

STEVE SOMMERVILLE

POWER SYSTEMS · AI · RESEARCH

Technical Note 004: WECC Plant Controller (REPC_C) Modelling in PSCAD using FORTRAN

N-004

This is preliminary research and has not been validated or peer-reviewed. It could therefore be totally wrong.

Key Concepts

- Modelling of individual and bespoke controllers in PSCAD is often a slow and tedious process, especially for complex systems.
- As a human, I like to build models directly using transfer functions and blocks. It is a challenging and sometimes 'fun' process, but it does take time and bug-hunting can be frustrating.
- Instead of using transfer function it is possible to code directly into FORTRAN, without using building blocks. But as FORTRAN is quite a few years older than me, and I am not a fan of coding, AI seems the obvious choice.
- Using AI (Claude) with an MCP interface and an appropriate toolkit it is possible to directly interface to PSCAD and use it to build models. AIs are **great** at writing code and this is simple compared to most cases.
- In this case I developed and tested a model of a WECC – REPC_C Plant Controller.

Plant Controller – WECC REPC_C

- The WECC Plant Controller model REPC_C has been around for a while and is used extensively in simple open-source RMS studies.
- Most OEM models are built around a similar topology but with additional functions and hardware implementation levels.
- The REPC_C contains all the core functions for a plant controller that can work with Solar, Wind and Storage:
 - Active Power Control.
 - Frequency Response (under frequency and over frequency).
 - Reactive Power Control – Fixed MVar and Fixed Power Factor.
 - Voltage Control.
 - Various anti-windup and limit functions.

Plant Controller – WECC REPC_C

- The version below is from the WECC document library and shows the REPC_D variant – it is the same as the REPC_C with some additional scaling functions (shown in red).

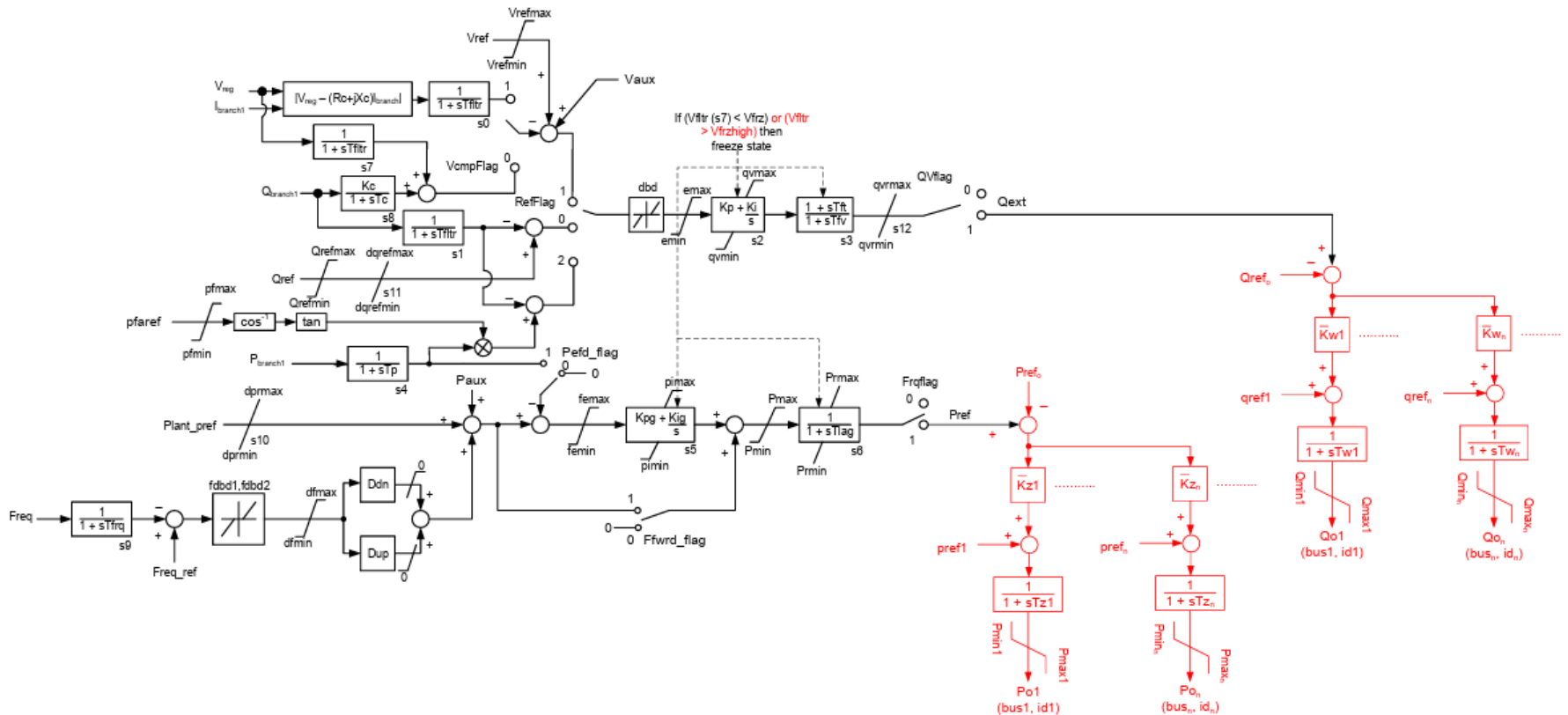
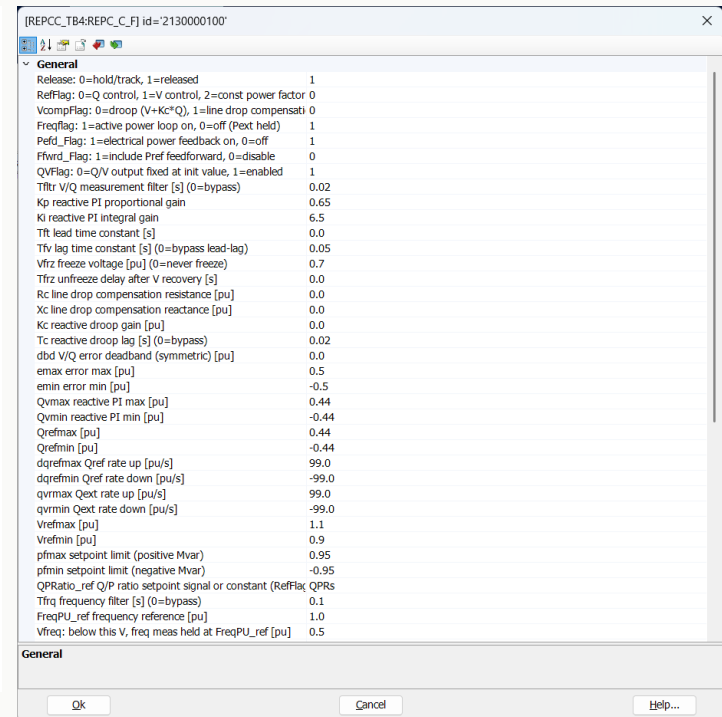
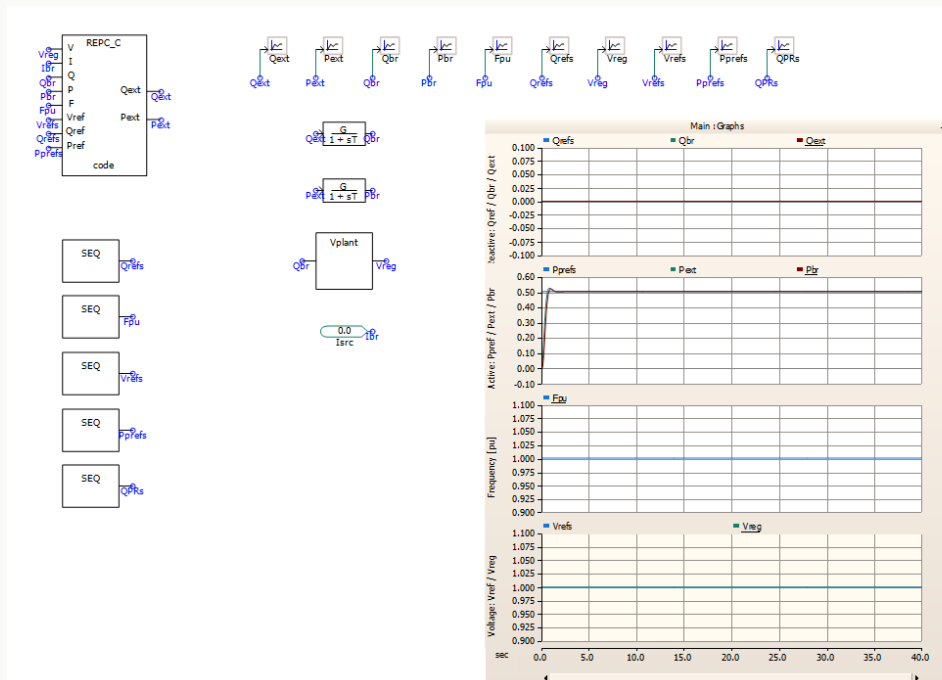


Figure 2: REPC_D model

Plant Controller – WECC REPC_C

- The implementation was done by uploading the WECC REPC_C model and associated transfer functions and WECC guidance note into Claude.
- Claude Fable was used to implement directly in PSCAD as direct FORTRAN code, using some skills I had previously taught it for creating submodules.
- It managed to extract all the required ports and define the parameters as external settable values.



Testing

- The interesting part of the project was letting Claude define, implement and evaluate a test plan and the output results – all as an agentic flow.
- I gave high level instructions of making sure it tested all the functions. It produced a test regime (below) and then checked with me for approval.
- Testing was done as step response against as a closed feedback loop.

Series	Commands	What it proves
P setpoint	1.0 → 0.5 → 0.0 → -0.5 → -1.0 pu	Full four-quadrant P dispatch incl. BESS charging
Q setpoint	unity → 0.98 → 0.95 → -0.95 → -0.98 pf	MVAr mode tracks the pf-equivalent Q at P = 0.5
PF setpoint	same pf ladder, commanded as Q/P ratio	Power-factor mode; ratio setpoint now a live signal
V setpoint	+2% → +4% → -2% → -4%	Voltage mode with 4% droop; ±4% brushes the Q limit
Limits	Qref ±0.6 pu, tight rate limits	Qrefmax clamp and qvr ramp behave, anti-windup clean
Grid events	f = 0.99 / 1.01 pu while charging	Droop and deadband arithmetic around a negative setpoint

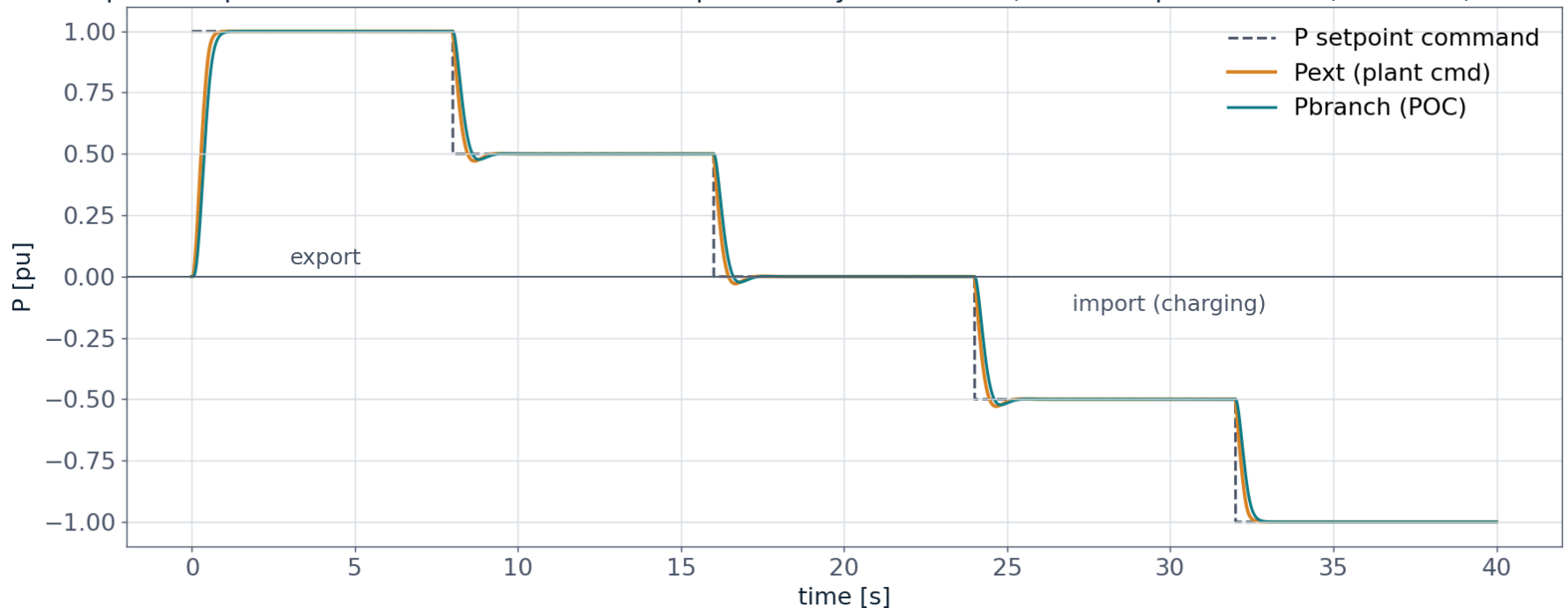
Testing – Pt2

- There were several expected results and a few surprises on the test regime. It autotuned and parameterised the system easily, and identified a few bugs in the responses where it didn't quite meet the test regime – Claude happily updated and tweaked the code.
- The unexpected behaviour was that Claude came up with a clever way of creating the tests. It developed an additional small submodule it called 'SEQ'. Each SEQ box was a custom FORTRAN code that allowed definition of:
 - Five input setpoint values
 - Four time stamps for events
 - Some internal logic to avoid spurious operation.
- The SEQ modules were a way to avoid using a huge pile of the basic components from the library and chain them all together (which is what I would have done). It is a simple and elegant solution.
- The other unexpected behaviour was that Claude, without being asked, decided to build a model of the system directly in Python, and then validate the PSCAD model against its own Python model!

P Setpoint

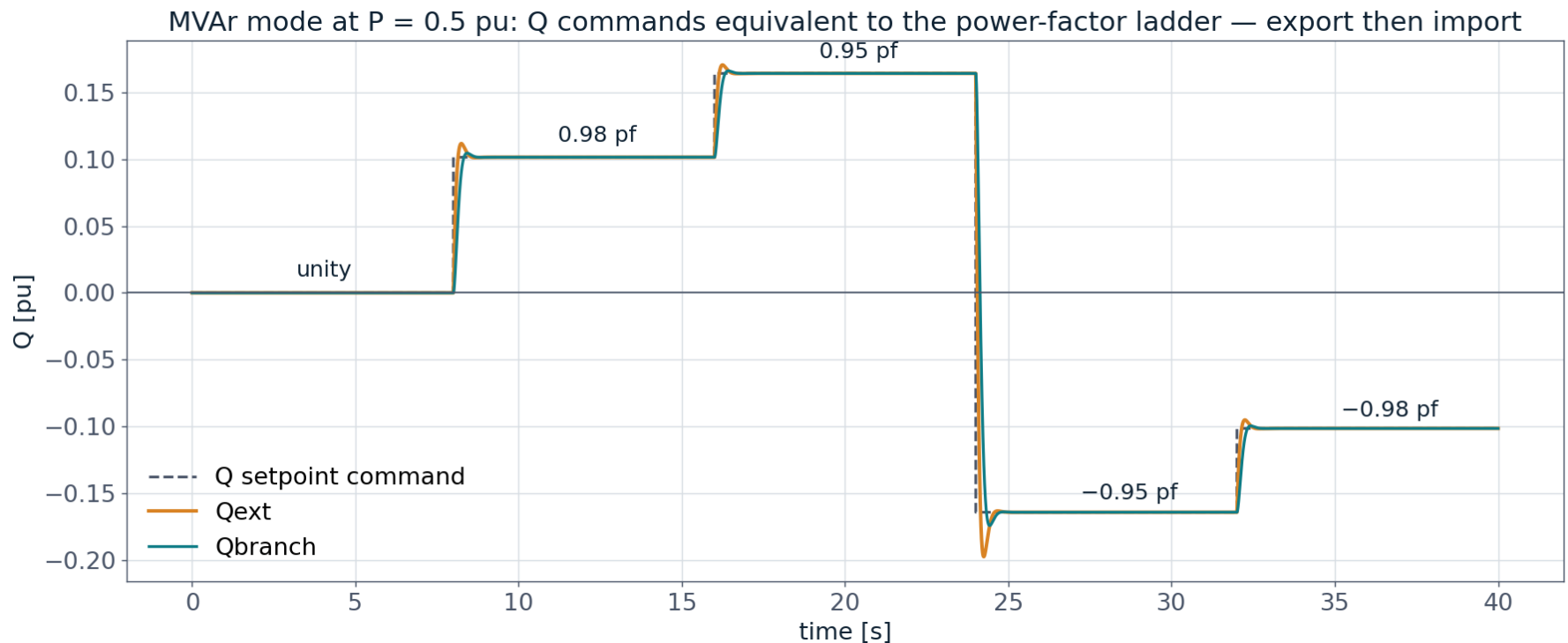
- Testing of simple MW setpoints, using the Plant_pref command.

P setpoint sequence 1.0 / 0.5 / 0.0 / -0.5 / -1.0 pu — every dwell exact, worst step: rise 0.79 s, OS 4.6%, settle 1.



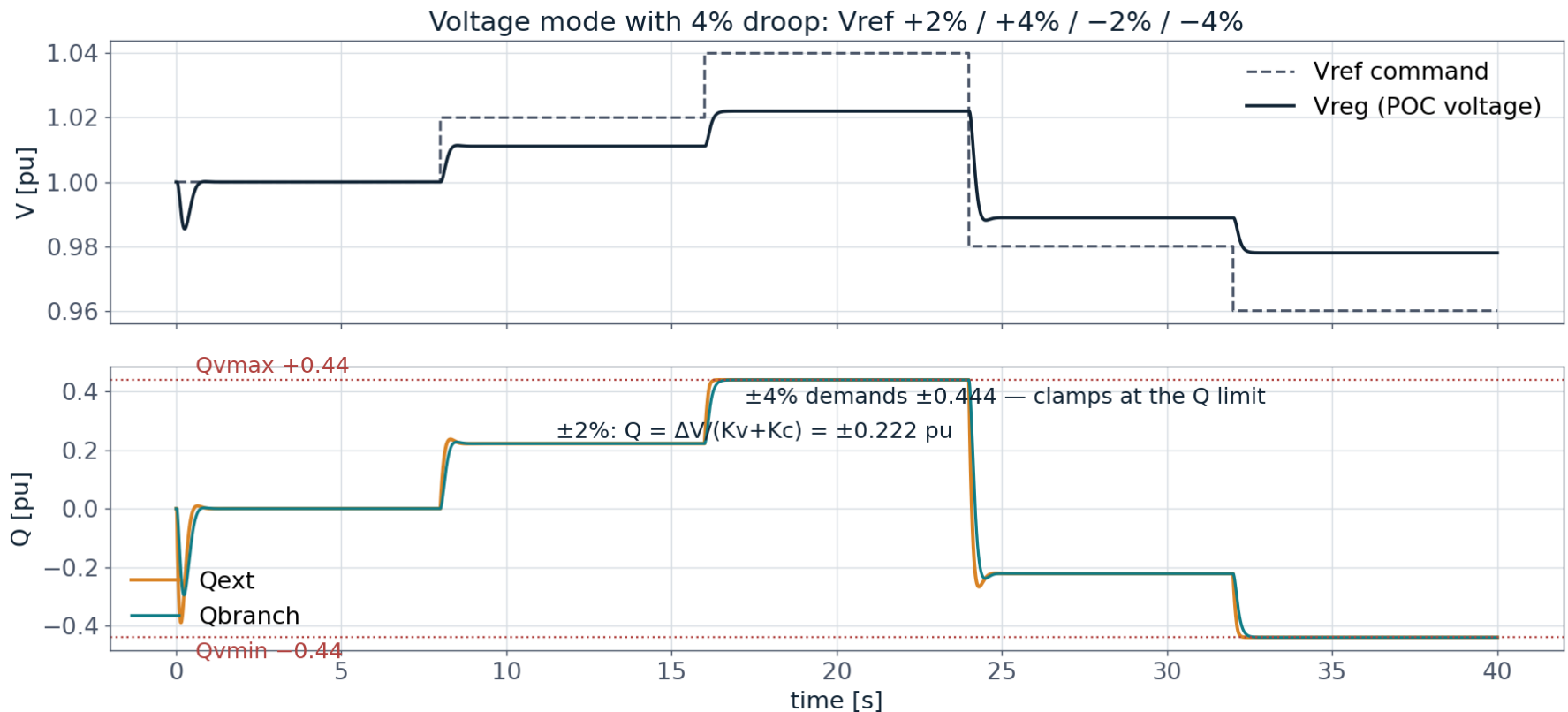
Q Setpoint – Fixed MVAR

- Testing of simple MVAR setpoints, using the Qref command.
- Power factor control was also tested (not shown)



Voltage Control

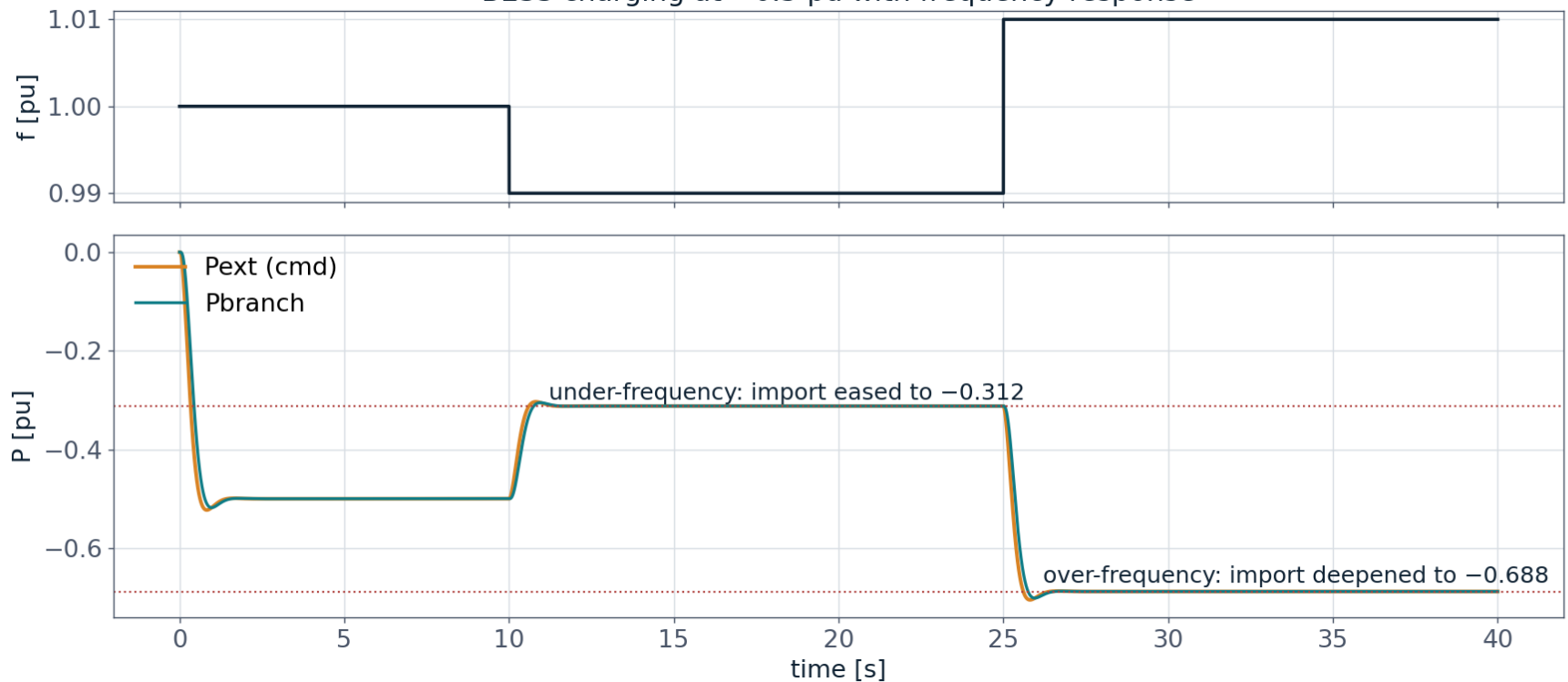
- Testing of voltage control, by adjusting the input voltage reference
- Controller set on a typical 4% voltage droop



Frequency Response

- Testing of frequency response whilst the site is set to import mode.
- Negative and positive frequency step

BESS charging at -0.5 pu with frequency response



Summary

- The use of AI was a huge help in this and is very much playing to the AI strengths, of mathematical modelling and a direct MCP plug-in to PSCAD.
- Using AI to write FORTRAN code directly into a submodule in PSCAD, then define the input and output ports and produce and evaluate a test plan, is where the AI and power systems blend really starts to get interesting.
- The bit that surprised me the most was Claude deciding that trying to build the test ladders and input signals with conventional signal blocks was too messy and came up with a clever bit of custom code.
- If this was a real project and more complex control system, trying to do this by hand would have been expensive and cost \$\$\$.
- Next steps are to produce a REEC model of an inverter and then string everything together in a full WECC aggregated model of a simple IBR plant.