

Phenomenological Dual Koide Relation for Down-Type Quarks: Brannen-Type Geometric Skeleton and QCD RG Dressing

Stafy Nem*
Independent Researcher

Tetsuro Eguchi
Independent Researcher

Abstract

Koide’s mass formula and Brannen’s phase-parametrized extension provide a compact empirical description of the charged-lepton spectrum. In this paper we examine whether a structurally similar ansatz can organize the down-type quark sector. The construction is presented as an empirical benchmark parametrization rather than a derived common-scale mass law: the down-type pattern is read as a *Brannen-type geometric skeleton*—a phase-parametrized square-root pattern—together with *QCD renormalization-group dressing*. Adopting $\phi_d = \frac{1}{3}$ and $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$ yields percent-level consistency with the PDG 2024 central values, comparable to the current input uncertainties.

With m_d, m_s quoted at $\mu = 2 \text{ GeV}$ and m_b at its own reference scale $\mu = m_b$, the benchmark is a mixed-scale skeleton; light-quark entries acquire a multiplicative dressing under transport to a common scale, with four-loop $\overline{\text{MS}}$ running giving a nearly common factor ≈ 0.9191 for $\sqrt{m_d}$ and $\sqrt{m_s}$ between these scales. At a common scale, the same three-mass tuple can be recast in a Brannen-type form with negligible numerical residual; we treat this as a diagnostic parametrization rather than as an independent prediction, and the residual fit parameters T_d^{fit} and ϕ_d^{fit} are kept approximately scale-independent. In this form, the inverse-Koide quantity K_{inv} —the Koide combination with each mass replaced by its reciprocal—remains close to $\frac{2}{3}$; since QCD running is flavour-universal for these masses in the approximation used here, the shape parameters of the inverse square-root tuple are kinematically preserved, and the empirical content is the proximity of the corresponding ratio parameter to $\frac{1}{\sqrt{2}}$. Following correspondence with Rivero, we further parametrize the inverse square-root tuple in the same Brannen-type form and observe that the corresponding fit parameter $T_{d\text{inv}}^{\text{fit}}$ is exactly scale-independent within the QCD-only, flavour-universal running approximation adopted in this work and lies within $\sim 8 \times 10^{-4}$ of the symmetric value $\frac{1}{\sqrt{2}}$. Whether full Standard-Model running further reduces this residual is left as an external open question and is not assumed in the present analysis. The whole construction is presented as a compact empirical organization of the down-type spectrum, not as a first-principles derivation, together with a structurally compact statement of the dual Koide relation.

1 Introduction

1.1 Koide and Brannen relations

In 1982, Yoshio Koide proposed a mass relation for the three charged leptons, based on earlier work by Harari, Haut, and Weyers [1–3]:

$$\frac{m_e + m_\mu + m_\tau}{(\sqrt{m_e} + \sqrt{m_\mu} + \sqrt{m_\tau})^2} = \frac{2}{3}, \quad (1)$$

where m_e, m_μ , and m_τ denote the masses of e^- , μ^- , and τ^- , respectively. This empirical relation reproduces the charged-lepton hierarchy at the percent level and has motivated a long line of phenomenological studies of flavour structure.

A convenient reformulation was later proposed by Carl A. Brannen [4, 5]. If we relabel the charged-lepton masses as m_{e1} , m_{e2} , and m_{e3} , their square roots may be parameterized as

$$\sqrt{m_{en}} = 17.716 \left(1 + \sqrt{2} \cos \left(\frac{2}{9} + \frac{2}{3} n\pi \right) \right) \text{ MeV}^{\frac{1}{2}}, \quad n = 1, 2, 3. \quad (2)$$

An important detail is that the phase is $\frac{2}{9}$, not $\frac{2}{9}\pi$. Geometric interpretations of Koide-type relations have also been discussed from other viewpoints, notably by Foot and, in a different mathematical setting, by Kocik [8, 12]. This parametrization motivates the broader question of whether quark masses may admit a related phase-based description.

1.2 Aim of the present work

A direct transplantation of Koide-type formulas from charged leptons to quarks is nontrivial because quark masses are renormalization-scale dependent and experimentally less precise. In the quark sector, one must therefore distinguish between a numerical pattern observed at quoted reference scales and a relation that remains meaningful under renormalization-group evolution. Earlier extensions and related quark-sector discussions include the empirical analyses of Rodejohann and Zhang, Kartavtsev, and Koide–Nishiura [10, 11, 13]. The issue of radiative stability has also motivated model-building work such as Sumino’s family-gauge approach [9].

The present paper focuses on the down-type quarks. The aim is not to derive a mass formula from an underlying symmetry, but to determine whether a Brannen-type ansatz can provide a coherent phenomenological organization of the d -, s -, and b -quark masses. The viewpoint adopted here is that the observed pattern is most naturally read as

a Brannen-type geometric skeleton together with QCD renormalization-group dressing.

In this language, the benchmark phase structure organizes the three-generation hierarchy, while QCD running describes how the quoted light-quark inputs move coherently when they are transported to a common scale. Throughout this paper, the expression “Brannen-type geometric skeleton” refers precisely to such a phase-parametrized square-root mass pattern with a prescribed benchmark pair (ϕ, δ) , considered independently of the scales at which the individual masses are quoted. “QCD renormalization-group dressing” then refers to the additional multiplicative factor acquired by each square-root mass when the quoted inputs are transported from their reference scales to a common renormalization scale μ . The benchmark values introduced below are compact phenomenological choices near the fitted region, not theoretically derived constants.

The present paper contributes the following four points. First, it proposes a specific benchmark ansatz for the down-type spectrum, with $\phi_d = \frac{1}{3}$ and $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$. Second, it separates the mixed-scale benchmark pattern from the QCD running that dresses it, including the near-common square-root reduction factor ≈ 0.9191 shared by the d - and s -quark branches between $\mu = 2 \text{ GeV}$ and $\mu = m_b$. Third, it verifies numerically that the inverse Koide quantity K_{inv} remains close to $\frac{2}{3}$ and is transport-stable between the mixed PDG reference scales and the common-scale reconstruction. Fourth, it rewrites the inverse square-root tuple in a Brannen-type form, clarifying that its QCD-only shape stability is algebraic while its proximity to the symmetric value $\frac{1}{\sqrt{2}}$ is empirical.

The strategy has three parts. First, we identify a benchmark functional form at the standard PDG reference scales. Second, we allow the overall normalization to depend on the running bottom scale μ_{d3} . Third, we reconstruct the spectrum at a common renormalization scale μ . This progression makes it possible to separate the mixed-scale benchmark from the genuinely scale-stable part of the construction and to see more clearly why the inverse-Koide quantity K_{inv} is numerically more robust than the direct Koide combination.

To summarize the scope of this work: (i) it offers a phenomenological organization of the down-type quark masses, not a derivation from a more fundamental theory; (ii) the benchmark phase pair is a fit-motivated benchmark rather than a first-principles prediction; and (iii) the main technical contribution is the separation of a mixed-scale benchmark from a common-scale RG reconstruction, together with the transport stability of the inverse-Koide quantity documented in the latter. The low-energy extrapolation shown in Figure S.1 is included for reference only and has no standalone physical status.

2 Numerical Inputs and Conventions

2.1 Experimental inputs and renormalization conventions

We adopt the following PDG 2024 central values as numerical inputs [14]:

$$\begin{aligned} m_{d1}(2\text{ GeV}) &= 4.70\text{ MeV}, \\ m_{d2}(2\text{ GeV}) &= 93.50\text{ MeV}, \\ m_{d3}(4.183\text{ GeV}) &= 4183.00\text{ MeV}, \end{aligned} \tag{3}$$

together with

$$\alpha_s(m_Z^2) = 0.1180, \tag{4}$$

where m_{d1} , m_{d2} , and m_{d3} denote the d -, s -, and b -quark masses, respectively. Throughout the paper we use both notations interchangeably: the generational indices m_{d1}, m_{d2}, m_{d3} are preferred in contexts involving the Brannen-type phase-indexed parametrization and the ordered structure of the spectrum, while the flavour names m_d, m_s, m_b are used in contexts involving renormalization scales and standard PDG conventions such as $m_b(m_b)$. Quark masses are understood as $\overline{\text{MS}}$ running masses. For the threshold parameters entering the numerical running, we use

$$\begin{aligned} m_c(m_c) &= 1.273\text{ GeV}, \\ m_b(m_b) &= 4.183\text{ GeV}, \\ m_t(m_t) &= 162.500\text{ GeV}, \end{aligned} \tag{5}$$

again following the PDG 2024 central values.

2.2 Computational procedure

Most algebraic and tabulated numerical evaluations were carried out in Microsoft Excel 2021 using default double-precision arithmetic. Python was used for the renormalization-group running, root finding, and the production of the two-dimensional plots shown below.

The running masses were evaluated in the $\overline{\text{MS}}$ scheme with four-loop QCD running. Heavy-flavour thresholds were implemented through continuous matching at $\mu = m_q(m_q)$. In the present work this continuous-matching prescription is a numerical convention, not a substitute for a full effective-field-theory decoupling analysis. The explicit numerical assumptions are collected in the Supplemental Material, and the Python scripts used to generate the figures are reproduced in Appendices A–D (with the auxiliary script for Section S.3 in Appendix E).

Numerical values are displayed according to their role: PDG input masses are quoted with a precision commensurate with the input uncertainties; exact benchmark constants are shown with extra digits (or the ellipsis ...) at their defining occurrence and rounded to four digits in subsequent appearances; renormalization-group running masses are quoted to three decimals in MeV; dimensionless ratios such as K_{inv} , the QCD square-root dressing factor, and the fitted Brannen parameters are shown to three or four significant figures in the body text; and the additional digits retained in diagnostic tables (e.g., Table 2) serve only to exhibit the numerical stability of the central-value calculation. These display conventions do not affect the underlying numerical calculations.

3 Phenomenological Construction: Brannen-Type Skeleton and QCD RG Dressing

3.1 General three-generation form

Following Brannen’s parametrization of the charged-lepton spectrum, we describe a three-generation mass pattern in terms of an amplitude, a geometric parameter, and a phase. To avoid confusion between the amplitude and the renormalization scale, we denote the amplitude by A rather than μ .

Define

$$T \equiv \sqrt{\frac{1 + \cos \delta}{2}}. \quad (6)$$

Then the generalized Koide form and the corresponding Brannen-type parametrization may be written as

(i) **Generalized Koide form.**

$$\frac{m_1 + m_2 + m_3}{(\sqrt{m_1} + \sqrt{m_2} + \sqrt{m_3})^2} = \frac{1 + 2T^2}{3}. \quad (7)$$

(ii) **Brannen-type form.**

$$\sqrt{m_n} = A \left(1 + 2T \cos \left(\frac{1}{3}\phi + \frac{2}{3}n\pi \right) \right), \quad n = 1, 2, 3. \quad (8)$$

For the charged-lepton realization, we use the reference values

$$\phi_e = \frac{2}{3}, \quad \cos(\delta_e) = 0, \quad A_e = \frac{53.148}{3} \text{ MeV}^{\frac{1}{2}}, \quad (9)$$

with

$$\sqrt{m_{e1}} + \sqrt{m_{e2}} + \sqrt{m_{e3}} = 53.148 \text{ MeV}^{\frac{1}{2}}. \quad (10)$$

Further remarks on the charged-lepton reference parameters are collected in Section S.1 of the Supplemental Material. We note already at this stage that, for any positive three-mass tuple, the Brannen-type three-parameter form admits a closed-form inversion in (A, T, ϕ) , so that any common-scale Brannen reconstruction of three masses is, on its own, a parametrization rather than an independent fit-based prediction.

3.2 Benchmark ansatz at the quoted reference scales

Using the PDG 2024 central values listed in Section 2.1, we ask whether the down-type quark spectrum admits a Brannen-type description of the same structural form. At this stage the proposal is purely phenomenological: the goal is to identify a compact benchmark ansatz that is in agreement with the observed hierarchy within the present accuracy of the quark-mass inputs before addressing the renormalization-scale problem more carefully.

As an exploratory first step, we keep the quoted reference scales μ_{d1} , μ_{d2} , and μ_{d3} separate. One then finds

$$\sqrt{m_{d1}} + \sqrt{m_{d2}} + \sqrt{m_{d3}} \approx 76.224 - 76.879 \text{ MeV}^{\frac{1}{2}} \approx 53.148 \times 3^{\frac{1}{3}} \text{ MeV}^{\frac{1}{2}}. \quad (11)$$

The interval in Eq. (11) is obtained by inserting the quoted PDG 2024 1σ input ranges for $m_d(2 \text{ GeV})$, $m_s(2 \text{ GeV})$, and $m_b(m_b)$ into the mixed-scale sum $\sqrt{m_{d1}} + \sqrt{m_{d2}} + \sqrt{m_{d3}}$; the numerical analysis below otherwise uses the corresponding central values. This motivates the benchmark ansatz

$$\sqrt{m_{dn}} = \frac{53.148}{3^{\frac{2}{3}}} \left(1 + 2T_d \cos \left(\frac{1}{3}\phi_d + \frac{2}{3}n\pi \right) \right) \text{ MeV}^{\frac{1}{2}}, \quad n = 1, 2, 3, \quad (12)$$

with

$$T_d \equiv \sqrt{\frac{1 + \cos(\delta_d)}{2}}. \quad (13)$$

A simple and suggestive benchmark choice is

$$\phi_d = \frac{1}{2}\phi_e = \frac{1}{3}. \quad (14)$$

If one inserts the trial value $2T_d = 1.543$ into Eq. (12), the resulting masses are already close to the PDG central values:

$$\begin{aligned} m_{d1} &\approx 4.727 \text{ MeV}, \\ m_{d2} &\approx 94.981 \text{ MeV}, \\ m_{d3} &\approx 4190.336 \text{ MeV}. \end{aligned}$$

3.2.1 Benchmark choice of $\cos(\delta_d)$

The empirical value $2T_d \approx 1.543$ implies $\cos(\delta_d) \approx 0.190$. We therefore adopt the nearby exact benchmark value

$$\cos(\delta_d) = \frac{1}{2} \cos\left(\frac{3}{8}\pi\right) = \frac{\sqrt{2-\sqrt{2}}}{4} \approx 0.191342\dots \quad (\Rightarrow \delta_d \approx 1.378267\dots [\text{rad}]). \quad (15)$$

The rationale for this choice is discussed in Section S.2 of the Supplemental Material. We emphasize that the present paper treats Eq. (15) as a compact benchmark ansatz, not as a rigorous geometric derivation.

With

$$\phi_d = \frac{1}{3}, \quad \cos(\delta_d) = \frac{1}{2} \cos\left(\frac{3}{8}\pi\right), \quad (16)$$

the benchmark ansatz in Eq. (12) should be read carefully. Formally it contains three quantities (A, T_d, ϕ_d) , but in the present work $\phi_d = \frac{1}{3}$ and $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$ are not treated as fit parameters. They are prescribed *a priori* as structural benchmark choices, in analogy with the charged-lepton reference values $\phi_e = \frac{2}{3}$ and $\cos(\delta_e) = 0$. Only the overall normalization $A = 53.148 \times 3^{-\frac{2}{3}}$ is fixed from the mixed-scale square-root sum in Eq. (11). The benchmark construction should therefore be regarded as a conditional one-parameter ansatz for three masses, yielding two non-trivial constraints: the mass-ratio pattern and the associated benchmark value of the direct Koide ratio. The benchmark is not claimed to be a parameter-free prediction; its role is to test whether a compact fixed phase pair can organize the observed hierarchy once the overall scale is set.

One then obtains

$$T_d = \sqrt{\frac{1}{2} + \frac{\sqrt{2-\sqrt{2}}}{8}}. \quad (17)$$

The quoted interval is obtained by propagating the quoted PDG 2024 1σ input ranges of the mixed-scale masses through the square-root sum; unless stated otherwise, the numerical analysis below uses the central values. The factor $3^{\frac{1}{3}}$ is used here only as a numerical bridge between the charged-lepton square-root sum $53.148 \text{ MeV}^{\frac{1}{2}}$ and the down-type mixed-scale square-root sum. No group-theoretic meaning is claimed for this factor by itself.

Accordingly, the direct Koide ratio becomes

$$\frac{m_{d1} + m_{d2} + m_{d3}}{(\sqrt{m_{d1}} + \sqrt{m_{d2}} + \sqrt{m_{d3}})^2} = \frac{2}{3} + \frac{\sqrt{2-\sqrt{2}}}{12} \approx 0.730447\dots \quad (18)$$

and the benchmark Brannen-type expression is

$$\sqrt{m_{dn}} = \frac{53.148}{3^{\frac{2}{3}}} \left(1 + \sqrt{2 + \frac{\sqrt{2-\sqrt{2}}}{2}} \cos\left(\frac{1}{9} + \frac{2}{3}n\pi\right) \right) \text{ MeV}^{\frac{1}{2}}, \quad n = 1, 2, 3. \quad (19)$$

At this benchmark point, the predicted masses are

$$\begin{aligned} m_{d1} &\approx 4.687 \text{ MeV}, \\ m_{d2} &\approx 94.862 \text{ MeV}, \\ m_{d3} &\approx 4192.291 \text{ MeV}. \end{aligned} \quad (20)$$

At this point the benchmark formula should be interpreted carefully. Because m_d and m_s are quoted at $\mu = 2 \text{ GeV}$, whereas m_b is quoted at $\mu = m_b$, Eq. (19) is not a common-scale mass law. It is better read as a mixed-scale *Brannen-type geometric skeleton*. The common-scale description is obtained only after renormalization-group transport.

Schematically, if the quoted inputs are evolved from a reference scale μ_0 to a common scale μ , then

$$m_q(\mu) = Z_m(\mu, \mu_0) m_q(\mu_0), \quad \sqrt{m_q(\mu)} = Z_m^{\frac{1}{2}}(\mu, \mu_0) \sqrt{m_q(\mu_0)}. \quad (21)$$

Using the same four-loop running conventions implemented in Scripts 1 and 3 (Appendices A and C), evolution of the light branches from $\mu = 2 \text{ GeV}$ to $\mu = m_b$ gives

$$m_d(m_b) \approx 3.970 \text{ MeV}, \quad m_s(m_b) \approx 78.984 \text{ MeV}, \quad (22)$$

and therefore

$$\sqrt{\frac{m_d(m_b)}{m_d(2 \text{ GeV})}} \approx 0.9191, \quad \sqrt{\frac{m_s(m_b)}{m_s(2 \text{ GeV})}} \approx 0.9191. \quad (23)$$

The near equality of these two factors is the numerical reason for the language adopted here: the benchmark phase pattern supplies the geometric skeleton, while QCD running supplies an almost common multiplicative dressing for the light-quark branches. The b branch, by contrast, remains nearly unchanged because it is already anchored at its own reference scale.

3.3 Running extension in terms of μ_{d3}

The constant prefactor $53.148 \times 3^{-\frac{2}{3}}$ is adequate only at the benchmark point. To incorporate the running of the bottom mass, we replace it by a scale-dependent amplitude $A_d(\mu_{d3})$ and thereby define a scale-dependent predicted spectrum $m_{dn}^{\text{pred}}(\mu_{d3})$:

$$\sqrt{m_{dn}^{\text{pred}}(\mu_{d3})} = A_d(\mu_{d3}) \left(1 + \sqrt{2 + \frac{\sqrt{2 - \sqrt{2}}}{2} \cos\left(\frac{1}{9} + \frac{2}{3}n\pi\right)} \right) \text{MeV}^{\frac{1}{2}}, \quad n = 1, 2, 3. \quad (24)$$

The explicit numerical procedure is given in Appendix A.

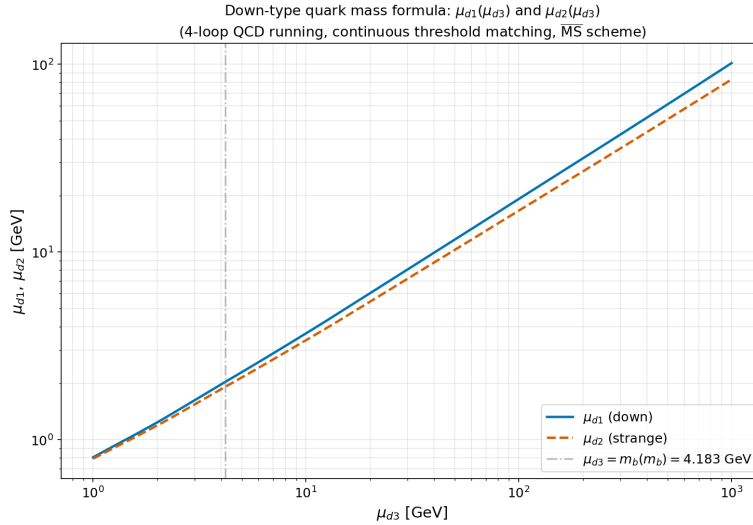


Figure 1. Reconstructed scales $\mu_{d1}(\mu_{d3})$ and $\mu_{d2}(\mu_{d3})$ obtained from the running extension of the down-type mass formula.

Figure 1 illustrates the resulting dependence of μ_{d1} and μ_{d2} on μ_{d3} . A more detailed inspection of the low-energy region, in which the reconstructed scales appear to cluster together, is collected in Section S.3 of the Supplemental Material, since the apparent infrared coincidence is an extrapolation artifact close to the Landau-pole region rather than a physical claim.

At this stage, following a subsequent remark by Rivero [15], we examine the inverse Koide quantity K_{inv} , obtained from the Koide combination by replacing each mass with its reciprocal:

$$K_{\text{inv}}^{\text{pred}}(\mu_{d3}) \equiv \frac{\frac{1}{m_{d1}^{\text{pred}}(\mu_{d3})} + \frac{1}{m_{d2}^{\text{pred}}(\mu_{d3})} + \frac{1}{m_{d3}^{\text{pred}}(\mu_{d3})}}{\left(\frac{1}{\sqrt{m_{d1}^{\text{pred}}(\mu_{d3})}} + \frac{1}{\sqrt{m_{d2}^{\text{pred}}(\mu_{d3})}} + \frac{1}{\sqrt{m_{d3}^{\text{pred}}(\mu_{d3})}} \right)^2} \approx 0.6662 \approx \frac{2}{3}. \quad (25)$$

After our initial mixed-scale down-type mass-formula observation, Rivero pointed out that this near equality remains numerically stable, both in the quoted mixed-scale form (where $K_{\text{inv}}^{\text{pred}} \approx 0.6662$ is computed from the benchmark-predicted spectrum) and in the common-scale reconstruction (where the corresponding quantity computed from the running masses is denoted $K_{\text{inv}}(\mu) \approx 0.6674$; see Section 4.2 and Table 2), and suggested describing it as a *phenomenological dual Koide* relation. That observation helped motivate the common-scale analysis developed below.

3.4 Common-scale reconstruction at μ

We next reformulate the analysis at a common renormalization scale. The negligible residual obtained below in the common-scale reconstruction is an internal numerical check of a three-parameter representation of three masses, not an independent physical prediction; the non-trivial empirical content resides instead in the restricted benchmark choice (Section 3.2) and, as developed in Section 4.3, in the proximity of the inverse-tuple parameter $T_{d\text{inv}}^{\text{fit}}$ to $\frac{1}{\sqrt{2}}$. With this caveat, let

$$\sqrt{m_{dn}}(\mu) = A_d^{\text{fit}}(\mu) \left(1 + 2T_d^{\text{fit}} \cos \left(\frac{1}{3}\phi_d^{\text{fit}} + \frac{2}{3}n\pi \right) \right) \text{MeV}^{\frac{1}{2}}. \quad (26)$$

From the running masses $m_{d1}(\mu)$, $m_{d2}(\mu)$, and $m_{d3}(\mu)$, we reconstruct the scale-dependent amplitude $A_d^{\text{fit}}(\mu)$ while treating T_d^{fit} and ϕ_d^{fit} as scale-independent fit constants. This separation holds within the QCD-only, flavour-universal running approximation adopted in this work; we do not assume that it remains exact once electroweak gauge and Yukawa contributions to the full Standard-Model running are included at $\mu \gg M_Z$. The explicit algorithm is given in Appendix B.

At $\mu = 4.183 \text{ GeV}$, the fitted parameters are approximately

$$A_d^{\text{fit}}(4.183 \text{ GeV}) \approx 25.19, \quad T_d^{\text{fit}} \approx 0.788, \quad \phi_d^{\text{fit}} \approx 0.301. \quad (27)$$

The fit values in Eq. (27) necessarily differ from the benchmark parameters $(T_d^{\text{pred}}, \phi_d^{\text{pred}}) = (\frac{1}{2} \cos(\frac{3}{8}\pi), \frac{1}{3})$ fixed in Eqs. (15)–(16). This shift is not an independent phenomenological adjustment: it is the direct algebraic consequence of QCD renormalization-group dressing of the two light branches. Since $\sqrt{m_d}$ and $\sqrt{m_s}$ acquire the common factor ≈ 0.9191 between $\mu = 2 \text{ GeV}$ and $\mu = m_b$ (Eq. (23)) while $\sqrt{m_b}$ is already evaluated at $\mu = m_b$, the trigonometric pattern that simultaneously reproduces the three masses at a single common scale must absorb this scale-dependent reshaping of the ratios $\sqrt{m_{d1}/m_{d3}}$ and $\sqrt{m_{d2}/m_{d3}}$. In this sense $(T_d^{\text{fit}}, \phi_d^{\text{fit}})$ are not free parameters of a new model; they are the RG-dressed image of $(T_d^{\text{pred}}, \phi_d^{\text{pred}})$ at the chosen common scale. The residuals quoted in the “with RG dressing” column of Table 1 are a logically separate quantity: because flavour-universal QCD running multiplies both the benchmark prediction and the PDG input by the same factor between $\mu = 2 \text{ GeV}$ and $\mu = m_b$, the ratio is preserved and the residuals simply inherit the quality of the benchmark fit at the PDG input scales. The asymmetry between the m_{d1} and m_{d2} residuals (-0.49% vs $+1.23\%$) is therefore an intrinsic property of the Brannen-type fit at $\mu = 2 \text{ GeV}$, not a flavour-dependent RG effect.

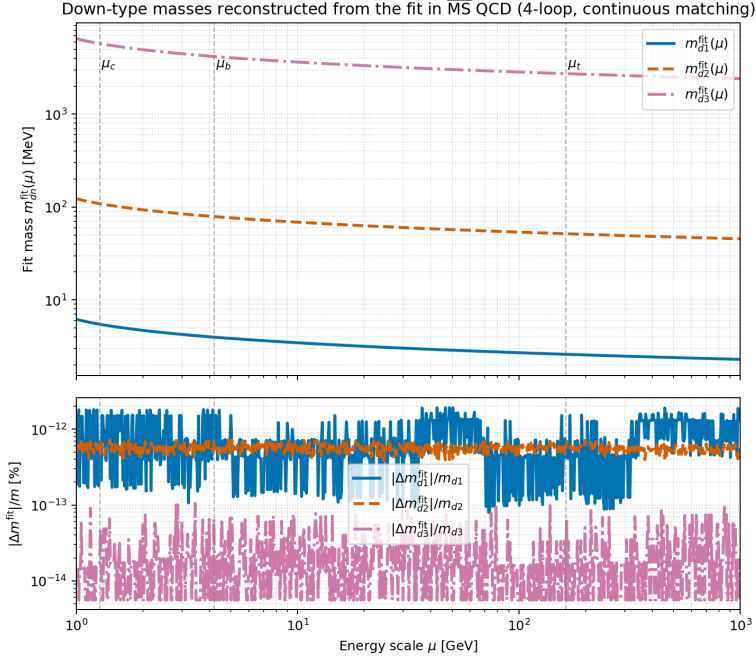


Figure 2. Common-scale reconstruction of the down-type running masses, produced by Script 2 (Appendix B). The top panel shows the Brannen-type form of Eq. (26) evaluated with the fit parameters $(A_d^{\text{fit}}(\mu), T_d^{\text{fit}}, \phi_d^{\text{fit}})$, denoted $m_{dn}^{\text{fit}}(\mu)$. The bottom panel shows the absolute relative residual $|\Delta m_{dn}^{\text{fit}}|/m_{dn}$, where $\Delta m_{dn}^{\text{fit}}(\mu) \equiv m_{dn}^{\text{fit}}(\mu) - m_{dn}(\mu)$ and $m_{dn}(\mu)$ denotes the PDG 2024 running masses obtained by 4-loop $\overline{\text{MS}}$ evolution from the quoted inputs of Eq. (3) under the continuous-matching convention of Section 2.2. The $\sim 10^{-12}$ residual reflects the numerical closure of the three-parameter reconstruction and should not be read as phenomenological accuracy.

The reconstruction error stays at the 10^{-12} level over the plotted range. In this common-scale formulation as well, the inverse Koide quantity $K_{\text{inv}}(\mu)$, computed from the running masses, remains close to $\frac{2}{3}$ and is effectively scale independent:

$$K_{\text{inv}}(\mu) = \frac{\frac{1}{m_{d1}(\mu)} + \frac{1}{m_{d2}(\mu)} + \frac{1}{m_{d3}(\mu)}}{\left(\frac{1}{\sqrt{m_{d1}(\mu)}} + \frac{1}{\sqrt{m_{d2}(\mu)}} + \frac{1}{\sqrt{m_{d3}(\mu)}}\right)^2} \approx 0.6674 \approx \frac{2}{3}. \quad (28)$$

This is the most persistent numerical feature of the construction and the main reason the dual Koide viewpoint survives the transition from the benchmark skeleton to the common-scale, RG-dressed description.

4 Results and Interpretation

4.1 Benchmark reproduction of the down-type spectrum

At the benchmark point defined by Eq. (19), the Brannen-type ansatz is in agreement with the observed hierarchy of the down-type sector within the present PDG accuracy. For a clean benchmark comparison at a *common* renormalization scale, we fix $\mu_{d3} = m_b(m_b) = 4.183 \text{ GeV}$ and evaluate both the common-scale formula of Eq. (26) and the PDG 2024 central inputs at the same scale. The common-scale predictions are

$$\begin{aligned} m_{d1}^{\text{pred}}(\mu_{d3}) &\approx 4.677 \text{ MeV}, \\ m_{d2}^{\text{pred}}(\mu_{d3}) &\approx 94.652 \text{ MeV}, \\ m_{d3}^{\text{pred}}(\mu_{d3}) &\approx 4183.000 \text{ MeV}, \end{aligned} \quad (29)$$

which numerically coincide with the Brannen-type values associated with the PDG input scales (2 GeV for d, s ; m_b for b). Table 1 summarizes the comparison after the common-scale transport.

Table 1. Benchmark comparison with all entries evaluated at the common scale $\mu_{d3} = m_b(m_b) = 4.183 \text{ GeV}$. “Mixed-scale benchmark”: Eq. (26) evaluated at μ_{d3} using the benchmark parameters $(T_d^{\text{pred}}, \phi_d^{\text{pred}}) = (\frac{1}{2} \cos(\frac{3}{8}\pi), \frac{1}{3})$ of Eqs. (15)–(16), not the fit values of Eq. (27). “with RG dressing”: the same benchmark prediction transported from its PDG input scale (2 GeV for d, s ; $m_b(m_b)$ for b) to μ_{d3} via 4-loop $\overline{\text{MS}}$ running. “Evolved PDG input”: the PDG 2024 central inputs transported to μ_{d3} analogously (these are inputs, not fit outputs). Relative deviation $\equiv (\text{with RG dressing} - \text{Evolved PDG input})/\text{Evolved PDG input}$. Algorithm: Script 3 (Appendix C); visual summary: Figure 3.

Quantity	Mixed-scale benchmark	with RG dressing	Evolved PDG input	Relative deviation
m_{d1}	4.677 MeV	3.951 MeV	3.970 MeV	−0.49%
m_{d2}	94.652 MeV	79.957 MeV	78.984 MeV	+1.23%
m_{d3}	4183.000 MeV	4183.000 MeV	4183.000 MeV	±0.00%

Table~1 benchmark: Prediction (Eq.~22) vs RG-dressed vs PDG 2024 evolved, common scale $\mu_{d3} = m_b(m_b)$

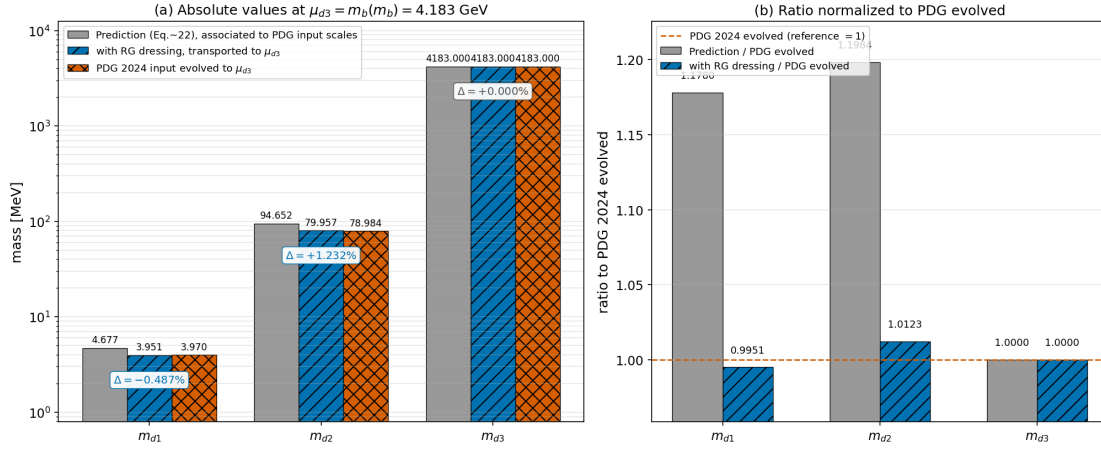
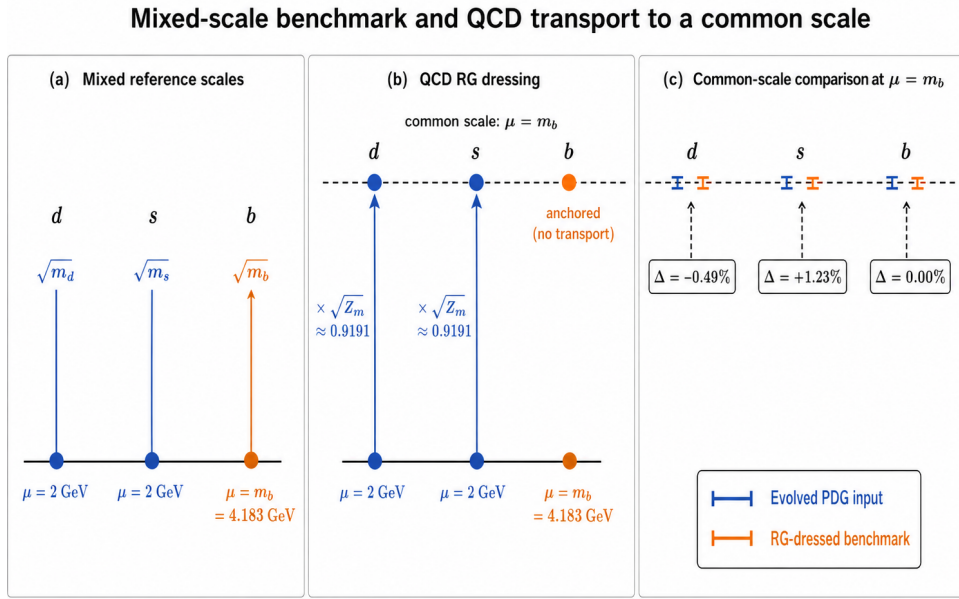


Figure 3. Visual summary of Table 1: Brannen-type common-scale prediction (Eq. (26)) versus the same prediction after RG dressing, compared with the PDG 2024 central inputs evolved to the common scale $\mu_{d3} = m_b(m_b) = 4.183 \text{ GeV}$. Panel (a) shows the absolute values on a logarithmic axis with the residual deviations Δ after RG dressing. Panel (b) shows the same comparison as a ratio normalized to the PDG 2024 evolved values (reference line at unity). The plot is produced by Script 3 (Appendix C).

Figure 4 provides a three-panel schematic accompanying Table 1. Panel (a) shows the mixed reference scales of the down-type masses: $\sqrt{m_d}$ and $\sqrt{m_s}$ at $\mu = 2 \text{ GeV}$ and $\sqrt{m_b}$ at $\mu = m_b$. Panel (b) shows the QCD transport to the common scale $\mu = m_b$: the d and s branches each pick up the multiplicative factor $\sqrt{Z_m} \approx 0.9191$ (with Z_m evaluated between $\mu = 2 \text{ GeV}$ and $\mu = m_b$, in agreement with Eq. (23)), while the b branch is anchored at its own reference scale and receives no transport. Panel (c) compares, at the common scale $\mu = m_b$, the RG-dressed benchmark prediction with the evolved PDG 2024 input, displaying the residuals $\Delta = -0.49\%$, $+1.23\%$, and $\pm 0.00\%$ for the d , s , and b entries respectively. The figure is a schematic illustration only; it is not generated by any of Scripts 1–4, and the corresponding quantitative content is given by Table 1 and Figure 3.



Schematic only; not to scale.

Figure 4. Schematic scale-flow diagram accompanying Table 1. (a) Mixed reference scales: $\sqrt{m_d}$, $\sqrt{m_s}$ at $\mu = 2 \text{ GeV}$ and $\sqrt{m_b}$ at $\mu = m_b$. (b) QCD transport to the common scale $\mu = m_b$: the d and s square-root masses each acquire the multiplicative factor $\sqrt{Z_m} \approx 0.9191$ under the QCD-only running adopted in this work (Eq. (23)), while the b branch is anchored at its own reference scale and receives no transport. (c) Common-scale comparison between the RG-dressed benchmark and the evolved PDG 2024 input, with residuals $\Delta = -0.49\%$, $+1.23\%$, $\pm 0.00\%$ for the d, s, b entries (cf. Table 1). The figure is schematic only and is not produced by any of Scripts 1–4.

For the benchmark comparison summarized in Table 1, only the overall normalization A is fixed from Eq. (11); the quantities T_d and ϕ_d are not refit to the three down-type masses but are held fixed by the structural benchmark choice in Eq. (16). The residual deviations quoted in Table 1 (-0.49% for m_{d1} and $+1.23\%$ for m_{d2}) sit well within the PDG 2024 quoted uncertainties on $m_d(2\text{ GeV})$ and $m_s(2\text{ GeV})$, so the benchmark agreement should not be over-interpreted as a precision test. The heaviest entry m_{d3} is identical across the three columns by construction, because the common scale is itself chosen at $m_b(m_b)$ and the overall amplitude is normalized so that the b branch reproduces $m_b(m_b)$ exactly. The non-trivial benchmark check is therefore the simultaneous percent-level placement of the d and s entries after the same QCD transport, not the b entry on its own.

Using the same running conventions as in Scripts 1 and 3, the light-quark inputs transported from 2 GeV to $\mu = m_b$ become $m_d(m_b) \approx 3.970\text{ MeV}$ and $m_s(m_b) \approx 78.984\text{ MeV}$. The corresponding square-root dressing factors are both approximately 0.9191 , in agreement with Eq. (23). This explicit recalculation supports the view that the benchmark relation should be read as a mixed-scale geometric skeleton, while the common rescaling of the light branches is supplied by QCD running.

The agreement is not exact, nor should one expect exact agreement at the present phenomenological level. What matters is that the ansatz captures both the strong hierarchy and the correct overall scale of the three down-type masses within current PDG uncertainties, using only a small set of benchmark parameters.

4.2 Dual Koide behavior under running

The central numerical result is not the benchmark mass table itself but the transport stability of the inverse Koide quantity under the reduction from mixed quoted scales to a common renormalization scale. In the common-scale reconstruction, the more robust quantity is

$$K_{\text{inv}}(\mu) \equiv \frac{\frac{1}{m_{d1}(\mu)} + \frac{1}{m_{d2}(\mu)} + \frac{1}{m_{d3}(\mu)}}{\left(\frac{1}{\sqrt{m_{d1}(\mu)}} + \frac{1}{\sqrt{m_{d2}(\mu)}} + \frac{1}{\sqrt{m_{d3}(\mu)}} \right)^2}. \quad (30)$$

Representative values are listed in Table 2.

Table 2. Illustrative values of the inverse Koide quantity $K_{\text{inv}}(\mu)$, computed numerically from the running masses in Script 2 (Appendix B). Within numerical precision and within the QCD-only, flavour-universal running convention adopted in this work, the quantity is effectively scale independent. The six-digit display of $K_{\text{inv}}(\mu)$ is retained only to exhibit the numerical stability across the listed scales; the body text quotes the rounded value 0.6674 .

Scale μ	$K_{\text{inv}}(\mu)$
2 GeV	0.667416
$m_b(= 4.183\text{ GeV})$	0.667416
$M_Z(= 91.2\text{ GeV})$	0.667416
Reference value $\frac{2}{3}$	0.666667

Numerically, $K_{\text{inv}}(\mu)$, as computed from the running masses in Script 2 (Appendix B), stays very close to $\frac{2}{3}$ over the scale range examined here and is effectively scale independent within numerical precision. The scale stability itself follows from the homogeneity of K_{inv} under a common rescaling of the three masses in the QCD-only approximation; the empirical observation is that the scale-stable value lies close to $\frac{2}{3}$. The same homogeneity protects the direct Koide ratio under a common-scale QCD-only rescaling, but the mixed-to-common-scale transport treats the b branch and the light branches asymmetrically (m_b is anchored at $\mu = m_b$ while $m_{d,s}$ are evolved from $\mu = 2\text{ GeV}$); this asymmetry distorts the direct benchmark relation, but leaves the inverse-tuple shape (and hence K_{inv}) intact under the same transport, which is the empirical asymmetry between the two ratios documented here. It is also consistent with Rivero’s subsequent observation, following our initial mixed-scale mass-formula construction, that the inverse relation is unusually stable under renormalization-group evolution [15].

4.3 Inverse-tuple Brannen parametrization

The empirical content of Section 4.2 can be recast in a structurally compact form by parametrizing the inverse square-root tuple in the same Brannen-type form as Eq. (26). Define

$$\frac{1}{\sqrt{m_{dn}(\mu)}} = A_{d\text{inv}}^{\text{fit}}(\mu) \left(1 + 2T_{d\text{inv}}^{\text{fit}}(\mu) \cos\left(\frac{1}{3}\phi_{d\text{inv}}^{\text{fit}}(\mu) + \frac{2}{3}k\pi\right) \right) \text{MeV}^{-\frac{1}{2}}, \quad k = 4 - n, \quad n = 1, 2, 3. \quad (31)$$

For three masses at any given μ , Eq. (31) admits a closed-form solution in $(A_{d\text{inv}}^{\text{fit}}, T_{d\text{inv}}^{\text{fit}}, \phi_{d\text{inv}}^{\text{fit}})$ by the same inversion as in the direct case (Section S.4); the explicit algorithm is provided in Appendix D.

By the same algebra used to derive the direct identity $K = \frac{1}{3}(1 + 2T_d^2)$, the inverse Koide quantity satisfies the closed-form identity

$$K_{\text{inv}}(\mu) = \frac{1 + 2[T_{d\text{inv}}^{\text{fit}}(\mu)]^2}{3}, \quad (32)$$

so that $K_{\text{inv}} = \frac{2}{3}$ is equivalent to $T_{d\text{inv}}^{\text{fit}} = \frac{1}{\sqrt{2}}$. At the PDG 2024 input scales the Brannen extraction yields

$$A_{d\text{inv}}^{\text{fit}}(2 \text{ GeV}) \approx 0.193 \text{ MeV}^{-\frac{1}{2}}, \quad T_{d\text{inv}}^{\text{fit}} \approx 0.7079, \quad \phi_{d\text{inv}}^{\text{fit}} \approx +0.569 \text{ rad}, \quad (33)$$

the empirical proximity $T_{d\text{inv}}^{\text{fit}} \approx \frac{1}{\sqrt{2}} \approx 0.7071$ being the same dual-Koide observation expressed in the inverse-tuple language.

For comparison, applying the same closed-form inversion to the benchmark predicted masses of Eq. (19), evaluated at the same mixed PDG reference scales, yields

$$A_{d\text{inv}}^{\text{pred}}(2 \text{ GeV}) \approx 0.193 \text{ MeV}^{-\frac{1}{2}}, \quad T_{d\text{inv}}^{\text{pred}} \approx 0.7066, \quad \phi_{d\text{inv}}^{\text{pred}} \approx +0.556 \text{ rad}, \quad (34)$$

which differ from the PDG-input fit values of Eq. (33) at the percent level, mirroring the direct benchmark residuals reported in Table 1. Notably, numerically the symmetric value $\frac{1}{\sqrt{2}} \approx 0.7071$ lies between $T_{d\text{inv}}^{\text{pred}} \approx 0.7066$ and $T_{d\text{inv}}^{\text{fit}} \approx 0.7079$, so that the residual gap to $K_{\text{inv}} = \frac{2}{3}$ closes from opposite sides under the two parametrizations.

The structural advantage of Eq. (31) is that the scale dependence of the three Brannen quantities is now fully transparent. Because flavour-universal QCD running multiplies $\sqrt{m_{dn}(\mu)}$ by a common factor $\eta(\mu, \mu_0)$ (Section S.4), the inverse tuple acquires the inverse common factor:

$$\frac{1}{\sqrt{m_{dn}(\mu)}} = \frac{1}{\eta(\mu, \mu_0)} \frac{1}{\sqrt{m_{dn}(\mu_0)}}. \quad (35)$$

The amplitude $A_{d\text{inv}}^{\text{fit}}(\mu)$ absorbs this dressing factor exactly, while $T_{d\text{inv}}^{\text{fit}}$ and $\phi_{d\text{inv}}^{\text{fit}}$ remain manifestly scale-independent under QCD-only running. Figure 5 confirms this numerically across the full perturbative window: $T_{d\text{inv}}^{\text{fit}}(\mu) \approx 0.7079$ and $\phi_{d\text{inv}}^{\text{fit}}(\mu) \approx +0.569 \text{ rad}$ are both flat to numerical precision. Compared with the direct mass formula Eq. (26), the inverse-tuple form Eq. (31) keeps the cosine in the same Brannen shape $\cos(\frac{1}{3}\phi + \frac{2}{3}\text{idx}\pi)$ but with the index reversed by $k = 4 - n$. This single discrete relabelling reflects the inversion of the underlying mass ordering – $\sqrt{m_{dn}}$ is monotonically increasing in n whereas $\frac{1}{\sqrt{m_{dn}}}$ is monotonically decreasing – so that the Brannen reference (the dominant branch at angle 0) is taken at index 3 in both cases: $n = 3$ (the bottom branch) for the direct form and $k = 3$ (i.e. $n = 1$, the down branch) for the inverse form. In Brannen’s geometric reading, the inverse-tuple parametrization corresponds to the same spherical triangle traversed in the opposite orientation, and the resulting $\phi_{d\text{inv}}^{\text{fit}} \approx +0.569 \text{ rad}$ lies in the same sign range as the direct $\phi_d = \frac{1}{3}$. Via Eq. (32) the constancy of $T_{d\text{inv}}^{\text{fit}}$ is equivalent to the constancy of K_{inv} recorded in Table 2; the constancy of $\phi_{d\text{inv}}^{\text{fit}}$ further fixes the relative orientation of the three inverse-tuple branches in Brannen space.

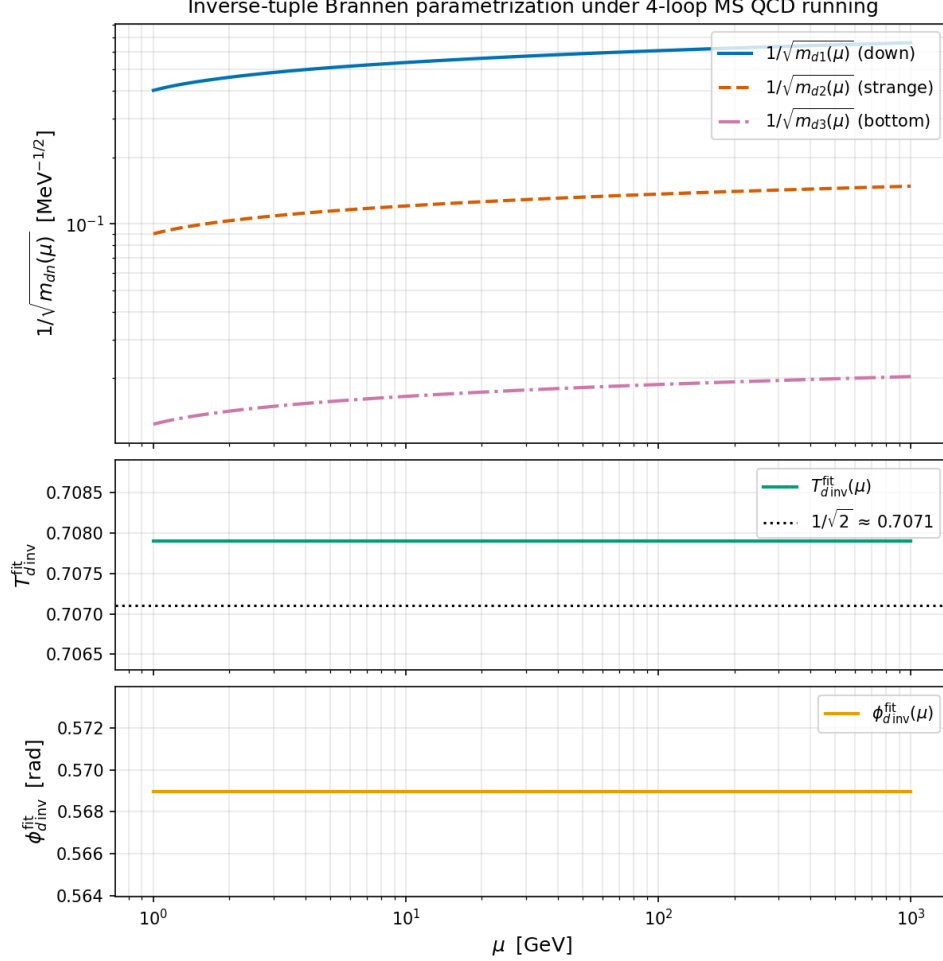


Figure 5. Inverse-tuple Brannen parametrization under 4-loop $\overline{\text{MS}}$ QCD running. Top: the three inverse square-root masses $\frac{1}{\sqrt{m_{dn}(\mu)}}$ for $n = 1, 2, 3$ (down, strange, bottom) on a log-log axis; only the overall amplitude $A_{dinv}^{\text{fit}}(\mu)$ carries the QCD dressing. Middle: the Brannen ratio $T_{dinv}^{\text{fit}}(\mu)$ compared with the symmetric reference $\frac{1}{\sqrt{2}} \approx 0.7071$; within the QCD-only, flavour-universal running approximation adopted in this work, $T_{dinv}^{\text{fit}} \approx 0.7079$ is exactly scale-independent, with the residual gap $\sim 8 \times 10^{-4}$ invisible to QCD evolution. Bottom: the Brannen phase $\phi_{dinv}^{\text{fit}}(\mu)$, likewise constant at $\approx +0.569$ rad over the entire perturbative window, confirming that the full Brannen shape of the inverse tuple (both T_{dinv}^{fit} and ϕ_{dinv}^{fit}) is preserved under QCD running and only the amplitude runs. Since Eq. (31) is solved in closed form (three equations for three unknowns), the reconstruction error stays at the 10^{-12} level, identical to that exhibited in the bottom panel of Figure 2; an analogous error panel is therefore omitted. The plot is produced by Script 4 (Appendix D).

The residual deviation $K_{\text{inv}} - \frac{2}{3} \approx 7 \times 10^{-4}$ is, in the language of Eq. (32), a residual $T_{d\text{inv}}^{\text{fit}} - \frac{1}{\sqrt{2}} \approx 8 \times 10^{-4}$, and is invisible to the QCD-only RG evolution implemented in this work. Whether the inclusion of the full Standard-Model running, including the top-Yukawa contribution and the electroweak gauge contributions at $\mu \gg M_Z$, further reduces this residual is left as an external open question and is not assumed in the present analysis. A specific proposal in this direction has been discussed in Ref. [15]; the present paper neither verifies nor relies on it, and treats the proximity $T_{d\text{inv}}^{\text{fit}} \approx \frac{1}{\sqrt{2}}$ strictly as an empirical observation at the scales considered here.

4.4 Interpretation, scope, and limitations

The main conceptual gain of the present reconstruction is not a new algebraic identity but a cleaner separation between two logically different ingredients (its value is to separate the fixed benchmark skeleton from the RG dressing and to identify which inverse-tuple shape parameter is close to its symmetric value):

a Brannen-type geometric skeleton and QCD renormalization-group dressing.

The benchmark phase choice organizes the mixed-scale hierarchy, while the common-scale running describes why the two light branches move coherently under transport from 2 GeV to $\mu = m_b$. In particular, the recalculated factors in Eq. (23) show that the common reduction of the light square-root masses is a QCD effect, not a second independent trigonometric coincidence.

This perspective also clarifies why a separate long Results section is unnecessary in the present manuscript. Most of the substantive content consists of the construction itself: the benchmark ansatz, the running extension, the common-scale reconstruction, and the associated numerical plots. The function of the present section is therefore to isolate the final numerical messages from the derivation, not to repeat the derivation in expanded form.

The principal theoretical gap remains unchanged. The benchmark choice $\phi_d = \frac{1}{3}$ together with $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$ works well numerically, but its microscopic origin is unknown. The stability of K_{inv} should therefore be understood as an empirical consistency property, not as a first-principles explanation. Clarifying whether this pattern is accidental, emergent, or symmetry-based remains the main open problem.

A second limitation is methodological. The running analysis is carried out with standard $\overline{\text{MS}}$ evolution and continuous threshold matching, which is adequate for the present exploratory purpose but does not replace a more systematic effective-field-theory treatment of threshold decoupling. Even so, the near constancy of K_{inv} appears sufficiently robust to justify further investigation.

5 Conclusions

We have presented a Brannen-type phenomenological parametrization of the down-type quark masses, summarized in Eqs. (12) and (19), with the benchmark phase choice $\phi_d = \frac{1}{3}$ and $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$. The empirical content of the present analysis can be summarized in three points:

1. the fixed benchmark phase pair organizes the mixed-scale down-type hierarchy at the percent level, comparable to the current PDG 2024 input uncertainties;
2. QCD transport from the mixed PDG reference scales to a common scale gives a nearly common multiplicative dressing factor $\sqrt{Z_m} \approx 0.9191$ for the d and s square-root masses, while the b branch is anchored at its own reference scale $\mu = m_b$;
3. the inverse-tuple shape parameter $T_{d\text{inv}}^{\text{fit}}$ is scale-stable within the QCD-only, flavour-universal running approximation adopted here and lies empirically close to $\frac{1}{\sqrt{2}}$, equivalent to K_{inv} close to $\frac{2}{3}$ via the closed-form identity $K_{\text{inv}} = \frac{1}{3}(1 + 2[T_{d\text{inv}}^{\text{fit}}]^2)$.

The preferred reading of the construction is a mixed-scale Brannen-type benchmark together with QCD RG dressing, and the common-scale reconstruction cleanly separates these two ingredients.

Formally the benchmark expression contains three quantities (A, T_d, ϕ_d) , but in the present work two of them are fixed to phenomenologically motivated benchmark values, while the overall normalization is

fixed from the mixed-scale square-root sum. In that sense the benchmark construction is best regarded as a three-parameter ansatz with two frozen benchmark inputs, and its non-trivial content lies in the resulting mass hierarchy and the associated benchmark value of the direct Koide ratio.

The preferred reading of the present construction is that the benchmark formula provides a *Brannen-type geometric skeleton*, while the common-scale evolution provides *QCD RG dressing*. With the same numerical conventions used in the scripts, the evolution of the light branches from 2 GeV to $\mu = m_b$ gives

$$m_d(m_b) \approx 3.970 \text{ MeV}, \quad m_s(m_b) \approx 78.984 \text{ MeV},$$

so that both light square-root masses acquire an almost common multiplicative factor ≈ 0.9191 . In this sense, the common-scale fit is not a replacement for the benchmark skeleton but its dressed version.

We then reformulated the construction at a common renormalization scale μ . In that form, the running masses are recovered with negligible numerical residual as a diagnostic reconstruction (i.e. a closed-form three-parameter representation of three masses, not an independent prediction) when the amplitude is allowed to run as $A_d^{\text{fit}}(\mu)$, while the fit parameters T_d^{fit} and ϕ_d^{fit} , which differ from the mixed-scale benchmark values, are kept approximately scale independent. The most robust *a posteriori* observation is not the benchmark mass table itself, but the near constancy and transport stability of the inverse-Koide quantity K_{inv} between the mixed-scale and common-scale descriptions, while the direct Koide combination is neither numerically close to $\frac{2}{3}$ at the quoted scales nor as well protected under that transport. As shown in Section 4.3 and Figure 5, the same observation can be recast as a Brannen-type parametrization of the inverse tuple $\frac{1}{\sqrt{m_{dn}(\mu)}}$, in which the ratio parameter $T_{d\text{inv}}^{\text{fit}} \approx 0.7079$ is exactly scale-independent within the QCD-only, flavour-universal running approximation adopted in this work and lies within $\sim 8 \times 10^{-4}$ of the symmetric value $\frac{1}{\sqrt{2}}$; whether the residual gap $T_{d\text{inv}}^{\text{fit}} - \frac{1}{\sqrt{2}}$ is further reduced once full Standard-Model running is included at $\mu \gg M_Z$ is left as an external open question. A discussion in this direction is given in Ref. [15], but the present paper does not assume any such UV-exact statement.

Several limitations should be kept in mind. No first-principles derivation of the benchmark values is given; the phase pair $(\phi_d, \cos \delta_d)$, the proximity $T_{d\text{inv}}^{\text{fit}} \approx \frac{1}{\sqrt{2}}$, and the structural identity $K_{\text{inv}} = \frac{1}{3} (1 + 2[T_{d\text{inv}}^{\text{fit}}]^2)$ are documented as numerical and algebraic observations rather than predictions of an underlying theory, and their microscopic origin remains an open theoretical question. The work should not be read as evidence for a fundamental common-scale mass law. The low-energy extrapolation of Figure S.1 is illustrative only, sits close to the lower edge of the perturbative window, and has no standalone physical status. The construction is therefore best regarded as a compact empirical organization rather than a completed theory; its value is to make explicit which part of the down-type structure is captured by the Brannen-type phase skeleton, which part is RG kinematics, and why the inverse-Koide viewpoint is more robust than the direct one for running quark masses.

As one potential avenue for a first-principles understanding, we note in passing that Brannen's three-generation circulant matrix $\Gamma(\mu, \eta, \delta)$ can be read as the Gram matrix of three qutrit states in \mathbb{C}^3 , with η playing the role of the overlap magnitude $|\langle a|b \rangle|$ and δ being proportional to a three-vertex Bargmann (geometric) phase $\Phi_B = \arg \langle r|g \rangle \langle g|b \rangle \langle b|r \rangle$ via $\delta = \frac{1}{3} \Phi_B$ [6, 7]. In this reading, Koide's condition $\eta^2 = \frac{1}{2}$ corresponds to an equilateral configuration of the three generation states in \mathbb{CP}^2 , whereas the down-type benchmark $\cos(\delta_d) = \frac{1}{2} \cos(\frac{3}{8}\pi)$ departs from that configuration in a specific way. Whether this qutrit / \mathbb{CP}^2 framework can accommodate the down-type benchmark values, and in particular the overall factor $\frac{1}{2}$, is an interesting question which we leave for future work.

Specific items that deserve further investigation include: robustness of the pattern under alternative threshold prescriptions and higher-order matching, a systematic comparison with neighbouring benchmark choices and with unconstrained free fits, a possible relation to Brannen's original circulant framework or to qutrit / \mathbb{CP}^2 interpretations, and, if any, an extension to quark mixing or to wider flavour-sector observables. The construction should be judged as a compact empirical parametrization and scale-bookkeeping exercise, not as a completed theory of flavour; the value of the present work lies in its empirical organization of the down-type spectrum rather than in any theoretical claim beyond it.

Acknowledgements

The authors thank Alejandro Rivero (Universidad de Zaragoza) for valuable correspondence. After our initial observation of the non-unified down-type mass formula, he pointed out that the associated inverse Koide quantity $K_{\text{inv}}(d, s, b)$ remains numerically close to $\frac{2}{3}$, both in the quoted mixed-scale form and in the common-scale reconstruction, and highlighted its renormalization-group stability [15]. His observation helped motivate the common-scale analysis summarized in Section 4 and detailed further in Section S.4.

Supplemental Material

This Supplemental Material collects reference information that would interrupt the flow of the main text if presented there in full. Sections S.1–S.4 clarify the parameter conventions and the interpretation of the dual Koide quantity. The Python scripts used for the numerical figures are reproduced separately as Appendices A–D, and the auxiliary script for the visualization remark in Section S.3 as Appendix E.

S.1 Reference charged-lepton parameters

Following Brannen, for the charged-lepton realization we denote the relevant parameters by δ_e and ϕ_e . The quantity δ_e is associated with the angle formed by three mutually orthogonal generation vectors, so that

$$\delta_e = \frac{1}{2}\pi [\text{rad}] \quad \Rightarrow \quad \cos \delta_e = 0. \quad (36)$$

The phase ϕ_e is taken to be half of the oriented area of the corresponding spherical triangle:

$$\phi_e \equiv \frac{1}{2} \text{Area} \left(\triangle(\vec{r}, \vec{g}, \vec{b}) \right). \quad (37)$$

Brannen [4] reported the numerical value

$$\phi_e = \frac{2}{3}, \quad (38)$$

which we adopt throughout the present paper as the charged-lepton reference point.

S.2 Benchmark interpretation of $\cos(\frac{3}{8}\pi)$ and the origin of the $\frac{1}{2}$ factor

In the main text, the down-type mass formula is written as

$$\sqrt{m_{dn}} = \frac{53.148}{3^{\frac{2}{3}}} \left(1 + 2T_d \cos \left(\frac{1}{3}\phi_d + \frac{2n\pi}{3} \right) \right) \text{ MeV}^{\frac{1}{2}}, \quad n = 1, 2, 3, \quad (39)$$

with

$$T_d = \sqrt{\frac{1 + \cos(\delta_d)}{2}}. \quad (40)$$

From the observed down-type quark masses, one finds phenomenologically that

$$2T_d \approx 1.543, \quad (41)$$

which implies

$$\cos(\delta_d) \approx 0.190. \quad (42)$$

We therefore adopt the nearby exact benchmark value

$$\cos(\delta_d) = \frac{1}{2} \cos \left(\frac{3}{8}\pi \right) = \frac{\sqrt{2 - \sqrt{2}}}{4} \approx 0.191342 \dots \quad (43)$$

It is important to emphasize that, in the present work, this value is introduced as a phenomenological benchmark motivated by the mass fit. The primary input is the fitted value $\cos(\delta_d) \approx 0.190$, while

$$\frac{1}{2} \cos \left(\frac{3}{8}\pi \right) = \frac{\sqrt{2 - \sqrt{2}}}{4}$$

is chosen as a compact nearby exact expression that lies well within the PDG 2024 input window.

We do not attempt to derive this benchmark from an underlying geometric or symmetry construction in the present paper. For earlier geometric viewpoints on Koide-type relations that might inspire such a derivation, see Refs. [8, 12]. A possible reformulation in a qutrit / \mathbb{CP}^2 language is mentioned briefly as a future direction in Section 5.

S.3 Low-energy behavior of the running extension: a visualization remark

This auxiliary section documents the low-energy behavior of the running extension introduced in Section 3.3. The discussion is collected here, separately from the main argument, because the apparent infrared clustering visible below is a visualization artifact of the chosen extrapolation procedure rather than evidence for a physical unification scale.

Figure 1 shows that the two reconstructed low-energy scales move toward the same infrared region as μ_{d3} decreases. Within the displayed perturbative window, one may visualize an apparent low-energy clustering near the infrared singular region, around $\mu_{lp} \approx 0.587$ GeV, but this trend must be interpreted with great care. The low-energy behavior is displayed more clearly in Figure S.1. The plotting routine used for Figure S.1 is provided in the script reproduced below.

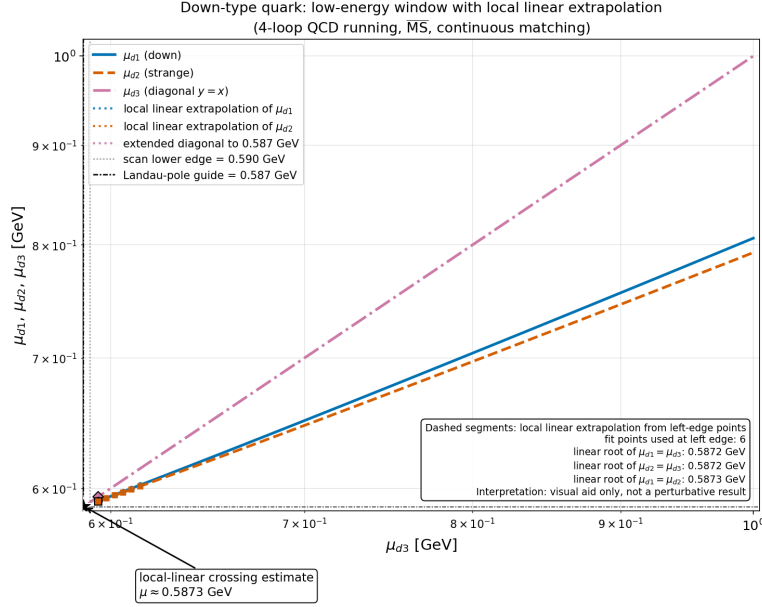


Figure S.1. For reference only: infrared behavior of the running-extension construction. The dashed local-linear extension is only a visualization aid near the lower edge of the perturbative window; the indicated low-energy crossing is an extrapolation artifact close to the Landau-pole region and should not be interpreted as evidence for a physical unification scale.

The apparent convergence in Figure S.1 is not evidence for a dynamical infrared unification. As $\mu_{d3} \rightarrow \mu_{lp}$, the running amplitude satisfies $A_d(\mu_{d3}) \rightarrow +\infty$, so the reconstructed scales μ_{d1} and μ_{d2} are pushed toward the same infrared region largely for kinematic reasons. Moreover, the plotted “crossing” comes from a local linear extrapolation performed extremely close to the lower edge of the perturbative window, where α_s is already large and perturbative control is poor. In that sense the crossing is a visualization artifact of the chosen extrapolation procedure, not a robust prediction of the model. No physical unification scale is claimed here.

The Python script that produced Figure S.1, including the local-linear extrapolation routine, is reproduced as Appendix E.

S.4 Renormalization-scale dependence and the stability of K_{inv}

In the present work we quote down-type quark masses as \overline{MS} running masses at a common renormalization scale μ [14]. At sufficiently low scales, strong-coupling effects grow and perturbation theory becomes less reliable. Correspondingly, heavy-quark pole-mass intuition should be used with care because pole masses suffer from infrared renormalon ambiguities of order $\mathcal{O}(\Lambda_{QCD})$ [16].

The key point is that under flavour-universal one-loop QCD running, the square roots acquire a common

multiplicative factor:

$$\sqrt{m_{dn}(\mu)} = \eta(\mu, \mu_0) \sqrt{m_{dn}(\mu_0)}. \quad (44)$$

Therefore,

$$\frac{1}{\sqrt{m_{dn}(\mu)}} = \frac{1}{\eta(\mu, \mu_0)} \frac{1}{\sqrt{m_{dn}(\mu_0)}}, \quad (45)$$

and all ratios of inverse square roots remain unchanged:

$$\frac{1/\sqrt{m_{dn}(\mu)}}{1/\sqrt{m_{dm}(\mu)}} = \frac{1/\sqrt{m_{dn}(\mu_0)}}{1/\sqrt{m_{dm}(\mu_0)}}. \quad (46)$$

Hence the normalized inverse pattern and K_{inv} are scale independent at this level.

It should also be emphasized that, once all three masses are quoted at a common scale, the same multiplicative cancellation holds for the direct Koide combination as well. The empirical content of the present paper is therefore not that K_{inv} obeys a fundamentally different renormalization-group law, but rather that K_{inv} is already numerically close to $\frac{2}{3}$ at the mixed PDG reference scales and remains close to that value after transport to a common scale, whereas the direct Koide ratio is neither close to $\frac{2}{3}$ nor as stable under the mixed-scale to common-scale comparison.

By contrast, the direct mass formula for $\sqrt{m_{dn}(\mu)}$ can appear distorted when one moves far away from the benchmark scale. Using standard four-loop running of α_s and m_q in the $\overline{\text{MS}}$ scheme [20, 21] together with the conventional treatment of heavy-flavour thresholds [17–19], one finds that evolving the masses changes the fitted prefactor and the effective phase in the direct Brannen form. In this restricted but important sense, the inverse Koide relation is the more robust object.

A Python Script 1: 4-loop $\overline{\text{MS}}$ running and μ_{d3} scan

```

1  #!/usr/bin/env python3
2  """
3  =====
4  Generate a two-dimensional log-log plot based on the down-type quark mass
5  formula
6  =====
7
8  Problem setup
9  -----
10 Horizontal axis:  $\mu_{d3}$  [GeV] (evaluation scale of the bottom quark)
11 Vertical axis:  $\mu_{d1}$  [GeV],  $\mu_{d2}$  [GeV]
12
13 The script computes the predicted masses from the down-type quark mass
14 formula and then determines the running-mass scales that reproduce those
15 masses by root finding.
16
17 Formula and algorithm
18 -----
19  $\sqrt{m_{dn}} = A(\mu_{d3}) * (1 + \sqrt{2 + \sqrt{2 - \sqrt{2}}}) / 2$ 
20  $\mu_{dn} = \mu_{d1} * \cos(1/9 + 2n\pi/3)$  [MeV(1/2)]
21  $n = 1$ : down,  $n = 2$ : strange,  $n = 3$ : bottom
22
23  $A(\mu_{d3})$  is determined from  $m_b(\overline{\text{MS}})(\mu_{d3})$ :
24  $\sqrt{m_b(\mu_{d3})} = A(\mu_{d3}) * k_3$ 
25 therefore
26  $A(\mu_{d3}) = \sqrt{m_b(\mu_{d3}) [\text{MeV}]} / k_3$ 
27
28 predicted mass:  $\mu_{dn}^{\text{pred}} = (A(\mu_{d3}) * k_n)^2$  [MeV]
29
30  $\mu_{d1}$  and  $\mu_{d2}$  are defined by
31  $\mu_{d1}^{\text{pred}}(\mu_{d3}) = \mu_{d1}^{\text{pred}}(\mu_{d3})$ 
32  $\mu_{d2}^{\text{pred}}(\mu_{d3}) = \mu_{d2}^{\text{pred}}(\mu_{d3})$ 
33
34 Running-mass and matching assumptions
35 -----
36  $\overline{\text{MS}}$  scheme, 4-loop QCD running
37  $\alpha_s(m_Z^2) = 0.1180$  (PDG 2024)
38  $m_Z = 91.1876$  GeV (PDG 2024)
39 Quark thresholds ( $\overline{\text{MS}}$  masses):
40  $m_c = 1.273$  GeV (PDG 2024, charm)
41  $m_b = 4.183$  GeV (PDG 2024, bottom)
42  $m_t = 162.5$  GeV (PDG 2024, top)
43 Threshold treatment:  $n_f \rightarrow n_f - 1$  (or the reverse) at  $\mu = m_q$ 
44  $\alpha_s$  matching: continuous at  $\mu = m_q$ 
45 mass matching: continuous at  $\mu = m_q$ 
46 Finite higher-order matching corrections from decoupling constants are
47 not implemented in this script.
48 PDG 2024 input values:
49  $m_d = 4.7$  MeV at  $\mu = 2.0$  GeV
50  $m_s = 93.5$  MeV at  $\mu = 2.0$  GeV
51  $m_b = 4183$  MeV at  $\mu = 4.183$  GeV ( $= m_b(m_b)$ )
52
53 How to run
54 -----
55  $\$ python down\_quark\_mass.py$ 
56
57 Required libraries
58 -----
59 numpy, scipy, matplotlib
60 =====
61 """
62
63 import numpy as np
64 from scipy.integrate import solve_ivp
65 from scipy.optimize import brentq

```

```

66 import matplotlib.pyplot as plt
67 import warnings
68
69 # =====
70 # Constants
71 # =====
72 MZ = 91.1876      # Z boson mass [GeV] (PDG 2024)
73 ALPHA_S_MZ = 0.1180 # alpha_s(m_Z^2) (PDG 2024)
74
75 # Quark thresholds: MS-bar masses m_q(m_q) [GeV] (PDG 2024)
76 MC_MC = 1.273     # charm: m_c(m_c) = 1.273 GeV
77 MB_MB = 4.183     # bottom: m_b(m_b) = 4.183 GeV
78 MT_MT = 162.5     # top: m_t(m_t) = 162.5 GeV
79
80 # Input quark masses [MeV] at given scales [GeV] (PDG 2024 central values)
81 MD_INPUT = 4.7     # m_d at mu=2 GeV [MeV]
82 MS_INPUT = 93.5     # m_s at mu=2 GeV [MeV]
83 MB_INPUT = 4183.0   # m_b at mu=m_b [MeV]
84 MU_D_INPUT = 2.0    # evaluation scale for m_d, m_s [GeV]
85
86 # =====
87 # QCD beta-function coefficients (MS-bar scheme)
88 # beta(alpha_s) = -beta_0 * (alpha_s/pi)^2 - beta_1 * (alpha_s/pi)^3 - ...
89 # Using standard normalization: d(alpha_s/pi)/d(ln mu^2) = - beta_i (alpha_s/pi)^{i+2}
90 # =====
91 def beta_coeffs(nf):
92     """
93     4-loop beta-function coefficients for n_f active flavours.
94     beta_0, beta_1, beta_2, beta_3 in the convention:
95     d a_s / d ln mu^2 = - beta_0 a_s^2 - beta_1 a_s^3 - beta_2 a_s^4 - beta_3 a_s^5
96     where a_s = alpha_s / pi.
97     """
98     b0 = (11.0 - 2.0 * nf / 3.0) / 4.0
99     b1 = (102.0 - 38.0 * nf / 3.0) / 16.0
100     b2 = (2857.0 / 2.0 - 5033.0 * nf / 18.0 + 325.0 * nf**2 / 54.0) / 64.0
101     # 4-loop coefficient (van Ritbergen, Vermaseren, Larin)
102     b3 = ((149753.0 / 6.0 + 3564.0 * 1.2020569031595942)
103           - (1078361.0 / 162.0 + 6508.0 * 1.2020569031595942 / 27.0) * nf
104           + (50065.0 / 162.0 + 6472.0 * 1.2020569031595942 / 81.0) * nf**2
105           + 1093.0 * nf**3 / 729.0) / 256.0
106     return b0, b1, b2, b3
107
108 # =====
109 # Quark mass anomalous dimension coefficients (MS-bar)
110 # gamma_m(alpha_s) = gamma_0 a_s + gamma_1 a_s^2 + gamma_2 a_s^3 + gamma_3 a_s^4
111 # d ln m / d ln mu^2 = -gamma_m
112 # =====
113 def gamma_m_coeffs(nf):
114     """
115     4-loop mass anomalous dimension coefficients.
116     gamma_0, gamma_1, gamma_2, gamma_3 in the convention:
117     d ln m / d ln mu^2 = -gamma_0 a_s - gamma_1 a_s^2 - gamma_2 a_s^3 - gamma_3 a_s^4
118     where a_s = alpha_s / pi.
119     References: Chetyrkin, Vermaseren, Larin (1997); Baikov et al.
120     """
121     zeta3 = 1.2020569031595942
122     zeta4 = np.pi**4 / 90.0
123     zeta5 = 1.0369277551433699
124
125     g0 = 1.0
126     g1 = (202.0 / 3.0 - 20.0 * nf / 9.0) / 16.0
127     g2 = (1249.0 - (2216.0 / 27.0 + 160.0 * zeta3 / 3.0) * nf
128           - 140.0 * nf**2 / 81.0) / 64.0
129     g3 = ((4603055.0 / 162.0 + 135680.0 * zeta3 / 27.0
130           - 8800.0 * zeta5)
131           - (91723.0 / 27.0 + 34192.0 * zeta3 / 9.0
132           - 880.0 * zeta4 - 18400.0 * zeta5 / 9.0) * nf

```

```

134         + (5242.0 / 243.0 + 800.0 * zeta3 / 9.0
135             - 160.0 * zeta4 / 3.0) * nf**2
136         + (-332.0 / 243.0 + 64.0 * zeta3 / 27.0) * nf**3
137     ) / 256.0
138     return g0, g1, g2, g3
139
140
141     # =====
142     # alpha_s threshold matching at mu = m_q(m_q)
143     # =====
144     def alpha_s_matching_up(a_s_low, nf_low):
145         """
146         Match alpha_s from n_f=nf_low to n_f=nf_low+1 at a quark threshold.
147         a_s=alpha_s/pi.
148
149         In this script finite decoupling constants are not included;
150         alpha_s is matched continuously at mu=m_q(m_q).
151         """
152         return a_s_low # continuous matching at mu = m_q(m_q)
153
154
155     def alpha_s_matching_down(a_s_high, nf_high):
156         """Match alpha_s from n_f=nf_high to n_f=nf_high-1.
157
158         In this script finite decoupling constants are not included;
159         alpha_s is matched continuously at mu=m_q(m_q).
160         """
161         return a_s_high # continuous matching at mu = m_q(m_q)
162
163
164     # =====
165     # RGE for a_s = alpha_s/pi as function of t = ln(mu^2/mu_0^2)
166     # =====
167     def das_dt(t, a_s, nf):
168         """
169         d(a_s)/dt where t=ln(mu^2/mu_0^2), a_s=alpha_s/pi.
170         4-loop beta-function.
171         """
172         b0, b1, b2, b3 = beta_coeffs(nf)
173         return -(b0 * a_s**2 + b1 * a_s**3 + b2 * a_s**4 + b3 * a_s**5)
174
175
176     def run_alpha_s_segment(a_s_start, mu_start, mu_end, nf, rtol=1e-12):
177         """
178         Run alpha_s/pi from mu_start to mu_end with n_f fixed flavours (4-loop).
179         Returns a_s at mu_end.
180         """
181         if abs(mu_end - mu_start) / mu_start < 1e-14:
182             return a_s_start
183         t_start = 0.0
184         t_end = np.log(mu_end**2 / mu_start**2)
185         sol = solve_ivp(lambda t, y: das_dt(t, y[0], nf),
186                        [t_start, t_end], [a_s_start],
187                        method='RK45', rtol=rtol, atol=1e-15,
188                        max_step=abs(t_end - t_start) / 10)
189         if not sol.success:
190             raise RuntimeError(f"ODE solver failed for alpha_s running: {sol.message}")
191         return sol.y[0, -1]
192
193
194     # =====
195     # Complete alpha_s running from m_Z to arbitrary mu, with threshold matching
196     # =====
197     def alpha_s_at_mu(mu, a_s_mz=ALPHA_S_MZ / np.pi, mu0=MZ):
198         """
199         Compute a_s=alpha_s/pi at scale mu[GeV], starting from alpha_s(m_Z)/pi.
200         Includes 4-loop running with threshold crossings at m_c, m_b, m_t.
201         Finite decoupling constants are not included; alpha_s is matched continuously.

```

```

202
203 #####Threshold structure:
204 #####mu<_m_c:#####n_f=3
205 #####m_c<=mu<_m_b:#####n_f=4
206 #####m_b<=mu<_m_t:#####n_f=5
207 #####mu>=m_t:#####n_f=6
208
209 #####Matching at mu= m_q(m_q) is continuous in this implementation.
210 #####
211     a_s_current = a_s_mz
212     mu_current = mu0
213
214     if mu >= mu0:
215         # Run upward from m_Z
216         # First check if we cross m_t
217         if mu < MT_MT:
218             return run_alpha_s_segment(a_s_current, mu_current, mu, nf=5)
219         else:
220             # Run to m_t, match, then run to mu
221             a_s_current = run_alpha_s_segment(a_s_current, mu_current, MT_MT, nf=5)
222             a_s_current = alpha_s_matching_up(a_s_current, 5)
223             return run_alpha_s_segment(a_s_current, MT_MT, mu, nf=6)
224     else:
225         # Run downward from m_Z
226         if mu >= MB_MB:
227             return run_alpha_s_segment(a_s_current, mu_current, mu, nf=5)
228         else:
229             # Run to m_b
230             a_s_current = run_alpha_s_segment(a_s_current, mu_current, MB_MB, nf=5)
231             a_s_current = alpha_s_matching_down(a_s_current, 5)
232             if mu >= MC_MC:
233                 return run_alpha_s_segment(a_s_current, MB_MB, mu, nf=4)
234             else:
235                 # Run to m_c
236                 a_s_current = run_alpha_s_segment(a_s_current, MB_MB, MC_MC, nf=4)
237                 a_s_current = alpha_s_matching_down(a_s_current, 4)
238                 return run_alpha_s_segment(a_s_current, MC_MC, mu, nf=3)
239
240
241 # =====
242 # Quark mass running (MS-bar, 4-loop)
243 # =====
244 def run_mass_segment(m_start, a_s_start, mu_start, mu_end, nf, rtol=1e-12):
245     """
246     #####Run quark mass from mu_start to mu_end with n_f fixed flavours (4-loop).
247
248     #####The coupled system:
249     #####d(a_s)/dt=-beta_0 a_s^2 - beta_1 a_s^3 - beta_2 a_s^4 - beta_3 a_s^5
250     #####d(ln m)/dt=-gamma_0 a_s - gamma_1 a_s^2 - gamma_2 a_s^3 - gamma_3 a_s^4
251     #####where t=ln(mu^2/mu_0^2).
252
253     #####Returns (m_end, a_s_end) at mu_end.
254     #####
255     if abs(mu_end - mu_start) / max(mu_start, 1e-30) < 1e-14:
256         return m_start, a_s_start
257
258     t_start = 0.0
259     t_end = np.log(mu_end**2 / mu_start**2)
260
261     b0, b1, b2, b3 = beta_coeffs(nf)
262     g0, g1, g2, g3 = gamma_m_coeffs(nf)
263
264     def rhs(t, y):
265         a_s = y[0]
266         ln_m = y[1]
267         da = -(b0 * a_s**2 + b1 * a_s**3 + b2 * a_s**4 + b3 * a_s**5)
268         dlnm = -(g0 * a_s + g1 * a_s**2 + g2 * a_s**3 + g3 * a_s**4)
269         return [da, dlnm]

```



```

270
271     sol = solve_ivp(rhs, [t_start, t_end],
272                    [a_s_start, np.log(m_start)],
273                    method='RK45', rtol=rtol, atol=1e-15,
274                    max_step=abs(t_end - t_start) / 10)
275     if not sol.success:
276         raise RuntimeError(f"ODE solver failed for mass running: {sol.message}")
277
278     a_s_end = sol.y[0, -1]
279     m_end = np.exp(sol.y[1, -1])
280     return m_end, a_s_end
281
282
283 def running_mass(m_ref, mu_ref, mu_target):
284     """
285     Compute MS-bar running mass  $m(\mu_{\text{target}})$  given  $m(\mu_{\text{ref}})$ .
286
287     Mass matching at heavy-quark thresholds is treated as continuous
288     in this implementation.
289
290     Strategy: run  $\alpha_s$  and mass together through thresholds.
291
292     Parameters
293     -----
294     m_ref: float
295     Quark mass at  $\mu_{\text{ref}}$  [MeV]
296     mu_ref: float
297     Reference scale [GeV]
298     mu_target: float
299     Target scale [GeV]
300
301     Returns
302     -----
303     float:  $m(\mu_{\text{target}})$  [MeV]
304     """
305     # Get  $a_s$  at  $\mu_{\text{ref}}$ 
306     a_s_ref = alpha_s_at_mu(mu_ref)
307
308     # Thresholds
309     thresholds = sorted([MC_MC, MB_MB, MT_MT])
310
311     def get_nf(mu):
312         """Number of active flavours at scale  $\mu$ ."""
313         nf = 3
314         if mu >= MC_MC: nf = 4
315         if mu >= MB_MB: nf = 5
316         if mu >= MT_MT: nf = 6
317         return nf
318
319     mu_current = mu_ref
320     m_current = m_ref
321     a_s_current = a_s_ref
322
323     if mu_target > mu_ref:
324         # Run upward: build waypoints [mu_ref, thr1, thr2, ..., mu_target]
325         waypoints = [mu_ref]
326         for thr in thresholds:
327             if mu_ref < thr < mu_target:
328                 waypoints.append(thr)
329         waypoints.append(mu_target)
330
331         for i in range(len(waypoints) - 1):
332             mu_from = waypoints[i]
333             mu_to = waypoints[i + 1]
334             # Determine  $n_f$  using midpoint (avoids boundary ambiguity)
335             nf = get_nf((mu_from + mu_to) / 2.0)
336             m_current, a_s_current = run_mass_segment(
337                 m_current, a_s_current, mu_from, mu_to, nf)

```

```

338         mu_current = mu_to
339     else:
340         # Run downward: build waypoints [mu_ref, thr_high, ..., thr_low, mu_target]
341         waypoints = [mu_ref]
342         for thr in reversed(thresholds):
343             if mu_target < thr < mu_ref:
344                 waypoints.append(thr)
345         waypoints.append(mu_target)
346
347         for i in range(len(waypoints) - 1):
348             mu_from = waypoints[i]
349             mu_to = waypoints[i + 1]
350             # Determine n_f using midpoint (avoids boundary ambiguity)
351             nf = get_nf((mu_from + mu_to) / 2.0)
352             m_current, a_s_current = run_mass_segment(
353                 m_current, a_s_current, mu_from, mu_to, nf)
354             mu_current = mu_to
355
356     return m_current
357
358 # =====
359 # Coefficients of the down-type quark mass formula
360 # =====
361 def mass_formula_k(n):
362     """
363     Bracket coefficient k_n of the mass formula:
364     k_n = 1 + sqrt(2 + sqrt(2 - sqrt(2))) / 2 * cos(1/9 + 2*n*pi/3)
365     n = 1 (down), 2 (strange), 3 (bottom)
366     Angle in radians.
367     """
368     prefactor = np.sqrt(2.0 + np.sqrt(2.0 - np.sqrt(2.0))) / 2.0
369     angle = 1.0 / 9.0 + 2.0 * n * np.pi / 3.0 # [radians]
370     return 1.0 + prefactor * np.cos(angle)
371
372
373 # Precompute the coefficients
374 K1 = mass_formula_k(1) # down
375 K2 = mass_formula_k(2) # strange
376 K3 = mass_formula_k(3) # bottom
377
378
379 def compute_A(mu_d3):
380     """
381     Compute the amplitude A(mu_d3).
382
383     A(mu_d3) = sqrt(m_b(mu_d3) [MeV]) / k_3
384
385     m_b(mu_d3) is the bottom-quark MS-bar running mass evaluated at the
386     scale mu_d3. Reference: m_b(m_b) = 4183 MeV.
387     """
388     # Obtain m_b(mu_d3) via RG running [MeV]
389     mb_at_mud3 = running_mass(MB_INPUT, MB_MB, mu_d3)
390     A = np.sqrt(mb_at_mud3) / K3
391     return A
392
393
394 def predicted_mass(A, n):
395     """
396     Compute the predicted mass from the mass formula [MeV].
397     m_dn^pred = (A * k_n)^2
398     """
399     kn = mass_formula_k(n)
400     return (A * kn) ** 2
401
402
403 # =====
404 # Numerical determination of mu_d1, mu_d2 via root finding
405

```

```

406 # =====
407 def find_mu_scale(m_pred, m_ref, mu_ref, mu_low=0.65, mu_high=1000.0):
408     """
409     Find the scale mu that satisfies  $m_q(\overline{MS})(\mu) = m_{\text{pred}}$  using brentq.
410
411     The running mass is a decreasing function of  $\mu$  (asymptotic freedom),
412     so we solve  $f(\mu) = \text{running\_mass}(m_{\text{ref}}, \mu_{\text{ref}}, \mu) - m_{\text{pred}} = 0$ .
413
414     Near the lower bound  $\alpha_s$  becomes large and perturbation theory is
415     unreliable; the code therefore searches dynamically for an effective
416     lower bound. Following the user's specification, tentative solutions
417     below 1 GeV are also allowed for  $\mu_{d1}$  and  $\mu_{d2}$ .
418
419     Parameters
420     -----
421     m_pred: float
422     Target mass [MeV]
423     m_ref: float
424     Reference mass [MeV] at  $\mu_{\text{ref}}$ 
425     mu_ref: float
426     Reference scale [GeV]
427     mu_low, mu_high: float
428     Search bounds [GeV]
429
430     Returns
431     -----
432     float:  $\mu$  [GeV] or NaN if no solution
433     """
434     def func(mu):
435         return running_mass(m_ref, mu_ref, mu) - m_pred
436
437     try:
438         # Dynamically determine an effective lower bound in case the ODE
439         # diverges at the nominal lower bound.
440         f_low = None
441         effective_mu_low = mu_low
442         # Try the nominal lower bound first; raise it step by step on failure.
443         test_points = [mu_low, mu_low * 1.1, mu_low * 1.3, mu_low * 1.5,
444                       mu_low * 2.0]
445         if 1.0 < mu_high:
446             test_points.append(1.0)
447         for test_mu in test_points:
448             if test_mu >= mu_high:
449                 break
450             try:
451                 f_low = func(test_mu)
452                 effective_mu_low = test_mu
453                 break
454             except (RuntimeError, ValueError):
455                 continue
456
457         if f_low is None:
458             return np.nan
459
460         f_high = func(mu_high)
461
462         if f_low * f_high > 0:
463             # No sign change: no solution in range
464             return np.nan
465
466         mu_sol = brentq(func, effective_mu_low, mu_high, xtol=1e-8, rtol=1e-10)
467         return mu_sol
468     except (ValueError, RuntimeError):
469         return np.nan
470
471 # =====
472 # Main calculation

```

```

474 # =====
475 def main():
476     print("=" * 70)
477     print("Down-type quark mass formula: generation of a 2D log-log plot")
478     print("=" * 70)
479
480     # -----
481     # 1. Print the mass-formula coefficients
482     # -----
483     print(f"\n* Mass-formula coefficients k_n:")
484     print(f"    k_1 (down) = {K1:.10f}")
485     print(f"    k_2 (strange) = {K2:.10f}")
486     print(f"    k_3 (bottom) = {K3:.10f}")
487
488     # -----
489     # 2. Consistency check for A(4.183 GeV)
490     # -----
491     A_check = compute_A(MB_MB)
492     print(f"\n* A(4.183 GeV) consistency check:")
493     print(f"    m_b(m_b) = {MB_INPUT:.1f} MeV")
494     print(f"    k_3 = {K3:.10f}")
495     print(f"    A(4.183) = sqrt({MB_INPUT:.1f}) / {K3:.10f}")
496     print(f"    A_check = {A_check:.6f}")
497     print(f"    Expected ~ 25.522566")
498     print(f"    Difference = {abs(A_check - 25.522566):.6e}")
499     assert abs(A_check - 25.522566) < 0.01, \
500         f"A(4.183 GeV) = {A_check}, expected ~ 25.522566"
501     print("    OK: consistency check passed")
502
503     # -----
504     # 3. Predicted masses at the reference point
505     # -----
506     print(f"\n* Predicted masses at the reference point (mu_d3 = 4.183 GeV):")
507     A_ref = A_check
508     m_d_pred_ref = predicted_mass(A_ref, 1)
509     m_s_pred_ref = predicted_mass(A_ref, 2)
510     m_b_pred_ref = predicted_mass(A_ref, 3)
511     print(f"    m_d^pred = {m_d_pred_ref:.4f} MeV (input: {MD_INPUT} MeV at 2 GeV)")
512     print(f"    m_s^pred = {m_s_pred_ref:.4f} MeV (input: {MS_INPUT} MeV at 2 GeV)")
513     print(f"    m_b^pred = {m_b_pred_ref:.4f} MeV (input: {MB_INPUT} MeV at m_b)")
514
515     # m_d at 2 GeV from running
516     m_d_at_2 = running_mass(MD_INPUT, MU_D_INPUT, 2.0)
517     m_s_at_2 = running_mass(MS_INPUT, MU_D_INPUT, 2.0)
518     print(f"\n* Running-mass check:")
519     print(f"    m_d(2 GeV) = {m_d_at_2:.4f} MeV (should be {MD_INPUT})")
520     print(f"    m_s(2 GeV) = {m_s_at_2:.4f} MeV (should be {MS_INPUT})")
521
522     # Additional internal consistency check: run to a scale and come back.
523     mu_test = 10.0
524     mb_at_10 = running_mass(MB_INPUT, MB_MB, mu_test)
525     mb_back = running_mass(mb_at_10, mu_test, MB_MB)
526     print(f"\n* Running-mass round-trip check:")
527     print(f"    m_b(10 GeV) = {mb_at_10:.4f} MeV")
528     print(f"    m_b(10 GeV -> m_b) = {mb_back:.4f} MeV")
529     print(f"    round-trip error = {abs(mb_back - MB_INPUT):.6e} MeV")
530
531     # -----
532     # 4. Scan mu_d3 and generate plot data
533     # -----
534     print(f"\n* mu_d3 scan range: 1.0 - 1000.0 GeV (log-spaced)")
535     N_POINTS = 100 # number of grid points
536     mu_d3_array = np.logspace(np.log10(1.0), np.log10(1000.0), N_POINTS)
537
538     mu_d1_array = np.full(N_POINTS, np.nan)
539     mu_d2_array = np.full(N_POINTS, np.nan)
540
541     print("    Computing...")

```

```

542 for i, mu_d3 in enumerate(mu_d3_array):
543     try:
544         # Step 1: Obtain the running bottom mass  $m_b(\mu_{d3})$ .
545         # Step 2: Determine  $A(\mu_{d3})$ .
546         A_val = compute_A(mu_d3)
547
548         # Step 3: Compute the predicted mass [MeV].
549         m_d1_pred = predicted_mass(A_val, 1) # down
550         m_d2_pred = predicted_mass(A_val, 2) # strange
551
552         # Step 4: Determine  $\mu_{d1}$ ,  $\mu_{d2}$  via root finding.
553         #  $m_d^{(MS-\bar{)}}(\mu_{d1}) = m_{d1\_pred}$ 
554         #  $\rightarrow$  running_mass(MD_INPUT, 2.0 GeV,  $\mu_{d1}$ ) =  $m_{d1\_pred}$ 
555         if m_d1_pred > 0:
556             mu_d1_array[i] = find_mu_scale(
557                 m_d1_pred, MD_INPUT, MU_D_INPUT, mu_low=0.65, mu_high=1000.0)
558
559         #  $m_s^{(MS-\bar{)}}(\mu_{d2}) = m_{d2\_pred}$ 
560         if m_d2_pred > 0:
561             mu_d2_array[i] = find_mu_scale(
562                 m_d2_pred, MS_INPUT, MU_D_INPUT, mu_low=0.65, mu_high=1000.0)
563
564     except Exception as e:
565         # Leave as NaN if no solution exists.
566         warnings.warn(f"mu_d3={mu_d3:.3f} GeV: {e}")
567         continue
568
569     if (i + 1) % 20 == 0:
570         print(f".....{i+1}/{N_POINTS} points completed")
571
572 print(f"All {N_POINTS} points computed")
573
574 # Count valid data points.
575 n_valid_d1 = np.sum(~np.isnan(mu_d1_array))
576 n_valid_d2 = np.sum(~np.isnan(mu_d2_array))
577 print(f"Valid data points: {mu_d1}={n_valid_d1}, {mu_d2}={n_valid_d2}")
578
579 # -----
580 # 5. Produce the plot
581 # -----
582 print("\n* Generating plot...")
583 fig, ax = plt.subplots(1, 1, figsize=(10, 7))
584
585 # Color-blind-safe palette (Okabe-Ito family) chosen to avoid the
586 # red/green confusion common in deuteranopia/protanopia.
587 # We also separate the series by line style so the curves remain
588 # distinguishable in grayscale printouts.
589 color_d1 = "#0072B2" # blue
590 color_d2 = "#D55E00" # vermillion
591 color_ref = "#999999" # neutral gray for the reference marker
592 ls_d1 = "--"
593 ls_d2 = "--"
594 ls_ref = "-."
595
596 #  $\mu_{d1}$  curve (down quark)
597 mask1 = ~np.isnan(mu_d1_array)
598 ax.plot(mu_d3_array[mask1], mu_d1_array[mask1],
599         color=color_d1, linestyle=ls_d1, linewidth=2.2,
600         label=r'$\mu_{d1}$ (down)')
601
602 #  $\mu_{d2}$  curve (strange quark)
603 mask2 = ~np.isnan(mu_d2_array)
604 ax.plot(mu_d3_array[mask2], mu_d2_array[mask2],
605         color=color_d2, linestyle=ls_d2, linewidth=2.2,
606         label=r'$\mu_{d2}$ (strange)')
607
608 # Reference-point marker
609 ax.axvline(x=MB_MB, color=color_ref, linestyle=ls_ref, alpha=0.7,

```

```

610         linewidth=1.5,
611         label=r'$\mu_{d3} = m_b(m_b) = 4.183$ GeV')
612
613     ax.set_xscale('log')
614     ax.set_yscale('log')
615     ax.set_xlabel(r'$\mu_{d3}$ [GeV]', fontsize=14)
616     ax.set_ylabel(r'$\mu_{d1}, \mu_{d2}$ [GeV]', fontsize=14)
617     ax.set_title(
618         r'Down-type quark mass formula: $\mu_{d1}(\mu_{d3})$ and $\mu_{d2}(\mu_{d3})$'
619         r'\n(4-loop QCD running, continuous threshold matching, $\overline{\mathrm{MS}}$'
620         r'scheme)',
621         fontsize=13)
622     ax.legend(fontsize=12, loc='best')
623     ax.grid(True, which='both', alpha=0.3)
624     ax.tick_params(labelsize=12)
625
626     plt.tight_layout()
627
628     # Save figure
629     output_path = 'down_quark_mass_plot_fixed.png'
630     fig.savefig(output_path, dpi=150, bbox_inches='tight')
631     print(f"Plot saved: {output_path}")
632
633     plt.close(fig)
634
635     # -----
636     # 6. Print numerical notes
637     # -----
638     print("\n" + "=" * 70)
639     print("* Numerical notes")
640     print("=" * 70)
641     print("""
642 1. Uses the 4-loop beta-function and 4-loop mass anomalous dimension.
643 2. At each threshold $n_f$ is switched and the RG is solved piecewise.
644    Finite higher-order decoupling corrections are not included;
645    alpha_s and the light-quark masses are matched continuously at
646    mu = m_q(m_q).
647 3. Root finding uses brentq (bracketing method, numerically stable).
648    Following the user's specification, the search range is
649    [0.65 GeV, 1000 GeV], and tentative solutions below 1 GeV
650    for mu_{d1}, mu_{d2} are accepted.
651 4. ODE solver: RK45 (relative tolerance 1e-12).
652 5. Units: masses in MeV, scales in GeV. Conversions are explicit.
653 6. PDG 2024 values used:
654    alpha_s(m_Z) = 0.1180
655    m_Z = 91.1876 GeV
656    m_c(m_c) = 1.273 GeV
657    m_b(m_b) = 4.183 GeV
658    m_t(m_t) = 162.5 GeV
659    m_d(2 GeV) = 4.7 MeV
660    m_s(2 GeV) = 93.5 MeV
661    """)
662
663     if __name__ == '__main__':
664         main()

```

B Python Script 2: common-scale reconstruction and K_{inv}

The following script reconstructs the down-type running masses at a common scale from the 4-loop $\overline{\text{MS}}$ running machinery, extracts the scale-dependent amplitude $A_d^{\text{fit}}(\mu)$, the ratio T_d^{fit} and the phase ϕ_d^{fit} of Eq. (26), and produces Figure 2 together with the numerical data underlying Table 2. The running machinery and threshold conventions are the same as in Script 1.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Log-x/ $\mu$  plot of the down-type running masses reconstructed from the
5  mass formula
6
7   $\sqrt{m_{dn}} = A_d(\mu) * [1 + 2 T_d(\mu) \cos(\phi_d/3 + 2\pi n/3)]$ 
8
9  with  $n = 1, 2, 3$ ,
10
11  where  $A_d(\mu)$ ,  $T_d(\mu)$ ,  $\phi_d(\mu)$  are extracted from the running masses
12   $m_{d1}(\mu)$ ,  $m_{d2}(\mu)$ ,  $m_{d3}(\mu)$  in  $\overline{MS}$  QCD.
13
14  What is plotted
15  -----
16  Top panel:
17   $\mu$ -masses reconstructed from the Brannen-type formula evaluated with the fit
18  parameters  $(A_d^{fit}(\mu), T_d^{fit}(\mu), \phi_d^{fit}(\mu))$ ,
19   $m_{d1}^{fit}(\mu)$ ,  $m_{d2}^{fit}(\mu)$ ,  $m_{d3}^{fit}(\mu)$ ,
20  with logarithmic y-axis.
21  Bottom panel:
22   $\mu$ -absolute relative difference between the fit-reconstructed masses and the
23  PDG-2024 running masses  $m_{dn}(\mu)$  obtained by 4-loop  $\overline{MS}$ -bar evolution,
24   $|\Delta m^{fit}| / m_{dn}^{fit} * 100 [\%]$ ,
25   $\Delta m^{fit}(\mu) = m_{dn}^{fit}(\mu) - m_{dn}(\mu)$ ,
26  also with logarithmic y-axis.
27
28  Why percentage difference is recommended
29  -----
30  The three down-type masses span orders of magnitude from a few MeV to several GeV.
31  Therefore a raw  $\Delta [MeV]$  plot is dominated by the heaviest state and is less useful
32  for comparing reconstruction quality across all three curves. For that reason the
33  bottom panel shows
34
35   $|\Delta m| / m_{dn} * 100 [\%]$ 
36
37  on a logarithmic scale.
38
39  What is saved internally
40  -----
41  The CSV/NPZ files contain:
42   $\mu$ ,  $\alpha_s$ , running masses  $m_{d1}$ ,  $m_{d2}$ ,  $m_{d3}$ 
43   $\sqrt{m_{dn}}$ ,  $1/\sqrt{m_{dn}}$ 
44   $K$ ,  $K_{inv}$ 
45   $A_d(\mu)$ ,  $T_d(\mu)$ ,  $\phi_d(\mu)$ 
46  masses reconstructed from the mass formula
47  signed differences  $\Delta m [MeV]$ 
48  signed relative differences  $[\%]$ 
49  absolute relative differences  $[\%]$ 
50
51  Input reference values (given by the user)
52  -----
53   $m_{d1}(2.000 GeV) = 4.7 MeV$ 
54   $m_{d2}(2.000 GeV) = 93.5 MeV$ 
55   $m_{d3}(4.183 GeV) = 4183 MeV$ 
56   $m_u(1.273 GeV) = 1273 MeV \rightarrow$  used here as the charm threshold scale
57   $m_u(162.500 GeV) = 162500 MeV \rightarrow$  used here as the top threshold scale
58   $\alpha_s(M_Z^2) = 0.1180$ 
59
60  Physics assumptions implemented here
61  -----
62  1)  $\overline{MS}$  scheme for quark masses and  $\alpha_s$ .
63  2) 4-loop QCD running for both  $\alpha_s$  and the quark-mass anomalous dimension.
64  3) Piecewise running with flavour thresholds at
65   $m_c = m_c(m_c) = 1.273 GeV$ ,
66   $m_b = m_b(m_b) = 4.183 GeV$ ,
67   $m_t = m_t(m_t) = 162.5 GeV$ .
68  4) "Continuous matching" baseline, as requested:

```

```

69  alpha_s^(nf)(mu_th)=alpha_s^(nf+1)(mu_th)
70  m_q^(nf)(mu_th)=m_q^(nf+1)(mu_th)
71  i.e. finite MSbar decoupling constants at the thresholds are NOT applied.
72  This is the simplest continuous prescription.
73
74  Important note on matching
75  -----
76  In strict EFT matching inside MSbar, finite threshold corrections start beyond leading order
77
78  So the present script is intentionally a continuous-matching approximation, not a full
79  4-loop decoupling implementation. If you want the strict EFT treatment, replace the two
80  functions
81  continuous_match_alpha(...)
82  continuous_match_mass(...)
83  by the appropriate decoupling factors.
84
85  Important note on the bottom contribution below mu_b
86  -----
87  The d3 input is interpreted as the bottom quark mass m_b(mu). To compute single continuous
88  curves over the full range 1--1000 GeV, this script continues m_b(mu) below mu_b using the
89  same continuous-matching convention. For a stricter EFT-oriented treatment, set
90  INCLUDE_HEAVY_BELOW_OWN_THRESHOLD=False. In that case all three-mass derived quantities
91  are set to NaN for mu < mu_b because m_d3(mu) is then excluded from the three-mass formula.
92  ""
93
94  from __future__ import annotations
95
96  import csv
97  import math
98  from dataclasses import dataclass
99  from typing import Callable, List, Tuple
100
101  import matplotlib.pyplot as plt
102  import numpy as np
103  from scipy.integrate import solve_ivp
104
105  # -----
106  # User-facing switches
107  # -----
108  INCLUDE_HEAVY_BELOW_OWN_THRESHOLD = True
109  SAVEFIG = True
110  SAVE_INTERNAL_DATA = True
111  OUTPUT_FIG = "Figure2.png"
112  OUTPUT_CSV = "quark_running_mass_formula_compare_v5_colorblind_data.csv"
113  OUTPUT_NPZ = "quark_running_mass_formula_compare_v5_colorblind_data.npz"
114  N_GRID = 800
115
116  # -----
117  # Input parameters
118  # -----
119  MZ_GEV = 91.1876          # Z-boson pole mass [GeV]
120  ALPHA_S_MZ = 0.1180      # alpha_s(M_Z^2)
121
122  MU_MIN = 1.0
123  MU_MAX = 1000.0
124
125  # Quark thresholds used for piecewise running.
126  # These are the assumptions requested to be stated explicitly in the code.
127  MU_C = 1.273              # charm threshold scale mu_c = m_c(m_c) [GeV]
128  MU_B = 4.183              # bottom threshold scale mu_b = m_b(m_b) [GeV]
129  MU_T = 162.5              # top threshold scale mu_t = m_t(m_t) [GeV]
130
131  M_D1_REF = 4.7            # MeV at mu = 2 GeV
132  MU_D1_REF = 2.0           # GeV
133  M_D2_REF = 93.5           # MeV at mu = 2 GeV
134  MU_D2_REF = 2.0           # GeV
135  M_D3_REF = 4183.0         # MeV at mu = 4.183 GeV (interpreted as bottom)
136  MU_D3_REF = MU_B          # GeV

```



```

136
137 # Mathematical constants
138 PI = math.pi
139 ZETA3 = 1.2020569031595942854
140 ZETA4 = PI**4 / 90.0
141 ZETA5 = 1.0369277551433699263
142
143
144 # -----
145 # 4-loop MSbar beta function and mass anomalous dimension
146 # Convention: a = alpha_s / (4*pi), and t = ln(mu^2)
147 # Then
148 # da/dt = - sum_{n=0}^3 beta_n a^{n+2}
149 # d ln m / dt = - sum_{n=0}^3 gamma_n a^{n+1}
150 # -----
151 def beta_coeffs(nf: int) -> Tuple[float, float, float, float]:
152     beta0 = 11.0 - 2.0 / 3.0 * nf
153     beta1 = 102.0 - 38.0 / 3.0 * nf
154     beta2 = 2857.0 / 2.0 - 5033.0 / 18.0 * nf + 325.0 / 54.0 * nf**2
155     beta3 = (
156         149753.0 / 6.0
157         + 3564.0 * ZETA3
158         - (1078361.0 / 162.0 + 6508.0 / 27.0 * ZETA3) * nf
159         + (50065.0 / 162.0 + 6472.0 / 81.0 * ZETA3) * nf**2
160         + 1093.0 / 729.0 * nf**3
161     )
162     return beta0, beta1, beta2, beta3
163
164
165 def gamma_coeffs(nf: int) -> Tuple[float, float, float, float]:
166     gamma0 = 4.0
167     gamma1 = 202.0 / 3.0 - 20.0 / 9.0 * nf
168     gamma2 = 1249.0 - (2216.0 / 27.0 + 160.0 / 3.0 * ZETA3) * nf - 140.0 / 81.0 * nf**2
169     gamma3 = (
170         4603055.0 / 162.0
171         + 135680.0 / 27.0 * ZETA3
172         - 8800.0 * ZETA5
173         + (-91723.0 / 27.0 - 34192.0 / 9.0 * ZETA3 + 880.0 * ZETA4 + 18400.0 / 9.0 * ZETA5)
174         * nf
175         + (5242.0 / 243.0 + 800.0 / 9.0 * ZETA3 - 160.0 / 3.0 * ZETA4) * nf**2
176         + (-332.0 / 243.0 + 64.0 / 27.0 * ZETA3) * nf**3
177     )
178     return gamma0, gamma1, gamma2, gamma3
179
180 def beta_a(a: float, nf: int) -> float:
181     beta0, beta1, beta2, beta3 = beta_coeffs(nf)
182     return -(beta0 * a**2 + beta1 * a**3 + beta2 * a**4 + beta3 * a**5)
183
184
185 def gamma_m(a: float, nf: int) -> float:
186     gamma0, gamma1, gamma2, gamma3 = gamma_coeffs(nf)
187     return gamma0 * a + gamma1 * a**2 + gamma2 * a**3 + gamma3 * a**4
188
189
190 # -----
191 # Threshold / matching assumptions
192 # -----
193 def active_nf(mu: float) -> int:
194     """Number of active flavours used for the EFT running at the scale mu."""
195     if mu < MU_C:
196         return 3
197     if mu < MU_B:
198         return 4
199     if mu < MU_T:
200         return 5
201     return 6
202

```

```

203
204 def continuous_match_alpha(alpha_in: float, nf_from: int, nf_to: int, mu_th: float) -> float
205 :
206     """
207     Continuous matching baseline requested by the user.
208
209     This enforces continuity of  $\alpha_s$  across the threshold and omits the finite
210     MSbar decoupling corrections.
211
212     To switch to strict EFT matching, replace the return value by the appropriate
213     decoupling relation  $\alpha_s^{(nf\_to)}(\mu\_th) = \zeta_g^2 \alpha_s^{(nf\_from)}(\mu\_th)$ .
214     """
215     _ = (nf_from, nf_to, mu_th)
216     return alpha_in
217
218 def continuous_match_mass(m_in: float, nf_from: int, nf_to: int, mu_th: float, heavy_name:
219     str | None = None) -> float:
220     """
221     Continuous matching baseline requested by the user.
222
223     For light quarks this keeps  $m_q$  continuous across the threshold.
224     For the bottom contribution (heavy_name='b') we also keep continuity so that
225     the three-mass formula can be evaluated continuously over the full  $x$ -range
226     when desired.
227
228     To switch to strict EFT matching, replace the return value by the appropriate
229     MSbar mass decoupling factor.
230     """
231     _ = (nf_from, nf_to, mu_th, heavy_name)
232     return m_in
233
234 # -----
235 # Dense piecewise solutions for  $\alpha_s(\mu)$ 
236 # -----
237 @dataclass
238 class AlphaSegment:
239     mu_lo: float
240     mu_hi: float
241     nf: int
242     dense: Callable[[float], np.ndarray]
243
244
245 def solve_alpha_segment(mu_start: float, mu_end: float, alpha_start: float, nf: int) ->
246     Tuple[AlphaSegment, float]:
247     a0 = alpha_start / (4.0 * PI)
248     t_start = math.log(mu_start**2)
249     t_end = math.log(mu_end**2)
250
251     sol = solve_ivp(
252         fun=lambda t, y: [beta_a(y[0], nf)],
253         t_span=(t_start, t_end),
254         y0=[a0],
255         method="DOP853",
256         rtol=1e-10,
257         atol=1e-12,
258         dense_output=True,
259         max_step=0.05,
260     )
261     if not sol.success:
262         raise RuntimeError(f"alpha_s segment solve failed between {mu_start} and {mu_end} GeV")
263
264     alpha_end = float(sol.y[0, -1]) * 4.0 * PI
265     seg = AlphaSegment(mu_lo=min(mu_start, mu_end), mu_hi=max(mu_start, mu_end), nf=nf,
266         dense=sol.sol)
267     return seg, alpha_end

```

```

266
267
268 def build_alpha_segments(mu_min: float, mu_max: float, mu_anchor: float, alpha_anchor: float
    ) -> List[AlphaSegment]:
269     segments: List[AlphaSegment] = []
270
271     # Downward from MZ.
272     alpha_curr = alpha_anchor
273     mu_curr = mu_anchor
274     for mu_th, nf_from, nf_to in [(MU_B, 5, 4), (MU_C, 4, 3)]:
275         if mu_min < mu_th < mu_curr:
276             seg, alpha_at_th = solve_alpha_segment(mu_curr, mu_th, alpha_curr, nf_from)
277             segments.append(seg)
278             alpha_curr = continuous_match_alpha(alpha_at_th, nf_from=nf_from, nf_to=nf_to,
279                 mu_th=mu_th)
280             mu_curr = mu_th
281         if mu_min < mu_curr:
282             seg, _ = solve_alpha_segment(mu_curr, mu_min, alpha_curr, active_nf(math.sqrt(
283                 mu_curr * mu_min)))
284             segments.append(seg)
285
286     # Upward from MZ.
287     alpha_curr = alpha_anchor
288     mu_curr = mu_anchor
289     for mu_th, nf_from, nf_to in [(MU_T, 5, 6)]:
290         if mu_curr < mu_th < mu_max:
291             seg, alpha_at_th = solve_alpha_segment(mu_curr, mu_th, alpha_curr, nf_from)
292             segments.append(seg)
293             alpha_curr = continuous_match_alpha(alpha_at_th, nf_from=nf_from, nf_to=nf_to,
294                 mu_th=mu_th)
295             mu_curr = mu_th
296         if mu_curr < mu_max:
297             seg, _ = solve_alpha_segment(mu_curr, mu_max, alpha_curr, active_nf(math.sqrt(
298                 mu_curr * mu_max)))
299             segments.append(seg)
300
301     return segments
302
303 def alpha_of_mu(mu: float, segments: List[AlphaSegment]) -> float:
304     for seg in segments:
305         if seg.mu_lo - 1e-14 <= mu <= seg.mu_hi + 1e-14:
306             t = math.log(mu**2)
307             a = float(seg.dense(t)[0])
308             return 4.0 * PI * a
309         raise ValueError(f"mu={mu} GeV is outside the solved range")
310
311 # -----
312 # Dense piecewise solutions for m_q(mu)
313 # -----
314 @dataclass
315 class MassSegment:
316     mu_lo: float
317     mu_hi: float
318     nf: int
319     dense: Callable[[float], np.ndarray]
320
321 def solve_mass_segment(
322     mu_start: float,
323     mu_end: float,
324     m_start: float,
325     nf: int,
326     alpha_fun: Callable[[float], float],
327 ) -> Tuple[MassSegment, float]:
328     t_start = math.log(mu_start**2)
329     t_end = math.log(mu_end**2)

```

```

329
330 def rhs(t: float, y: np.ndarray) -> List[float]:
331     mu = math.exp(0.5 * t)
332     a = alpha_fun(mu) / (4.0 * PI)
333     return [-gamma_m(a, nf) * y[0]]
334
335 sol = solve_ivp(
336     fun=rhs,
337     t_span=(t_start, t_end),
338     y0=[m_start],
339     method="DOP853",
340     rtol=1e-10,
341     atol=1e-12,
342     dense_output=True,
343     max_step=0.05,
344 )
345 if not sol.success:
346     raise RuntimeError(f"mass_segment_solve_failed_between_{mu_start}_and_{mu_end}_GeV")
347
348 m_end = float(sol.y[0, -1])
349 seg = MassSegment(mu_lo=min(mu_start, mu_end), mu_hi=max(mu_start, mu_end), nf=nf, dense
350                  =sol.sol)
351 return seg, m_end
352
353 def thresholds_between(mu_a: float, mu_b: float) -> List[float]:
354     lo, hi = sorted((mu_a, mu_b))
355     th = [mu for mu in (MU_C, MU_B, MU_T) if lo < mu < hi]
356     return th if mu_a < mu_b else th[::-1]
357
358
359 def build_mass_segments(
360     mu_ref: float,
361     m_ref: float,
362     mu_min: float,
363     mu_max: float,
364     alpha_fun: Callable[[float], float],
365     heavy_name: str | None = None,
366 ) -> List[MassSegment]:
367     segments: List[MassSegment] = []
368
369     # Downward from reference scale.
370     m_curr = m_ref
371     mu_curr = mu_ref
372     for mu_th in thresholds_between(mu_ref, mu_min):
373         nf_from = active_nf(math.sqrt(mu_curr * (mu_th * (1.0 + 1e-12))))
374         nf_to = active_nf(mu_th * (1.0 - 1e-12))
375         seg, m_at_th = solve_mass_segment(mu_curr, mu_th, m_curr, nf_from, alpha_fun)
376         segments.append(seg)
377         m_curr = continuous_match_mass(m_at_th, nf_from=nf_from, nf_to=nf_to, mu_th=mu_th,
378                                     heavy_name=heavy_name)
379         mu_curr = mu_th
380     if mu_min < mu_curr:
381         nf = active_nf(math.sqrt(mu_curr * mu_min))
382         seg, _ = solve_mass_segment(mu_curr, mu_min, m_curr, nf, alpha_fun)
383         segments.append(seg)
384
385     # Upward from reference scale.
386     m_curr = m_ref
387     mu_curr = mu_ref
388     for mu_th in thresholds_between(mu_ref, mu_max):
389         nf_from = active_nf(math.sqrt(mu_curr * (mu_th / (1.0 + 1e-12))))
390         nf_to = active_nf(mu_th * (1.0 + 1e-12))
391         seg, m_at_th = solve_mass_segment(mu_curr, mu_th, m_curr, nf_from, alpha_fun)
392         segments.append(seg)
393         m_curr = continuous_match_mass(m_at_th, nf_from=nf_from, nf_to=nf_to, mu_th=mu_th,
394                                     heavy_name=heavy_name)
395         mu_curr = mu_th

```

```

394     if mu_curr < mu_max:
395         nf = active_nf(math.sqrt(mu_curr * mu_max))
396         seg, _ = solve_mass_segment(mu_curr, mu_max, m_curr, nf, alpha_fun)
397         segments.append(seg)
398
399     return segments
400
401
402 def mass_of_mu(mu: float, segments: List[MassSegment]) -> float:
403     for seg in segments:
404         if seg.mu_lo - 1e-14 <= mu <= seg.mu_hi + 1e-14:
405             t = math.log(mu**2)
406             return float(seg.dense(t)[0])
407     raise ValueError(f"mu={mu} GeV is outside the mass solution range")
408
409
410 # -----
411 # Koide-type quantities and the A_d, T_d, phi_d parametrization
412 # -----
413 def koide_k_from_x(x1: np.ndarray, x2: np.ndarray, x3: np.ndarray) -> np.ndarray:
414     denom = (x1 + x2 + x3) ** 2
415     return (x1**2 + x2**2 + x3**2) / denom
416
417
418 def koide_k(m1: np.ndarray, m2: np.ndarray, m3: np.ndarray) -> np.ndarray:
419     s1 = np.sqrt(m1)
420     s2 = np.sqrt(m2)
421     s3 = np.sqrt(m3)
422     return koide_k_from_x(s1, s2, s3)
423
424
425 def koide_k_inv(m1: np.ndarray, m2: np.ndarray, m3: np.ndarray) -> np.ndarray:
426     invs1 = 1.0 / np.sqrt(m1)
427     invs2 = 1.0 / np.sqrt(m2)
428     invs3 = 1.0 / np.sqrt(m3)
429     return koide_k_from_x(invs1, invs2, invs3)
430
431
432 def koide_ad_t_phi_from_sqrt(x1: np.ndarray, x2: np.ndarray, x3: np.ndarray) -> Tuple[np.
433     ndarray, np.ndarray, np.ndarray]:
434     """
435     Compute A_d(mu), T_d(mu), phi_d(mu) from
436     x_n = sqrt(m_dn) = A_d * [1 + 2 T_d cos(phi_d/3 + 2 pi n / 3)],
437     with n = 1, 2, 3.
438     Writing theta = phi_d / 3 and
439     u_n = (x_n / A_d - 1) / (2 T_d),
440     the three equations become
441     u_1 = cos(theta + 2 pi / 3),
442     u_2 = cos(theta + 4 pi / 3),
443     u_3 = cos(theta).
444     Hence
445     cos(theta) = u_3,
446     sin(theta) = (u_2 - u_1) / sqrt(3),
447     and a stable inversion is
448     theta = atan2((u_2 - u_1) / sqrt(3), u_3),
449     phi_d = 3 * theta.
450     """
451     a_d = (x1 + x2 + x3) / 3.0

```

```

461     k = koide_k_from_x(x1, x2, x3)
462     t_d = np.sqrt((3.0 * k - 1.0) / 2.0)
463
464     u1 = (x1 / a_d - 1.0) / (2.0 * t_d)
465     u2 = (x2 / a_d - 1.0) / (2.0 * t_d)
466     u3 = (x3 / a_d - 1.0) / (2.0 * t_d)
467
468     cos_theta = u3
469     sin_theta = (u2 - u1) / np.sqrt(3.0)
470
471     norm = np.hypot(cos_theta, sin_theta)
472     cos_theta = cos_theta / norm
473     sin_theta = sin_theta / norm
474
475     theta = np.arctan2(sin_theta, cos_theta)
476     phi_d = 3.0 * theta
477     return a_d, t_d, phi_d
478
479
480 def masses_from_formula(a_d: np.ndarray, t_d: np.ndarray, phi_d: np.ndarray) -> Tuple[np.
    ndarray, np.ndarray, np.ndarray]:
481     """
482     Reconstruct  $m_{d1}$ ,  $m_{d2}$ ,  $m_{d3}$  from
483
484      $\sqrt{m_{dn}} = A_d * [1 + 2 T_d \cos(\phi_d/3 + 2 \pi n / 3)]$ 
485
486     using  $n = 1, 2, 3$  exactly as requested.
487     """
488     theta = phi_d / 3.0
489     x1 = a_d * (1.0 + 2.0 * t_d * np.cos(theta + 2.0 * PI / 3.0))
490     x2 = a_d * (1.0 + 2.0 * t_d * np.cos(theta + 4.0 * PI / 3.0))
491     x3 = a_d * (1.0 + 2.0 * t_d * np.cos(theta + 6.0 * PI / 3.0))
492     return x1**2, x2**2, x3**2
493
494
495 # -----
496 # Internal data export
497 # -----
498 def save_internal_data(
499     mu_grid: np.ndarray,
500     alpha_values: np.ndarray,
501     m_d1: np.ndarray,
502     m_d2: np.ndarray,
503     m_d3: np.ndarray,
504     sqrt_d1: np.ndarray,
505     sqrt_d2: np.ndarray,
506     sqrt_d3: np.ndarray,
507     invsqrt_d1: np.ndarray,
508     invsqrt_d2: np.ndarray,
509     invsqrt_d3: np.ndarray,
510     k_values: np.ndarray,
511     k_inv_values: np.ndarray,
512     a_d_values: np.ndarray,
513     t_d_values: np.ndarray,
514     phi_d_values: np.ndarray,
515     m_d1_formula: np.ndarray,
516     m_d2_formula: np.ndarray,
517     m_d3_formula: np.ndarray,
518     delta_m_d1: np.ndarray,
519     delta_m_d2: np.ndarray,
520     delta_m_d3: np.ndarray,
521     rel_pct_d1: np.ndarray,
522     rel_pct_d2: np.ndarray,
523     rel_pct_d3: np.ndarray,
524     abs_rel_pct_d1: np.ndarray,
525     abs_rel_pct_d2: np.ndarray,
526     abs_rel_pct_d3: np.ndarray,
527 ) -> None:

```

```

528     with open(OUTPUT_CSV, "w", newline="", encoding="utf-8") as f:
529         writer = csv.writer(f)
530         writer.writerow([
531             "mu_GeV",
532             "alpha_s",
533             "m_d1_MeV",
534             "m_d2_MeV",
535             "m_d3_MeV",
536             "sqrt_m_d1_MeV_half",
537             "sqrt_m_d2_MeV_half",
538             "sqrt_m_d3_MeV_half",
539             "inv_sqrt_m_d1_MeV_minus_half",
540             "inv_sqrt_m_d2_MeV_minus_half",
541             "inv_sqrt_m_d3_MeV_minus_half",
542             "K",
543             "K_inv",
544             "A_d_MeV_half",
545             "T_d",
546             "phi_d",
547             "m_d1_formula_MeV",
548             "m_d2_formula_MeV",
549             "m_d3_formula_MeV",
550             "delta_m_d1_MeV",
551             "delta_m_d2_MeV",
552             "delta_m_d3_MeV",
553             "delta_m_d1_percent",
554             "delta_m_d2_percent",
555             "delta_m_d3_percent",
556             "abs_delta_m_d1_percent",
557             "abs_delta_m_d2_percent",
558             "abs_delta_m_d3_percent",
559         ])
560         for values in zip(
561             mu_grid,
562             alpha_values,
563             m_d1,
564             m_d2,
565             m_d3,
566             sqrt_d1,
567             sqrt_d2,
568             sqrt_d3,
569             invsqrt_d1,
570             invsqrt_d2,
571             invsqrt_d3,
572             k_values,
573             k_inv_values,
574             a_d_values,
575             t_d_values,
576             phi_d_values,
577             m_d1_formula,
578             m_d2_formula,
579             m_d3_formula,
580             delta_m_d1,
581             delta_m_d2,
582             delta_m_d3,
583             rel_pct_d1,
584             rel_pct_d2,
585             rel_pct_d3,
586             abs_rel_pct_d1,
587             abs_rel_pct_d2,
588             abs_rel_pct_d3,
589         ):
590             writer.writerow([f"{x:.16e}" for x in values])
591
592     np.savez(
593         OUTPUT_NPZ,
594         mu_GeV=mu_grid,
595         alpha_s=alpha_values,

```

```

596         m_d1_MeV=m_d1,
597         m_d2_MeV=m_d2,
598         m_d3_MeV=m_d3,
599         sqrt_m_d1_MeV_half=sqrt_d1,
600         sqrt_m_d2_MeV_half=sqrt_d2,
601         sqrt_m_d3_MeV_half=sqrt_d3,
602         inv_sqrt_m_d1_MeV_minus_half=invsqrt_d1,
603         inv_sqrt_m_d2_MeV_minus_half=invsqrt_d2,
604         inv_sqrt_m_d3_MeV_minus_half=invsqrt_d3,
605         K=k_values,
606         K_inv=k_inv_values,
607         A_d_MeV_half=a_d_values,
608         T_d=t_d_values,
609         phi_d=phi_d_values,
610         m_d1_formula_MeV=m_d1_formula,
611         m_d2_formula_MeV=m_d2_formula,
612         m_d3_formula_MeV=m_d3_formula,
613         delta_m_d1_MeV=delta_m_d1,
614         delta_m_d2_MeV=delta_m_d2,
615         delta_m_d3_MeV=delta_m_d3,
616         delta_m_d1_percent=rel_pct_d1,
617         delta_m_d2_percent=rel_pct_d2,
618         delta_m_d3_percent=rel_pct_d3,
619         abs_delta_m_d1_percent=abs_rel_pct_d1,
620         abs_delta_m_d2_percent=abs_rel_pct_d2,
621         abs_delta_m_d3_percent=abs_rel_pct_d3,
622     )
623
624
625     # -----
626     # mu-grid construction
627     # -----
628     def build_mu_grid() -> np.ndarray:
629         """
630         Build the plotting/export grid and force exact inclusion of important scales.
631
632         In particular, the user requested that the reference scale  $\mu = 4.183 \text{ GeV}$ 
633         be present explicitly in the CSV output. A pure logspace grid does not
634         generally hit that value exactly, so we merge the logspace grid with the
635         exact reference/threshold scales and remove duplicates.
636         """
637         base_grid = np.logspace(math.log10(MU_MIN), math.log10(MU_MAX), N_GRID)
638         forced_points = np.array(sorted({MU_D1_REF, MU_D2_REF, MU_D3_REF, MU_C, MU_B, MU_T}),
639                                   dtype=float)
640         mu_grid = np.unique(np.concatenate([base_grid, forced_points]))
641         mu_grid.sort()
642         return mu_grid
643
644     def finite_positive_floor(*arrays: np.ndarray) -> float:
645         positive = np.concatenate([arr[np.isfinite(arr) & (arr > 0.0)] for arr in arrays])
646         if positive.size == 0:
647             return 1e-30
648         return max(float(np.min(positive)) * 0.5, 1e-30)
649
650
651     def add_threshold_lines(ax: plt.Axes) -> None:
652         ax.axvline(MU_C, ls="--", lw=1, color="gray", alpha=0.6)
653         ax.axvline(MU_B, ls="--", lw=1, color="gray", alpha=0.6)
654         ax.axvline(MU_T, ls="--", lw=1, color="gray", alpha=0.6)
655
656
657     # -----
658     # Main execution
659     # -----
660     def main() -> None:
661         mu_grid = build_mu_grid()
662

```



```

663     alpha_segments = build_alpha_segments(
664         mu_min=MU_MIN,
665         mu_max=MU_MAX,
666         mu_anchor=MZ_GEV,
667         alpha_anchor=ALPHA_S_MZ,
668     )
669     alpha_fun = lambda mu: alpha_of_mu(mu, alpha_segments)
670
671     d1_segments = build_mass_segments(MU_D1_REF, M_D1_REF, MU_MIN, MU_MAX, alpha_fun,
672         heavy_name=None)
673     d2_segments = build_mass_segments(MU_D2_REF, M_D2_REF, MU_MIN, MU_MAX, alpha_fun,
674         heavy_name=None)
675     d3_segments = build_mass_segments(MU_D3_REF, M_D3_REF, MU_MIN, MU_MAX, alpha_fun,
676         heavy_name="b")
677
678     alpha_values = np.array([alpha_fun(mu) for mu in mu_grid])
679     m_d1 = np.array([mass_of_mu(mu, d1_segments) for mu in mu_grid])
680     m_d2 = np.array([mass_of_mu(mu, d2_segments) for mu in mu_grid])
681     m_d3 = np.array([mass_of_mu(mu, d3_segments) for mu in mu_grid])
682
683     if not INCLUDE_HEAVY_BELOW_OWN_THRESHOLD:
684         mask = mu_grid < MU_B
685         m_d3 = np.where(mask, np.nan, m_d3)
686
687     sqrt_d1 = np.sqrt(m_d1)
688     sqrt_d2 = np.sqrt(m_d2)
689     sqrt_d3 = np.sqrt(m_d3)
690     invsqrt_d1 = 1.0 / sqrt_d1
691     invsqrt_d2 = 1.0 / sqrt_d2
692     invsqrt_d3 = 1.0 / sqrt_d3
693
694     k_values = koide_k(m_d1, m_d2, m_d3)
695     k_inv_values = koide_k_inv(m_d1, m_d2, m_d3)
696     a_d_values, t_d_values, phi_d_values = koide_ad_t_phi_from_sqrt(sqrt_d1, sqrt_d2,
697         sqrt_d3)
698
699     m_d1_formula, m_d2_formula, m_d3_formula = masses_from_formula(a_d_values, t_d_values,
700         phi_d_values)
701
702     delta_m_d1 = m_d1_formula - m_d1
703     delta_m_d2 = m_d2_formula - m_d2
704     delta_m_d3 = m_d3_formula - m_d3
705
706     rel_pct_d1 = 100.0 * delta_m_d1 / m_d1
707     rel_pct_d2 = 100.0 * delta_m_d2 / m_d2
708     rel_pct_d3 = 100.0 * delta_m_d3 / m_d3
709
710     abs_rel_pct_d1 = np.abs(rel_pct_d1)
711     abs_rel_pct_d2 = np.abs(rel_pct_d2)
712     abs_rel_pct_d3 = np.abs(rel_pct_d3)
713
714     if SAVE_INTERNAL_DATA:
715         save_internal_data(
716             mu_grid,
717             alpha_values,
718             m_d1,
719             m_d2,
720             m_d3,
721             sqrt_d1,
722             sqrt_d2,
723             sqrt_d3,
724             invsqrt_d1,
725             invsqrt_d2,
726             invsqrt_d3,
727             k_values,
728             k_inv_values,
729             a_d_values,
730             t_d_values,

```

```

726         phi_d_values,
727         m_d1_formula,
728         m_d2_formula,
729         m_d3_formula,
730         delta_m_d1,
731         delta_m_d2,
732         delta_m_d3,
733         rel_pct_d1,
734         rel_pct_d2,
735         rel_pct_d3,
736         abs_rel_pct_d1,
737         abs_rel_pct_d2,
738         abs_rel_pct_d3,
739     )
740     print(f"Saved internal data to: {OUTPUT_CSV}")
741     print(f"Saved internal data to: {OUTPUT_NPZ}")
742
743     fig, (ax1, ax2) = plt.subplots(
744         2,
745         1,
746         figsize=(9, 8),
747         sharex=True,
748         gridspec_kw={"height_ratios": [3.0, 1.8], "hspace": 0.08},
749     )
750
751     # Color-blind-safe palette (Okabe-Ito family) chosen to avoid the
752     # red/green confusion common in deuteranopia/protanopia.
753     # We also separate the series by line style so the curves remain
754     # distinguishable in grayscale printouts.
755     color_d1 = "#0072B2" # blue
756     color_d2 = "#D55E00" # vermillion
757     color_d3 = "#CC79A7" # reddish purple
758     ls_d1 = "-"
759     ls_d2 = "--"
760     ls_d3 = "-."
761
762     # Top panel: formula-reconstructed masses (evaluated with the fit parameters).
763     ax1.plot(mu_grid, m_d1_formula, lw=2.2, color=color_d1, ls=ls_d1, label=r"$m_{d1}^{\mathrm{fit}}(\mu)$")
764     ax1.plot(mu_grid, m_d2_formula, lw=2.2, color=color_d2, ls=ls_d2, label=r"$m_{d2}^{\mathrm{fit}}(\mu)$")
765     ax1.plot(mu_grid, m_d3_formula, lw=2.2, color=color_d3, ls=ls_d3, label=r"$m_{d3}^{\mathrm{fit}}(\mu)$")
766     add_threshold_lines(ax1)
767     ax1.set_xscale("log")
768     ax1.set_yscale("log")
769     ax1.set_xlim(MU_MIN, MU_MAX)
770     ax1.set_ylabel(r"Fit mass $m_{dn}^{\mathrm{fit}}(\mu)$ [MeV]")
771     ax1.set_title(r"Down-type masses reconstructed from the fit in $\overline{\mathrm{MS}}$ QCD (4-loop, continuous matching)")
772     ax1.grid(True, which="both", ls=":", alpha=0.5)
773     ax1.legend(loc="best")
774
775     # Bottom panel: recommended comparison metric = absolute relative difference [%].
776     # Delta $m^{\mathrm{fit}}_{dn}(\mu) = m^{\mathrm{fit}}_{dn}(\mu) - m_{dn}(\mu)$, where $m_{dn}(\mu)$ is the
777     # PDG-2024 running mass obtained by 4-loop $\overline{\mathrm{MS}}$ evolution from the quoted inputs.
778     plot_abs_rel_d1 = np.maximum(abs_rel_pct_d1, finite_positive_floor(abs_rel_pct_d1,
779         abs_rel_pct_d2, abs_rel_pct_d3))
779     plot_abs_rel_d2 = np.maximum(abs_rel_pct_d2, finite_positive_floor(abs_rel_pct_d1,
780         abs_rel_pct_d2, abs_rel_pct_d3))
780     plot_abs_rel_d3 = np.maximum(abs_rel_pct_d3, finite_positive_floor(abs_rel_pct_d1,
781         abs_rel_pct_d2, abs_rel_pct_d3))
782
783     ax2.plot(mu_grid, plot_abs_rel_d1, lw=2.2, color=color_d1, ls=ls_d1, label=r"$|\Delta m_{d1}^{\mathrm{fit}}|/m_{d1}$")
784     ax2.plot(mu_grid, plot_abs_rel_d2, lw=2.2, color=color_d2, ls=ls_d2, label=r"$|\Delta m_{d2}^{\mathrm{fit}}|/m_{d2}$")
785     ax2.plot(mu_grid, plot_abs_rel_d3, lw=2.2, color=color_d3, ls=ls_d3, label=r"$|\Delta m_{d3}^{\mathrm{fit}}|/m_{d3}$")

```

```

785         {d3}^{\mathrm{fit}}/m_{d3}$")
786     add_threshold_lines(ax2)
787     ax2.set_xscale("log")
788     ax2.set_yscale("log")
789     ax2.set_xlim(MU_MIN, MU_MAX)
790     ax2.set_xlabel(r"Energy\scale_\mu$[GeV]")
791     ax2.set_ylabel(r"$\Delta_m^{\mathrm{fit}}/m$[%]")
792     ax2.grid(True, which="both", ls=":", alpha=0.5)
793     ax2.legend(loc="best")
794
795     # Threshold labels on the upper panel only.
796     y_top = np.nanmax(m_d3_formula[np.isfinite(m_d3_formula)])
797     ax1.text(MU_C * 1.02, y_top / 1.6, r"$\mu_c$", va="top")
798     ax1.text(MU_B * 1.02, y_top / 1.6, r"$\mu_b$", va="top")
799     ax1.text(MU_T * 1.02, y_top / 1.6, r"$\mu_t$", va="top")
800
801     fig.tight_layout()
802
803     if SAVEFIG:
804         fig.savefig(OUTPUT_FIG, dpi=180, bbox_inches="tight")
805         print(f"Saved figure to: {OUTPUT_FIG}")
806
807     # Diagnostics.
808     print("\nDiagnostic\alpha_s values:")
809     for mu in [1.0, MU_C, 2.0, MU_B, MZ_GEV, MU_T, 1000.0]:
810         print(f"\alpha_s({mu:8.3f}GeV)={alpha_fun(mu):.6f}")
811
812     print("\nReference-point check:")
813     print(f"\m_d1({MU_D1_REF:.3f}GeV)={mass_of_mu(MU_D1_REF, d1_segments):.6f}MeV")
814     print(f"\m_d2({MU_D2_REF:.3f}GeV)={mass_of_mu(MU_D2_REF, d2_segments):.6f}MeV")
815     print(f"\m_d3({MU_D3_REF:.3f}GeV)={mass_of_mu(MU_D3_REF, d3_segments):.6f}MeV")
816
817     print("\nSample A_d(mu), T_d(mu), phi_d(mu) values:")
818     for mu in [1.0, 2.0, MU_B, 10.0, MZ_GEV, MU_T, 1000.0]:
819         i = int(np.argmin(np.abs(mu_grid - mu)))
820         print(
821             f"A_d({mu_grid[i]:8.3f}GeV)={a_d_values[i]:.10f}, "
822             f"T_d({mu_grid[i]:8.3f}GeV)={t_d_values[i]:.10f}, "
823             f"phi_d({mu_grid[i]:8.3f}GeV)={phi_d_values[i]:.10f}"
824         )
825
826     print("\nMaximum absolute reconstruction differences [%]:")
827     print(f"d1: {np.nanmax(abs_rel_pct_d1):.16e}")
828     print(f"d2: {np.nanmax(abs_rel_pct_d2):.16e}")
829     print(f"d3: {np.nanmax(abs_rel_pct_d3):.16e}")
830
831     plt.show()
832
833 if __name__ == "__main__":
834     main()

```

C Python Script 3: benchmark Table 1 verification and Figure 3

The following script verifies the benchmark comparison of Table 1 and produces Figure 3. At the common scale $\mu_{d3} = m_b(m_b) = 4.183 \text{ GeV}$ it computes (A) the Brannen-type prediction of Eq. (26), (B) the same prediction after 4-loop $\overline{\text{MS}}$ transport from its associated PDG input scale to μ_{d3} , and (C) the PDG 2024 central inputs evolved to μ_{d3} . The running machinery and constants are imported directly from Script 1.

```

1  #!/usr/bin/env python3
2  """
3  =====
4  Table 1 (Benchmark) verification + comparison figure
5  (2-panel layout; Okabe-Ito color-blind-safe palette)
6

```

```

7 At the common scale  $\mu_d = m_b(m_b) = 4.183 \text{ GeV}$ , compare:
8
9  $A$  Prediction [Eq. (26) at  $\mu_d = m_b(m_b)$ ]
10  $\sqrt{m_{dn}} = A_d(\mu_d) * (1 + \beta * \cos(1/9 + 2\pi/3))$ 
11 Values numerically coincide with the Brannen form and are
12 naturally associated with the PDG input scales
13  $(2 \text{ GeV for } d, s; m_b \text{ for } b)$ .
14
15  $B$  With RG dressing
16 Take Column  $A$  and transport each value from its PDG input scale
17 to the common scale  $\mu_d = 4.183 \text{ GeV}$  via 4-loop  $\overline{\text{MS}}$ -bar running.
18
19  $C$  Fit value from PDG 2024 input
20 Take PDG 2024 inputs and transport them to  $\mu_d = 4.183 \text{ GeV}$ ,
21 again via 4-loop  $\overline{\text{MS}}$ -bar running.
22
23 Relative deviation  $= (B - C) / C$  [percent, shown in left panel].
24 Ratio (right panel)  $= (A, B, C) / C$  [dimensionless, normalized].
25
26 Outputs:
27  $A$  5-column table (stdout) mirroring the revised Table 1 of the paper.
28  $\text{Figure3.png}$ : 2-panel comparison figure.
29 =====
30 " "
31
32 from __future__ import annotations
33
34 import sys
35 from pathlib import Path
36 sys.path.insert(0, str(Path(__file__).resolve().parent))
37
38 import numpy as np
39 import matplotlib
40 matplotlib.use("Agg")
41 import matplotlib.pyplot as plt
42
43 from Script1_en import (
44     compute_A,
45     predicted_mass,
46     running_mass,
47     mass_formula_k,
48     MD_INPUT,
49     MS_INPUT,
50     MU_D_INPUT,
51     MB_INPUT,
52     MB_MB,
53 )
54
55 # --- Okabe-Ito color-blind-safe palette -----
56 COLOR_PRED = "#999999" # neutral gray - Prediction (A)
57 COLOR_RG = "#0072B2" # blue - with RG dressing (B)
58 COLOR_PDG = "#D55E00" # vermillion - PDG 2024 evolved (C)
59
60
61 def prediction_mev(mu_d3: float, n: int) -> float:
62     """Brannen-phase prediction at  $\mu_d$  (Column A)."""
63     A = compute_A(mu_d3)
64     return float(predicted_mass(A, n))
65
66
67 def rg_dressed_mev(m_pred: float, mu_from: float, mu_to: float) -> float:
68     """Transport mass value  $m_{\text{pred}}$  from  $\mu_{\text{from}}$  to  $\mu_{\text{to}}$  via 4-loop  $\overline{\text{MS}}$ -bar (Column B)."""
69     return float(running_mass(m_pred, mu_from, mu_to))
70
71
72 def pdg_evolved_mev(mu_target: float):
73     """PDG 2024 inputs evolved to  $\mu_{\text{target}}$  (Column C)."""
74     m_d = float(running_mass(MD_INPUT, MU_D_INPUT, mu_target))

```

```

75     m_s = float(running_mass(MS_INPUT, MU_D_INPUT, mu_target))
76     m_b = float(running_mass(MB_INPUT, MB_MB, mu_target))
77     return m_d, m_s, m_b
78
79
80 def _panel_absolute(ax, labels, pred, rg, pdg, devs):
81     """Left panel: grouped-bar comparison, log y."""
82     x = np.arange(len(labels), dtype=float)
83     width = 0.26
84
85     b1 = ax.bar(x - width, pred, width,
86                 color=COLOR_PRED, edgecolor="black", linewidth=0.6,
87                 label=r"Prediction (Eq.~26), associated to PDG input scales")
88     b2 = ax.bar(x, rg, width,
89                 color=COLOR_RG, edgecolor="black", linewidth=0.6,
90                 hatch="//",
91                 label=r"with RG dressing, transported to  $\mu_{d3}$ ")
92     b3 = ax.bar(x + width, pdg, width,
93                 color=COLOR_PDG, edgecolor="black", linewidth=0.6,
94                 hatch="xx",
95                 label=r"PDG 2024 input evolved to  $\mu_{d3}$ ")
96
97     ax.set_yscale("log")
98     ax.set_xticks(x)
99     ax.set_xticklabels(labels, fontsize=13)
100    ax.set_ylabel(r"mass [MeV]", fontsize=12)
101    ax.set_title(
102        r"(a) Absolute values at  $\mu_{d3}=m_b(m_b)=4.183\text{ GeV}$ ",
103        fontsize=12)
104    ax.legend(fontsize=8.8, loc="upper left")
105    ax.grid(True, which="both", axis="y", alpha=0.3)
106    ax.tick_params(labelsize=11)
107
108    # value labels on each bar
109    for bars, values in [(b1, pred), (b2, rg), (b3, pdg)]:
110        for rect, v in zip(bars, values):
111            ax.text(rect.get_x() + rect.get_width() / 2.0,
112                    v * 1.08, f"{v:.3f}",
113                    ha="center", va="bottom", fontsize=8.2)
114
115    # Fix y-axis so labels fit inside the plot on log scale
116    ymin_data = min(min(pred), min(rg), min(pdg))
117    ymax_data = max(max(pred), max(rg), max(pdg))
118    ax.set_ylim(ymin_data * 0.2, ymax_data * 3.0)
119
120    # deviation labels placed just below each RG bar (log-scale safe)
121    for xi, v_rg, dev in zip(x, rg, devs):
122        color_box = COLOR_RG if abs(dev) > 1e-6 else "#555555"
123        ax.annotate(rf"$\Delta={dev:+.3f}\%",
124                    xy=(xi, v_rg * 0.55),
125                    xytext=(xi, v_rg * 0.55),
126                    ha="center", va="center",
127                    fontsize=9.5, color=color_box,
128                    bbox=dict(boxstyle="round,pad=0.3",
129                            facecolor="white", alpha=0.95,
130                            edgecolor=color_box, linewidth=0.7))
131
132
133 def _panel_ratio(ax, labels, pred, rg, pdg):
134     """Right panel: ratio (A/B) on linear scale with C as reference=1."""
135     x = np.arange(len(labels), dtype=float)
136     width = 0.32
137
138     r_pred = np.array(pred) / np.array(pdg)
139     r_rg = np.array(rg) / np.array(pdg)
140
141     b1 = ax.bar(x - width / 2.0, r_pred, width,
142                 color=COLOR_PRED, edgecolor="black", linewidth=0.6,

```

```

143         label=r"Prediction_/_PDG_evolved")
144     b2 = ax.bar(x + width / 2.0, r_rg, width,
145                color=COLOR_RG, edgecolor="black", linewidth=0.6,
146                hatch="//",
147                label=r"with_RG_dressing_/_PDG_evolved")
148
149     # Reference line at 1.0 (PDG evolved)
150     ax.axhline(1.0, color=COLOR_PDG, linestyle="--", linewidth=1.3,
151                label=r"PDG_2024_evolved_(reference_=$1$)")
152
153     ax.set_xticks(x)
154     ax.set_xticklabels(labels, fontsize=13)
155     ax.set_ylabel(r"ratio_to_PDG_2024_evolved", fontsize=12)
156     ax.set_title(r"(b) Ratio_normalized_to_PDG_evolved", fontsize=12)
157     ax.legend(fontsize=8.8, loc="upper_left")
158     ax.grid(True, which="major", axis="y", alpha=0.3)
159     ax.tick_params(labelsize=11)
160
161     # Value labels on each bar (with four decimals for visibility of small diffs)
162     for bars, values in [(b1, r_pred), (b2, r_rg)]:
163         for rect, v in zip(bars, values):
164             ax.text(rect.get_x() + rect.get_width() / 2.0,
165                    v + 0.008, f"{v:.4f}",
166                    ha="center", va="bottom", fontsize=8.6)
167
168     # Choose y-limits that always include 1.0 and the value range
169     all_vals = np.concatenate([r_pred, r_rg, np.array([1.0])])
170     lo = min(all_vals.min(), 0.97)
171     hi = max(all_vals.max(), 1.03)
172     span = hi - lo
173     ax.set_ylim(lo - 0.05 * span, hi + 0.12 * span)
174
175
176 def make_figure(mu_d3, pred, rg, pdg, devs, output_path):
177     labels = [r"$m_{d1}$", r"$m_{d2}$", r"$m_{d3}$"]
178
179     fig, axes = plt.subplots(1, 2, figsize=(13.6, 6.0),
180                             gridspec_kw={"width_ratios": [1.15, 1.0]})
181     _panel_absolute(axes[0], labels, pred, rg, pdg, devs)
182     _panel_ratio(axes[1], labels, pred, rg, pdg)
183
184     fig.suptitle(
185         r"Table~1_benchmark:_Prediction_(Eq.~26)_vs_RG-dressed_vs_PDG_2024"
186         r"_evolved,_common_scale_$_{m_{d3}=m_b(m_b)}$",
187         fontsize=13, y=1.01)
188
189     fig.tight_layout()
190     fig.savefig(output_path, dpi=150, bbox_inches="tight")
191     plt.close(fig)
192
193
194 def main():
195     mu_d3 = MB_MB # 4.183 GeV
196
197     # (A) Prediction at mu_d3 (from Eq. 26 with A_d(mu_d3))
198     m_d1_pred = prediction_mev(mu_d3, 1)
199     m_d2_pred = prediction_mev(mu_d3, 2)
200     m_d3_pred = prediction_mev(mu_d3, 3)
201
202     # (B) with RG dressing: transport from PDG input scale to mu_d3
203     m_d1_rg = rg_dressed_mev(m_d1_pred, MU_D_INPUT, mu_d3) # 2 GeV -> 4.183 GeV
204     m_d2_rg = rg_dressed_mev(m_d2_pred, MU_D_INPUT, mu_d3) # 2 GeV -> 4.183 GeV
205     m_d3_rg = rg_dressed_mev(m_d3_pred, MB_MB, mu_d3) # m_b -> 4.183 GeV (identity)
206
207     # (C) PDG 2024 evolved
208     md_pdg, ms_pdg, mb_pdg = pdg_evolved_mev(mu_d3)
209
210     # Relative deviation = (B - C) / C

```

```

211 dev_d1 = (m_d1_rg - md_pdg) / md_pdg * 100.0
212 dev_d2 = (m_d2_rg - ms_pdg) / ms_pdg * 100.0
213 dev_d3 = (m_d3_rg - mb_pdg) / mb_pdg * 100.0
214
215 sep = "=" * 100
216 hbar = "-" * 100
217 print(sep)
218 print(f"Table 1 (Benchmark): Down-type quark masses at  $\mu_{d3} = \mu_{d3:.3f}$  GeV =  $m_b(m_b)$ 
    ")
219 print(sep)
220 print(f"{'Quantity':<22}{'Prediction [MeV]':>18}"
221       f"{'with RG dressing [MeV]':>24}"
222       f"{'Fit from PDG 2024 [MeV]':>26}"
223       f"{'Rel. dev.':>12}")
224 print(hbar)
225 rows = [
226     ("m_d1(4.183 GeV)", m_d1_pred, m_d1_rg, md_pdg, dev_d1),
227     ("m_d2(4.183 GeV)", m_d2_pred, m_d2_rg, ms_pdg, dev_d2),
228     ("m_d3(4.183 GeV)", m_d3_pred, m_d3_rg, mb_pdg, dev_d3),
229 ]
230 for name, a, b, c, d in rows:
231     print(f"{name:<22}{a:18.3f}{b:24.3f}{c:26.3f}{d:+11.3f}%")
232 print(sep)
233 print("Rel. dev. = (RG-dressed - PDG evolved) / PDG evolved, at  $\mu_{d3} = m_b(m_b)$ ")
234 print(sep)
235
236 output_path = "Figure3.png"
237 make_figure(
238     mu_d3,
239     [m_d1_pred, m_d2_pred, m_d3_pred],
240     [m_d1_rg, m_d2_rg, m_d3_rg],
241     [md_pdg, ms_pdg, mb_pdg],
242     [dev_d1, dev_d2, dev_d3],
243     output_path,
244 )
245 print(f"Figure saved to: {output_path}")
246
247
248 if __name__ == "__main__":
249     main()

```

D Python Script 4: inverse-tuple Brannen parametrization and Figure 5

The following script imports the running-mass machinery from Script 1 (Appendix A), evaluates the 4-loop $\overline{\text{MS}}$ running masses $m_d(\mu)$, $m_s(\mu)$, $m_b(\mu)$ on a logarithmic grid in μ , forms the inverse square-root tuple, fits Eq. (31) for $(A_{d\text{inv}}^{\text{fit}}, T_{d\text{inv}}^{\text{fit}}, \phi_{d\text{inv}}^{\text{fit}})$ at each point, and produces Figure 5.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  =====
5  Inverse-tuple Brannen parametrization: scale dependence under 4-loop
6  MS-bar QCD running.
7
8  Generates Figure 5 of the manuscript.
9
10 What is computed
11 -----
12 For each scale  $\mu$  in the range [1, 1000] GeV, we evaluate the 4-loop
13 MS-bar running masses  $m_d(\mu)$ ,  $m_s(\mu)$ ,  $m_b(\mu)$  using the same machinery
14 as Script 1 (Appendix A), then form the inverse square-root tuple
15  $1/\sqrt{m_{\text{dn}}(\mu)}$  and fit it to the Brannen-type form
16
17  $1/\sqrt{m_{\text{dn}}(\mu)} = A_{\text{inv}}(\mu) * (1 + 2 T_{\text{inv}}(\mu) \cos(\phi_{\text{inv}}(\mu)/3 + 2 k \pi/3)$ ,
18  $k = 0, 1, 2, 3$ .
19
20 Compared with the direct mass formula  $\cos(\phi_d/3 + 2 n \pi/3)$ , this inverse
21 form keeps the cosine in the same Brannen shape but reverses the index by
22  $k = 4 - n$ . This single discrete relabelling reflects the inversion of the
23 underlying mass ordering:  $\sqrt{m_{\text{dn}}}$  is monotonically increasing in  $n$ ,
24 whereas  $1/\sqrt{m_{\text{dn}}}$  is monotonically decreasing, so that the Brannen
25 reference (the dominant branch at angle 0) is taken at index 3 in both
26 cases:  $n = 3$  (bottom) for the direct form and  $k = 3$  (i.e.  $n = 1$ , down)
27 for the inverse form. In Brannen's geometric reading, the inverse-tuple
28 parametrization corresponds to the same spherical triangle traversed in
29 the opposite orientation.
30
31 The fit is closed-form (3 unknowns, 3 data points). Under flavour-universal
32 QCD running the parameters  $T_{\text{inv}}$  and  $\phi_{\text{inv}}$  are scale-independent by
33 construction; only the overall amplitude  $A_{\text{inv}}(\mu)$  carries the
34 multiplicative dressing.
35
36 Output
37 -----
38 Figure5.png: top panel =  $1/\sqrt{m_{\text{dn}}(\mu)}$  vs  $\mu$  (log-log);
39 bottom panel =  $T_{\text{inv}}(\mu)$  with the reference
40 line  $1/\sqrt{2} \approx 0.7071$  marked.
41
42 Imports the running-mass machinery from Script1_en.py (Appendix A).
43 =====
44 """
45
46 from __future__ import annotations
47
48 import math
49 import numpy as np
50 import matplotlib
51 matplotlib.use("Agg")
52 import matplotlib.pyplot as plt
53
54 from Script1_en import (
55     running_mass,
56     MD_INPUT,
57     MS_INPUT,
58     MB_INPUT,
59     MU_D_INPUT,

```



```

60     MB_MB,
61 )
62
63 PI = math.pi
64
65
66 def inverse_brannen_fit(m1: float, m2: float, m3: float):
67     """
68     Solve
69
70     1/sqrt(m_n) = A_inv * (1 + 2 T_inv cos(phi_inv/3 + 2 k pi/3)),
71     k = 0, 1, 2, 3,
72
73     for (A_inv, T_inv, phi_inv) given the three masses (m1, m2, m3).
74
75     The inversion is closed-form. With theta = phi_inv/3 and
76     u_n = (1/sqrt(m_n) - A_inv) / 2 we have
77
78     u_1 (down): k=3, u_1 = T_inv cos(theta + 2 pi)
79     u_2 (strange): k=2, u_2 = T_inv cos(theta + 4 pi/3),
80     u_3 (bottom): k=1, u_3 = T_inv cos(theta + 2 pi/3),
81
82     so the closed-form inversion is
83
84     T_inv cos(theta) = u_1,
85     T_inv sin(theta) = (u_2 - u_3) / sqrt(3),
86
87     which is identical (modulo the period-2 pi equivalence
88     cos(theta + 4 pi/3) = cos(theta - 2 pi/3)) to the relations of any
89     Brannen-type three-mass inversion.
90     """
91     inv1 = 1.0 / np.sqrt(m1)
92     inv2 = 1.0 / np.sqrt(m2)
93     inv3 = 1.0 / np.sqrt(m3)
94     A = (inv1 + inv2 + inv3) / 3.0
95
96     u1 = (inv1 / A - 1.0) / 2.0
97     u2 = (inv2 / A - 1.0) / 2.0
98     u3 = (inv3 / A - 1.0) / 2.0
99
100     cos_theta_T = u1
101     sin_theta_T = (u2 - u3) / np.sqrt(3.0)
102     T = np.hypot(cos_theta_T, sin_theta_T)
103
104     cos_theta = cos_theta_T / T
105     sin_theta = sin_theta_T / T
106     theta = np.arctan2(sin_theta, cos_theta)
107     phi = 3.0 * theta
108     return A, T, phi
109
110
111
112 def main():
113     mu_grid = np.logspace(0.0, 3.0, 200) # 1 GeV ... 1000 GeV
114
115     inv_d1 = np.zeros_like(mu_grid)
116     inv_d2 = np.zeros_like(mu_grid)
117     inv_d3 = np.zeros_like(mu_grid)
118     A_inv = np.zeros_like(mu_grid)
119     T_inv = np.zeros_like(mu_grid)
120     phi_inv = np.zeros_like(mu_grid)
121
122     print("Computing 4-loop MS-bar running masses and inverse Brannen fit...")
123     for i, mu in enumerate(mu_grid):
124         md = running_mass(MD_INPUT, MU_D_INPUT, float(mu))
125         ms = running_mass(MS_INPUT, MU_D_INPUT, float(mu))
126         mb = running_mass(MB_INPUT, MB_MB, float(mu))
127         inv_d1[i] = 1.0 / np.sqrt(md)

```

```

128     inv_d2[i] = 1.0 / np.sqrt(ms)
129     inv_d3[i] = 1.0 / np.sqrt(mb)
130     A, T, phi = inverse_brannen_fit(md, ms, mb)
131     A_inv[i] = A
132     T_inv[i] = T
133     phi_inv[i] = phi
134     if (i + 1) % 50 == 0:
135         print(f"_{i+1}/{len(mu_grid)}_points")
136
137 # Diagnostic at a few representative scales
138 for mu_target in [2.0, MB_MB, 91.1876]:
139     i_t = int(np.argmin(np.abs(mu_grid - mu_target)))
140     print(
141         f"_{mu_grid[i_t]:8.3f}_GeV_{A_inv[i_t]:.6e}MeV^{(-1/2)},_"
142         f"T_inv_{T_inv[i_t]:.6f},_"
143         f"phi_inv_{phi_inv[i_t]:.6f}_rad"
144     )
145
146 # ----- plot (3 panels) -----
147 fig, (ax1, ax2, ax3) = plt.subplots(
148     3, 1, figsize=(9, 9.5), sharex=True,
149     gridspec_kw={"height_ratios": [2.4, 1.2, 1.2], "hspace": 0.06},
150 )
151
152 color_d = "#0072B2"
153 color_s = "#D55E00"
154 color_b = "#CC79A7"
155 color_T = "#009E73"
156 color_phi = "#E69F00"
157
158 # Top panel: 1/sqrt(m_dn) vs mu
159 ax1.plot(mu_grid, inv_d1, lw=2.0, color=color_d, ls="-",
160         label=r"$1/\sqrt{m_{d1}}(\mu)$ (down)")
161 ax1.plot(mu_grid, inv_d2, lw=2.0, color=color_s, ls="--",
162         label=r"$1/\sqrt{m_{d2}}(\mu)$ (strange)")
163 ax1.plot(mu_grid, inv_d3, lw=2.0, color=color_b, ls="-. ",
164         label=r"$1/\sqrt{m_{d3}}(\mu)$ (bottom)")
165 ax1.set_yscale("log")
166 ax1.set_xscale("log")
167 ax1.set_ylabel(r"$1/\sqrt{m_{dn}}(\mu)$; [\mathrm{MeV}]^{-1/2}$",
168             fontsize=12)
169 ax1.legend(loc="best", fontsize=10)
170 ax1.grid(True, which="both", alpha=0.3)
171 ax1.set_title(
172     r"Inverse-tuple Brannen parametrization under 4-loop_"
173     r"$\overline{\mathrm{MS}}$ QCD running",
174     fontsize=12,
175 )
176
177 # Middle panel: T_inv(mu) and reference 1/sqrt(2)
178 ax2.plot(mu_grid, T_inv, lw=2.0, color=color_T, ls="-",
179         label=r"$T_{d, \mathrm{inv}}^{\mathrm{fit}}(\mu)$")
180 ax2.axhline(1.0 / np.sqrt(2.0), color="black", ls=":", lw=1.5,
181         label=r"$1/\sqrt{2}$, \approx 0.7071$")
182 ax2.set_ylabel(r"$T_{d, \mathrm{inv}}^{\mathrm{fit}}$", fontsize=12)
183 ax2.set_xscale("log")
184 band_T = max(0.001, 4 * (T_inv.max() - T_inv.min()))
185 centre_T = 0.5 * (T_inv.max() + T_inv.min())
186 ax2.set_ylim(min(centre_T - band_T, 1.0 / np.sqrt(2.0) - 0.0008),
187             max(centre_T + band_T, 1.0 / np.sqrt(2.0) + 0.0015))
188 ax2.legend(loc="best", fontsize=10)
189 ax2.grid(True, which="both", alpha=0.3)
190
191 # Bottom panel: phi_inv(mu)
192 ax3.plot(mu_grid, phi_inv, lw=2.0, color=color_phi, ls="-",
193         label=r"$\phi_{d, \mathrm{inv}}^{\mathrm{fit}}(\mu)$")
194 ax3.set_xlabel(r"$\mu$; [\mathrm{GeV}]$", fontsize=12)

```

```

196     ax3.set_ylabel(r"$\phi_{d\,,\mathrm{inv}}^{\mathrm{fit}}\;;\;[\mathrm{rad}]$",
197                   fontsize=12)
198     ax3.set_xscale("log")
199     band_phi = max(0.005, 4 * (phi_inv.max() - phi_inv.min()))
200     centre_phi = 0.5 * (phi_inv.max() + phi_inv.min())
201     ax3.set_ylim(centre_phi - band_phi, centre_phi + band_phi)
202     ax3.legend(loc="best", fontsize=10)
203     ax3.grid(True, which="both", alpha=0.3)
204
205     fig.tight_layout()
206     fig.savefig("Figure5.png", dpi=150, bbox_inches="tight")
207     plt.close(fig)
208     print("Saved Figure5.png")
209
210
211 if __name__ == "__main__":
212     main()

```

E Python Script S.1: low-energy window with local linear extrapolation

```

1  #!/usr/bin/env python3
2  """
3
4  Down-type quark mass formula: low-energy window with local linear
5  extrapolation near the left edge
6  =====
7
8  Purpose:
9  Start from the low-energy plot generated by Script1_en.py (Appendix~A),
10 approximate the finite points near the left edge by local straight lines,
11 and extend them visually down to  $\mu_{D3} = 0.587$  GeV.
12
13 Important:
14 This is not a physical prediction. It is only a visualization of a local
15 linear extrapolation near the left edge of the scanned window.
16 Below about 0.59 GeV the calculation is already close to the Landau-pole
17 region and perturbation theory is not reliable.
18 The figure is therefore only a visual aid for the apparent crossing trend
19 at low energy, not evidence for an actual unification scale.
20
21 Model assumptions:
22 same 4-loop  $\overline{MS}$ -bar running as in Script1_en.py (Appendix~A)
23 continuous matching at flavour thresholds
24 finite decoupling constants are not included
25 =====
26 """
27
28 from __future__ import annotations
29
30 import numpy as np
31 import matplotlib
32 matplotlib.use("Agg")
33 import matplotlib.pyplot as plt
34 import warnings
35
36 from Script1_en import (
37     alpha_s_at_mu,
38     compute_A,
39     predicted_mass,
40     find_mu_scale,
41     MD_INPUT,
42     MS_INPUT,
43     MU_D_INPUT,
44 )
45
46 LANDAU_POLE_APPROX = 0.5870 # [GeV], visual marker only
47 MU_D3_MIN = 0.5900 # scanned lower edge [GeV]
48 MU_D3_MAX = 1.0000 # scanned upper edge [GeV]
49 MU_EXTRAP_MIN = 0.5870 # extrapolation stop [GeV]
50 N_POINTS = 80
51 N_FIT = 6 # number of left-edge points used for local linear fit
52
53
54 def compute_scan(mu_min: float = MU_D3_MIN,
55                 mu_max: float = MU_D3_MAX,
56                 n_points: int = N_POINTS):
57     """Scan  $\mu_{D3}$  and compute  $\mu_{D1}$ ,  $\mu_{D2}$  at  $\mu_{D3}$ ."""
58     mu_d3_array = np.logspace(np.log10(mu_min), np.log10(mu_max), n_points)
59     mu_d1_array = np.full(n_points, np.nan)
60     mu_d2_array = np.full(n_points, np.nan)
61
62     for i, mu_d3 in enumerate(mu_d3_array):
63         if (i + 1) % 20 == 0:

```

```

64         print(f"_{i+1}/{n_points}_points")
65     try:
66         A_val = compute_A(mu_d3)
67         m_d1_pred = predicted_mass(A_val, 1)
68         m_d2_pred = predicted_mass(A_val, 2)
69
70         if m_d1_pred > 0.0:
71             mu_d1_array[i] = find_mu_scale(
72                 m_d1_pred,
73                 MD_INPUT,
74                 MU_D_INPUT,
75                 mu_low=mu_min,
76                 mu_high=1000.0,
77             )
78
79         if m_d2_pred > 0.0:
80             mu_d2_array[i] = find_mu_scale(
81                 m_d2_pred,
82                 MS_INPUT,
83                 MU_D_INPUT,
84                 mu_low=mu_min,
85                 mu_high=1000.0,
86             )
87     except Exception as exc:
88         warnings.warn(f"mu_d3={mu_d3:.6f} GeV: {exc}")
89
90     return mu_d3_array, mu_d1_array, mu_d2_array
91
92
93 def fit_local_line(x: np.ndarray, y: np.ndarray, n_fit: int = N_FIT):
94     """Fit y~a*x+b using the first n_fit finite points at the left edge."""
95     mask = np.isfinite(x) & np.isfinite(y)
96     xv = x[mask]
97     yv = y[mask]
98     if len(xv) < max(2, n_fit):
99         raise ValueError("Not enough finite points for local linear extrapolation")
100
101     xf = xv[:n_fit]
102     yf = yv[:n_fit]
103     a, b = np.polyfit(xf, yf, deg=1)
104     return float(a), float(b), xf, yf
105
106
107 def line_crossing_with_diagonal(a: float, b: float):
108     """Return x where a*x+b=0, if well defined."""
109     if abs(a - 1.0) < 1e-14:
110         return np.nan
111     return -b / (a - 1.0)
112
113
114 def line_crossing(a1: float, b1: float, a2: float, b2: float):
115     """Return x where a1*x+b1=a2*x+b2, if well defined."""
116     if abs(a1 - a2) < 1e-14:
117         return np.nan
118     return -(b1 - b2) / (a1 - a2)
119
120
121 def local_extrapolation(mu_d3: np.ndarray, mu_d1: np.ndarray, mu_d2: np.ndarray):
122     """Build local linear extrapolations from the left edge down to MU_EXTRAP_MIN."""
123     a1, b1, xf1, yf1 = fit_local_line(mu_d3, mu_d1, N_FIT)
124     a2, b2, xf2, yf2 = fit_local_line(mu_d3, mu_d2, N_FIT)
125
126     x_ext = np.linspace(MU_EXTRAP_MIN, MU_D3_MIN, 60)
127     y1_ext = a1 * x_ext + b1
128     y2_ext = a2 * x_ext + b2
129     y3_ext = x_ext # diagonal
130
131     cross_d1 = line_crossing_with_diagonal(a1, b1)

```

```

132     cross_d2 = line_crossing_with_diagonal(a2, b2)
133     cross_12 = line_crossing(a1, b1, a2, b2)
134
135     return {
136         "x_ext": x_ext,
137         "y1_ext": y1_ext,
138         "y2_ext": y2_ext,
139         "y3_ext": y3_ext,
140         "a1": a1,
141         "b1": b1,
142         "a2": a2,
143         "b2": b2,
144         "fit_x1": xf1,
145         "fit_y1": yf1,
146         "fit_x2": xf2,
147         "fit_y2": yf2,
148         "cross_d1": float(cross_d1),
149         "cross_d2": float(cross_d2),
150         "cross_12": float(cross_12),
151     }
152
153
154 def closest_approach(mu_d3: np.ndarray, mu_d1: np.ndarray, mu_d2: np.ndarray):
155     """Find the closest approach to mu_d1=mu_d2=mu_d3 in the scanned window."""
156     mask = np.isfinite(mu_d3) & np.isfinite(mu_d1) & np.isfinite(mu_d2)
157     if not np.any(mask):
158         return None
159
160     x = mu_d3[mask]
161     y1 = mu_d1[mask]
162     y2 = mu_d2[mask]
163     spread = np.maximum.reduce([
164         np.abs(y1 - x),
165         np.abs(y2 - x),
166         np.abs(y1 - y2),
167     ])
168     idx = np.argmin(spread)
169     return {
170         "mu_d3": float(x[idx]),
171         "mu_d1": float(y1[idx]),
172         "mu_d2": float(y2[idx]),
173         "spread": float(spread[idx]),
174     }
175
176
177 def make_plot(mu_d3_array: np.ndarray,
178              mu_d1_array: np.ndarray,
179              mu_d2_array: np.ndarray,
180              closest: dict | None,
181              extrap: dict,
182              output_path: str):
183     fig, ax = plt.subplots(figsize=(10, 8))
184
185     # Color-blind-safe palette (Okabe-Ito family) chosen to avoid the
186     # red/green confusion common in deuteranopia/protanopia.
187     # We also separate the series by line style so the curves remain
188     # distinguishable in grayscale printouts.
189     color_d1 = "#0072B2" # blue
190     color_d2 = "#D55E00" # vermillion
191     color_d3 = "#CC79A7" # reddish purple
192     color_aux = "#999999" # neutral gray for scan-edge / Landau guide
193     ls_d1 = "-"
194     ls_d2 = "--"
195     ls_d3 = "-."
196
197     m1 = np.isfinite(mu_d1_array)
198     m2 = np.isfinite(mu_d2_array)
199

```

```

200 # Main curves in the scanned window
201 ax.plot(mu_d3_array[m1], mu_d1_array[m1], linewidth=2.5,
202         color=color_d1, linestyle=ls_d1,
203         label=r'$\mu_{d1}$ (down)', zorder=3)
204 ax.plot(mu_d3_array[m2], mu_d2_array[m2], linewidth=2.5,
205         color=color_d2, linestyle=ls_d2,
206         label=r'$\mu_{d2}$ (strange)', zorder=3)
207 ax.plot(mu_d3_array, mu_d3_array, linewidth=2.5,
208         color=color_d3, linestyle=ls_d3,
209         label=r'$\mu_{d3}$ (diagonal $y=x$)', zorder=2)
210
211 # Extrapolation helper lines -- dotted to distinguish from scanned curves
212 ax.plot(extrap["x_ext"], extrap["y1_ext"], linestyle=':', linewidth=2.0,
213         color=color_d1, alpha=0.85,
214         label=r'local linear extrapolation of $\mu_{d1}$', zorder=2)
215 ax.plot(extrap["x_ext"], extrap["y2_ext"], linestyle=':', linewidth=2.0,
216         color=color_d2, alpha=0.85,
217         label=r'local linear extrapolation of $\mu_{d2}$', zorder=2)
218 ax.plot(extrap["x_ext"], extrap["y3_ext"], linestyle=':', linewidth=2.0,
219         color=color_d3, alpha=0.85,
220         label=r'extended diagonal to 0.587 GeV', zorder=1)
221
222 # Mark the scan lower edge and the Landau-pole guide
223 ax.axvline(MU_D3_MIN, linestyle=(0, (1, 1)), alpha=0.75, linewidth=1.5,
224           color=color_aux,
225           label=r'scan lower edge = {MU_D3_MIN:.3f} GeV')
226 ax.axvline(LANDAU_POLE_APPROX, linestyle=(0, (3, 1, 1, 1)),
227           alpha=0.9, linewidth=1.5, color='black',
228           label=r'Landau-pole guide = {LANDAU_POLE_APPROX:.3f} GeV')
229 ax.axhline(LANDAU_POLE_APPROX, linestyle=(0, (3, 1, 1, 1)),
230           alpha=0.6, linewidth=1.2, color='black')
231
232 # Show the points used for the left-edge fit
233 ax.scatter(extrap["fit_x1"], extrap["fit_y1"],
234           color=color_d1, marker='o', s=20, alpha=0.9, zorder=4)
235 ax.scatter(extrap["fit_x2"], extrap["fit_y2"],
236           color=color_d2, marker='s', s=20, alpha=0.9, zorder=4)
237
238 # Closest approach within the scanned window
239 if closest is not None:
240     x0 = closest["mu_d3"]
241     y1 = closest["mu_d1"]
242     y2 = closest["mu_d2"]
243     ax.plot([x0], [x0], marker='D', color=color_d3,
244           markersize=7, zorder=5, markeredgecolor='black')
245     ax.plot([x0], [y1], marker='o', color=color_d1,
246           markersize=7, zorder=5, markeredgecolor='black')
247     ax.plot([x0], [y2], marker='s', color=color_d2,
248           markersize=7, zorder=5, markeredgecolor='black')
249
250 # Estimated extrapolated crossing if the three lines cluster in the extension window
251 crosses = np.array([extrap["cross_d1"], extrap["cross_d2"], extrap["cross_12"]], dtype=
252                    float)
253 finite_crosses = crosses[np.isfinite(crosses)]
254 if len(finite_crosses) > 0:
255     x_cross = float(np.mean(finite_crosses))
256     if MU_EXTRAP_MIN * 0.995 <= x_cross <= MU_D3_MIN * 1.01:
257         ax.plot([x_cross], [x_cross], marker='*', color='black',
258               markersize=18, markeredgecolor='white', zorder=6)
259         ax.annotate(
260             'local-linear crossing estimate\n'
261             rf'$\mu_{\text{approx}}\{x\_cross:.4f\}$ GeV',
262             xy=(x_cross, x_cross),
263             xytext=(x_cross * 1.045, x_cross * 0.90),
264             fontsize=11,
265             arrowprops=dict(arrowstyle='->', lw=1.4, color='black'),
266             bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.9),
267             color='black',

```

```

267         zorder=7,
268     )
269
270     info_lines = [
271         'Dashed segments: local linear extrapolation from left-edge points',
272         rf'fit points used at left edge: {N_FIT}',
273         rf'linear root of  $\mu_{\{d1\}} = \mu_{\{d3\}}$ : {extrap["cross_d1"]:.4f} GeV',
274         rf'linear root of  $\mu_{\{d2\}} = \mu_{\{d3\}}$ : {extrap["cross_d2"]:.4f} GeV',
275         rf'linear root of  $\mu_{\{d1\}} = \mu_{\{d2\}}$ : {extrap["cross_12"]:.4f} GeV',
276         'Interpretation: visual aid only, not a perturbative result',
277     ]
278     ax.text(0.98, 0.03, '\n'.join(info_lines), transform=ax.transAxes,
279            fontsize=9, ha='right', va='bottom',
280            bbox=dict(boxstyle='round,pad=0.4', facecolor='white', alpha=0.92))
281
282     ax.set_xscale('log')
283     ax.set_yscale('log')
284     ax.set_xlim(MU_EXTRAP_MIN, MU_D3_MAX * 1.005)
285     ax.set_ylim(MU_EXTRAP_MIN * 0.995, MU_D3_MAX * 1.02)
286     ax.set_xlabel(r'$\mu_{d3}$ [GeV]', fontsize=14)
287     ax.set_ylabel(r'$\mu_{d1}$, $\mu_{d2}$, $\mu_{d3}$ [GeV]', fontsize=14)
288     ax.set_title(
289         r'Down-type quark: low-energy window with local linear extrapolation'
290         r'\n(4-loop QCD running,  $\overline{\mathrm{MS}}$ , continuous matching)',
291         fontsize=13,
292     )
293     ax.legend(fontsize=10, loc='upper left')
294     ax.grid(True, which='both', alpha=0.3)
295     ax.tick_params(labelsize=12)
296
297     fig.tight_layout()
298     fig.savefig(output_path, dpi=150, bbox_inches='tight')
299     plt.close(fig)
300
301
302 def main():
303     print('=' * 78)
304     print('Low-energy window with local linear extrapolation near the left edge')
305     print(f' scanned window:  $\mu_{d3} \in [\text{MU\_D3\_MIN} : .3f] \cup [\text{MU\_D3\_MAX} : .3f]$  GeV')
306     print(f' helper extrapolation down to: {MU_EXTRAP_MIN:.3f} GeV')
307     print(f' left-edge fit points used: {N_FIT}')
308     print(' note: below 0.59 GeV this is only a simple extrapolation into the'
309           ' nonperturbative region')
310     print('=' * 78)
311
312     mu_d3_array, mu_d1_array, mu_d2_array = compute_scan()
313     closest = closest_approach(mu_d3_array, mu_d1_array, mu_d2_array)
314     extrap = local_extrapolation(mu_d3_array, mu_d1_array, mu_d2_array)
315
316     print('local linear fit results near the left edge:')
317     print(f'  $\mu_{d1} \sim \{extrap[\'a1\'] : .6f\} \cup \mu_{d3} + \{extrap[\'b1\'] : .6f\}$ ')
318     print(f'  $\mu_{d2} \sim \{extrap[\'a2\'] : .6f\} \cup \mu_{d3} + \{extrap[\'b2\'] : .6f\}$ ')
319     print(f' linear root  $\mu_{d1} = \mu_{d3}$ : {extrap[\'cross_d1\']:.6f} GeV')
320     print(f' linear root  $\mu_{d2} = \mu_{d3}$ : {extrap[\'cross_d2\']:.6f} GeV')
321     print(f' linear root  $\mu_{d1} = \mu_{d2}$ : {extrap[\'cross_12\']:.6f} GeV')
322
323     probe_mu = MU_EXTRAP_MIN
324     y1_probe = extrap['a1'] * probe_mu + extrap['b1']
325     y2_probe = extrap['a2'] * probe_mu + extrap['b2']
326     print(f' extrapolated values at  $\mu_{d3} = \{probe\_mu : .3f\}$  GeV:')
327     print(f'  $\mu_{d1} \sim \{y1\_probe : .6f\}$  GeV')
328     print(f'  $\mu_{d2} \sim \{y2\_probe : .6f\}$  GeV')
329     print(f' diagonal  $\mu = \{probe\_mu : .6f\}$  GeV')
330     try:
331         alpha_s_value = alpha_s_at_mu(probe_mu) * np.pi
332         print(f'  $\alpha_s(\{probe\_mu : .3f\} \text{ GeV}) \sim \{alpha\_s\_value : .3f\}$ ')
333         if alpha_s_value > 1.0:
334             print(' warning: this probe point is deep in the nonperturbative regime')

```



```

334     except Exception as exc:
335         print(f'alpha_s_evaluation_failed_near_the_extrapolation_edge:{exc}')
336
337     output_path = 'unification_scale_with_extrapolation.png'
338     make_plot(mu_d3_array, mu_d1_array, mu_d2_array, closest, extrap, output_path)
339     print(f'plot_saved_to:{output_path}')
340
341
342 if __name__ == '__main__':
343     main()

```

References

- [1] H. Harari, H. Haut, and J. Weyers, “Quark masses and Cabibbo angles,” *Physics Letters B* **78**(4), 459–461 (1978). doi:10.1016/0370-2693(78)90668-2.
- [2] Y. Koide, “Fermion-boson two body model of quarks and leptons and Cabibbo mixing,” *Lettere al Nuovo Cimento* **34**(7), 201–206 (1982). doi:10.1007/BF02773988.
- [3] Y. Koide, “A fermion-boson composite model of quarks and leptons,” *Physics Letters B* **120**(1–3), 161–165 (1983). doi:10.1016/0370-2693(83)90628-0.
- [4] C. A. Brannen, “The lepton masses,” *Brannen Works* (2006). Available at <https://brannenworks.com/MASSES2.pdf>.
- [5] C. A. Brannen, “Spin path integrals and generations,” *Foundations of Physics* **40**, 1681–1699 (2010). doi:10.1007/s10701-010-9465-8.
- [6] V. Bargmann, “Note on Wigner’s theorem on symmetry operations,” *Journal of Mathematical Physics* **5**(7), 862–868 (1964). doi:10.1063/1.1704188.
- [7] N. Mukunda and R. Simon, “Quantum kinematic approach to the geometric phase. I. General formalism,” *Annals of Physics* **228**(2), 205–268 (1993). doi:10.1006/aphy.1993.1093.
- [8] R. Foot, “A note on Koide’s lepton mass relation,” arXiv:hep-ph/9402242 (1994).
- [9] Y. Sumino, “Family Gauge Symmetry and Koide’s Mass Formula,” *Physics Letters B* **671**(4), 477–480 (2009). arXiv:0812.2090 [hep-ph].
- [10] W. Rodejohann and H. Zhang, “Extended Empirical Fermion Mass Relation,” *Physics Letters B* **698**(2), 152–156 (2011). arXiv:1101.5525 [hep-ph].
- [11] A. Kartavtsev, “A remark on the Koide relation for quarks,” arXiv:1111.0480 [hep-ph] (2011).
- [12] J. Kocik, “The Koide Lepton Mass Formula and Geometry of Circles,” arXiv:1201.2067 [physics.gen-ph] (2012).
- [13] Y. Koide and H. Nishiura, “Parameter-Independent Quark Mass Relation in the $U(3) \times U(3)'$ Model,” arXiv:1805.07334 [hep-ph] (2018).
- [14] Particle Data Group Collaboration, “Review of particle physics,” *Physical Review D* **110**(3), 030001 (2024). doi:10.1103/PhysRevD.110.030001.
- [15] A. Rivero, “The UV renormalization-group run of Koide-type mass relations: a reference dataset from M_Z to M_{Pl} ,” ai.viXra.org:2603.0048 (2026).
- [16] M. Beneke, “Renormalons,” *Physics Reports* **317**(1–2), 1–142 (1999). doi:10.1016/S0370-1573(98)00130-6.
- [17] T. Appelquist and J. Carazzone, “Infrared singularities and massive fields,” *Physical Review D* **11**(10), 2856–2861 (1975). doi:10.1103/PhysRevD.11.2856.

- [18] K. G. Chetyrkin, B. A. Kniehl, and M. Steinhauser, “Decoupling relations to $\mathcal{O}(\alpha_s^3)$ and their connection to low-energy theorems,” *Nuclear Physics B* **510**(1–2), 61–87 (1998). doi:10.1016/S0550-3213(97)00649-4.
- [19] K. G. Chetyrkin, J. H. Kühn, and M. Steinhauser, “RunDec: a Mathematica package for running and decoupling of the strong coupling and quark masses,” *Computer Physics Communications* **133**(1), 43–65 (2000). doi:10.1016/S0010-4655(00)00155-7.
- [20] T. van Ritbergen, J. A. M. Vermaseren, and S. A. Larin, “The four-loop β -function in quantum chromodynamics,” *Physics Letters B* **400**(3–4), 379–384 (1997). doi:10.1016/S0370-2693(97)00370-5.
- [21] K. G. Chetyrkin, “Quark mass anomalous dimension to $\mathcal{O}(\alpha_s^4)$,” *Physics Letters B* **404**(1–2), 161–165 (1997). doi:10.1016/S0370-2693(97)00535-2.