ETERNITY: FUNCTIONS, F4: Logarithm

SOEN 6011, Team C, Student N4, Kenlo DINH VAN

August, 12th ,2019

Retrospective's Summary

- Function Description.
- Important Properties.
- Project's Decisions.
- Code Review.
- Testing.

Logarithms

Quick Facts

- introduced by **John Napier** in 1614.
- **inverse** function to **exponentiation**, y = x as axis of symmetry.
- $log_b(x) = y$ can be read: **x** equals **base b** to the **power y**.
- commonly used logarithms: **binary** (b = 2), **natural** (b = e), **common** (b = 10).
- variable $\mathbf{x} > 0$ (in $\mathbf{R}$).
- base $\mathbf{b} \neq 1$ and $b > 0$ (in $\mathbf{R}$).
- co-domain: real numbers $\mathbf{R}$.

Base b

b determines the behaviour of the logarithm:

- $x \in (0; 1)$ decreasing function.
- $x \in (1; +\infty)$ increasing function.

Graph of Logarithmic Functions

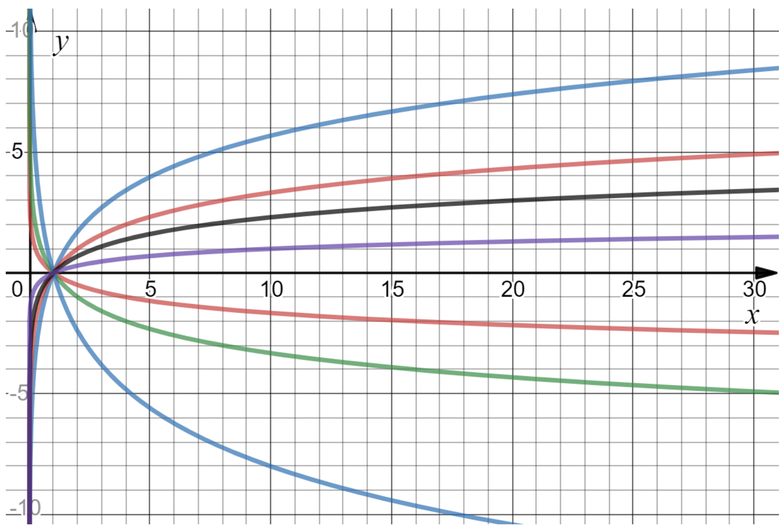

Figure 1: Graph of $f(x) = log_{\{1.5,2,e,10,0.25,0.5,0.75\}}(x)$

Important properties

Change of base:

$$\log_b x = \frac{\log_k x}{\log_k b}$$

Natural Logarithm identity:

$$\ln(x) = \lim_{n \to \infty} n\,(\,x^{1/n} - 1)$$

Those properties were crucial in implementing the logarithmic function. We first approximate ln(x), then convert it to the desired base.

Critical Decisions

1. Algorithm Selection: ln(x) approximation
Trade-offs: ease of implementation and efficiency versus accuracy. Allowed function development within the project's time constraints.

2. Coding Style: Google Java Style Guide.
Inner-team coding style uniformity, favored readable and predictable code. Wide range of tools and example available.

3. Tools Selection:
The set of tools used during this project eased the development process, as well as helped time efficiency. LaTex editor: Overleaf, VCS: GitHub. IDE: Jetbrains IntelliJ, CheckStyle, JUnit, Reviewable.

Lessons Learned

The reviewing and testing experiences of this project has allowed us to extract the following lessons.

Code Review

- Code review is simpler when **VCS** resources are properly managed, allows seamless integration with code review tools and git features. (branches, pull requests)
- Code review is more efficient when the **coding style** is uniform and predictable within a team, peer reviewers read and understand the code faster.
- Choosing the right **code review tools** allows spending less efforts in sometimes tedious tasks.
- Bad practice: sharing compressed source code. Time wasted in importing and configuring projects, absence of version history.

Testing

- **Functional Requirements** should have been formulated in concert **with the team** for homogeneous requirements (existence of common requirements, as well as better, project-wide, requirement identifiers).
- All Unit tests should have been written with the **same version** of a **unit test framework** (ex: JUnit 4 or 5), a lot of time was spent on adjusting configuration, **dependencies**.
- A **common unit testing guide line** should have been selected by the whole team. (disparate unit testing practices among team members)

Repository

All project artifacts are available on GitHub

- https://github.com/k3nlo/SOEN6011_TeamC_N4