**1. Modifications to the Weibull Function (Parameter *f*)**

The user’s modified Weibull model likely replaces log(x) with log(x + f) to avoid taking the log of zero or negative values. In the standard four-parameter Weibull (Type 1) model (as used in drc::W1.4), the mean function is:

f(x)=c+(d−c)  exp⁡ ⁣(−exp⁡ ⁣[ b (log⁡x−log⁡e)]),f(x) = c + (d - c) \; \exp\!\Big(-\exp\!\big[\,b\,(\log x - \log e)\big]\Big),f(x)=c+(d−c)exp(−exp[b(logx−loge)]),

where b is the slope, c lower asymptote, d upper asymptote, and e the inflection point (ED₅₀)​

[stackoverflow.com](https://stackoverflow.com/questions/77092406/not-correct-upperl-argument-when-using-drm#:~:text=f%28x%29%20%3D%20c%20%2B%20%28d,log%28e)

​

[doseresponse.github.io](https://doseresponse.github.io/drc/reference/weibull1.html#:~:text=The%20four,log%28e)

. This formula fails when x=0 (since $\log(0)$ is undefined)​

[stackoverflow.com](https://stackoverflow.com/questions/77092406/not-correct-upperl-argument-when-using-drm#:~:text=f%28x%29%20%3D%20c%20%2B%20%28d,log%28e)

, which motivates introducing a small offset f so that $\log(x+f)$ is always defined. The modified version would be something like:

f(x)=c+(d−c)  exp⁡ ⁣(−exp⁡ ⁣[ b (log⁡(x+f)−log⁡e)]),f(x) = c + (d - c) \; \exp\!\Big(-\exp\!\big[\,b\,(\log(x+f) - \log e)\big]\Big),f(x)=c+(d−c)exp(−exp[b(log(x+f)−loge)]),

or an analogous change for a Type 2 Weibull function. This new parameter f acts as a **location/threshold offset** ensuring the log argument is positive.

**Potential issues:** Adding f increases model flexibility but also complexity. The parameters f and e can become **correlated or confounded** – shifting the curve along the x-axis can be accomplished by changing either parameter. For example, if f grows, the effect could be counteracted by adjusting e (since the inflection point in terms of the original x-scale might shift). This can lead to a **flat log-likelihood or sum-of-squares surface** where many combinations of f and e give similar fits, impeding convergence. In fact, introducing a threshold (location) parameter is known to make Weibull models harder to estimate reliably​

[mathworks.com](https://www.mathworks.com/help/stats/three-parameter-weibull-distribution.html#:~:text=If%20the%20scale%20parameter%20,maximum%20is%20degenerate%20when)

. It often requires strong data signal near the lower end; if the data don’t clearly determine the threshold, the optimizer may wander.

**Recommendation:** Double-check that the modified formula is implemented correctly everywhere (including any internal uses of e). Typically, one would **not** add f to the e term’s log (since e remains a scale parameter on the original dose axis), but confirm that the model definition is self-consistent. Also, ensure f is constrained to a sensible range (it should be ≥0 and likely small). If f is allowed to go negative during optimization, the model will again evaluate \log(x+f) on non-positive inputs, causing undefined values or NaNs. Using drm(..., lowerl=c(…,0)) to set a non-negativity bound for f can enforce this domain​

[stackoverflow.com](https://stackoverflow.com/questions/77092406/not-correct-upperl-argument-when-using-drm#:~:text=f%28x%29%20%3D%20c%20%2B%20%28d,log%28e)

. (If using drmc(constr=TRUE), note that it forces **all** parameters non-negative​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

, which may not be appropriate for parameters like b that could be negative, so better to use lowerl/upperl vectors for selective bounds.)

**2. Numerical Stability and Scaling Issues**

Nonlinear models can easily encounter numerical instability if transformations and parameter scales are not handled carefully. Several points to consider:

* **Domain and Range of the Function:** The Weibull function involves a double exponential. When $x+f$ is very small relative to e, $\log(x+f) - \log(e)$ becomes a large negative number in magnitude. The inner term $\exp[b(\log(x+f)-\log e)]might underflow to 0 or overflow to an extremely large value depending on the sign ofb. For instance, if $b$ is positive and $x+f \ll e$, then $b(\log(x+f)-\log e)$ is a large negative number, $\exp(\text{that}) \approx 0, and the outer $\exp(-(\cdot)) \approx \exp(0) = 1$. If $b$ is negative (decreasing curve) and $x+f \ll e$, then $\exp(b(\log(x+f)-\log e))$ might overflow to $\infty, and $\exp(-\infty)=0$. \*\*Extremes like 0 and 1\*\* in these expressions are not problematic by themselves, but if the computation crosses into actual InforNaNdue to floating-point limits, the optimizer can fail. Monitor if for some parameter guesses the model returns NaN/Inf residuals or gradients. This might happen if, say,fis extremely small andblarge in magnitude, causing $\exp(b(\log(x+f)-\log e))$ to overflow. Using higher precision isn’t an option in R’s base optimizer, but \*scaling inputs\* can mitigate this. For example, if dosesxare very large or very small, rescaling the dose variable (and accordinglyeandf` initial values) to a more moderate range can help. In practice, one might work in units where the median dose is ~1 or ~10 for numerical convenience.
* **Improper transformations or offset handling:** Ensure that the **offset f is applied consistently** only to avoid log-domain issues and not inadvertently elsewhere. Sometimes users log-transform the data or predictor outside the model and also include an offset, which can double-count or mis-scale the effect. The model should treat f as just an additive constant inside the log. Also, consider the interpretation: with the offset, the effective dose that yields the inflection point is no longer exactly e on the original scale – it’s roughly x = e - f (if the formula is $\log(x+f) - \log e$). If the self-starter or initial guesses assumed e was near the midpoint of the observed $x$ range, that logic might break. You may need to adjust how initial e is chosen (for example, if a good guess for inflection is around the median dose, maybe set initial e to median($x$)+ some fraction of it, and f to a small fraction of that).
* **Initial parameter estimates:** Poor starting values are a common cause of divergence or non-convergence in nonlinear regression​

[cran.r-project.org](https://cran.r-project.org/web/packages/nlstools/vignettes/vignetteJSS.pdf#:~:text=,close%20to%20the%20unknown)

. The default self-starter in drc for Weibull (without f) uses heuristics based on the data. But with a custom five-parameter function, those self-starting values might not be available (unless you wrote a self-starter). If you didn’t supply start values, drm might have tried generic defaults, which could be far off. It’s crucial to provide reasonable starting values for all five parameters. For example:

* + Start c and d near the observed minimum and maximum response, respectively.
  + Start e near a mid-range dose (or where you guess the curve’s inflection is).
  + Start b (slope) with a modest value (e.g. 1 or -1) indicating an increasing or decreasing trend as appropriate.
  + Start f as a small positive number (e.g. a fraction of the smallest non-zero dose, or something like 0.1 if doses are around 1, or 1 if doses are around 100, etc.). Essentially, f should be small enough that $x+f \approx x$ for most nonzero $x$, but not so tiny that $x+f$ is indistinguishable from $x$ in machine precision or that the optimizer must traverse orders of magnitude to adjust it. If the data includes an actual dose of 0, a sensible initial f is slightly below the lowest non-zero dose, so that at $x=0$ the term $\log(0+f)$ is defined and yields a value in the ballpark of the other log-transformed doses.
* **Parameter scaling:** The five parameters can have very different scales. For instance, c and d might be around the scale of your response measurements, e and f are on the dose scale (which could be completely different units/magnitude), and b is dimensionless but can vary widely. If one parameter’s scale is orders of magnitude different from another’s, the optimizer’s default step sizes or tolerances may not suit both simultaneously. This can lead to one parameter barely budging while another oscillates. If you observe something like f not updating or overshooting, consider **scaling the problem**. You can manually scale x (and accordingly interpret e and f in that scaled space) or use a parameter transformation. For example, sometimes one fits $\log(e)$ instead of e to keep it unconstrained and of order 0-1, or one might fit $\log(f)to ensure positivity and manageable scale (since iff is expected ~$10^{-3}, $\log(f) \approx -7$ which is a more moderate number than dealing with $0.001directly). Indrm, direct log-transformation of parameters isn’t built-in, but you can internally reparameterize: for instance, define a new parameter $f^\* = \log(f)$ and use $\log(x + \exp(f^\*)) in the model function. This way, $f = \exp(f^*)$ is always positive and the optimizer works on $f^*$ which can vary on the real line. This can greatly stabilize the search for f. (Ensure you also transform your initial guess for f accordingly if you do this.)
* **Handling 0 or negative log inputs gracefully:** Even with f, if the optimizer *during its search* tries a value of f that is very small or negative, you can get $\log(x+f)$ invalid for some data points. Using bounds as mentioned (e.g. lowerl in drm) is one way to prevent that. Another strategy is to modify the model function to **safeguard against domain errors** – e.g., return a very large residual or cost if $x+f \le 0$ for any point. In a custom objective function one might do if(x+f <= 0) return(very\_large\_value) to create a penalty, but with drm’s structured model interface, that’s tricky. It’s better to enforce constraints via the optimizer or reparameterize to inherently avoid the invalid domain.

**3. Checking Derivative Calculations**

If you supplied analytic derivatives (the gradient function) for your custom Weibull, it’s essential that they are 100% correct. Even a sign error or a minor algebraic mistake can prevent convergence. An optimizer like **BFGS** (which drm uses by default​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

) will rely on those gradients to navigate the parameter space. Incorrect gradients may send the search in the wrong direction or cause it to miss the optimum entirely. Symptoms of a bad gradient include: very tiny or random changes in parameters each iteration, oscillating objective values, or the algorithm terminating early with “gradient is zero” when it really isn’t at the optimum.

**Steps to verify:**

* Derive the partial derivatives of the modified function with respect to each parameter (b, c, d, e, f) on paper or using a computer algebra system. The presence of $\log(x+f)$ and nested exponentials makes it easy to slip up. For example, for a Type 1 Weibull with the $f$ offset, one should get something like (for each observation $x\_i$):
  + $\displaystyle \frac{\partial y}{\partial c} = 1 - \exp!{-\exp[z\_i]},$
  + $\displaystyle \frac{\partial y}{\partial d} = \exp!{-\exp[z\_i]},$
  + $\displaystyle \frac{\partial y}{\partial b} = -(d-c),\exp!{-\exp[z\_i]};\exp[z\_i];\big(\log(x\_i+f) - \log e\big),$
  + $\displaystyle \frac{\partial y}{\partial e} = +(d-c),\exp!{-\exp[z\_i]};\exp[z\_i];\frac{b}{e},$
  + $\displaystyle \frac{\partial y}{\partial f} = -(d-c),\exp!{-\exp[z\_i]};\exp[z\_i];\frac{b}{,x\_i+f,},$

where $z\_i = b,[\log(x\_i+f) - \log e]$. (The signs and form will differ slightly if you’re using the Type 2 Weibull formulation, but the process is analogous.) Ensure your code matches these results.

* If you did **not** supply an analytic gradient, drm/optim computes it via finite differences. In that case, consider that finite difference approximation can be unstable around boundaries. For example, if $f$ is very small, a symmetric difference formula might try evaluating the function at $f - \epsilon$ (which could make $x+f-\epsilon$ negative for the smallest $x$) and at $f + \epsilon$. The evaluation at $f-\epsilon$ would yield NA/Inf, so the gradient for $f$ comes out NA. This could derail the optimization. When using numeric gradients, it’s crucial to start well *inside* the domain (not right at the boundary). Starting f at a moderately positive value (not exactly 0) helps. Also, you might switch to a one-sided difference for f (though optim doesn’t allow choosing that easily). A more practical approach is: once again, enforcing f’s lower bound or using a log-transform for f avoids the possibility of sampling an invalid point when computing the gradient.
* As a troubleshooting step, you can **turn off analytic gradients** to see if the model converges with a numeric approach (or vice versa). In drm, this can be controlled by the useD argument in drmc(). By default useD = FALSE (meaning even if you provided a gradient, it won’t use it unless you explicitly set useD=TRUE)​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

. If you suspect your gradient function is wrong, leave useD=FALSE so that optim does its own numeric gradient – this might converge albeit more slowly. If the numeric approach converges but the analytic one doesn’t, that’s a strong indicator the derivatives are implemented incorrectly. Fixing them should then allow the faster analytic method to work. Conversely, if numeric gradients were giving trouble due to the $f$ boundary issue, providing a correct analytic gradient and setting useD=TRUE can sometimes **improve stability** (since the analytic derivative “knows” not to step outside the domain, whereas a blind finite difference might).

* Double-check subtle points in the derivative: the derivative with respect to e often trips people up because of the negative sign inside $\log(e)$ – make sure you have the correct sign (it should be positive as shown above, since increasing e usually shifts the curve rightward, which for an increasing curve means a higher predicted $y$ at a given $x$, hence a positive partial derivative). The derivative w.r.t. f should be negative (if $b$ and $d>c$ are positive), meaning increasing f usually lowers the response (because $x+f$ larger means $\log(x+f)$ slightly bigger, making the inner exponent argument larger, thereby the outer exp$(-\text{something})$ smaller). These intuitive checks can catch sign mistakes.

**4. Optimization Method in drm for a 5-Parameter Model**

The drm() function from the **drc** package uses R’s general-purpose optimizer **optim()** under the hood​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=match%20at%20L1711%20This%20function,least%20squares%20estimation%2C%20which%20is)

. By default it calls the “BFGS” method (a quasi-Newton algorithm) with finite-difference gradients (since useD=FALSE unless changed)​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

. BFGS is usually efficient, but it can struggle if the surface is not smooth or if starting values are poor. A few considerations:

* **Is BFGS appropriate?** BFGS assumes a reasonably well-behaved (differentiable) surface. If your modified Weibull function yields NA/Inf in regions (due to log issues) or has a very flat region, BFGS might have trouble. A classic Newton-type method might also be confused if the Hessian is nearly singular (which can happen if parameters are highly correlated, as noted with f and e). In such cases, a **simplex method like Nelder–Mead** can be more robust because it doesn’t use gradients at all. It may handle weird surfaces more gracefully (at the cost of more function evaluations). You can try Nelder–Mead by calling drm(..., control=drmc(method="Nelder-Mead")). If Nelder–Mead converges, it suggests the problem might have been with gradient-based search. Often a strategy is to first get close to the optimum with Nelder–Mead, then, if high precision is needed, switch to BFGS (or another method) using the Nelder–Mead solution as the new start.
* **Constrained optimization:** If you set bounds for parameters via lowerl/upperl in drm, it will switch to the **“L-BFGS-B”** method automatically (a variant of BFGS that handles box constraints)​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=The%20control%20arguments%20are%20specified,is%20used%20to%20estimate%20the)

. L-BFGS-B can enforce $f \ge 0$ nicely, but be aware that when a parameter hits its bound, the optimization might proceed along the boundary which can also be tricky. Monitor whether the algorithm is slamming f against 0; if so, it might be telling you that the best fit is actually at the boundary (or that it *wants* to go below 0 but can’t). If the best fit is at f = 0 exactly, that means perhaps the data don’t actually require an offset (the model might fit fine with just 4 parameters). In that scenario, it might be better to remove f and handle the zero-dose another way (see Point 5 below).

* **Alternate algorithms:** Besides BFGS and Nelder–Mead, optim() offers **CG** (conjugate gradients) and **SANN** (simulated annealing) among others. Simulated annealing (SANN) is more of a global search method — it’s very slow but might escape a bad local minimum or flat region. You could attempt a two-stage approach: use SANN for a large number of iterations with a wide exploration, then switch to BFGS. However, this is rarely needed for standard dose-response models; it’s a last resort if you suspect multiple minima. Conjugate gradient (CG) is another gradient-based method that sometimes behaves differently than BFGS – it might be worth a shot if others fail, but generally if BFGS fails it’s due to gradients or ill-conditioning, which CG won’t magically fix.
* **Increase iterations or adjust tolerances:** By default, drm allows up to 500 iterations (maxIt=500)​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

. If the model is **close** to converging but not quite, you might simply need to raise this limit. Watch the optimization trace (use trace=TRUE in control) to see if it’s still improving when it stops. If so, increasing maxIt could help. Likewise, loosening the convergence tolerance (relTol) might declare convergence even if the last steps are tiny oscillations. But be cautious: if the algorithm is bouncing around a plateau without improving, more iterations won’t help; you need a different strategy or better starting point.

* **Use of self-starters vs manual start:** drm typically has built-in self-starter functions for its built-in models that give good initial values. For a custom model, if you provided an ssfct (self-start) function, verify it returns sensible starts. If not using one, you *must* provide start values in the drm() call to override the default. Without a good self-starter or user-supplied start, drm might have simply used (0.1, 0.1, 0.1, 0.1, 0.1) or something arbitrary for five parameters, which is almost guaranteed to fail for a nonlinear model of this complexity.

In summary, the default optimization method (BFGS) is generally fine **if** the model is well-behaved in the region of the optimum. But given the modified model’s challenges, you should be open to trying Nelder–Mead or constrained BFGS (L-BFGS-B) to handle domain restrictions. The drm control interface lets you do this easily​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=Description%20Set%20control%20arguments%20in,15)

. Also consider using the **nls() function** (which uses Gauss-Newton or “Port” algorithm) for a sanity check – sometimes nls with the port algorithm (which can enforce bounds) may converge where drm did not, or vice versa, due to differences in algorithms. You can supply the model formula to nls() directly using log(x+f) and see what happens (you’ll have to provide start vals there too).

**5. Debugging Strategies to Improve Convergence**

Given the above, here are concrete steps and alternatives to resolve the convergence issue:

* **Try simpler models first:** As a diagnostic, see if the model converges *without* the f parameter. For example, omit the zero-dose data point and fit the standard 4-parameter Weibull (W1.4() or W2.4() in drc). Does that converge and give sensible estimates for b, c, d, e? If not, the problem might not be f at all but something fundamental with the data or the Weibull assumption. If the 4-parameter model converges fine, then add the zero-dose back in and see if it breaks. If it does, that confirms the zero-dose is the culprit. In that case, assess how that zero-dose behaves: is the response at zero dose near the expected lower asymptote (c)? If yes, one could argue you don’t actually need a special f (the model’s $c$ should accommodate the 0-dose response). If the 0-dose response is unexpectedly high or low (e.g., higher than some responses at small non-zero doses), then a standard Weibull might indeed struggle – that could indicate some form of hormesis or other effect not captured by the basic shape.
* **Use built-in alternatives:** Note that drc **does have a five-parameter Weibull** in the form of weibull2x() (with parameter t0)​

[doseresponse.github.io](https://doseresponse.github.io/drc/reference/weibull1.html#:~:text=weibull2x,ssfct%20%3D%20NULL)

. This effectively adds a location parameter (like your f). If available in your drc version, you could try drm(..., fct = weibull1(names=c("b","c","d","e","f"))) or the weibull2x() convenience function. The built-in may handle self-starting and scaling better internally. Check the documentation for weibull2x: it likely defines $t\_0$ such that the model is $\log(x - t\_0)$ (i.e. shifting the dose by a threshold $t\_0$) – analogous to your $\log(x+f)$ depending on sign conventions. If weibull2x is parameterized as $\log(x - t0)$, you’d want to use it only if your data has a **natural** threshold > 0 (which in your case is not true; you just want to avoid log(0)). But you could still attempt it by treating $t0 = -f$ (which would mean x - t0 = x + f). In any case, using the package’s native implementation (if it exists) can at least serve as a check: does their version converge on your data? If it does, perhaps examine how their self-starter picks initial values for the threshold.

* **Vary the starting values systematically:** When a model doesn’t converge, try a wide range of starting guesses. This can be done manually by calling drm (or optim/nls) in a loop over different starting points. For f, try starts ranging from very small (e.g. 1e-6) up to a larger fraction of your lowest nonzero dose. For b, try both positive and negative values if you aren’t sure of the direction of the response. It’s not uncommon to run dozens or hundreds of combinations (this can be automated with a grid or random search) to find a set that converges. The {drc} package doesn’t do this automatically, but you can script it. The **nls2** package is also handy for trying multiple start sets. A successful convergence with one of these starts will give you a clue – you can then refine around that start or use those estimates as fixed starting values in drm to see if it consistently converges.
* **Check intermediate iterations (trace):** Run drm(..., control=drmc(trace=TRUE, otrace=TRUE)). This will output the trace from optim as it iterates​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=match%20at%20L1769%20trace%20logical,See)

. By inspecting this, you can see how parameters are evolving each iteration. If you see f becoming negative or extremely large during the search, that’s immediately informative – it means you need to constrain it. If you see one parameter hardly changing at all, it might mean it’s not identifiable from the data. If the objective (residual sum of squares) is not decreasing or is oscillating, you might guess the algorithm is stuck on a flat ridge. This information can guide you to adjust starts or method.

* **Consider data transformations:** Sometimes a log-dose model can be fit by transforming the dose axis beforehand. For example, rather than modeling $y = f(x)$ with $x$ in original units, you model $y = g(z)$ with $z = \log(x + \text{constant})$. Essentially, you perform the log transform outside of the nonlinear function. If one adds a small constant to all doses (e.g. 0.1) and then takes $\log$, you avoid $-\infty$ for zero dose. Then you could fit a standard asymmetrical sigmoid (like a logistic or Gompertz) on that transformed scale. The results are not identical to the Weibull model, but they might be close. This is more of a workaround and could bias the fit slightly (choice of constant affects it), so it’s a last resort if the direct modeling absolutely refuses to converge. The preferable solution is to get the direct model working with proper techniques mentioned above.
* **Evaluate model at solution (if any) and residuals:** If the model *does* converge after tweaks but the fit is poor (or residuals show patterns), reassess if the modified Weibull is appropriate. Convergence issues sometimes indicate model mis-specification. For instance, if there is a slight upward tick at zero dose (hormesis effect), a Weibull might not capture it well unless further modified. In such cases, a different functional form or adding a term might fit better. The drc package has a class of “Brain-Cousens” hormesis models (with an additional parameter for low-dose stimulation)​

[cran.r-project.org](https://cran.r-project.org/web/packages/drc/drc.pdf#:~:text=%5BPDF%5D%20Package%20%27drc%27%20,should%20be%20positive%20in)

– if relevant, that could be another path (though that’s more if the zero-dose response is higher than the response at small doses).

* **Lastly, patience and small tweaks:** Nonlinear optimization can be fickle. Changing the optimizer’s settings like step size or tolerance can make a difference. If using BFGS, ensure the numeric gradient step size (ndeps in optim, not directly exposed in drm control) isn’t too large relative to f’s scale – by default it’s something like 1e-6. If your f optimum is ~1e-4, a step of 1e-6 is okay, but if f is ~1e-8, the default step might be too big (or too small relative to f – it’s tricky). While drm doesn’t expose ndeps, you could call optim yourself on a custom sum-of-squares function to have more control. This is advanced, but if desperate you can bypass drm: use optim(par=c(b,c,d,e,f), fn=sum\_of\_squares, gr=gradient\_fn, method=...) directly to tinker with such settings.

**In summary,** the model likely fails to converge because the introduction of f makes the optimization landscape difficult: there may be numerical problems at the boundary (log(x+f)), an extra parameter to estimate with possible identifiability issues, and a need for better starting values or constraints. To fix this, ensure the model is defined correctly (no algebra mistakes), pick good starting values for all parameters (especially the new f), constrain f to a valid range, and experiment with alternative optimization methods if the default struggles. By systematically debugging – checking gradients, trying different algorithms, and simplifying where possible – you should be able to pinpoint the issue. Often, simply providing a well-informed starting guess for the offset f and using an algorithm like Nelder–Mead or a bounded BFGS will resolve the convergence failure. Good luck!