# Testing Shock Absorbers: Towards a Faster Parallelizable Algorithm

Christian Servin

Computational Sciences Program
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
christians@miners.utep.edu

# 1. How to Describe Shock Absorbers

- In general, we can use Newton's law

$$m \cdot \frac{d^2x}{dt^2} = f(t) + F\left(x, \frac{dx}{dt}\right).$$

- In practice, the vertical displacement and the vertical velocity are relatively small.
- Therefore, we can expand the dependence $F$ in Taylor series and keep only linear terms: $F = F_0 - b \cdot x - a \cdot \frac{dx}{dt}$.
- On the ideally smooth road, with no vertical motion $\left(x = \frac{dx}{dt} = 0\right)$, we should have $F = 0$, so $F_0 = 0$ and

$$m \cdot \frac{d^2x}{dt^2} + a \cdot \frac{dx}{dt} + b \cdot x = f(t).$$

- So, to describe the reaction of the shock absorbers to arbitrary road conditions, we must know $m$, $a$, and $b$.

## 2. How to Predict the Reaction of the Shock Absorber on the Given Force $f(t)$

- The equations are linear and time-shift-invariant:

$$m \cdot \frac{d^2x}{dt^2} + a \cdot \frac{dx}{dt} + b \cdot x = f(t).$$

- So, the reaction $x(t)$ to $f(t)$ is also linear and time-shift-invariant: $x(t) = \int R(t-s) \cdot f(s)\, ds$.
- From the above differential equation, we get

$$R(t) = A \cdot \exp(-k \cdot t) \cdot \cos(\omega \cdot t + \varphi).$$

- When we apply an impulse force, and measurement errors are negligible, we measure the following values:

$$x_i = A \cdot \exp(-k \cdot t_i) \cdot \cos(\omega \cdot t_i + \varphi).$$

- How to find $A$, $k$, $\omega$, and $\varphi$ from these measurement results?

# 3. Analysis of the Problem

- General formula: reminder

$$x_i = A \cdot \exp(-k \cdot t_i) \cdot \cos(\omega \cdot t_i + \varphi).$$

- There are many techniques for determining frequency $\omega$ – and the corresponding phase $\varphi$.
- It is also relatively easy to find maxima and minima within each cycle, i.e., at the moments $t_i$ when

$$\cos(\omega \cdot t_i + \varphi) = \pm 1.$$

- For these moments of time, we have

$$x_i \approx \pm A \cdot \exp(-k \cdot t_i) \text{ and } |x_i| \approx A \cdot \exp(-k \cdot t_i).$$

- Thus, the main remaining problem is to estimate the values $A$ and $k$ based on the corresponding values $x_i$.

# 4. How to Estimate the Values of $A$ and $k$

- *Reminder:* we know values $t_i$ and $x_i$ for which

$$|x_i| = A \cdot \exp(-k \cdot t_i).$$

- *Problem:* dependence on $k$ is non-linear.
- *Idea:* take logarithms:

$$\ln(|x_i|) \approx \ln(A) - k \cdot t_i.$$

- *Result:* this equation is linear in terms of unknowns $k$ and $z \stackrel{\text{def}}{=} \ln(A)$:

$$\ln(|x_i|) \approx z - k \cdot t_i.$$

- *Solution:* use the Least Squares Method to solve the corresponding system of linear equations.

## 5. Need to Take Uncertainty into Account

- In practice, measurements are never 100% accurate.
- Often, we only know the upper bound $\Delta_i$ on the measurement error $x_i - x(t_i)$.
- So, we only know that $|x(t_i)| = A \cdot \exp(-k \cdot t_i)$ is in $[\underline{X}_i, \overline{X}_i]$, where

$$\underline{X}_i \stackrel{\text{def}}{=} \max(0, |x_i| - \Delta_i); \quad \overline{X}_i \stackrel{\text{def}}{=} |x_i| + \Delta_i.$$

- Please note that we cut off the range at 0 from below, since the absolutely value is always non-negative.
- In general, different values $A$ and $k$ are consistent with these inequalities.
- Our objective is to find the range of possible values of $A$ and $k$.
- Thus, we arrive at the following problem.

# 6. Formulation of the Problem

- We are given the values $x_i$ and $\Delta_i$.
- Based on these values, we compute

$$\underline{X}_i = \max(0, |x_i| - \Delta_i) \text{ and } \overline{X}_i = |x_i| + \Delta_i.$$

- Our objective is to find:
  - the smallest $\underline{A}$ and the largest $\overline{A}$ values of $A$ and
  - the smallest $\underline{k}$ and the largest $\overline{k}$ values of $k \geq 0$

  among all the values of $A$ and $k$ that, for all $n$ measurements $i = 1, \ldots, n$, satisfy the constraints

$$\underline{X}_i \leq A \cdot \exp(-k \cdot t_i) \leq \overline{X}_i.$$

# 7. Case of Fuzzy Uncertainty

- We have guaranteed upper bounds $\Delta_i$ on the measurement error $x_i - x(t_i)$: $|x_i - x(t_i)| \le \Delta_i$.
- We often also have smaller bounds $\Delta_i(\alpha) < \Delta_i$ about which we are not 100% certain.
- So, for different $\alpha \in (0,1)$, we have an interval that contains $x(t_i)$ with this degree of uncertainty:

$$[x_i - \Delta_i(\alpha), x_i + \Delta_i(\alpha)].$$

- These intervals form a *fuzzy set* containing all this knowledge.
- In this case, we need to solve several interval problems – corresponding to different values $\alpha$.
- So, in the following text, we will concentrate on the case of interval uncertainty.

# 8. Simplification of the Problem

- *Idea:* we apply logarithm to both sides of the inequality

$$\underline{X}_i \le A \cdot \exp(-k \cdot t_i) \le \overline{X}_i.$$

- *Result:* constraints $\underline{y}_i \le z - k \cdot t_i \le \overline{y}_i$, where we denoted

$$\underline{y}_i \stackrel{\text{def}}{=} \ln(\underline{X}_i), \quad \overline{y}_i \stackrel{\text{def}}{=} \ln(\overline{X}_i), \text{ and } z \stackrel{\text{def}}{=} \ln(A).$$

- *Problem:* find the ranges $[\underline{z}, \overline{z}]$ and $[\underline{k}, \overline{k}]$ of values $z$ and $k$ for which, for all $i$,

$$\underline{y}_i \le z - k \cdot t_i \le \overline{y}_i.$$

- Once we have the bounds $\underline{z}$ and $\overline{z}$, we compute

$$\underline{A} = \exp(\underline{z}) \text{ and } \overline{A} = \exp(\overline{z}).$$

# 9. How This Problem Is Solved Now

- To find $\underline{z}$, we minimize $z$ under the constraints

$$\underline{y}_i \le z - k \cdot t_i \le \overline{y}_i.$$

- We also maximize $z$ and minimize and maximize $k$.
- In all these problems, we optimize a linear function under linear constraints.
- There exist efficient algorithms for solving such *linear programming* problems.
- These algorithms take time that grows with the problem size as $O(n^{3.5})$.
- While this dependence is polynomial, it still grows very fast for large $n$.
- It is therefore desirable to find faster algorithms.

# 10. Need for Parallelization

- *Reminder:* we need to find faster algorithms for solving our problem.
- *In general:* one way to speed up computations is to *parallelize* them, i.e.:
  - to divide the computations
  - between several computers working in parallel.
- *Alas:* linear programming is known to be the provably hardest problem to parallelize (to be precise, P-hard).
- *Thus:* a new algorithm is needed.
- *What we do:* in this paper, we propose a new faster and easy-to-parallelize algorithm for solving our problem.

## 11. Analysis of the Problem: Bounds for $k$

- We need to satisfy the constraints $\underline{y}_i \le z - k \cdot t_i \le \overline{y}_i$.
- If we add $k \cdot t_i$ to all three sides, we conclude that

$$\underline{y}_i + k \cdot t_i \le z \le \overline{y}_i + k \cdot t_i.$$

- The value $k$ is possible if there exists a value $z$ that satisfies all these inequalities.
- Such a value exists if and only if each lower bounds is smaller than or equal to each upper bound:

$$\underline{y}_i + k \cdot t_i \le \overline{y}_j + k \cdot t_j.$$

- Moving all the terms proportional to $k$ to the left-hand sides and all the others to the right-hand side, we get:

$$k \cdot (t_i - t_j) \le \overline{y}_j - \underline{y}_i.$$

# 12. Bounds for $k$ (cont-d)

- We need to satisfy the constraints $k \cdot (t_i - t_j) \le \overline{y}_j - \underline{y}_i$.
- When $i > j$, we have $t_i > t_j$, so $t_i - t_j > 0$, and we can divide both side by this positive value, resulting in

$$k \le \frac{\overline{y}_j - \underline{y}_i}{t_i - t_j}.$$

- When $i < j$, then $t_i - t_j < 0$, so $k \ge \dfrac{\underline{y}_i - \overline{y}_j}{t_j - t_i}$.
- A value $k$ is possible if it is $\ge$ the largest of the lower bounds and $\le$ the smallest of the lower bounds:

$$\max_{i>j} \frac{\underline{y}_i - \overline{y}_j}{t_j - t_i} \le k \le \min_{i<j} \frac{\overline{y}_j - \underline{y}_i}{t_i - t_j}.$$

- So, $\underline{k} = \max\left(0, \max\limits_{i>j} \dfrac{\underline{y}_i - \overline{y}_j}{t_j - t_i}\right)$; $\overline{k} = \min\limits_{i<j} \dfrac{\overline{y}_j - \underline{y}_i}{t_i - t_j}$.
- Please note: since $k \ge 0$, we cut off $\underline{k}$ at 0.

# 13. Finding Bounds for $z$

- Similarly, the value $z$ is possible if and only if there exists $k$ for which $\underline{y}_i + k \cdot t_i \le z; \quad z \le \overline{y}_i + k \cdot t_i$.
- By moving $\underline{y}_i$ and $\overline{y}_i$ to other side, we get

$$z - \overline{y}_i \le k \cdot t_i \le z - \underline{y}_i.$$

- Dividing by $t_i > 0$ (we can always assume that we start counting time with 0), we get $\dfrac{z - \overline{y}_i}{t_i} \le k \le \dfrac{z - \underline{y}_i}{t_i}$.
- Such $k$ exists if and only if every lower bound is $\le$ every upper bound: $\dfrac{z - \overline{y}_i}{t_i} \le \dfrac{z - \underline{y}_j}{t_j}$.
- By moving all the terms proportional to $z$ to one side, we get:

$$z \cdot \left(\frac{1}{t_i} - \frac{1}{t_j}\right) \le \frac{\overline{y}_i}{t_i} - \frac{\underline{y}_j}{t_j}.$$

# 14. Finding Bounds for $z$ (cont-d)

- We have $z \cdot \left(\frac{1}{t_i} - \frac{1}{t_j}\right) \le \frac{\overline{y}_i}{t_i} - \frac{\underline{y}_j}{t_j}$.
- When $i < j$, then $t_i < t_j$, hence $\frac{1}{t_i} - \frac{1}{t_j} > 0$.
- When $i' > j'$, then $t_{i'} > t_{j'}$, hence $\frac{1}{t_{i'}} - \frac{1}{t_{j'}} < 0$.
- So, dividing by a coefficient at $z$, we get

$$\frac{\underline{y}_{j'} \cdot t_{i'} - \overline{y}_{i'} \cdot t_{j'}}{t_{i'} - t_{j'}} \le z \le \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

- So, $z$ is possible if and only if it is $\ge$ the largest of the lower bounds and $\le$ the smallest of the upper bounds:

$$\max_{i>j} \frac{\underline{y}_j \cdot t_i - \overline{y}_i \cdot t_j}{t_i - t_j} \le z \le \min_{i<j} \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

## 15. Resulting Bounds for $z$

- Reminder: $z$ is possible if and only if:
  - $z$ is larger than or equal to the largest of the lower bounds, and
  - $z$ is smaller than or equal to the smallest of the upper bounds:

$$\max_{i>j} \frac{\underline{y}_j \cdot t_i - \overline{y}_i \cdot t_j}{t_i - t_j} \le z \le \min_{i<j} \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

- So, the resulting bounds on $z$ are as follows:

$$\underline{z} = \max_{i>j} \frac{\underline{y}_j \cdot t_i - \overline{y}_i \cdot t_j}{t_i - t_j}; \quad \overline{z} = \min_{i<j} \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

# 16. Resulting Algorithm and Its Analysis

- *Given:* the values $x_i$ and $t_i$.
- *Preliminary stage:* we compute the values

$$\underline{X}_i = \max(0, |x_i| - \Delta_i), \quad \overline{X}_i = |x_i| + \Delta_i,$$

$$\underline{y}_i = \ln(\underline{X}_i), \quad \overline{y}_i = \ln(\overline{X}_i).$$

- *Main stage:* we compute

$$\underline{k} = \max\left(0, \max_{i>j} \frac{\underline{y}_i - \overline{y}_j}{t_j - t_i}\right); \quad \overline{k} = \min_{i<j} \frac{\overline{y}_j - \underline{y}_i}{t_i - t_j};$$

$$\underline{z} = \max_{i>j} \frac{\underline{y}_j \cdot t_i - \overline{y}_i \cdot t_j}{t_i - t_j}; \quad \overline{z} = \min_{i<j} \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

- *Final stage:* we compute $\underline{A} = \exp(\underline{z})$ and $\overline{A} = \exp(\overline{z})$.
- *Computation time:* $O(n^2)$, since we need to go through all $O(n^2)$ pairs $(i, j)$, $1 \le i, j \le n$.

# 17. Our New Algorithm Is Faster than Linear Programming

- *Main stage:* we compute

$$\underline{k} = \max\left(0, \max_{i>j} \frac{\underline{y}_i - \overline{y}_j}{t_j - t_i}\right); \quad \overline{k} = \min_{i<j} \frac{\overline{y}_j - \underline{y}_i}{t_i - t_j};$$

$$\underline{z} = \max_{i>j} \frac{\underline{y}_j \cdot t_i - \overline{y}_i \cdot t_j}{t_i - t_j}; \quad \overline{z} = \min_{i<j} \frac{\overline{y}_i \cdot t_j - \underline{y}_j \cdot t_i}{t_j - t_i}.$$

- To compute each of the 4 bounds, we find $n^2$ ratios corresponding to $n \cdot n = n^2$ possible pairs $(i, j)$.
- Then, we take $n^2$ steps to find the smallest and the largest of these ratios.
- Thus, computations take time $O(n^2)$.
- For large $n$, this is much smaller than the time $O(n^{3.5})$ that is used by the linear programming algorithm.

## 18. Comment

- Our formulas assume that we have correctly estimated the bounds $\Delta_i$ on the measurement errors.
- Thus, we assume the desired values $A$ and $k$ satisfy the inequalities $|x_i| - \Delta_i \le A \cdot \exp(-k \cdot t_i) \le |x_i| + \Delta_i$.
- If we underestimate these bounds, the actual values $A$ and $k$ may not satisfy these inequalities.
- Moreover, it may be possible that no values $A$ and $k$ satisfy all these inequalities.
- This means that the corresponding intervals $[\underline{k}, \overline{k}]$ and/or $[\underline{A}, \overline{A}]$ are empty, i.e., that $\underline{k} > \overline{k}$ and/or $\underline{A} > \overline{A}$. So:
  - if we apply the algorithm and get $\underline{k} > \overline{k}$ and/or $\underline{A} > \overline{A}$,
  - this means that we have underestimated the measurement errors.

## 19. Possibility of Parallelization

- The main part of the algorithm is computing the minima and maxima of $n^2$ ratios.
- We can use $n^2$ processors to compute all the ratios in parallel – i.e., in time of one computational step.
- In general, if we have $N$ numbers $r_1, \ldots, r_N$, we need $\log_2(N)$ time to find $\min r_i$ and $\max r_i$.
- At $t = 1$, the 1st processor computes $r'_1 = \max(r_1, r_2)$, the 2nd $r'_2 = \max(r_3, r_4)$, the 3rd $r'_3 = \max(r_5, r_6)$, etc.
- At $t = 2$, the 1st and 2nd processors compute

$$r''_1 = \max(r'_1, r'_2) = \max(r_1, r_2, r_3, r_4),$$

$$r''_2 = \max(r'_3, r'_4) = \max(r_5, r_6, r_7, r_8).$$

- At $t = 3$, the 1st computes $\max(r''_1, r''_2) = \max(r_1, r_2, \ldots, r_8)$.
- At $t = s$, the 1st processor computes $\max(r_1, r_2, \ldots, r_{2^s})$.

## 20. Parallelization (cont-d)

- At $t = s$, the 1st processor computes $\max(r_1, r_2, \ldots, r_{2^s})$.
- When $2^s = N$, i.e., when $s = \log_2(N)$, then we compute the desired maximum in $\log_2(N)$ steps.
- In our case, we have $N \le n^2$ numbers, so computing the maximum takes

$$\log_2(N) \le \log_2(n^2) = 2 \cdot \log_2(n) = O(\log_2(n)) \text{ steps}$$

- Thus, the new algorithm is indeed highly parallelizable.
- To be more precise, it belongs to the class NC of all the problems that can be solved:
  - in polylog time (i.e., in time bounded by a polynomial of $\log_2(n)$)
  - on a polynomial number of processors.

## 21. Acknowledgments

The author is thankful:

- to Martine Ceberio,
- to Eric Freudenthal, and
- to the anonymous referees

for valuable suggestions.