

Derivatives of the generalized chi-square CDF with respect to its parameters

1 Gradient and Hessian derivations

The generalized chi-square distribution is the distribution of a quadratic function of a normal random vector, or equivalently of a weighted sum of independent chi-square variables. It appears wherever a squared or quadratic quantity is formed from correlated Gaussian noise, most notably as the decision variable of a Bayes-optimal classifier between two normal populations. This appendix derives the gradient and the Hessian — the vector of first derivatives and the matrix of second derivatives — of the generalized chi-square cumulative distribution function (cdf) with respect to the parameters that specify it. Throughout we follow the notation of the generalized chi-square paper.²

A generalized chi-square variable, which we write $\tilde{\chi}$, can be described in two equivalent ways, and each way carries its own natural set of parameters. We will need the derivatives of its cdf F with respect to both sets, so we introduce them here.

The canonical form, and the native parameters. In its canonical form, a generalized chi-square variable is a weighted sum of independent non-central chi-square variables, plus an independent normal term:

$$\tilde{\chi}_{\mathbf{w}, \mathbf{k}, \boldsymbol{\lambda}, s, m} = \sum_i w_i \chi_{k_i, \lambda_i}^{\prime 2} + s z + m.$$

Here each $\chi_{k_i, \lambda_i}^{\prime 2}$ is a non-central chi-square variable with k_i degrees of freedom (the number of squared standard normals summed) and non-centrality λ_i (the summed squared means of those normals); w_i is the weight applied to it; z is a standard normal; s scales that normal term; and m is a constant offset. We call $(\mathbf{w}, \mathbf{k}, \boldsymbol{\lambda}, s, m)$ the *native parameters*, since they are intrinsic to the distribution and do not refer to any underlying vector.

The quadratic-form, and the boundary parameters. Equivalently, $\tilde{\chi}$ can be written as a quadratic function of a normal vector $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

$$\tilde{\chi} = q(\mathbf{x}) = \mathbf{x}' \mathbf{Q}_2 \mathbf{x} + \mathbf{q}_1' \mathbf{x} + q_0,$$

with a symmetric matrix $\mathbf{Q}_2$, a vector $\mathbf{q}_1$, and a scalar q_0 . Because the triple $(\mathbf{Q}_2, \mathbf{q}_1, q_0)$ is exactly the coefficient set of a quadratic classification boundary, we call these the *boundary parameters*.

Diagonalizing this quadratic form recovers the canonical form above: the eigenvalues of the matrix $\boldsymbol{\Sigma}^{1/2} \mathbf{Q}_2 \boldsymbol{\Sigma}^{1/2}$ are the weights $\mathbf{w}$, and the multiplicity of each eigenvalue is the corresponding degree of freedom k_i . This is the conversion between the two representations derived by Das and Geisler.¹

We will derive the gradient and Hessian of F with respect to each set of parameters.

The Gil-Pelaez inversion and a master formula

Both parameter sets rest on the same piece of machinery, which we set up once here. However $\tilde{\chi}$ is parametrized, it has a characteristic function $\phi(t) = \mathbb{E}[e^{it\tilde{\chi}}]$, the Fourier transform of its density, and we recover the cdf and the probability

density function (pdf) from ϕ by the Gil-Pelaez inversion formula, in the numerically convenient form given by Imhof:²

$$F(x) = \frac{1}{2} - \frac{1}{\pi} \int_0^\infty \frac{\operatorname{Im}[\phi(t) e^{-itx}]}{t} dt, \quad f(x) = \frac{1}{\pi} \int_0^\infty \operatorname{Re}[\phi(t) e^{-itx}] dt.$$

The key observation is that both of these are *linear* in ϕ : the characteristic function enters only through the bracketed product, while the kernel e^{-itx} and the weight $1/t$ depend on the argument x and the integration variable t , but on none of the distribution's parameters. It will be convenient to give the cdf integral its own name as an operator acting on a function ψ , dropping the constant $\frac{1}{2}$ since no parameter depends on it:

$$T[\psi](x) := -\frac{1}{\pi} \int_0^\infty \frac{\operatorname{Im}[\psi(t) e^{-itx}]}{t} dt, \quad \text{so that } T[\phi] = F - \frac{1}{2}.$$

Since every parameter enters F only through ϕ , differentiating F in a parameter θ amounts to differentiating ϕ inside the integral. Writing the derivative of ϕ through its logarithm, $\partial_\theta \phi = (\partial_\theta \log \phi) \phi$, and using the linearity of T , we obtain the master formula used throughout this appendix:

$$\partial_\theta F = T[(\partial_\theta \log \phi) \phi].$$

Each derivative then costs only one line of algebra to find $\partial_\theta \log \phi$, after which we are left with an integrand to recognize. Two simple rules do the recognizing every time.

Rule R1: a factor $(1 - 2iw_j t)^{-1}$ raises a degree of freedom by two. The native characteristic function (§1.1) contains the factor $(1 - 2iw_j t)^{-k_j/2}$ for the j -th component. Multiplying the integrand by one more power $(1 - 2iw_j t)^{-1}$ therefore turns it into $(1 - 2iw_j t)^{-(k_j+2)/2}$ and changes nothing else, which is simply the same distribution with its j -th degree of freedom raised by two. Writing $[k_j + n]$ for the distribution with k_j replaced by $k_j + n$,

$$(1 - 2iw_j t)^{-1} \phi = \phi_{[k_j+2]}.$$

Because T is linear, this shift passes straight through to the cdf and pdf, so that $T[\phi_{[k_j+2]}] = F_{[k_j+2]} - \frac{1}{2}$, and similarly for the density. Whenever such a factor appears, we may simply read it as "evaluate at a raised degree of freedom," with no new integral to perform.

Rule R2: a factor it is an x -derivative. The argument x enters the integrand only through the kernel e^{-itx} , and differentiating that kernel in x brings down a factor $-it$. Hence

$$T[(it) \psi] = -\partial_x T[\psi],$$

so that $T[(it)\phi] = -f$, and more generally $T[(it)^n \phi] = (-\partial_x)^n F$. The same rule is what lets us compute the derivatives of the density in its argument without any finite differencing: inserting a factor t^n into the inversion integral for f returns $f^{(n)}$ exactly. For instance $f'(x) = \frac{1}{\pi} \int_0^\infty t \operatorname{Im}[\phi e^{-itx}] dt$, an integral of the same kind as the one for f . We use this repeatedly, first for $\partial_s F$ and then throughout the Hessian.

Between them, these two rules turn every derivative below into one of three kinds of object:

1. a cdf or pdf at shifted degrees of freedom, by rule R1;
2. a derivative of the density in its argument, f' , f'' , $\dots$, obtained from the same inversion integral by rule R2;

3. or, only when we differentiate in a degree of freedom k_j , a single convergent integral carrying a logarithmic weight (§1.1).

Every one of these is exact — we never finite-difference, and never regularize. The one object that needs care is the second kind: when the normal term vanishes ($s = 0$) and the total degrees of freedom are few, the inversion integral for the density derivatives loses its convergence, and §1.5 computes them there from an exact series instead.

Why we keep two separate parameter sets

The native-parameter derivatives (§1.1–§1.2) and the boundary-parameter derivatives (§1.3–§1.4) answer different questions, and neither is obtained from the other.

The native-parameter derivatives are the general-purpose ones, needed whenever a generalized chi-square variable is the object of interest in its own right, as in signal detection or the study of quadratic forms in Gaussian variables. They also serve as a chain-rule hub for any other parametrization: if some parameter θ maps into $(\mathbf{w}, \mathbf{k}, \boldsymbol{\lambda}, s, m)$, its derivative follows by combining the native derivatives with the chain rule, as illustrated at the end of §1.1.

The boundary-parameter derivatives serve the quadratic-classification problem, such as computing the sensitivity of a classification error to the boundary coefficients.³ One might hope to reach them indirectly, by chaining the native derivatives of §1.1 through the quadratic-to-native conversion of Das and Geisler.¹ That route, however, is fragile. The conversion passes through the eigendecomposition of $\boldsymbol{\Sigma}^{1/2} \mathbf{Q}_2 \boldsymbol{\Sigma}^{1/2}$, whose eigenvalue multiplicities are the degrees of freedom $\mathbf{k}$. These multiplicities are integers, so they are not differentiable functions of the coefficients; moreover the eigenvectors turn singular wherever two eigenvalues coincide. A chain rule built on this map is not stable.

We avoid the difficulty by differentiating the characteristic function of $q(\mathbf{x})$ *directly* in the coefficients (§1.3). That route never introduces $\mathbf{k}$ as a variable at all: it differentiates the ordinary matrix inverse $\mathbf{M}^{-1}(t) = (\boldsymbol{\Sigma}^{-1} - 2it \mathbf{Q}_2)^{-1}$ through the identity $d(\mathbf{M}^{-1}) = 2it \mathbf{M}^{-1}(d\mathbf{Q}_2)\mathbf{M}^{-1}$, which is smooth in $\mathbf{Q}_2$ and tracks no individual eigenvalue or eigenvector, so no integer multiplicity ever appears to spoil differentiability. The eigendecomposition returns later, but only as a device for evaluating the resulting integral quickly (§1.3), and even there only through the eigenvalues, never the multiplicities.

So we have two derivative routines for two genuinely different sets of variables, not one computation written twice. What they share is the layer beneath them: the same x -weighted inversion integrands (for f' and for $\partial_{k_j} F$) and the same two rules. The natural design is therefore a single common integrand at the core, with the two routines built on top of it.

1.1 Gradient with respect to the native parameters

In the canonical form, the characteristic function is

$$\phi(t) = \frac{\exp\left(imt - \frac{1}{2}s^2t^2 + \sum_j \frac{i\lambda_j w_j t}{1 - 2iw_j t}\right)}{\prod_j (1 - 2iw_j t)^{k_j/2}}.$$

Its logarithm is a plain sum of terms,

$$\log \phi(t) = imt - \frac{1}{2}s^2t^2 + \sum_j \frac{i\lambda_j w_j t}{1 - 2iw_j t} - \sum_j \frac{k_j}{2} \log(1 - 2iw_j t),$$

so each parameter derivative is straightforward. We take the parameters one at a time: differentiate $\log \phi$, substitute into

the master formula $\partial_\theta F = T[(\partial_\theta \log \phi) \phi]$, and read off the result with rules R1 and R2.

Offset m . The offset appears only in the single term imt , so $\partial_m \log \phi = it$, and by rule R2 this factor it is one x -derivative:

$$\partial_m F = T[(it)\phi] = -\partial_x F = -f(x).$$

Normal scale s . Only the term $-\frac{1}{2}s^2 t^2$ depends on s . Using $(it)^2 = -t^2$, its derivative is $\partial_s \log \phi = -s t^2 = s(it)^2$, and two factors of it are two derivatives in the argument:

$$\partial_s F = s T[(it)^2 \phi] = s \partial_x^2 F = s f'(x).$$

Here f' comes from the x -weighted inversion of rule R2, not from finite differencing.

Non-centralities λ_j . Only the exponent's sum involves λ_j , giving

$$\partial_{\lambda_j} \log \phi = \frac{i w_j t}{1 - 2i w_j t}.$$

To recognize this, substitute $u = 2i w_j t$ (so $i w_j t = u/2$) and split off a constant:

$$\frac{i w_j t}{1 - 2i w_j t} = \frac{u/2}{1 - u} = \frac{1}{2} \left(\frac{1}{1 - u} - 1 \right) = \frac{1}{2} \left[(1 - 2i w_j t)^{-1} - 1 \right].$$

The first piece is a factor $(1 - 2i w_j t)^{-1}$, which by rule R1 raises k_j to $k_j + 2$; the second is ϕ itself. Hence

$$\partial_{\lambda_j} F = \frac{1}{2} (F_{[k_j+2]} - F).$$

Weights w_j . The weight w_j appears both in the exponent and in the power of the denominator, so its derivative has two contributions. Using $\frac{d}{dw_j} \frac{w_j}{1 - 2i w_j t} = \frac{1}{(1 - 2i w_j t)^2}$,

$$\partial_{w_j} \log \phi = \underbrace{\frac{i \lambda_j t}{(1 - 2i w_j t)^2}}_{\text{from the exponent}} + \underbrace{\frac{i k_j t}{1 - 2i w_j t}}_{\text{from the power}} = k_j (it) (1 - 2i w_j t)^{-1} + \lambda_j (it) (1 - 2i w_j t)^{-2}.$$

Each term is a factor it times one or two factors of $(1 - 2i w_j t)^{-1}$. Rule R1 (applied once, then twice) reads off the degree-of-freedom shifts, and rule R2 turns each it into $-\partial_x$:

$$\partial_{w_j} F = k_j (-\partial_x) F_{[k_j+2]} + \lambda_j (-\partial_x) F_{[k_j+4]},$$

that is,

$$\partial_{w_j} F = -k_j f_{[k_j+2]}(x) - \lambda_j f_{[k_j+4]}(x).$$

As a check, setting $\lambda_j = 0$ leaves $-k_j f_{[k_j+2]}$, the weighted-sum analogue of the elementary identity $x f_{\chi_k^2}(x) = k f_{\chi_{k+2}^2}(x)$.

Degrees of freedom k_j . Here the derivative is a logarithm,

$$\partial_{k_j} \log \phi = -\frac{1}{2} \log(1 - 2iw_j t) \implies \partial_{k_j} F = \frac{1}{2\pi} \int_0^\infty \frac{\text{Im}[\log(1 - 2iw_j t) \phi e^{-itx}]}{t} dt.$$

This is the one derivative with no finite shifted-dof form, because a logarithm cannot be absorbed by rule R1, so it remains an integral. It nonetheless converges without difficulty: $\log(1 - 2iw_j t)$ grows only like $\log t$, which the weight $1/t$ easily controls, and it is evaluated by the same inversion machinery as F itself.

All five derivatives converge at first order. The only two that pull down positive powers of t — ∂_m (a factor it) and ∂_s (a factor t^2) — are precisely the ones that are x -derivatives of the integrable density, $-f$ and $s f'$. A genuine loss of convergence appears only at *second* order in the location and scale directions, and we defer that discussion to §1.5.

Chaining to any other parametrization. If a downstream parameter θ maps into $(\mathbf{w}, \mathbf{k}, \boldsymbol{\lambda}, s, m)$, its derivative assembles from the blocks above by the chain rule, $\partial_\theta F = \sum_j (\partial_\theta w_j) \partial_{w_j} F + \dots$. The single case where we do not recommend this — the quadratic boundary coefficients — is treated directly, and more stably, in §1.3.

1.2 Hessian with respect to the native parameters

The Hessian collects the second derivatives $H_{ab} = \partial_a \partial_b F$ over all pairs of parameters $a, b \in \{w_j, k_j, \lambda_j, s, m\}$. It is a symmetric matrix, of size $(3n + 2) \times (3n + 2)$ for a distribution with n components. As with the gradient, the two rules reduce nearly every entry to shifted-dof cdfs and their x -derivatives; the only entries that remain as integrals are those that differentiate a degree of freedom, and these are the same convergent log-weighted integrals met in §1.1.

The second-order master formula. Differentiating the gradient $\partial_b F = T[L_b \phi]$ once more, and writing $L_a \equiv \partial_a \log \phi$, the product rule (with $\partial_a \phi = L_a \phi$) gives

$$\partial_a \partial_b F = T[(L_{ab} + L_a L_b) \phi], \quad L_{ab} \equiv \partial_a \partial_b \log \phi.$$

Everything new is contained in the second derivatives L_{ab} of $\log \phi$; the products $L_a L_b$ are built from the first derivatives already in hand. Rules R1 and R2 then turn each resulting symbol into a computable quantity.

The derivatives of $\log \phi$. It helps to abbreviate the three recurring pieces: $p \equiv it$, $g_j \equiv (1 - 2iw_j t)^{-1}$, and $\ell_j \equiv -\frac{1}{2} \log(1 - 2iw_j t)$. Note $p^2 = -t^2$. Under the two rules, a factor p^a acts as $(-\partial_x)^a$, a factor g_j^n raises k_j by $2n$, and a factor ℓ_j carries the logarithmic weight of a k_j -derivative. In these symbols the first derivatives of §1.1 read

$$L_m = p, \quad L_s = s p^2, \quad L_{\lambda_j} = \frac{1}{2}(g_j - 1), \quad L_{w_j} = p(k_j g_j + \lambda_j g_j^2), \quad L_{k_j} = \ell_j.$$

Because $\log \phi$ splits into independent per-component terms plus the global m and s terms, a mixed second derivative L_{ab} vanishes unless a and b belong to the same component, or are global. The surviving ones are

$$L_{ss} = p^2, \quad L_{w_j w_j} = 2p^2(k_j g_j^2 + 2\lambda_j g_j^3), \quad L_{\lambda_j w_j} = p g_j^2, \quad L_{k_j w_j} = p g_j,$$

with all others zero; in particular every $L_{m \cdot} = 0$, and $L_{\lambda_j \lambda_j} = L_{k_j k_j} = L_{\lambda_j k_j} = 0$.

We now assemble $H_{ab} = T[(L_{ab} + L_a L_b) \phi]$, grouping the entries by whether they differentiate a degree of freedom.

Entries with no degree-of-freedom derivative (closed form). For any pair that does not differentiate a k , rules R1 and R2 leave only shifted-dof cdfs and their x -derivatives $f = \partial_x F$, $f' = \partial_x^2 F$, $\dots$, with no integral surviving. We sort the entries below by how many components the two derivatives touch — none, one, the same one twice, or two different ones — since that alone decides how many degree-of-freedom shifts show up and how large they are.

Global: both derivatives fall on m or s , touching no component.

$$H_{mm} = f', \quad H_{ms} = -s f'', \quad H_{ss} = f' + s^2 f''.$$

Global $\times$ component: one derivative is global, the other lands on a single component j , shifting only that component's degrees of freedom.

$$\begin{aligned} H_{m\lambda_j} &= -\frac{1}{2}(f_{[k_j+2]} - f), & H_{mw_j} &= k_j f'_{[k_j+2]} + \lambda_j f'_{[k_j+4]}, \\ H_{s\lambda_j} &= \frac{1}{2}s(f'_{[k_j+2]} - f'), & H_{sw_j} &= -s(k_j f''_{[k_j+2]} + \lambda_j f''_{[k_j+4]}). \end{aligned}$$

Same component: both derivatives land on the same component j , so the shifts stack, up to $k_j + 8$ below.

$$\begin{aligned} H_{\lambda_j\lambda_j} &= \frac{1}{4}(F_{[k_j+4]} - 2F_{[k_j+2]} + F), \\ H_{\lambda_j w_j} &= \frac{1}{2}k_j f_{[k_j+2]} + \frac{1}{2}(\lambda_j - k_j - 2)f_{[k_j+4]} - \frac{1}{2}\lambda_j f_{[k_j+6]}, \\ H_{w_j w_j} &= k_j(k_j + 2)f'_{[k_j+4]} + 2\lambda_j(k_j + 2)f'_{[k_j+6]} + \lambda_j^2 f'_{[k_j+8]}. \end{aligned}$$

Cross component ($i \neq j$): the two derivatives land on different components, each shifting its own degrees of freedom independently.

$$\begin{aligned} H_{\lambda_i\lambda_j} &= \frac{1}{4}(F_{[k_i+2, k_j+2]} - F_{[k_i+2]} - F_{[k_j+2]} + F), \\ H_{\lambda_i w_j} &= -\frac{1}{2}\left[k_j(f_{[k_i+2, k_j+2]} - f_{[k_j+2]}) + \lambda_j(f_{[k_i+2, k_j+4]} - f_{[k_j+4]})\right], \\ H_{w_i w_j} &= k_i k_j f'_{[k_i+2, k_j+2]} + k_i \lambda_j f'_{[k_i+2, k_j+4]} + \lambda_i k_j f'_{[k_i+4, k_j+2]} + \lambda_i \lambda_j f'_{[k_i+4, k_j+4]}. \end{aligned}$$

Two identities make convenient sanity checks: the whole m -row is minus the x -derivative of the gradient, $H_{mb} = -\partial_x(\partial_b F)$; and for $b \neq s$, $H_{sb} = s \partial_x^2(\partial_b F)$.

Entries that differentiate a degree of freedom (convergent integrals). The remaining entries each carry at least one factor ℓ_j , so they stay as the k -derivative integral of §1.1, now with any extra factors p^a or g^n realized as x -derivatives and degree-of-freedom shifts. Written through the basic integral $\partial_{k_j} F$ and its shifted or x -differentiated versions,

$$\begin{aligned} H_{mk_j} &= -\partial_x \partial_{k_j} F, & H_{sk_j} &= s \partial_x^2 \partial_{k_j} F, \\ H_{\lambda_j k_j} &= \frac{1}{2}(\partial_{k_j} F_{[k_j+2]} - \partial_{k_j} F), & H_{\lambda_i k_j} &= \frac{1}{2}(\partial_{k_j} F_{[k_i+2]} - \partial_{k_j} F) \quad (i \neq j), \end{aligned}$$

$$H_{w_j k_j} = -f_{[k_j+2]} - k_j \partial_{k_j} f_{[k_j+2]} - \lambda_j \partial_{k_j} f_{[k_j+4]}, \quad H_{w_i k_j} = -k_i \partial_{k_j} f_{[k_i+2]} - \lambda_i \partial_{k_j} f_{[k_i+4]} \quad (i \neq j),$$

$$H_{k_j k_j} = \partial_{k_j}^2 F, \quad H_{k_i k_j} = \partial_{k_i} \partial_{k_j} F \quad (i \neq j).$$

Here $\partial_{k_j} f_{[\dots]}$ is the k -derivative of a shifted pdf — the same log-weighted integrand with one extra $-\partial_x$ — and $\partial_{k_j} F_{[\dots]}$ is the k -derivative of a shifted cdf. The last two entries carry a squared or product logarithmic weight, ℓ_j^2 or $\ell_i \ell_j$. All of these converge for the reason given in §1.1: a single logarithm, or the product of two, grows slowly enough that the weight $1/t$ controls it.

As in the gradient, the only entries that raise the net power of t are the m and s ones, and they emerge as finite density x -derivatives f', f'', f''' . These are the objects whose robust computation §1.5 takes up.

1.3 Gradient with respect to the boundary parameters

Say $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and we have its probability content in the quadratic region:

$$q(\mathbf{x}) = \mathbf{x}' \mathbf{Q}_2 \mathbf{x} + \mathbf{q}_1' \mathbf{x} + q_0 > 0.$$

We ask how this probability content changes as we vary the boundary coefficients $(\mathbf{Q}_2, \mathbf{q}_1, q_0)$. Writing F for the generalized chi-square cdf of $q(\mathbf{x})$, this contained probability is $P(q(\mathbf{x}) > 0) = 1 - F(0)$, so its gradient is minus that of $F(0)$; we therefore work with the cdf F and negate at the end. As explained above ("Why we keep two separate parameter sets"), we obtain the gradient by differentiating the characteristic function of $q(\mathbf{x})$ directly in the coefficients, rather than chaining through the non-differentiable conversion to native parameters.

The characteristic function of a quadratic form. Completing the Gaussian integral for $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ gives

$$\phi(t) = \det(\mathbf{I} - 2it \boldsymbol{\Sigma} \mathbf{Q}_2)^{-1/2} \exp\left(\frac{1}{2} \mathbf{p}' \mathbf{M}^{-1} \mathbf{p} - \frac{1}{2} \boldsymbol{\mu}' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + it q_0\right),$$

where

$$\mathbf{M}(t) = \boldsymbol{\Sigma}^{-1} - 2it \mathbf{Q}_2, \quad \mathbf{p}(t) = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + it \mathbf{q}_1, \quad \tilde{\boldsymbol{\mu}}(t) \equiv \mathbf{M}^{-1} \mathbf{p}.$$

The matrix $\mathbf{M}^{-1}$ plays the role of a complex "tilted" covariance and $\tilde{\boldsymbol{\mu}}$ that of a tilted mean. Taking the logarithm,

$$\log \phi = -\frac{1}{2} \log \det(\mathbf{I} - 2it \boldsymbol{\Sigma} \mathbf{Q}_2) + \frac{1}{2} \mathbf{p}' \mathbf{M}^{-1} \mathbf{p} - \frac{1}{2} \boldsymbol{\mu}' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + it q_0.$$

Differentiating. By the master formula, for any coefficient $\theta \in \{\mathbf{Q}_2, \mathbf{q}_1, q_0\}$ we have $\partial_\theta F = T[(\partial_\theta \log \phi) \phi]$, so all that is new is $\partial_\theta \log \phi$. Using the standard matrix identity $d \log \det \mathbf{X} = \text{tr}(\mathbf{X}^{-1} d\mathbf{X})$, the factorization $\mathbf{I} - 2it \boldsymbol{\Sigma} \mathbf{Q}_2 = \boldsymbol{\Sigma} \mathbf{M}$, and the inverse-differential $d(\mathbf{M}^{-1}) = 2it \mathbf{M}^{-1} (d\mathbf{Q}_2) \mathbf{M}^{-1}$, each derivative is a single line:

$$\partial_{q_0} \log \phi = it, \quad \nabla_{\mathbf{q}_1} \log \phi = it \tilde{\boldsymbol{\mu}}, \quad \nabla_{\mathbf{Q}_2} \log \phi = it(\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}').$$

To see where these come from: q_0 appears only in the term $it q_0$, giving the first. For $\mathbf{q}_1$, we have $\partial \mathbf{p} / \partial \mathbf{q}_1 = it \mathbf{I}$, so the term $\frac{1}{2} \mathbf{p}' \mathbf{M}^{-1} \mathbf{p}$ contributes $it \mathbf{M}^{-1} \mathbf{p} = it \tilde{\boldsymbol{\mu}}$. And for $\mathbf{Q}_2$, the log-determinant contributes $it \text{tr}(\mathbf{M}^{-1} d\mathbf{Q}_2)$, while $\frac{1}{2} \mathbf{p}' \mathbf{M}^{-1} \mathbf{p}$ contributes $it \tilde{\boldsymbol{\mu}}' d\mathbf{Q}_2 \tilde{\boldsymbol{\mu}} = it \text{tr}(\tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}' d\mathbf{Q}_2)$, which together give the symmetric matrix shown.

Substituting into the master formula gives the cdf gradient:

$$\partial_{q_0} F = T[(it) \phi] = -f(0), \quad \nabla_{\mathbf{q}_1} F = T[(it) \tilde{\boldsymbol{\mu}} \phi], \quad \nabla_{\mathbf{Q}_2} F = T[(it)(\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}') \phi],$$

all evaluated at the boundary level 0. The first mirrors $\partial_m F$ of §1.1: since q_0 shifts $q(\mathbf{x})$ rigidly (just as the native offset m shifts $\tilde{\chi}$), differentiating the cdf in q_0 is minus differentiating it in its level, and rule R2 gives $\partial_{q_0} F = -f(0)$, the density at the boundary. The other two are density-type Gil-Pelaez integrals, weighted respectively by the vector $\tilde{\boldsymbol{\mu}}$ and the matrix $\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}'$.

The symmetry of the $\mathbf{Q}_2$ derivative. Because $\nabla_{\mathbf{Q}_2} \log \phi$ is symmetric, so is the gradient $\mathbf{G} \equiv \partial F / \partial \mathbf{Q}_2$: it is the symmetric matrix defined by $dF \approx \text{tr}(\mathbf{G} d\mathbf{Q}_2)$ for symmetric perturbations $d\mathbf{Q}_2$. This is what a matrix-space optimizer

wants, since a step $\mathbf{Q}_2 \leftarrow \mathbf{Q}_2 - \eta \mathbf{G}$ keeps $\mathbf{Q}_2$ symmetric.

Evaluating the gradient efficiently. Each derivative has the form $(it) \cdot (\text{weight}) \cdot \phi$, a single Gil-Pelaez integral. The only apparent cost is that $\mathbf{M}^{-1}$ looks like a fresh $d \times d$ complex inverse at every quadrature node. But the conversion to native parameters already supplies the eigendecomposition $\Sigma^{1/2} \mathbf{Q}_2 \Sigma^{1/2} = \mathbf{V} \text{diag}(w_j) \mathbf{V}'$, whose eigenvalues w_j are exactly the weights. In that basis,

$$\mathbf{M}^{-1} = \Sigma^{1/2} \mathbf{V} \text{diag}\left(\frac{1}{1-2it w_j}\right) \mathbf{V}' \Sigma^{1/2},$$

so the per-node cost collapses to a diagonal scaling by $1/(1 - 2it w_j)$ — the very factor of rule R1. And since $\mathbf{M}^{-1}$ and $\tilde{\boldsymbol{\mu}}$ are shared across all the entries, a single integration returns the whole $(\mathbf{Q}_2, \mathbf{q}_1, q_0)$ gradient at once.

The classification error. The optimal quadratic boundary and the error it produces are derived by Das and Geisler,¹ and the gradient of that error follows directly. Writing the error as $\mathcal{E} = p_0 P(\eta > 0 \mid 0) + p_1 P(\eta < 0 \mid 1)$ for a log-odds boundary η (the same coefficients enter both classes, with opposite sign), its gradient is $\nabla_{\theta} \mathcal{E} = p_0 \nabla_{\theta} P_0 + p_1 \nabla_{\theta} P_1$, each term being the gradient above, evaluated with that class's $(\boldsymbol{\mu}_y, \Sigma_y)$ and the appropriate sign and tail.

1.4 Hessian with respect to the boundary parameters

For the second derivatives we use the same second-order master formula as in §1.2, now ranging over $a, b \in \{\mathbf{Q}_2, \mathbf{q}_1, q_0\}$:

$$\partial_a \partial_b F = T[(L_{ab} + L_a L_b) \phi], \quad L_a \equiv \partial_a \log \phi, \quad L_{ab} \equiv \partial_a \partial_b \log \phi.$$

On top of the first derivatives L_a from §1.3, we need only the second derivatives L_{ab} of $\log \phi$. Differentiating the building blocks of §1.3 once more (writing δ for a perturbation),

$$\delta \mathbf{p} = it \delta \mathbf{q}_1, \quad \delta(\mathbf{M}^{-1}) = 2it \mathbf{M}^{-1} \delta \mathbf{Q}_2 \mathbf{M}^{-1}, \quad \delta \tilde{\boldsymbol{\mu}} = it \mathbf{M}^{-1} \delta \mathbf{q}_1 + 2it \mathbf{M}^{-1} \delta \mathbf{Q}_2 \tilde{\boldsymbol{\mu}}.$$

Since $L_{q_0} = it$ does not depend on any coefficient, every second derivative involving q_0 vanishes. Differentiating the other two,

$$L_{q_1 q_1} = (it)^2 \mathbf{M}^{-1}, \quad L_{q_1 \mathbf{Q}_2}[\cdot, \delta \mathbf{Q}_2] = 2(it)^2 \mathbf{M}^{-1} \delta \mathbf{Q}_2 \tilde{\boldsymbol{\mu}},$$

$$L_{\mathbf{Q}_2 \mathbf{Q}_2}[\cdot, \delta \mathbf{Q}_2] = 2(it)^2 (\mathbf{M}^{-1} \delta \mathbf{Q}_2 \mathbf{M}^{-1} + \mathbf{M}^{-1} \delta \mathbf{Q}_2 \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}' + \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}' \delta \mathbf{Q}_2 \mathbf{M}^{-1}).$$

We now assemble the Hessian block by block, $H_{ab} = T[(L_{ab} + L_a L_b) \phi]$.

The q_0 **row** mirrors the m -row of §1.2. Because $L_{q_0, b} = 0$, only the product term survives, $H_{q_0, b} = T[(it) L_b \phi]$, and by rule R2 the extra factor it is one more q_0 -derivative (again because q_0 shifts $q(\mathbf{x})$ rigidly). So each entry of this row is the q_0 -derivative of the corresponding gradient entry, obtained by inserting one more factor it with no new integral:

$$\boxed{H_{q_0 q_0} = f'(0), \quad H_{q_0, \mathbf{q}_1} = \partial_{q_0}(\partial_{\mathbf{q}_1} F), \quad H_{q_0, \mathbf{Q}_2} = \partial_{q_0}(\partial_{\mathbf{Q}_2} F).}$$

For the $\mathbf{q}_1$ - $\mathbf{q}_1$ **block**, the two contributions combine neatly: $L_{q_1 q_1} + L_{q_1} L'_{q_1} = (it)^2 \mathbf{M}^{-1} + (it)^2 \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}' = (it)^2 (\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}} \tilde{\boldsymbol{\mu}}')$, which is the $\mathbf{Q}_2$ -gradient weight times an extra factor it . Hence

$$H_{q_1 q_1} = T[(it)^2(\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}')\phi] = \partial_{q_0}(\partial_{\mathbf{Q}_2} F),$$

a clean identity: the $\mathbf{q}_1 \mathbf{q}_1$ block equals the q_0 -derivative of the $\mathbf{Q}_2$ gradient — that is, the $(q_0, \mathbf{Q}_2)$ entry of the row above.

The $\mathbf{q}_1$ – $\mathbf{Q}_2$ **block** is a vector for each perturbation $\delta \mathbf{Q}_2$:

$$H_{q_1, \mathbf{Q}_2}[\delta \mathbf{Q}_2] = T\left[\left(2(it)^2 \mathbf{M}^{-1} \delta \mathbf{Q}_2 \tilde{\boldsymbol{\mu}} + (it)^2 \tilde{\boldsymbol{\mu}} \operatorname{tr}((\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}')\delta \mathbf{Q}_2)\right)\phi\right].$$

And the $\mathbf{Q}_2$ – $\mathbf{Q}_2$ **block** is a bilinear form in two perturbations $\delta \mathbf{A}, \delta \mathbf{B}$:

$$H_{\mathbf{Q}_2 \mathbf{Q}_2}[\delta \mathbf{A}, \delta \mathbf{B}] = T\left[\left(2(it)^2 \operatorname{tr}(\delta \mathbf{A} (\mathbf{M}^{-1} \delta \mathbf{B} \mathbf{M}^{-1} + \mathbf{M}^{-1} \delta \mathbf{B} \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}' + \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}' \delta \mathbf{B} \mathbf{M}^{-1})) + (it)^2 \operatorname{tr}((\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}')\delta \mathbf{A}) \operatorname{tr}((\mathbf{M}^{-1} + \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}')\delta \mathbf{B})\right)\phi\right].$$

Every weight here is again built from $\mathbf{M}^{-1}$ and $\tilde{\boldsymbol{\mu}}$, so, exactly as in §1.3, a single Gil-Pelaez pass returns the full boundary Hessian, reusing the same eigen-structured $\mathbf{M}^{-1}$ with only a diagonal rescaling per node.

The obstruction at $s = 0$, and its cure. When $s \neq 0$ the normal term's Gaussian damping $e^{-s^2 t^2/2}$ makes all these integrals converge quickly, and the direct inversion above is the method of choice. At $s = 0$ — the classification regime, where a full-rank boundary forces the normal term to vanish (§1.5) — it fails: each Hessian block carries one factor it beyond the gradient, so, after the $1/t$ that T contributes, its integrand decays only like $t^{1-D/2}$, at best conditionally convergent for $D \leq 4$. Crucially this afflicts *every* weighted block, not just the threshold (q_0) directions, so the scalar series of §1.5 cannot simply be dropped in — the whole matrix- and tensor-valued weight must be reduced.

The cure is to perform the inversion *before* passing to the tail limit, by expanding the weights in the eigenbasis so that each block collapses to a finite combination of shifted-degree-of-freedom density derivatives — the objects §1.5 evaluates robustly at any D . Recall from §1.3 the eigendecomposition of the pencil $\boldsymbol{\Sigma}^{1/2} \mathbf{Q}_2 \boldsymbol{\Sigma}^{1/2} = \sum_j w_j \mathbf{v}_j \mathbf{v}_j'$, whose eigenvalues w_j are the generalized-chi-square weights. Writing $\mathbf{u}_j \equiv \boldsymbol{\Sigma}^{1/2} \mathbf{v}_j$ for the corresponding directions in the original coordinates and $g_j(t) \equiv (1 - 2it w_j)^{-1}$ for the resolvent factor of the j -th mode,

$$\mathbf{M}^{-1} = \sum_j g_j \mathbf{u}_j \mathbf{u}_j', \quad \tilde{\boldsymbol{\mu}} = \sum_j g_j c_j \mathbf{u}_j, \quad c_j \equiv \mathbf{u}_j' \mathbf{p} = \alpha_j + it \beta_j,$$

where $\alpha_j \equiv \mathbf{u}_j' \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}$ and $\beta_j \equiv \mathbf{u}_j' \mathbf{q}_1$ are real constants. Every weight in §1.4 is a polynomial in these two objects, hence, once expanded, a sum of terms of the single form $(it)^p g_{j_1} \cdots g_{j_r} \phi$. Two elementary facts collapse each such term. First, by rule R1 each factor g_j advances the j -th component's degrees of freedom by two, $g_j \phi = \phi_{[k_j+2]}$, so a product advances several components at once (a repeated index compounding). Second, by rule R2 a power $(it)^p$ is p argument-derivatives, $T[(it)^p \psi] = (-\partial_x)^p T[\psi]$. Together they give the master reduction

$$T[(it)^p g_{j_1} \cdots g_{j_r} \phi] = (-1)^p f_{[k_{j_1}+2, \dots, k_{j_r}+2]}^{(p-1)}(x_0),$$

a density derivative of order $p - 1$, evaluated at the parameter set with each listed component's degrees of freedom raised by two. Substituting the eigen-expansions into a block weight and collecting powers of it — from the explicit $(it)^2$ and from the linear-in- t factors c_j — thus turns every block into a finite sum of these shifted-dof density derivatives. For the $\mathbf{q}_1 \mathbf{q}_1$ block,

$$H_{\mathbf{q}_1 \mathbf{q}_1} = \sum_j \mathbf{u}_j \mathbf{u}'_j f'_{[k_j+2]} + \sum_{j,l} \mathbf{u}_j \mathbf{u}'_l \left(\alpha_j \alpha_l f' - (\alpha_j \beta_l + \alpha_l \beta_j) f'' + \beta_j \beta_l f''' \right)_{[k_j+2, k_l+2]},$$

and the remaining blocks follow by the same substitution, the tensor blocks $H_{\mathbf{q}_1 \mathbf{Q}_2}$ and $H_{\mathbf{Q}_2 \mathbf{Q}_2}$ merely carrying more eigen-indices and, through their extra c_j factors, higher derivative orders — up to $f^{(4)}$ and $f^{(5)}$ respectively. This reduction is *algebra*, not a convergence fix: the worst term of each block is a shifted density whose own inversion integrand still decays like $t^{1-D/2}$ — the raised degrees of freedom exactly cancelled by the raised derivative order — so it too must come from the series of §1.5, evaluated through the mixed-sign convolution (a two-class boundary has weights of both signs), not from inversion. This is also the " ω^2 term diverges" of the background note:³ what reads as a divergence is a density derivative in disguise, recovered exactly here rather than integrated.

1.5 Robust density derivatives when $s = 0$

Both Hessians rest on the same three x -derivatives of the density, f' , f'' , f''' . In each parametrization one parameter is a pure threshold shift — m natively, q_0 for the boundary — so its second derivative is just the curvature of F at the threshold, which rule R2 delivers as f' ; a factor of s^2 raises this by two orders to f''' where it occurs. Concretely, $H_{mm} = f'$ and $H_{ss} = f' + s^2 f'''$ (§1.2), and the entire boundary q_0 -row, $H_{q_0 q_0} = f'$ together with H_{q_0, q_1} and $H_{q_0, \mathbf{Q}_2}$ (§1.4), are built from these.

Rule R2 obtains $f^{(n)}$ from an x -weighted Gil-Pelaez integral, so its convergence is governed by the tail of the integrand. Writing $u = 2t$ and $D = \sum_j k_j$ for the total degrees of freedom, that integrand behaves like $u^{n-D/2}$ as $u \rightarrow \infty$. A non-zero normal term contributes a Gaussian damping factor $\exp(-\frac{1}{2}s^2 t^2)$ that makes every order converge; but at $s = 0$ this damping is absent, and the integral is only *conditionally* convergent once $n \geq D/2 - \frac{1}{2}$, and *divergent* once $n \geq D/2$. At $D = 4$, for instance, f' decays only as u^{-1} and converges conditionally; at $D = 2$ it does not decay at all, and the inversion integral breaks down numerically.

The boundary Hessian lives squarely in this regime. A full-rank quadratic boundary has $s = 0$ exactly, since no flat direction remains to carry a linear normal term, and its total degrees of freedom equal the ambient dimension, $D = \text{rank}(\mathbf{Q}_2) = d$; so a classification problem in $d = 2, 3, 4$ sits right on the divergent and borderline cases. Moreover its weights are generically of mixed sign, because a two-class boundary has $\mathbf{Q}_2 = \frac{1}{2}(\Sigma_1^{-1} - \Sigma_0^{-1})$. The regime we must therefore handle robustly is $s = 0$, small D , and mixed-sign weights.

The way out is to stop inverting the characteristic function for these derivatives, and to use instead a representation that converges term by term. At $s = 0$, the variable is a pure weighted sum of non-central chi-squares, and we first split it by the sign of its weights into a positive part and a (negated) negative part, $q = q_+ - q_-$; each part alone is then a weighted sum with weights of a single sign.

When the weights of a generalized chi-square are all of one sign — so that the underlying quadratic form is an ellipsoid — its density admits a convergent series in chi-square densities of increasing degrees of freedom, due to Ruben.² Introducing a positive scale β and writing $M = \sum_j k_j$ for the total degrees of freedom, the series expresses the cdf and pdf as mixtures of central chi-squares,

$$F(x) = \sum_{r \geq 0} a_r G_{M+2r}(y), \quad f(x) = \frac{1}{\beta} \sum_{r \geq 0} a_r g_{M+2r}(y), \quad y = \frac{x - m}{\beta},$$

where G_ν and g_ν are the central χ_ν^2 cdf and density, and the non-negative coefficients a_r (with $\sum_r a_r = 1$) are fixed by Ruben's recursion. The scale β is arbitrary in principle, but a *genuine* mixture — one with non-negative coefficients — is

guaranteed only for $0 < \beta \leq \min_j |w_j|$, the smallest weight; we take β to be a fixed fraction of $\min_j |w_j|$, placed just inside this bound. This applies to q_+ and to q_- individually, since each is now a same-sign generalized chi-square.

The point of this form is that its x -derivatives are *closed form*, because differentiating a central chi-square density in its argument only shifts its degrees of freedom. With the shift operator S defined by $(Sg)_\nu \equiv g_{\nu-2}$,

$$\frac{d}{dy} g_\nu = \frac{1}{2}(g_{\nu-2} - g_\nu) = \frac{1}{2}(S - I) g_\nu,$$

so that, iterating,

$$\frac{d^n}{dy^n} g_\nu = 2^{-n}(S - I)^n g_\nu = 2^{-n} \sum_{i=0}^n \binom{n}{i} (-1)^{n-i} g_{\nu-2i}.$$

The first identity is the elementary relation $g'_\nu(y) = g_\nu(y) [(\nu - 2)/(2y) - \frac{1}{2}]$, rewritten as a shift. Differentiating the series n times in x , and collecting the factor $1/\beta$ that each derivative picks up from $y = (x - m)/\beta$, we obtain

$$f^{(n)}(x) = \beta^{-(n+1)} \sum_{r \geq 0} a_r 2^{-n} \sum_{i=0}^n \binom{n}{i} (-1)^{n-i} g_{M+2r-2i}(y), \quad y = \frac{x-m}{\beta}.$$

This is the same series, with the same coefficients a_r , only shifted down in degrees of freedom by up to $2n$. There is no new series to build and no integral to perform, and it converges for every D . (For the few low-index terms with $M + 2r - 2i \leq 0$, one uses the elementary form of $g_\nu^{(n)}$ directly; only the smallest r are affected, and $M \geq 1$.)

When the original weights were already of one sign, this is the whole answer. Otherwise we must assemble the density from the two parts. Since q_+ and q_- are independent and $q = q_+ - q_-$, the density of q is the cross-correlation of their densities; carrying out the difference-integral in one variable or the other places the argument x on either factor at will,

$$f(x) = \int_0^\infty f_{q_+}(x+v) f_{q_-}(v) dv = \int_0^\infty f_{q_+}(u) f_{q_-}(u-x) du.$$

This is an ordinary, non-oscillatory one-dimensional quadrature — it never meets the oscillatory algebraic tail that defeated the Gil-Pelaez integral — and it returns the density itself cleanly. (A residual normal term $s \neq 0$ would add one further convolution, with $\mathcal{N}(0, s^2)$; but this route is used only at $s = 0$, where that factor is absent.)

Its x -derivatives need one more idea, because a naive differentiation under the integral sign fails in exactly the regime we care about. That x may ride on either factor means a derivative of the convolution may be carried by *either* factor too, so we may write $f^{(n)}$ with the derivatives on f_{q_+} or, equally, on f_{q_-} :

$$f^{(n)}(x) = \int_0^\infty f_{q_+}^{(n)}(x+v) f_{q_-}(v) dv = (-1)^n \int_0^\infty f_{q_+}(x+v) f_{q_-}^{(n)}(v) dv.$$

The two are equal as identities, but not equally usable. Each is a convolution of two densities — usually a harmless, smoothing operation — and the catch is that a chi-square density can carry a spike that convolving cannot smooth away. A same-sign generalized chi-square lives on $[m, \infty)$, its smallest value being the offset m (reached when every chi-square in it is zero). At that lower edge the density can blow up: for a single degree of freedom it behaves like $(y - m)^{-1/2}$. On

its own this spike is mild enough to integrate, but each x -derivative makes it sharper — like $(y - m)^{M/2-1-n}$ after n derivatives, where M is the degrees of freedom of that part — until it is too sharp to integrate at all, once $n \geq M/2$.

This is why it matters which factor carries the derivatives. In the first form the sharpened spike of $f_{q_+}^{(n)}$ sits where $x + v = m$, i.e. at $v = m - x$. If the threshold is below the positive part's floor, $x < m$, that spike lands in the middle of the integration range, where f_{q_-} is nonzero, so the convolution runs straight over it and no longer converges — even though the true derivative $f^{(n)}(x)$ is perfectly finite. (The plain density, $n = 0$, has only the mild edge and convolves without trouble; only its derivatives fail.) This is not an exotic corner. A full-rank two-class boundary has $\mathbf{Q}_2 = \frac{1}{2}(\mathbf{\Sigma}_1^{-1} - \mathbf{\Sigma}_0^{-1})$, generically indefinite with a low-dof positive part, and its floor m ordinarily lies well above the decision threshold $x = 0$; so this is the *typical* boundary, not a rare one, which is why it slipped past a validation that happened to test only $m < x$.

The second form cures it. There the differentiated — hence singular — factor is $f_{q_-}^{(n)}$, whose edge is at $v = 0$; but its companion $f_{q_+}(x + v)$ vanishes for $x + v < m$, i.e. for $v < m - x$, so the integrand is supported on $v \geq m - x > 0$ and $f_{q_-}^{(n)}$ is sampled only in the smooth interior of q_- . The one remaining singularity is the integrable $(v - (m - x))^{M/2-1}$ edge of the *undifferentiated* f_{q_+} — the same mild edge already present in the density itself. When instead $x \geq m$ the first form is the healthy one: its edge $v = m - x \leq 0$ is out of range and $f_{q_+}^{(n)}$ is sampled strictly inside q_+ . The prescription is therefore

$$f^{(n)}(x) = \begin{cases} (-1)^n \int_0^\infty f_{q_+}(x + v) f_{q_-}^{(n)}(v) dv, & x < m, \\ \int_0^\infty f_{q_+}^{(n)}(x + v) f_{q_-}(v) dv, & x \geq m : \end{cases}$$

we carry the derivatives on whichever part is being sampled away from its own support edge — on q_- below the floor, on q_+ above it. The two branches agree wherever both converge; they part only at $x = m$, where the two floors coincide and the shielding fails for both branches at once, since each factor's own edge now sits exactly at the other factor's range boundary.

At the minimum degrees of freedom, $k_j = 1$ for the part whose floor is being probed, this is not merely a derivative singularity but a divergence of the density itself: $f_q(x) \rightarrow \infty$ logarithmically as $x \rightarrow m$. (Confirmed by evaluating the same convolution with a small artificial normal term $s \neq 0$ — whose Gaussian damping restores the Gil-Pelaez inversion's convergence — at a shrinking sequence of such s : the value grows by an unshrinking, constant increment per decade of s , the signature of a genuine log divergence rather than a slow-converging limit; the same probe converges cleanly for $k_j \geq 2$.) This is a reachable corner, not a formal one: a quadratic classification boundary built from exact antipodal symmetry between two classes (equal-and-opposite means, swapped covariance eigenvalues) places its decision threshold exactly at $x_0 = m$, at $k = 1$. `cdf_grad_norm_quad` detects this coincidence directly and reports the honest value — a correctly-signed $\pm\infty$ for a divergent entry, decided by the same shrinking- s probe used above — rather than the arbitrary finite number a naive evaluation of the singular integral would return.

This series route replaces the density x -derivatives f', f'', f''' only in the problematic regime of $s = 0$ with small D . Everywhere else — whenever $s \neq 0$, or $s = 0$ with D large enough that the x -weighted integrand converges comfortably — the Gil-Pelaez inversion is both accurate and cheaper, and we keep it. In practice a single density-derivative routine chooses between the two methods from (s, D, n) , in the same spirit as the method selection the cdf and pdf routines already perform. This routine is implemented (§2.1), and the native Hessian's m, s blocks and their shifted-dof densities (§1.2) call it.

One thing this scalar route does *not* cover on its own: the Hessian entries that also differentiate a degree of freedom (§1.2, e.g. $H_{w_j k_j}$) involve $\partial_x \partial_{k_j} F$, an object Ruben's series does not produce, so at $s = 0$ with small D these keep the inversion

route. The boundary Hessian (§1.4) needs no separate treatment here: its blocks reduce, as shown there, to sums of shifted-dof density derivatives — exactly the objects this routine evaluates.

2 Code implementations: benchmarks against finite-difference

2.1 Gradient and Hessian with respect to the $\tilde{\chi}$ parameters

`cdf_grad_gx2` was benchmarked against finite differencing of `gx2cdf` on three distributions (same/mixed-sign weights, $s \neq 0/s = 0$; ground truth a near-exact analytic evaluation — `vpa` at $s \neq 0$, tight double precision at $s = 0$ — and FD swept over step size and matched to its own best-achievable error). Python results are from the `gx2-py` port (`cdf_grad_gx2`), run under the identical protocol, same parameters, same cases:

case	w	k	λ	s	m	x	P
1	[1 -2]	[2 3]	[1 0.5]	1.5	1	2	8
2	[1 -2 3]	[1 2 1]	[0.5 1 0.5]	1	2	5	11
3	[1 -1.5]	[1 1]	[0.5 0.3]	0	0	0.5	8

Table 2.1.1. Case parameters. Case 3 is the hard corner: $s = 0$, small total dof ($D = 2$), mixed-sign weights.

`cdf_grad_gx2` :

case	quantity	analytic time		rel. speed vs. FD		analytic error		rel. acc. vs FD	
		python	matlab	python	matlab	python	matlab	python	matlab
1 ($s \neq 0$)	gradient	54 ms	11 ms	0.91	1.0	6e-17	3e-17	~2e5	~2e5
1 ($s \neq 0$)	Hessian	0.29 s	61 ms	1.4	1.4	7e-14	1e-16	~1e5	~2e8
2 ($s \neq 0$)	gradient	0.11 s	17 ms	0.79	0.88	8e-17	6e-17	~8e4	~1e5
2 ($s \neq 0$)	Hessian	0.88 s	0.15 s	1.1	1.1	5e-13	1e-15	~2e3	~9e6
3 ($s=0$)†‡	gradient	56 s	1.1 s	1.7§	0.47	1e-9§	2e-6	~6.3e3§	~5.7
3 ($s=0$)†‡	Hessian	324 s	2.2 s	2.5§	1.3	2e-9§	1e-4	~9.3e4§	~12

Table 2.1.2. Benchmark results. †Excludes the degree-of-freedom-derivative entries, unreliable at $s = 0$ small dof for both methods and tracked as open item 1 in §3. ‡Case 3 in Python departs from the matched-accuracy protocol because a single evaluation there already costs seconds; see open item 6 in §3. §Filled in by a dedicated, deliberately scaled-down benchmark (single FD step, a moderately- rather than near-exact-tight ground truth, and a sampled rather than exhaustive Hessian); see the table just below for the raw numbers and the departures from the standard protocol.

Filling in the python case-3 cells. The four cells above left blank in the original benchmark needed a dedicated follow-up run, since a single analytic evaluation in this corner already costs tens of seconds to minutes. That run departs from the matched-accuracy protocol in three ways, each purely for cost: the ground truth is a moderately tight double-precision analytic evaluation (`AbsTol=1e-11`, `RelTol=1e-8`) rather than the near-exact one used elsewhere in this table (tightening further was prohibitively slow here), so the python analytic-error/rel.-acc. figures above are not quite on the same absolute footing as the near-exact-ground-truth figures used for cases 1–2 and for MATLAB, though the qualitative conclusion — analytic far more accurate than FD in this corner — is unaffected; FD uses a single central-difference step ($h=0.01$ in each parameter's natural units) rather than a swept-and-matched one; and the FD Hessian was evaluated at

only one representative index pair per parameter-block type (14 of the 36 unique entries of the symmetric 8×8 matrix, after excluding the k -derivative blocks per the $\dagger$ convention above — this exclusion was reconfirmed directly here, since the raw, unexcluded FD-vs-analytic gap on $H_{k_1, m}$ was ~ 0.6 , wrong by an order of magnitude, exactly the known small-dof failure mode of open item 3.1) rather than the full 36. The Hessian's reported rel.-speed is extrapolated from that sample to the full matrix, since FD cost scales with the number of entries evaluated (see below); the reported analytic-error and rel.-acc. do not need this extrapolation, since they reflect the accuracy of each distinct Hessian-block formula, which the one-pair-per-type sample already exercises.

quantity	python analytic time	python FD time	ground truth tolerance	python analytic error $\dagger$	python FD error $\dagger$
gradient	56.5 s	98.7 s (all 8 params)	AbsTol=1e-11, RelTol=1e-8	9.6e-10	6.1e-6
Hessian	324.1 s	340.6 s (14 of 36 entries sampled); ~ 794 s (extrapolated to the full matrix)	same	2.1e-9	1.9e-4

Table 2.1.3. Case-3 fill-in benchmark. $\dagger$ Excludes the degree-of-freedom-derivative entries (as in the main table above).

2.2 Gradient and Hessian with respect to the boundary coefficients

2.2.1 wider benchmark of the python `cdf_grad_norm_quad`

The cases above are made-up boundaries. We now run the python package on full classification problems, where we specify two Gaussians, and using a couple of additional functions, we compute the optimal boundary, and then the Hessian of the error wrt this boundary. (Only the Hessian, because the gradient is zero.) We compute a tight-tolerance analytic Hessian as the ground truth. Then we compute the default-tolerance analytic Hessian, and a `numdifftools`-based finite-difference Hessian, to compare against this ground-truth.

#	d	P	description	bd. type	quantity	FD time	an. time	FD err.	an. err.	gain
1	1	3	generic	elliptic	gradient	4.5s	4.4s	4.2e-13	5.6e-17	7.7e3
					Hessian	4.2s	3.4s	2.2e-4	0	∞
2	1	3	generic	elliptic	gradient	4.2s	4.4s	2.1e-12	7.7e-16	2.6e3
					Hessian	3.9s	3.3s	1.6e-3	0	∞
3	1	3	near-linear	elliptic	gradient	3.1s	4.8s	6.3e-13	2.2e-14	1.9e1
					Hessian	4.3s	10.8s	2.6e-3	1.5e-13	6.9e9
4	2	6	same covariance	linear	gradient	7.2m	1.4s	7.5e-14	6.1e-16	3.7e4
					Hessian	2.0h	3.0s	0.3	1.4e-16	5.6e18
5	2	6	same-sign contrast	elliptic	gradient	5.8m	1.3s	8.3e-12	1.2e-16	1.8e7
					Hessian	1.5h	3.4s	5.0e-4	0	∞
6	2	6	unequal priors	elliptic	gradient	5.5m	1.4s	2.9e-12	9.7e-17	7.0e6
					Hessian	1.6h	3.1s	5.7e-5	0	∞
7	2	6	generic, axis-aligned	hyperbolic	gradient	6.8m	2.1s	4.8e-12	9.7e-16	9.7e5
					Hessian	6.3h	14.1m	0.7	2.4e-8	8.0e8
8	2	6	generic, rotated	hyperbolic	gradient	4.1m	2.1s	2.4e-11	1.3e-15	2.1e6

#	d	P	description	bd. type	quantity	FD time	an. time	FD err.	an. err.	gain
					Hessian	6.4h	13.9m	0.4	7.0e-9	1.5e9
9	2	6	generic, crossed	crossing lines	gradient	6.6m	2.0s	3.9e-13	1.8e-16	4.3e5
					Hessian	6.2h	53.8s	∞^*	0*	∞
10	2	6	near-linear	hyperbolic	gradient	8.7m	1.9s	3.6e-11	1.7e-16	5.9e7
					Hessian	2.8h	12.6s	1.5	6.0e-9	2.0e11
11	2	6	same mean ($q_1 = 0$)‡	hyperbolic	gradient					
					Hessian	6.5h	6.0m	4.4e-2	4.8e-10	5.9e9
12	2	6	rank-deficient Q_2	parabolic	gradient	2.3s	1.7s	1.1e-13	2.4e-16	6.1e2
					Hessian	2.5h	3.2s	0.4	2.4e-15	4.3e17
13	3	10	same covariance	linear	gradient	4.4s	2.0s	7.8e-15	1.8e-17	9.5e2
					Hessian	44.5m	3.1s	3.5e-2	9.4e-16	3.2e16
14	3	10	same-sign contrast	elliptic	gradient	4.3s	1.3s	3.8e-12	2.9e-16	4.5e4
					Hessian	1.5h	3.6s	1.6e-4	0	∞
15	3	10	generic	hyperbolic	gradient	4.3s	1.9s	6.2e-12	1.6e-15	8.8e3
					Hessian	2.9h	15.1m	0.1	2.2e-9	7.7e8
16	3	10	generic, crossed	hyperbolic	gradient	7.5m	1.9s	1.4e-11	9.4e-16	3.4e6
					Hessian	3.1h	10.6m	2.4e-3	5.1e-10	8.3e7
17	3	10	near-linear	hyperbolic	gradient	8.3s	1.5s	2.1e-11	2.2e-16	5.4e5
					Hessian	1.3h	25.9s	1.7	1.6e-8	1.9e10
18	3	10	same mean ($q_1 = 0$)‡	hyperbolic	gradient					
					Hessian	6.6h	4.3m	0.8	2.3e-9	3.0e10
19	3	10	rank-deficient Q_2	parabolic	gradient	4.4s	1.6s	1.5e-13	2.4e-16	1.8e3
					Hessian	1.1h	3.0s	0.2	3.9e-16	7.7e17
20	4	15	same-sign, generic	elliptic	gradient	10.2s	3.4s	5.6e-12	1.1e-16	1.6e5
					Hessian	22.3m	3.8s	6.2e-4	0	∞
21	4	15	stress test	hyperbolic	gradient	7.0s	1.6s	3.2e-12	5.3e-16	2.6e4
					Hessian	37.6m	1.0h	0.06	9.4e-8	3.6e5
22	5	21	same-sign, generic	elliptic	gradient	15.1s	7.0s	4.1e-12	2.7e-16	3.3e4
					Hessian	13.2m	6.7s	9.3e-5	0	∞
23	5	21	stress test	hyperbolic	gradient	9.1s	1.4s	7.5e-12	5.4e-16	9.1e4
					Hessian	8.8m	23.8m	0.2	2.5e-8	2.5e6

time:	1.3s		6.6h
-------	------	--	------

error:	1.8e-17		1.7
--------	---------	--	-----

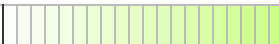
gain:	1e-18.7		1		1e18.7
-------	---------	---	---	--	--------

Table 2.2.1.1. Benchmark results, each problem split into a **gradient** row (first) and a **Hessian** row (second), reporting only the latest run of `bench_norm_err_bd.py` for every cell (a superseded, timeout-affected run preceded this one; its numbers are no longer shown). The gradient numbers come from a second, deliberately non-optimal boundary — the Fisher (pooled-covariance LDA) boundary — evaluated by `benchmarks/bench_norm_err_bd.py`'s `fisher_*` stages, since the gradient at the *optimal* boundary is ~ 0 by construction and so uninformative as an accuracy check. All color scales below run green–white–red (never through yellow/orange), so a cell's color is read as a blend of only two hues plus white, with exact white marking the scale's own midpoint. The **gain** column lists the overall gain (FD time / analytic time) $\times$ (FD error / analytic error) and is colored on a log scale symmetric about a gain of 1 (green above 1, white at exactly 1, red below), spanning $\pm$ the largest log-distance any row's gain reaches from 1 in either direction — since every computable gain here is well above 1, the whole column sits on the green half of that scale, with the least lopsided cases (still large wins, just less enormous ones) reading closest to white. An infinite gain (analytic error exactly 0, including problem 9's Hessian row — see * below) is shown as ∞ , colored at the greenest extreme; only the two blank same-mean gradient rows are left blank and uncolored. (The **quantity** cell itself — "gradient"/"Hessian" — carries no color.) The four time/error columns are each colored on their own uniform log scale from green (shortest time / smallest error) through white (the scale's midpoint) to red (longest time / largest error) — one shared scale across FD time and an. time, and a separate shared scale across FD err. and an. err., so cells are only comparable within their own color family (time vs. error), not across it. An exact-zero error is clipped to the greenest end of its scale and an infinite error to the reddest end. *Problem 9's boundary Hessian is genuinely infinite here — a density-cusp coincidence (see §1.5 and the cusp handling in `cdf_grad_norm_quad`). The tight-tolerance and default-tolerance analytic Hessians agree exactly (both $+\infty$ everywhere), so the analytic error is 0; the FD Hessian is finite by construction, so its error against the true divergence is ∞ . ‡Problems 11 and 18 have identical class means ($\mu_0 = \mu_1$), so their Fisher boundary is exactly degenerate ($q_2 = q_1 = \mathbf{0}$, i.e. no boundary at all under equal priors) — `gx2.norm_err` errors out on the resulting empty weight vector, so the gradient benchmark isn't computable for these two rows.

#	d	note	μ_0	Σ_0	μ_1	Σ_1
1	1	generic	0	1	3	4
2	1	generic	0	1	1.5	0.4
3	1	near-linear	0	1	2	1.05
4	2	same covariance	[-.3 -.3]	I	[.3 .3]	I
5	2	same-sign contrast	[-.3 -.3]	I	[.3 .3]	diag(.4 .6)
6	2	unequal priors	[-.3 -.3]	I	[.3 .3]	diag(.4 .6)
7	2	generic, axis-aligned	[-.3 -.3]	I	[.3 .3]	diag(3 .5)
8	2	generic, rotated	[-.3 -.3]	I	[.3 .3]	[[2 .8],[.8 1]]
9	2	generic, crossed	[.3 -.3]	diag(.5 2)	[-.3 .3]	diag(2 .5)
10	2	near-linear	[0 0]	[[1 .2],[.2 1]]	[.3 .2]	[[1.05 .2],[.2 .97]]
11	2	same mean ($q_1 = \mathbf{0}$)	[0 0]	I	[0 0]	diag(3 .5)
12	2	rank-deficient Q_2	[-.3 0]	I	[.3 0]	diag(1 3)
13	3	same covariance	[-.3 -.3 -.3]	I	[.3 .3 .3]	I
14	3	same-sign contrast	[-.3 -.3 -.3]	I	[.3 .3 .3]	diag(.4 .6 .5)
15	3	generic	[-.3 -.3 -.3]	I	[.3 .3 .3]	diag(2 .5 3)
16	3	generic, crossed	[.3 0 -.3]	diag(.5 2 .7)	[-.3 0 .3]	diag(2 .5 1.4)
17	3	near-linear	[0 0 0]	I	[.3 .2 0]	[[1.05 .03 0],[.03 1.02 .02],[0 .02 1.04]]

#	d	note	μ_0	Σ_0	μ_1	Σ_1
18	3	same mean ($q_1 = \mathbf{0}$)	[0 0 0]	$\mathbf{I}$	[0 0 0]	diag(2 .5 3)
19	3	rank-deficient $\mathbf{Q}_2$	[-.3 0 0]	$\mathbf{I}$	[.3 0 0]	diag(1 3 .5)
20	4	same-sign, generic	[-.3]^4	$\mathbf{I}$	[.3]^4	diag(.4 .5 .6 .7)
21	4	stress test	[-.3]^4	$\mathbf{I}$	[.3]^4	diag(2 .5 3 1.5)
22	5	same-sign, generic	[-.3]^5	$\mathbf{I}$	[.3]^5	diag(.4 .5 .6 .7 .5)
23	5	stress test	[-.3]^5	$\mathbf{I}$	[.3]^5	diag(2 .5 3 1.5 .7)

Table 2.2.1.2. Problem parameters.

3 Ongoing work and open items

- ☐ **3.1. Robust analytic k -derivatives at $s = 0$, small dof.** The native Hessian entries that differentiate a degree of freedom against the argument — $H_{w_j k_j}, H_{\lambda_j k_j}, H_{k_i k_j}, H_{s k_j}$, built from $\partial_x \partial_{k_j} F$ and $\partial_{k_i} \partial_{k_j} F$ — remain on the Imhof inversion (`gx2_imhof` `'k_deriv' / 'kk_deriv'`), because Ruben's series (§1.5) produces density derivatives $f^{(n)}$ but **not** derivatives in the degrees of freedom. They are accurate at $s \neq 0$ (Gaussian damping) or $s = 0$ with adequate dof, and lose accuracy only in the $s = 0$, small-dof corner. A robust fix needs new math — a series representation differentiable in k — which is not derived here. This affects only `gx2`'s own native-parameter Hessian; the boundary-coefficient derivatives (§1.3–§1.4) never differentiate in k , so the quadratic-classification use case is unaffected.
- ☐ **3.2. Faster mixed-sign density derivatives at $s = 0$.** The same-sign speedup already in place (prefer the Ruben series over the slow inversion) made same-sign $s = 0$ boundary derivatives competitive with finite differences. Mixed-sign weights still take the convolution route in `gx2_dens_deriv` (a cross-correlation of two Ruben series), which is slower; a mixed-sign boundary Hessian at $s = 0$ therefore remains in the seconds. Worth investigating whether the convolution can be accelerated (precomputed series coefficients, a coarser adaptive tolerance, or batching the derivative orders into one quadrature pass) to bring mixed-sign $s = 0$ down to the same-sign level.
- **3.4. Boundary optimization in `IntClassNorm`**
 - ☐ **3.4.1. Excess-risk from a mis-estimated boundary (delta method).** If a boundary is fit from finite data with coefficient covariance Σ_{bd} , the expected error above Bayes is second order in the deviation because the gradient vanishes there: $\mathbb{E}[p_e] - p_e^* \approx \frac{1}{2} \text{tr}(\mathbf{H} \Sigma_{\text{bd}})$, with $\mathbf{H}$ read off `norm_err_bd_hess`. This turns the empirical normal-approximation checks already in the toolbox (sample-boundary sweeps, `samp_opt_bd` vs. the true boundary) into a closed-form prediction instead of a Monte Carlo sweep, and is the direct match to the excess-classification-error question in Josiah Couch's background note (ref. 3).
 - ☐ **3.4.2. Analytic ROC slope.** The classic identity that the ROC slope at a criterion equals the likelihood ratio there follows immediately from the `q0` component of the gradient ($\partial p_h / \partial q_0, \partial p_f / \partial q_0$), giving the ROC curve's local slope at any point without sweeping the criterion and re-classifying.
 - ☐ **3.4.3. Sensitivity directions of the fitted boundary.**
Say we fit a classification boundary to data. There is some 'wobble room' around this boundary, in the sense that it can be morphed in some limited ways without losing accuracy much. The Hessian of the classification error at the optimum, `norm_err_bd_hess`, captures exactly this. Its eigendecomposition $\mathbf{H} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}'$ yields orthogonal directions $\mathbf{v}_k$ in the space of boundary coefficients, each with its own error curvature λ_k . A direction with large λ_k is *stiff*: moving the boundary even slightly along it raises the error quickly, so that combination of coefficients must be known precisely. A direction with small λ_k is *sloppy*: the error is nearly flat along it, so large uncertainty there costs almost nothing. This distinction is useful when we need to know how precisely the different aspects of a fitted boundary need to be nailed down. This stiff/sloppy split — together with the fact that the eigenvalues typically span many orders of magnitude rather than clustering around one scale — is exactly the phenomenon

that the model-fitting literature calls "sloppiness" (Gutenkunst et al., 2007, ref. 4; see also the review by Transtrum et al., 2015, ref. 5). The same Hessian eigenspectrum recurs whenever a model with several free parameters is fit to data, so it's worth reusing that framing here rather than re-deriving it.

Expand the error to second order around the fitted coefficients $\mathbf{c}^*$: since the error gradient vanishes there, $e(\mathbf{c}^* + \delta\mathbf{c}) \approx e^* + \frac{1}{2} \delta\mathbf{c}' \mathbf{H} \delta\mathbf{c}$ for a coefficient shift $\delta\mathbf{c}$. If we now ask which shifts keep the error increase within some tolerance Δe (say 1%), the answer is the set of $\delta\mathbf{c}$ satisfying $\frac{1}{2} \delta\mathbf{c}' \mathbf{H} \delta\mathbf{c} \leq \Delta e$. When $\mathbf{H}$ is positive definite, this sublevel set is a filled ellipsoid in $\delta\mathbf{c}$ -space — the *tolerance ellipsoid* — and its boundary $\frac{1}{2} \delta\mathbf{c}' \mathbf{H} \delta\mathbf{c} = \Delta e$ is the set of largest coefficient shifts allowed at that tolerance. Writing $\delta\mathbf{c}$ in the eigenbasis of $\mathbf{H}$ diagonalizes this quadratic form, which is exactly why the ellipsoid's axes line up with the eigenvectors $\mathbf{v}_k$: along $\mathbf{v}_k$ the constraint becomes $\frac{1}{2} \lambda_k \delta_k^2 = \Delta e$, i.e. $\delta_k = \sqrt{2\Delta e / \lambda_k}$. A stiff direction (large λ_k) gets a short semi-axis, since even a small shift there already spends the whole error budget; a sloppy direction (small λ_k) gets a long one. Plotting the boundary shifted by $\pm \delta_k \mathbf{v}_k$ for the largest and smallest λ_k shows the true stiffest and sloppiest boundary families.

Two implementation subtleties are worth flagging, both confirmed directly against a finite-difference Hessian while building this into `test.m`.

First, $\mathbf{c}$ should collect only $\mathbf{Q}_2$'s *symmetric* free entries, not all D^2 raw matrix entries. The boundary function $g(\mathbf{x}) = \mathbf{x}' \mathbf{Q}_2 \mathbf{x} + \mathbf{q}_1' \mathbf{x} + q_0$ depends on an off-diagonal pair $(\mathbf{Q}_2)_{ab}, (\mathbf{Q}_2)_{ba}$ only through their sum, so treating them as two independent raw coordinates double-counts a direction that has no effect on g at all — and `norm_err_bd_hess`'s `q2q2 / q1q2 / q0q2` blocks are built for exactly the reduced (symmetric) parameterization: a block whose index touches an off-diagonal entry of $\mathbf{Q}_2$ is the *sum* of that entry's two raw orderings, not either one alone (the two orderings are generally unequal individually, only their sum is meaningful). Using the raw, doubled coordinates instead produces a Hessian with a spurious extra near-zero — or even numerically negative — eigenvalue that reflects no real property of $e(\mathbf{c})$.

Second, even in the reduced parameterization $\mathbf{H}$ is only positive *semi*-definite, not definite: an overall positive rescaling $\mathbf{c} \rightarrow k\mathbf{c}$ ($k > 0$) leaves g 's zero-level-set — and so the error — exactly unchanged, which by Euler's identity forces $\mathbf{H}\mathbf{c}^* = \mathbf{0}$. So the true tolerance region is not a bounded ellipsoid but an elliptic *cylinder*: bounded in the directions with genuine curvature, unbounded along $\mathbf{c}^*$ itself. This costs nothing in practice — motion along that direction doesn't change the boundary at all, so it contributes no visually distinct family member — but the null direction (found by eigenvalue-thresholding $\mathbf{H}$) should be dropped before inverting, rather than treated as merely "very sloppy."

One thing to watch: the eigenvalues λ_k are only comparable to each other if the coefficients they multiply are measured in comparable units, and here they aren't. The boundary function is $g(\mathbf{x}) = \mathbf{x}' \mathbf{Q}_2 \mathbf{x} + \mathbf{q}_1' \mathbf{x} + q_0$, so a one-unit change in an entry of $\mathbf{Q}_2$ moves g by an amount that grows with $\|\mathbf{x}\|^2$, a one-unit change in $\mathbf{q}_1$ moves it by an amount that grows with $\|\mathbf{x}\|$, and a one-unit change in q_0 moves it by a fixed amount regardless of $\mathbf{x}$. Comparing raw curvatures across these three blocks would mostly be comparing unit choices, not real stiffness. The fix is to rescale coefficients before eigendecomposing so that a unit change means a comparable change in g everywhere: substitute $\delta\mathbf{c} = \mathbf{D} \delta\tilde{\mathbf{c}}$ for a diagonal matrix $\mathbf{D}$ and eigendecompose $\mathbf{D}' \mathbf{H} \mathbf{D}$ instead. A natural choice for $\mathbf{D}$ comes from a characteristic length scale ℓ of the data itself — e.g. $\ell = \sqrt{\text{tr } \Sigma / D}$ from the normal's own covariance — matched to each block's power of $\mathbf{x}$ in g : scale $\mathbf{Q}_2$'s entries by ℓ^2 , $\mathbf{q}_1$'s by ℓ , and leave q_0 unscaled. (Scaling each coefficient by its own fitted magnitude instead is another common convention in the sloppy-models literature; either is fine as long as it's applied consistently.)

For a continuous picture rather than just the two extreme directions, we can sample $\delta\mathbf{c} \sim \mathcal{N}(\mathbf{0}, 2\Delta e \mathbf{H}^{-1})$ and overlay the resulting family of boundaries — a Laplace-type approximation to the tolerance region. Collapsing this per-point envelope onto the fitted boundary curve itself, as a single shaded band, would also let stiffness and sloppiness be seen varying *along* the boundary — for instance, tightly pinned where the bulk of the data sits, floppier out in the tails.

This Gaussian is only a local approximation, though, and at a practical tolerance like $\Delta e = 1\%$ of e^* it can overshoot substantially: a draw's *typical* squared Mahalanobis radius under $\mathcal{N}(\mathbf{0}, 2\Delta e \mathbf{H}^{-1})$ is $2\Delta e$ per retained dimension (not $2\Delta e$ total), so in five or six dimensions a "typical" sample already sits several tolerance-

widths out along the quadratic form, and the true (non-quadratic) error there can overshoot the target by tens of percent — confirmed directly in `test.m`'s implementation. The fix used there: keep $\mathbf{H}$'s eigenbasis only to choose *directions* (so the stiff/sloppy character still shows), but calibrate the actual *distance* along each sampled direction by bisection on the real `norm_err` (via `classify_normals`) rather than trusting the quadratic form out to the full Δe budget.

□ **3.6. Python is much slower than MATLAB in the mixed-sign, $s = 0$ corner.** Repeating §2.2's case 3 (mixed-sign, $s = 0$) benchmark in Python:

quantity	MATLAB time	Python time
gradient	1.2 s	~38 s
Hessian	0.17 s	~115 s

Correctness is not in question (agreement to ~ 6 significant digits); the gap is pure runtime. Profiling traces it to `scipy.integrate.quad_vec`'s Gauss-Kronrod stepper calling the integrand once per scalar quadrature node in a Python loop, versus MATLAB's `quadgk`, which batches all nodes of a subinterval into one vectorized call — a genuine API-level difference, not a difference in the math. (An earlier hypothesis — that the Ruben-series convolution itself was the bottleneck — turned out to be a smaller effect; caching its coefficients across quadrature nodes gives a real $\sim 100\times$ speedup on an isolated call but barely moves the full Hessian benchmark, since the dominant cost is the separate Imhof-inversion fallback that many of the Hessian's shifted-dof terms take.)

Attempted fix, reverted. A from-scratch vectorized Gauss-Kronrod-15 integrator (reusing `scipy`'s published node/weight tables, not its internal driver) was written to batch quadrature nodes the way MATLAB does. The vectorized integrand itself is done and validated to machine precision. The batched adaptive-subdivision loop, however, produced `NaN` and failed to converge on the first real test case — likely a catastrophic-cancellation issue in the semi-infinite substitution near the integrand's removable singularity at $u = 0$, though the root cause was not isolated — and was reverted rather than shipped with an open instability. For whoever picks this up next: consider special-casing the $u \rightarrow 0$ limit analytically, or reusing `scipy`'s own outer subdivision/error-bookkeeping via a narrow monkey-patch of just its inner per-node loop rather than re-deriving that bookkeeping from scratch; re-profile after any fix rather than assuming a per-node speedup transforms the wall-clock total; and validate broadly (all output modes, both signs, $s = 0/s \neq 0$) before wiring in a replacement, since `imhof` is used throughout the library.

References

1. Das, A., & Geisler, W. S. (2020). *Methods to integrate multinormals and compute classification measures*. arXiv:2012.14331. — Derives the optimal (Bayes) quadratic classification boundary, and the conversion between the normal-quadratic coefficients $(\mathbf{Q}_2, \mathbf{q}_1, q_0, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and the native $\tilde{\chi}$ parameters $(\mathbf{w}, \mathbf{k}, \boldsymbol{\lambda}, s, m)$ (implemented in `norm_quad_to_gx2_params`).
2. Das, A. (2025). *New methods to compute the generalized chi-square distribution*. Journal of Statistical Computation and Simulation, 95(12), 2608–2642. doi:10.1080/00949655.2025.2501401. — The `gx2` paper: notation, and the Imhof / Gil-Pelaez inversion used throughout.
3. Couch, J. *Background note* (`derivatives/background/To_Abhranil.tex`) — the excess-classification-error motivation and the ω^2 "divergence" resolved by rule R2.
4. Gutenkunst, R. N., Waterfall, J. J., Casey, F. P., Brown, K. S., Myers, C. R., & Sethna, J. P. (2007). *Universally sloppy parameter sensitivities in systems biology models*. PLoS Computational Biology, 3(10), e189. doi:10.1371/journal.pcbi.0030189. — Introduces the stiff/sloppy eigenvalue-spectrum language for a Hessian or Fisher-information matrix at a fitted optimum, used in §3.4.3.
5. Transtrum, M. K., Machta, B. B., Brown, K. S., Daniels, B. C., Myers, C. R., & Sethna, J. P. (2015). *Perspective: Sloppiness and emergent theories in physics, biology, and engineering*. The Journal of Chemical Physics, 143(1), 010901. doi:10.1063/1.4923066. — Review of the same stiff/sloppy framework across fields.