From PLT to IR: A Code-Annotated Walkthrough

Gabriel Della Mattia
Ednacore AI

August 19, 2025

Abstract

This technical note provides an end-to-end explanation of the Pure-Load Tensor (PLT) pipeline and its connection to the Banister-style Impulse–Response (IR) model. Each mathematical step is directly mapped to its implementation in `implementation.py`, so that readers can follow the flow of information from raw training sessions to fitness–fatigue dynamics.

1 Step 1: Basic utilities

Implemented in `implementation.py`: `softplus()`, `robust_zscore()`, `scale_athlete()`.

Softplus.

$$\mathrm{softplus}(x) \;=\; \log\!\bigl(1 + e^{x}\bigr)$$

ensures daily impulses are nonnegative. (`softplus()`).

Robust z-score.

$$z_i \;=\; \frac{x_i - \mathrm{median}(x)}{Q_{75}(x) - Q_{25}(x)}\,.$$

This reduces outlier impact. (`robust_zscore()`).

Per-athlete scaling. Applies the above normalization per athlete, implemented in `scale_athlete()`.

2 Step 2: PLT configuration and feature builder

Implemented in: `compute_lambda_hrv()`, `compute_v_hrv()`, `inherit_hrv()`, `build_plt_vector()`, `aggregate_daily()`.

2.1 λ-HRV operator

$$\lambda_{\mathrm{HRV}} \;=\; \frac{1}{\Delta t_{12}} \ln\!\left(\frac{|H_1 - H_0|}{|H_2 - H_0|}\right). \tag{1}$$

Implemented in `compute_lambda_hrv()`. If H_2 is missing, a velocity proxy is used (`compute_v_hrv()`).

2.2 Intra-day inheritance

$$H_0^{(2)} \;=\; H_0^{(1)} + \bigl(H_1^{(1)} - H_0^{(1)}\bigr)e^{-\lambda\cdot \mathrm{gap}_h}. \tag{2}$$

Implemented in `inherit_hrv()`.

2.3 Session → daily vector

$$v_d = [\,\mathrm{TSS},\ E_{\mathrm{kg}},\ HR_{Q1..Q4},\ \mathrm{HRV}_{\mathrm{pre}},\ \mathrm{HRV}_{\mathrm{post}},\ \lambda_{\mathrm{HRV}}\,].$$

Implemented in `build_plt_vector()`. When multiple sessions occur, they are aggregated with weights via `aggregate_daily()`.

3 Step 3: From PLT vector to daily impulse

$$u_d \;=\; \mathrm{softplus}\big(\beta^{\top}\tilde{v}_d\big). \tag{3}$$

Implemented in `vector_to_impulse()`.

4 Step 4: Impulse–Response model

$$\begin{aligned}
F_t &= \rho_f F_{t-1} + u_{t-1}, & \rho_f &= e^{-1/\tau_f}, && (4)\\
G_t &= \rho_g G_{t-1} + u_{t-1}, & \rho_g &= e^{-1/\tau_g}, && (5)\\
\hat{P}_t &= P_0 + k_f F_t - k_g G_t\,. & & && (6)
\end{aligned}$$

Implemented in `simulate_IR()`.

5 Step 5: PMC baseline

$$\begin{aligned}
CTL_t &= CTL_{t-1} + \alpha_{CTL}(TSS_t - CTL_{t-1}), && (7)\\
ATL_t &= ATL_{t-1} + \alpha_{ATL}(TSS_t - ATL_{t-1}), && (8)\\
TSB_t &= CTL_t - ATL_t\,. && (9)
\end{aligned}$$

Implemented in `compute_PMC()`.

6 Step 6: End-to-end pipeline

The entire flow is orchestrated in `run_pipeline()`:

$$\text{Sessions} \xrightarrow{\texttt{build\_plt\_vector}} v_d \xrightarrow{\texttt{vector\_to\_impulse}} u_d \xrightarrow{\texttt{simulate\_IR}} \{F_t, G_t, \hat{P}_t\}.$$

For benchmarking, `compute_PMC()` runs in parallel.