A Fast Algorithm for Computing the Running-Time of Trains by Infinitesimal Calculus

IAROR RailRome 2011

Dr. Thomas Schank

February 17th 2011

About the Author

Dr. Thomas Schank

- **Physics** and **Mathematics**; University of Konstanz, Germany
- Ph.D. in **Computer Science**; University of Karlsruhe, Germany
- today: Swiss Federal Railways, K-IT Business Applications

Table of Contents

Use Case: Timetable Planning

Online Timetable ?

From: Bern **Date:** We, 16.02.11 « » » Calendar

To: Zürich Flughafen **Time:** 06:40 ○ Departure ⦿ Arrival

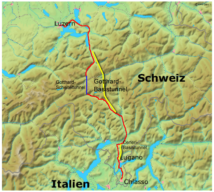
Via:

» Search connection

» Advanced search » New request » Return journey » Continue journey

Details	Station/Stop	Date	Time	Duration	Chg.	Travel with	Occupancy
1	Bern Zürich Flughafen	We, 16.02.11	dep 04:21 arr 05:30	1:09	0	IR	1. 2.
2	Bern Zürich Flughafen	We, 16.02.11	dep 04:40 arr 06:38	1:58	1	IR, S2	1. 2.
3	Bern Zürich Flughafen	We, 16.02.11	dep 05:30 arr 06:46	1:16	1	IC, IR	1. 2.
4	Bern Zürich Flughafen	We, 16.02.11	dep 05:30 arr 06:50	1:20	0	IC	1. 2.

Use Case: Infrastructure Planing



Use Case: Rolling Stock Acquisition

Definition

given

- **track** with parameters: signals, speed limits, inclination, curvature, ...
- **composition** with parameters: engine force, break force, weight, resistance, ...
- restrictions: limits on acceleration, limits on jerk, ...

result

- minimal **time required** to traverse from start to end
- location given time $s(t)$ and inverse $t(s)$
- speed at time $v(t)$
- **energy consumption**

Requirements

Basics

- correct
- precise
- reliable
- stable (small variation input $\rightarrow$ small variation output)

Timetable Optimization, Online Energy Conservation

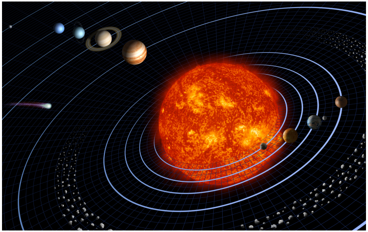
frequent (re-)evaluation $\rightarrow$ fast computation

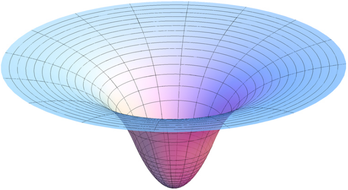
Forces and Motion

Newton

$$\vec{F} = \vec{a} \cdot m$$

Gravitational Field

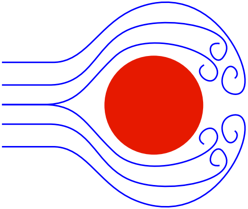


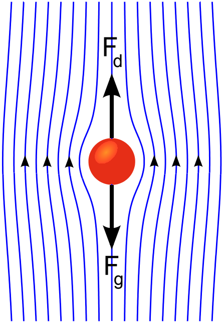


Approximation

$$F_H = m\,g \sin\varphi = h_1$$

Air Drag





Lord Rayleigh

$$F = \frac{1}{2}\rho C A v^2$$

Stokes' law

$$F = 6\pi\eta R v$$

Accumulated Forces Against Direction of Motion

more forces

- mechanical deformation
- rotation
- ...

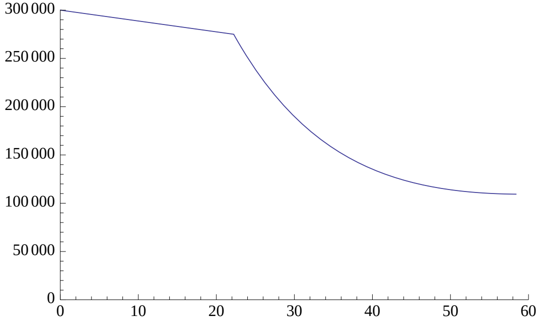
- empirical evaluation
- model with a **second order polynomial**

Resistance

$$F_R(v) = r_0 + r_1 v + r_2 v^2$$

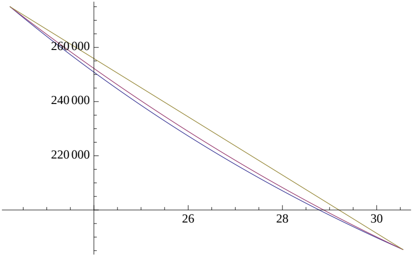
(*Strahl 1913, Davis 1926, Lukaszewicz 2001*)

Engine Force

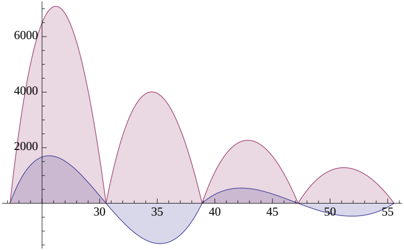


$$F_e\,[\mathrm{N}](v[m/s]) = \begin{cases} 3.\cdot 10^5 - 1.13\cdot 10^3 v & 0 \le v \le \frac{200}{9} \\ 1.72\cdot 10^4 + 1.05\cdot 10^6 e^{-6.84\cdot 10^{-2} v} + 1.25\cdot 10^3 v & \frac{200}{9} < v \le \frac{550}{9} \end{cases}$$

Engine Force - Second Order Approximation



(a) F_e original (blue), F_z quadratic (red), F_l linear (yello)



(b) differences: $F_z - F_e$ (blue), $F_l - F_e$ (red)

$$F_z[\mathrm{N}](v[m/s]) = \begin{cases} 3.000 \cdot 10^5 - 1.125 \cdot 10^3 v & 0 \leq v < 200/9 \\ 7.263 \cdot 10^5 - 2.726 \cdot 10^4 v + 3.128 \cdot 10^2 v^2 & 200/9 \leq v < 350/9 \\ 4.237 \cdot 10^5 - 1.120 \cdot 10^4 v + 1.000 \cdot 10^2 v^2 & 350/9 \leq v \end{cases}$$

Differential Equation of Train Dynamics

Combine Forces

$$F = F_R(v) + F_z(v)$$

Newton

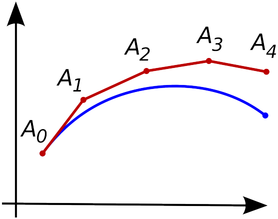
$$F = a \cdot m$$

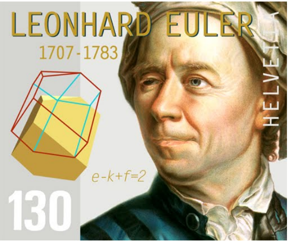
$a = \dot{v} = \frac{\mathrm{d}}{\mathrm{d}t} v$

Differential Equation of Train Dynamics

$$\dot{v} = \alpha + \beta v + \gamma v^2$$

Approximative Solutions, Euler Method





Eulers Method

- for ordinary differential equations with a given initial value
- first order approximation
- value at n is based on value at $n-1$ by linearization
- smaller the steps $\rightarrow$ better approximation
- de facto method
- standard step-width 1 second

The One Exact Solution

A Solution

let $\kappa = \beta^2 - 4\alpha\gamma$ then

$$v(t) = \frac{-\beta + \sqrt{-\kappa}\tan\left(\frac{1}{2}\sqrt{-\kappa}(t+T)\right)}{2\gamma} \tag{1}$$

is a solution for $\dot{v} = \alpha + \beta v + \gamma v^2$

(*CAS: Mathematica, Maple, ...; Brünger/Dahlhaus 2008; Wende 2003; ...*)

Existence and Uniqueness

there is a solution (obviously) and this solution is unique

(*Picard–Lindelöf 1894*)

Caveats

Complex Domain

$\kappa = \beta^2 - 4\alpha\gamma, \sqrt{-\kappa} \quad \Rightarrow \quad \mathbb{C} \to \mathbb{C}$
everything with physical correspondance is in $\mathbb{R}$

Periodic Functions

$$\left| \Re \left(\frac{1}{2} \sqrt{-\kappa} (t + T) \right) \right| < \pi/2$$

be aware of branches in inverse functions

Derived Equations

speed $\to$ time

$$t + T = \frac{2}{\sqrt{-\kappa}} \arctan \frac{\beta + 2\gamma v}{\sqrt{-\kappa}}$$

time $\to$ distance

let $\psi = \frac{1}{2}\sqrt{-\kappa}(t + T)$ then

$$s = -\frac{\beta t + 2 \ln \cos \psi}{2\gamma}$$

distance $\to$ time

use root finding, i.e. Brent-Dekker Algorithm

(*Brent 1973, Dekker 1969*)

Derived Equations

Acceleration

$$a = \frac{-\kappa \sec^2 \psi}{4\gamma}$$

Jerk

$$j = \frac{\sqrt{-\kappa}^3 \, \mathsf{sec}^2 \psi \tan \psi}{4\gamma}$$

Energy Consumption

$$W = \int \vec{F}(t) \cdot \vec{v}(t) \, \mathrm{d}t$$

Execution Time: Euler Method vs. Exact

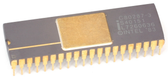
Execution Time: Euler Method vs. Exact

- euler: **many small**, simple and **inexpensive** steps
- exact: **few large** and **expensive** steps

Intuition

- exact implementation will take much longer
- consider power-series

however: FPUs



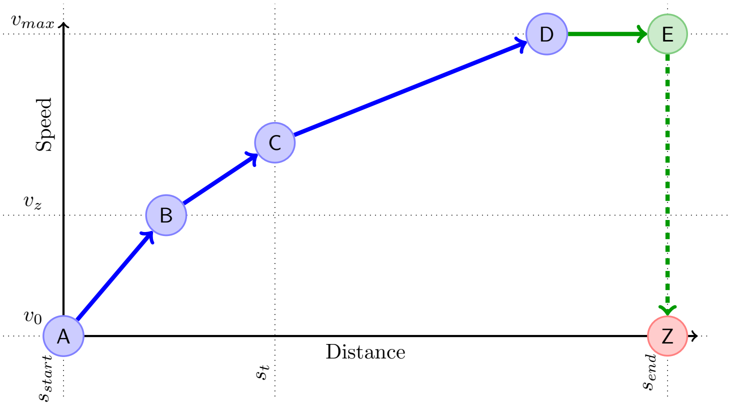
Chebyshev approximation, best uniform approximation, Padé approximation, Taylor and Laurent series with range reduction and table lookup; all in hardware

A Generic Algorithm

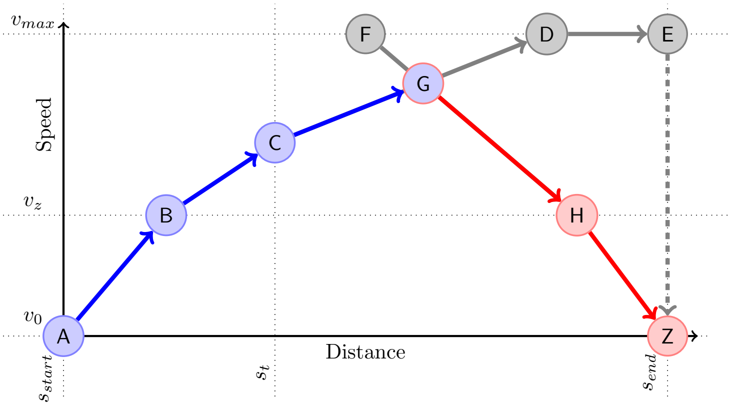
Phases

1. acceleration,
2. holding speed, and
3. deceleration.

Acceleration and Coasting



Deceleration



Why to Implement and How

Why?

1. **proof of concept**
2. **feasibility** with respect to **execution time**

Requirements

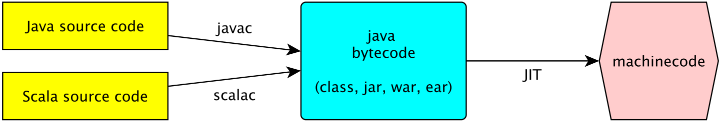
- enterprise proven platform
- must be efficient and fun to do

Conclusion

- platform: **Java Virtual Machine**
- language: **Scala**

About Scala

- Object Oriented and **Functional** hybrid language
- early adopter when starting out: one Book
- today sort of mainstream: about 20 Books, EU funding for next 5 years, used productively at twitter, ...



Why Scala - On a Superficial Level

Object Oriented Java

```
Polynomial z  = new PolynomialImp(z0,z1,z2);
z.times(2);
Polynomial r  = new PolynomialImp(r0,r1,r2);
r.negate();
r.times(3);
final Polynomial f = z.plus(r);
```

Functional Scala

```
val f =   ( - Polynomial(r0,r1,r2) ) * 3 +  Polynomial(z0,z1,z2) * 2
```

not convinced yet?

- what happens if you call `r.negate()` in Java?
- what is `f` referencing in Java anyways?

Why Scala - “meet and potatoes”

Why Scala - “meet and potatoes”

- functions as **first class values**, very good for **mathematical modelling**
- **immutability** by default, consistency and **correctness in provable sense**
- **lazy evaluation** and memoization; remember: computing $t(s)$ is (due to root finding) expensive, in particular if you don’t need it at all
- internal DSLs enable specifications as runnable code (BDD)

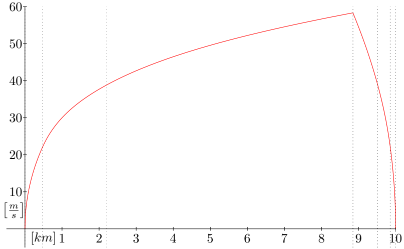
Setup: Composition and Track

Property	Value
Mass [t]	507
Rotational Mass Equivalent [t]	24.5
Brake-Force [kN]	596.6
Resistance-Parameter r_0	7122
Resistance-Parameter r_1	0.0
Resistance-Parameter r_2	13.0

Track

- 10 km, flat, straight
- speed limits: none but $v = 0$ at end
- no limits on acceleration or jerk

Computation Result



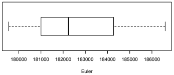
Point	Distance [m]	Time [s]	Speed [m/s]
1	0	0	0
2	481	42.5	200/9
3	2209	97.0	350/9
4	8848	230.6	58.34
5	9515	244.3	350/9
6	9853	255.3	200/9
7	10000	268.5	0

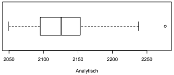
Execution Time

Result

The implementation based on the exact solution executes about 85 times faster than those based on the euler method.

	DEQ Solution	Euler	Factor
Min.	2050	179550	
1. Qu.	2095	181002	
Median	2125	182240	85.76
Mean	2129	182534	85.74
3. Qu.	2153	184273	
Max.	2276	186597	





Notes

- more parameters of the track will have a more severe impact on the exact solution
- implementation, platform and even the hardware (remember the FPU) will have an influence

however:

- factor 85 gives some way to go
- we haven't optimized our implementation in the slightest way

future:

- larger experiment with real data

Conclusion

- it is **feasible** to compute the running time of trains **exactly**
- it is actually more **faster** to do so
- **exact** solution **mitigates** the "rounding problem"
- in the context of **optimization** the exact solution is highly beneficial compared to Euler approximation

thank you!

Legal

The views and opinions expressed in this work do not necessarily represent those of my employer.