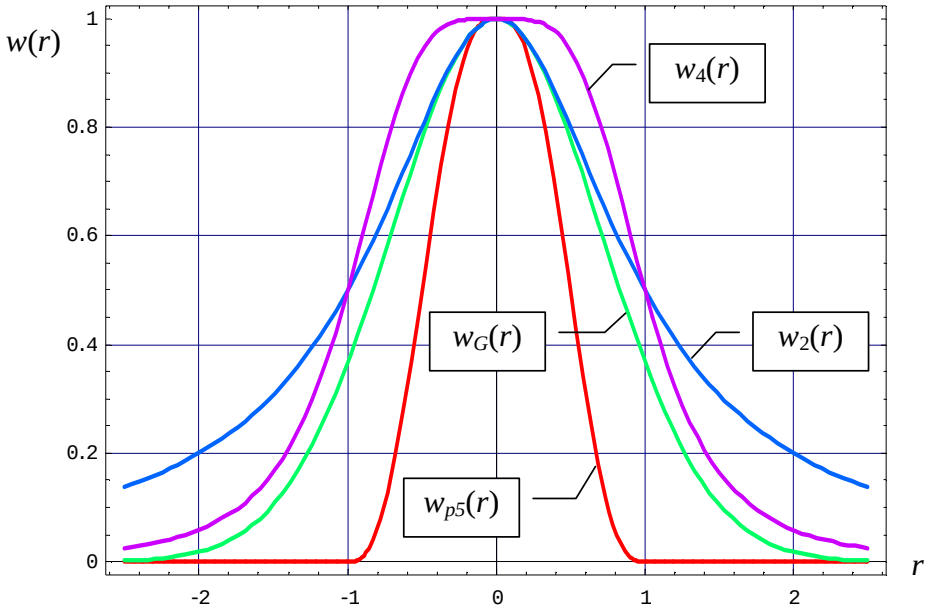
a)



b)

Building blocks: Weighting Functions for Approximations

- **Different forms of 1D weighting functions** $w(r)$
- **Multivariate weighting functions obtained by affine maps:**



Integration in algorithm scheme

- **To build adaptive approximations of response functions**
- **To solve approximated sub-problem with restricted step constraint**

Why Optimization is Important?

Industrial use of simulations:

- **Improvement of Current Processes & Designs**
- **Virtual Prototyping**
  - **Optimization used in parameter identification**

Development of numerical models:

- **Experimental Validation**
  - **Inverse identification of model parameters**

PART II:

THE IDEA OF UNIFIED SIMULATION FRAMEWORK

Interactions with Simulation Development

- **Definition of response functions**
- **Data exchange**
- **Model control (e.g. coarse/fine models)**
- **Model adjustment**
  - **For smoothness of response**
  - **Differentiation of numerical models**
- **Investigation of problem characteristics**

Conclusion:

- **Optimization depends on fitness of simulation software**

Simulation Software: Current State:

- **Several separated development threads**
- **Each simulation code developed for specific purpose**
- **Remarkable achievements made in narrow areas**
- **Modularity & extensibility usually not a primary issue**

Drawbacks

- **Duplication of work**
- **Expensive to maintain**
- **Difficult to extend & increase complexity**
- **Weak development potential per software unit**
- **Short lifetime (overall consequence)**

Alternative Way: Unified Simulation Framework

Impulses:

- **Industrial demands for solving complex problems; In long term, survival of group like ours depends on ability to service industrial needs.**
  - **Multiphysics**
  - **Multiphase**
  - **Multiscale**
  - **Multibody, with contact interactions**
  - **Complex 3D geometries**
- **Academic work in the field becomes increasingly multidisciplinary**

Motivation:

- **To alleviate drawbacks of the current approach**

BUT (no free lunch!): Requires Paradigm Shift

Importance of software design & planning is radically increased

- **Doesn't mean that software development is of greater importance than mathematical background & physical & numerical models**
- **Does mean that this area was neglected by now**

Well co-ordinated team work is required

Expected Benefits

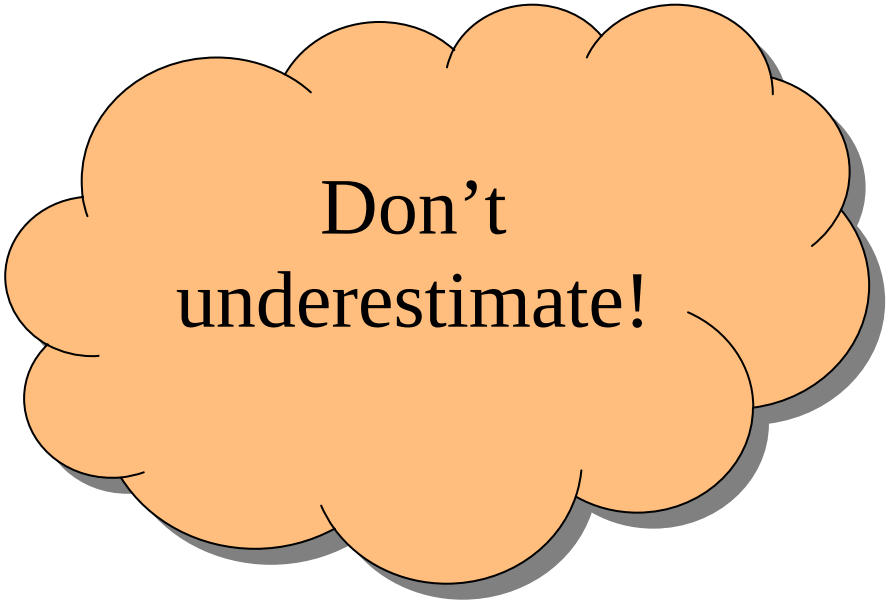
Industrial & Research Projects:

- **Increased efficiency**
- **Gain in Competitiveness**
- **More complex problems can be handled**
- **Attracts collaboration**

For Ph.D. students & researchers:

- **Well developed framework to start with**
- **Dynamic & inspiring team environment**
  - **Sharing of common tasks**
  - **Exchange of ideas**
- **Good starting point to deal with challenging problems**
- **Sharing of tasks will save a lot of work**

Things that Are Important for Team Work



- **Collaboration between team members**
- **Good communication**
- **Flow of ideas**
- **Efficient transfer of knowledge**
- **Awareness of broader context**
- **Discipline**
  - **Think how your actions will affect other team members**
  - **Number of interactions increases with team work**
- **Self-initiative**
- **Motivation of team members (since it influences all of the above)**
  - **Willingness to work in the team**

Organization of Work

- **Basic framework is set up by a small team**
  - **Simple example solved (heat conduction, then coupling of fluid flow & conduction)**
  - **Cleaning of code**
  - **Elaboration of design**
  - **Broader discussion of concepts, code review, etc.**
- **Incorporation of others:**
  - **Introductory tasks (solution of isolated problems under supervision)**
  - **Introduction to the code**
  - **Work under supervision**
  - **Independent work (but coordinated)**

Precondition: Working Environment

Workspace

- **Currently very critical, but being arranged**
  - **Promise from the Institute to get more room**

Equipment (computers, etc.)

- **Efforts to arrange this**
- **Bureaucracy in public institutions is a huge mystery (it could be employed in improbability propulsion).**
- **Strategy: In finite period of time, try to arrange solution that will do for a couple of years**
  - **I bet two beers this will succeed**

Others

- **Stable network, VPN access, Common service, system administration… must be established**
  - **We need to divide tasks**

Great Advantage: Huge Human Potential

- **We should really not waste that**
- **Enormous work can be done by such potential**
  - **We'll be having beginner's problems for some time**
- **Everybody can benefit from that**

Suggested Practices

Periodic overview meetings

- **To report on progress & current issues**
- **Draw attention to interesting things somebody has observed (conferences, software, books, projects) etc.**
- **To collect & discuss initiatives**

On-demand working meetings

- **Discuss technical issues**
- **Suggest solutions**
- **Co-ordinate actions**

Presentations on various topics

- **When doing something interesting, present to others!**
- **Cheap way to maintain a good overview**

Preliminary Task: Choice of Platform (Simulation Framework)

- **High priority (potential blocker!)**
- **Long-term importance**
- **Contribute arguments, opinions, ideas, knowledge!**
- **It concerns almost everybody**

- **Main optimization & control platform is fixed (C#)**

Choice of Platform – most relevant criteria

- Efficiency, especially in terms of CPU usage
- Availability and price
- Well elaborated language concepts suitable for development of complex applications
- Availability of numerical libraries
- Availability of suitable representation layer
  - Graphical libraries with good 3D support, suitable for representation of scientific results
  - Possibility of good integration with GUI
- Availability of basic utility libraries
  - Input/output
  - GUI building
  - Database connectivity
  - Web communication
- Availability of other libraries
- Possibility of deployment of stand-alone applications (independent of expensive packages)
- Portability

- Support (documentation, examples, etc.)
- Prospects for the future
- Popularity

- **Not everybody will be affected by all criteria**
- **All of them are important for the framework as a whole**

Programming Languages & Frameworks

Native languages

Non-object oriented:

- C
- Pascal
- Fortran

Object oriented:

- C++

Interpreted (scripting) languages

- Tcl/Tk (general purpose)
- Python (general purpose)
- JavaScript (web browsesr scripting)
- VBScript (common in MS applications)
- PHP (web server scripting)
- Mathematica (symbolic algebraic system)
- Matlab (numerical & symbolic system)

Managed languages

- Java
- .NET languages
  - C#
  - C++/CLI (managed C++)
  - Visual Basic

Current state: Main candidates are C# and C++

- **Main argument: languages developed from start for object-oriented programming (important for team work & software with high complexity)**

Main arguments against C#:

- **Slower than C++**
  - Fundamental difference: **native vs. managed**
  - Manages languages were designed to be safe (strong type checking, checking validity of operations such as array elements access)
  - May be improved in the future by different runtime modes (drop checks in well tested code) (?)
  - Use of quick “unsafe code” is also possible, but would we actually use it?
- **Availability of numerical libraries is not as good as with C++, especially open source**
  - Shorter tradition, developers communities are not so well established
  - Traditionally, primary target was IT
  - Quickly increasing range of commercial libraries & applications
  - Well developed & growing community of developers’ exchange
  - In many cases, this can be overcome by linking native libraries

Advantages of C#:

- **Clean, simple, well elaborated object oriented syntax**
- **Very strong language features (type safety, remoting, reflection) make many widely used common programming tasks much simpler (e.g. parallel computing, interpreter development – important for custom user routines)**
- **Elaborated helper tools for software development (GUI building, web services – automatic WSDL<=>proxy class generation, etc.) - many goodies from IT that increase developers' efficiency**
- **Automatic documentation tools available**
- **Extensive framework libraries – standardized, logically structured, simple to use (.NET)**
- **Truly platform independent (standard runtime environment takes care of that)**
  - But not all .NET libraries are implemented on Mono. Coverage is good and constantly increasing

My Personal Preference is towards C#

- **Very efficient development environment**
- **I've developed a set of base libraries for numerical applications; elaborated plan for further development, which can help developers of simulation code a lot.**
- **Most important:**
- **Programming in C++ is much more demanding (requires higher level of programming skills), particularly because of pointers**
- **My primary concern is whether we can achieve stable development with the current team, especially due to high expectations**
  - It would be much more difficult for me to combine other responsibilities with development in C++

However, we must critically weigh disadvantages of C#

- **Suggestions what else should be verified or considered! (tasks should be divided)**
- **Decision depends a lot on specific things we are likely to encounter in simulations, therefore we should engage experienced simulation people**
- **Arguments should be brought out & discussed, regardless of personal preferences. Keep the big picture in mind!**