Theory in the Computational Era

Stephen Boyd       Pablo A. Parrilo
(alphabetical order)

LIDS Paths Ahead, November 2009
What is Different Today?

- looking back at LIDS 40, 20 years ago
- what will be different 20 years from now?
- what is not (too) different: the (core) math

[we’ll focus on control; but similar stories for many other areas]
Computing power

- Moore’s law: various aggregate measures of computing ability double every 18–24 months or so

- and it’s not going to stop (GPUs, multi-core, cloud, . . .)

- 1969 → 1989 → 2009: $10^3 \times$ each step, $10^6 \times$ total
  no reason to doubt another $10^3 \times$ over next 20 years

- similar stories for sensing, networks, communications, . . .
Examples

<table>
<thead>
<tr>
<th>Computation</th>
<th>Then</th>
<th>Now (laptop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u = Kx$, $K \in \mathbb{R}^{10 \times 100}$</td>
<td>seconds</td>
<td>$\sim 1\mu s$</td>
</tr>
<tr>
<td>MPC (QP), 10 states, horizon 20</td>
<td>('80) 30s</td>
<td>$\sim 1ms$</td>
</tr>
<tr>
<td>SVD of $A \in \mathbb{R}^{100 \times 100}$</td>
<td>('77) 10s</td>
<td>$\sim 2ms$</td>
</tr>
</tbody>
</table>

- advances in raw speed; and also algorithms
Some Immediate Uses

[of massive computing power]

• dynamic simulation with detailed models
• Monte Carlo
• approximate worst-case analysis (pessimization)
• computational prototyping
• data visualization

[these are by now standard R&D tools . . . ]
What Constitutes a Solution?

[in the presence of huge computing resources]

- a formula or other ‘analytical solution’ (Black-Scholes, LQR, 2-Riccati)?
- a convex optimization problem?
- a polynomial-time algorithm?
- an algorithm that runs in 10s? (or 100μs?)
Example: Control Laws and Design Methods

- **PID**
  - design by rules/hand tuning
  - implement in analog; handful of operations

- **state-space linear control (LQR, LQG, $H_\infty$, . . .)**
  - solve AREs at design time
  - matrix-vector multiply at run time

- **LMIs/SOS**
  - solve nontrivial convex optimization problem at design time
  - run-time similar to state-space linear control

- **MPC/RHC/CLF/ADP**
  - solve nontrivial optimization problem at run time
• we like to think of these as advances in theory
• but they are enabled by Moore’s law
• each involves a theory whose time has arrived
Meerkov’s Law

written on Boyd’s whiteboard, 1990 or so

\[
\text{understanding } \times \text{ computing } = 1
\]

- “purpose of computing is insight, not numbers” (Hamming)
- computing gives numbers, not structure/architecture of solution
- computing solves problem instances, but doesn’t give intuition

all valid points; how much of a problem depends on (sub)field, application/purpose, and evolves with time
The Bad and the Ugly

- while (it doesn’t work) {tweak parameters; simulate}

- proof by matlab
The Good

- numerical experiments can suggest/motivate theory/understanding
  - phase transitions in combinatorial optimization (e.g., 3SAT)
  - $\ell_1$ minimization for sparsity
  - turbo decoding / message-passing algorithms
  - MPC/RHC

- numerical experiments/experience can re-train intuition

- theory coupled with algorithms/computation can yield far more than either alone
Why Theory is (Even More) Important

[in the presence of huge computing resources]

- theory helps define the right abstractions, frame problems
  - only after this is done can we start computing
  - abstraction needed to handle complexity

- theory helps determine the viability of a computational strategy
  - e.g., convexity in optimization; polynomial-time algorithm
Computation Alone Won’t Give You . . .

- theory gives guidance, intuition, ideas
  - e.g.: feedback; Lyapunov; DP; separation (as architecture); passivity
  - useful even when hypotheses don’t hold, models wrong

- theory helps us develop *narratives* about systems
  - concepts for back-of-envelope calculations, intuition
    (controllability, condition number, CAPM, graph conductance,
    time-frequency trade-offs, . . .)
  - simple short stories we tell our children
    (‘systems with RHP zeros are hard to control’)

LIDS Paths Ahead, November 2009
Actionable Theory

- theory with algorithmic teeth that can (and will) be applied/implemented

- what can be effectively computed is not obvious

- can actually do stuff
  - benefit of method/research is not abstract
  - can help relieve mild symptoms of analysis paralysis
  - huge help in transition, outreach

[non-actionable theory (scaling laws, negative results, performance limits, complexity analysis . . . ) is also very useful]
Blurring Discipline Boundaries

- control/estimation/ML/statistics/CS/OR . . .
  - ideas too powerful to be kept in ‘control’ (or other) subfield

- the good news: it’s an exciting world
  - lots of opportunities (many outside academia)

- if boundaries go away, do we have (need?) an intellectual home?
  - pragmatic (nurturing young talent, . . . )

- an answer: you can have a home and be worldly too
  - speak a dialect (say, control theory)
  - and high BBC applied math (SIAM Review)
Education/Training

• focus on
  – ideas (concepts, abstractions, narratives, . . . )
  – **together with** algorithmics

• recognizing and developing computation-friendly structures

• learning theory in (partially) algorithmic context far richer

• broad exposure to neighboring disciplines, application areas
Moving Forward

- need to get out more often
  - export ideas (but not in dialect)
  - see more styles, approaches, applications
  - like travel, improves us

- need to embrace the algorithmic