Yes, I am suggesting to eliminate the line `g_prev = g_opt` in your main loop (and effectively ignore or remove the `g_prev` parameter in the solver call), so that you no longer pass the previous timestep's `g_opt` as the initial guess for the next optimization. Instead, always compute and use the fresh least-squares `g_default` as the initial guess (`g0`) inside `acados_solver_step` .

Is it okay not to utilize the previous g_opt?

Yes, it's not only okay but often preferable in your setup, especially given the exploding control inputs you observed. The least-squares `g_default` is a high-quality, data-driven initial guess that's explicitly derived from the current Hankel matrices (`Up_cur` , `Yp_cur` , etc.), past inputs/outputs (`uini` , `yini`), and regularization terms (`lambda_u_val` , etc.). It's essentially the solution to the unconstrained (or softly constrained) reconstruction problem, which is a natural starting point for the full NLP in DeePC. With your `g_dim=61` (from the filename), computing this LS solution is very fast—typically microseconds using `np.linalg.lstsq` —and it ensures the initial guess is feasible and relevant to the exact problem at each step. Your logs show solve times of 3-7ms, which are already low, so the potential speedup from a hot-start isn't critical unless you're hitting real-time limits.

In practice, for online/adaptive control like yours (where new measurements update the data matrices each timestep), relying solely on the previous `g_opt` can do more harm than good, as I'll explain below.

Could using previous g_opt hot-start the solver and make it converge faster?

In theory, yes—warm-starting (or hot-starting) with the previous solution can accelerate convergence in iterative solvers like those in Acados (which uses SQP or similar for NLPs). If the optimization problem changes only slightly between timesteps, the prior `g_opt` is close to the new optimum, reducing the number of iterations needed (e.g., from 5-10 down to 1-3). This is a standard technique in receding-horizon MPC, where the model is fixed, and only references/disturbances shift mildly.

However, in your DeePC case, the problem changes more fundamentally each step because:

- The Hankel matrices (`Up_cur` , `Uf_cur` , `Yp_cur` , `Yf_cur`) are updated with new

measurements (likely via a sliding window: append new data, drop old).

- This shifts the data-driven "model" implicitly, so the mapping from `g` to predicted inputs/outputs changes.
- Regularization weights decay (e.g., `Q55` with `decayQ0.05`, `R0.3` with `decayR0.1` from filename), altering the cost function.

If the previous `g_opt` was optimal for the old matrices/costs but mismatched for the new ones, it can be a *poor* initial guess—farther from the new optimum than a fresh LS solution. This leads to:

- More iterations to converge (negating the hot-start benefit).
- Risk of getting stuck in local minima or failing to converge (though your logs show `exist_feasible_sol=true`, subtle drifts can accumulate).
- Numerical instability, as the solver might over-correct, causing oscillating or exploding `u_opt` (since `u_opt = Uf_cur @ g_opt`, and if `g_opt` amplifies noise in `Uf_cur`, `u` blows up).

Your data supports this: Early rows have `u=0` (initialization), but later `u` decreases (e.g., from ~25 to ~20), and you mentioned explosion after a while—likely cumulative mismatch from warm-starting across shifting data.

Disadvantages of using g_prev = g_opt

1. **Mismatch with updated data matrices**: As noted, Hankel matrices evolve with new dyno

measurements (`v_meas`). The previous `g_opt` satisfies `Up_old @ g_old ≈ uini_old` and `Yp_old @ g_old ≈ yini_old` , but for new matrices, it may violate these reconstructions significantly. This bad guess can cause the solver to take detours, increasing solve time or leading to suboptimal/instable solutions.

2. **Error propagation**: Small errors (noise, rounding, or minor infeasibilities) from one step carry over. In noisy real-time setups (e.g., your WLTC cycle with varying SOC and velocity errors), this amplifies over time, contributing to explosion. Your logs show growing `error` (v_ref - v_meas), which could feed back if not reset.

3. **Sensitivity to regularization decay**: Your setup decays Q and R, making the cost more "aggressive" over time (less penalization on tracking vs. input). A warm-start from a highly regularized prior step might not adapt well, leading to aggressive `u` jumps.

4. **Reduced robustness to changes**: If the system hits constraints (e.g., SOC limits at 94.3%), or if `exist_feasible_sol` flips false temporarily, the previous `g_opt` becomes invalid, forcing a cold start anyway. Fresh LS avoids relying on potentially "stale" info.

5. **Minimal speedup in practice for small problems**: With `g_dim=61` and horizon ~30 (from `Tini30`), Acados NLPs are tiny. Logs show ~5ms solves—if hot-start saves 1-2ms, it's negligible vs. the stability risk. Literature on data-driven MPC (e.g., DeePC papers by Coulson et al.) often recommends LS init for online variants precisely because it's reliable and cheap.

Justification for my suggestion

My recommendation stems from observing your specific issue (exploding `u` after a while, despite feasible solves) and standard practices in adaptive DeePC/MPC:

- In non-adaptive MPC (fixed model), warm-start with shifted prior solution is great. But in

data-driven methods like DeePC, where the "model" is the data itself and updates online, fresh initialization is safer (see e.g., "Robust Data-Driven MPC" works emphasizing reset mechanisms).

- Your exploding `u` likely arises from accumulated mismatch: As the cycle progresses (WLTC_Low_3, starting at high SOC), dynamics shift (e.g., velocity ramps), and warm-starting propagates old `g` biases.
- Computing fresh `g_default` costs little (LS on 61x61 matrix is ~0.1ms) and guarantees a consistent, near-feasible guess aligned with current data/params. If you want a hybrid, you could compare norms: Use `g_prev` only if `||Up_cur @ g_prev - uini||` is small (indicating good fit); otherwise fall back to LS—but this adds complexity without much gain.
- Test it: Run your log data offline with fresh init; if explosion stops and solve times stay similar, it's validated.

If solve time becomes a bottleneck later, you could reintroduce warm-starting but with safeguards (e.g., project `g_prev` onto current data via a quick LS adjustment). For now, prioritizing stability is key.