vreg

December 11, 2023

1 Autonomously Adjusting Reference Voltage

The purpose of autonomously adjusting reference voltage (AARV) is to mitigate rapid voltage change (RVC), without affecting the utility's control of steady-state voltage or the DER's steady-state reactive power. AARV may be used with a volt-watt function that mitigates steady-state overvoltage, but otherwise AARV does not regulate steady-state voltage.

Page 39 of IEEE 1547-2018 says: *The DER shall be capable of autonomously adjusting reference voltage (VRef) with VRef being equal to the low pass filtered measured voltage. The time constant shall be adjustable at least over the range of 300 s to 5000 s. The voltage-reactive power Volt-Var curve characteristic shall be adjusted autonomously as VRef changes. The approval of the Area EPS operator shall be required for the DER to autonomously adjust the reference voltage. Implementation of the autonomous VRef adjustability and the associated time constant shall be specified by the Area EPS operator.*

1.1 Historical Background

AARV was identified in the course of an applied research project on optimal smart inverter settings, which proved sensitive to location and size of the DER (2015 paper). AARV seemed to offer a uniform default setting approach that could be easier to apply for DER, and it was included in IEEE 1547-2018, but not specified as a default. The AARV function was implemented in the *ExpControl* component in *OpenDSS*, but using system parameters like gain, deadband, and reactive power bias instead of the V1..V4 and Q1..Q4 points (2019 paper). A companion notebook on this site shows how the system parameters are translated to the table parameters required in IEEE 1547-2018. In the mitigation of RVC, several examples have shown that AARV outperforms a static volt-var characteristic, with less reactive power needed from the DER (2019, 2023 papers).

References:

- 2015 Conference Paper: https://doi.org/10.1109/PESGM.2015.7286560
- 2019 Conference Paper: https://doi.org/10.1109/PVSC40753.2019.8981277
- 2023 Conference Paper: https://doi.org/10.1109/PESGM52003.2023.10252317

1.2 Code Samples

There are Python code cells in this notebook that create plots and text outputs to illustrate various points of application, implementation, and testing. A live version of this notebook is available at gridhub, under an open-source license and copyright that permit reuse and modification. Run the following Python code cell to define support functions using Numpy and Matplotlib.

```
[1]: import sys
     import os
     import matplotlib.pyplot as plt
     import numpy as np
     import math

     # convert center (Vref), deadband, slope, q limits, and q bias into a table of␣
      ↪V and Q points.
     # the function returns two arrays for the V and Q points
     #    the arrays have sentinel elements below V1 and above V4, so they are 6␣
      ↪elements long (not 4)
     #    the sentinel elements clarify that constant extrapolation is used outside␣
      ↪the range [V1..V4]
     #    this version of the function does not check any limits - the caller must␣
      ↪do that
     def set_characteristic (Vref=1.0, deadband=0.0, slope=22.0, qmax=0.44, qmin=-0.
      ↪44, qbias=0.0):
       Q1 = qmax
       Q2 = qbias
       Q3 = qbias
       Q4 = qmin
       V2 = Vref - 0.5 * deadband
       V3 = Vref + 0.5 * deadband
       V1 = V2 - (Q1 - Q2) / slope
       V4 = V3 - (Q4 - Q3) / slope
       VL = V1 - 0.01
       VH = V4 + 0.01
       vtable = np.array ([VL, V1, V2, V3, V4, VH])
       qtable = np.array ([Q1, Q1, Q2, Q3, Q4, Q4])
       return vtable, qtable

     # This function plots a volt-var characteristic, and returns the VQ points.
     # (Tabular display is not needed in this notebook.)
     def show_characteristic (label, Vref, deadband, slope, qmax, qmin, qbias=0):
       vtable, qtable = set_characteristic (Vref, deadband, slope, qmax, qmin, qbias)

       # bounds for plotting the horizontal axis
       vmin = vtable[0]-0.01
       vmax = vtable[-1]+0.01

       # evaluate the characteristic over 500 equal voltage intervals
       v = np.linspace (vmin, vmax, 501)
       # interpolating Q using the numpy library function
       q = np.interp (v, vtable, qtable)

       # create the plot
       fig, ax = plt.subplots(1, 1, figsize=(6,4), tight_layout=True)
```

```
  fig.suptitle ('{:s} volt-var characteristic'.format (label))
  ax.plot (vtable, qtable, marker='o', color='blue', label='Points and␣
 ↪Sentinels')
  ax.plot (v, q, color='red', label='Interpolated')
  ax.grid ()
  ax.set_xlabel ('V [pu]')
  ax.set_ylabel ('Q [pu]')
  ax.set_xlim (vmin, vmax)
  ax.legend ()

  plt.show ()
  return vtable, qtable

# This function calculates V and dV at the POC from the formula
# in clause 5.1.2 of IEEE 1547.2-2023
# All input and output in per-unit
def get_voltage (rpu, xpu, dP, dQ, vsrcpu):
  a1 = vsrcpu*vsrcpu + rpu*dP + xpu*dQ
  a2 = xpu*dP - rpu*dQ
  d = math.sqrt (a1*a1 + a2*a2) / vsrcpu / vsrcpu - 1.0
  vpu = vsrcpu + d
  return vpu, d

# use the current directory as default location for the "save plot" buttons
# optimize the graphic export for LaTex and PDF
plt.rcParams['savefig.directory'] = os.getcwd()
plt.rcParams['savefig.pad_inches'] = 0.05
plt.rcParams['savefig.dpi'] = 300.0
plt.rcParams['savefig.bbox'] = 'tight'
# invoke the Jupyter support for  Matplotlib graphics
%matplotlib widget
```

1.3 Example Data

Run the following Python code cell to define and example data for:

- A 6-MW battery energy storage system (BESS) for testing the AARV response to system conditions. At 1 pu source voltage and full power injection, this BESS will create a steady-state voltage of 1.0462 pu at the point of connection. If it turns on suddenly to full power injection, the RVC would be 0.0462 pu, which is greater than the allowed 0.03 pu (3%) at medium voltage. See the 2023 conference paper reference for more details.
- A category B volt-var characteristic with maximum allowed slope and zero deadband. See the other notebook, *vqpoints*, for the tabulated V and Q points if needed.

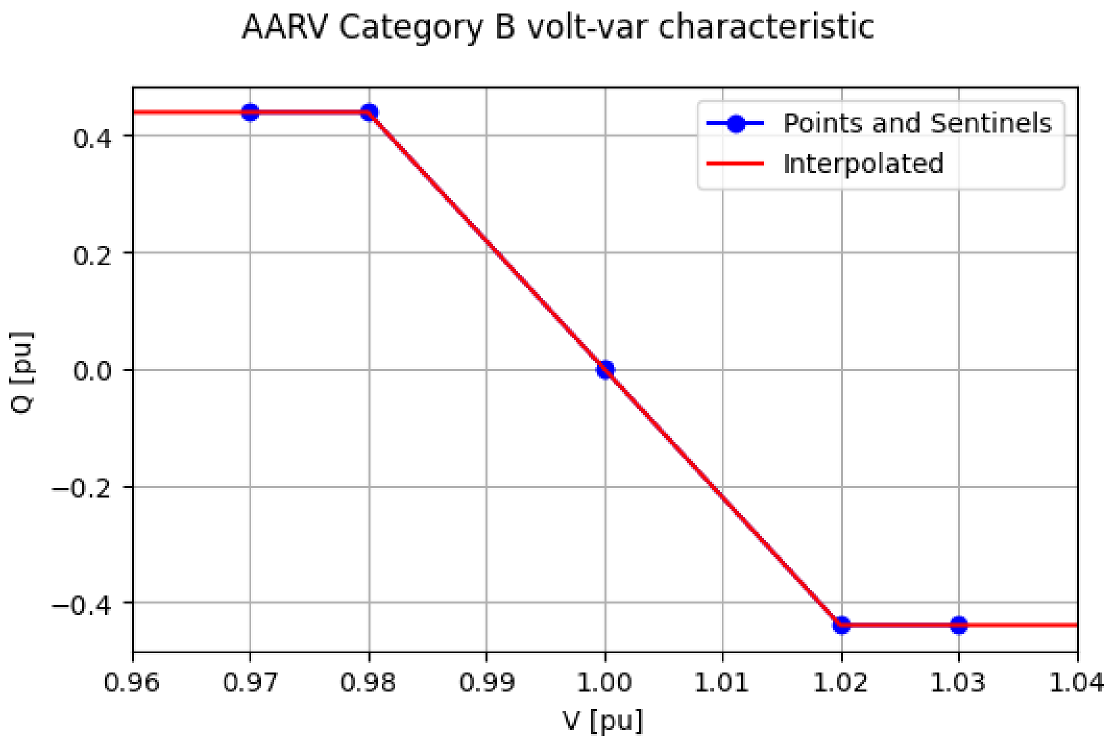
[2]:
```
# GLOBAL variables for the Nantucket BESS example; only the volt-var parameters␣
 ↪will change in most of the notebook examples
SBASE = 6.0e6
VBASE = 13200.0
```

```
ZBASE = VBASE*VBASE/SBASE
RPU = 1.210 / ZBASE # 1.744
XPU = 2.8339 / ZBASE # 2.6797
DP = 6.0e6 / SBASE
DQ = 0.0 / SBASE
VSRC = 1.0

vpoc, d = get_voltage (RPU, XPU, DP, DQ, VSRC)
print ('Nantucket BESS example for dP={:.3f}, dQ={:.3f}, R={:.3f}, X={:.3f} pu:␣
 ↪Vpoc={:.4f}, dV={:.4f} pu'.format (DP, DQ, RPU, XPU, vpoc, d))

# plot the default characteristic used for AARV examples
vtable, qtable = show_characteristic ('AARV Category B', Vref=1, deadband=0,␣
 ↪slope=22, qmax=0.44, qmin=-0.44)
```

```
Nantucket BESS example for dP=1.000, dQ=0.000, R=0.042, X=0.098 pu: Vpoc=1.0462,
dV=0.0462 pu
```



1.4 AARV Test from 1547.1-2020

Clause 5.14.5 of IEEE 1547.1-2020 describes a specific test for AARV. In all other tests, AARV is to be disabled. Four AARV tests are described, at the time constant values *Tref* = 300s and 5000s, each of those with source voltage steps to (V1+V2)/2 and (V3+V4)/2. Before each voltage step, the steady-state reactive power, Q, should be zero at V=Vref.

The voltage steps result in step changes of Q to 0.5*Q1 and 0.5*Q4, respectively, with some delay from the open-loop response. These new Q values should follow an exponential decay back to zero, governed by *Tref* = 300s or 5000s. The test criteria specifies that after one time constant, *Tref*, the value of Q should be less than 10% of Q1 or Q4, respectively. However, the expected value of Q, following an exponential decay from the stepped value, is Q = 0.5*Q1*exp(-Tref/Tref) = 0.1839*Q1 (or 0.1839*Q4). **The evaluation criteria at 1*Tref should be 20% instead of 10% of Q1 or Q4**. Note: at 2*Tref, the expected value of Q is 0.0677*Q1 or 0.0677*Q4.

Tref is defined as the time constant for AARV reference voltage. On the other hand, *Tresponse* is defined as an open-loop response time, which is converted to an exponential time constant in the Python code below. Furthermore, the simulation is run at a constant time step, *dt*, so both time constants may be implemented as constant decrement factors. If the real hardware controller operates at a constant sample interval, it may also use constant decrement factors to save the time of repeatedly evaluating the exponential functions.

Passing this test verifies implementation of *Tref*, but not the impact of AARV on a system with grid impedance.

1.4.1 Simulation of AARV Tests

Run the following Python code cell to define a function that simulates and plots an AARV test. The input parameters are:

- *tag* is a text label to appear in the plot.
- *Vref*, *dB*, *K*, *Qmax*, and *Qmin* are the system parameters to define the volt-var characteristic
- *Tref* is the AARV time constant, which ranges from 300s to 5000s. No default value was provided in IEEE 1547, but 300s works well whenever AARV is enabled.
- *Tresponse* is the open-loop response time, which ranges from 1s to 90s. The default in IEEE 1547 is 10s for category A and 5s for category B.
- *dt* is the time step for simulation and plotting.
- *bShiftTable* is a flag to choose between two different implementations of AARV:
  - *True* means that the V1..V4 points in the volt-var characteristic are shifted each time *VRef* changes. This procedure strictly follows language in the footnote in IEEE 1547-2018, and it performs as intended in simulation. However, it's not efficient and it may lose track of the original settings for V1..V4.
  - *False* means that instead of modifying V1..V4, the voltage used for interpolation is adjusted with an offset to the **original** value of *Vref*, denoted as *Vset* in the code. This method also performs as intended, without regenerating the interpolation table at each time step, and without losing track of the original settings for V1..V4. In the Python function signature, the default value is *False* to indicate that this might be the preferred implementation. (It is also close to the way *OpenDSS* implements AARV).

The function runs for one time constant, *Tref*, and it provides numerical outputs of the peak and final Q. The test procedure ensures that limits on *Vref* will be met, so the function does not check the limits during execution.

[3]:
```
def show_aarv_test (tag, Vref, dB, K, Qmax, Qmin, Tref, Tresponse, dt,␣
 ↪bShiftTable=False):
  TauOL = Tresponse / 2.3026
  IncRef = 1.0 - math.exp(-dt/Tref)
```

```
IncOL = 1.0 - math.exp(-dt/TauOL)
vtable, qtable = set_characteristic (Vref, dB, K, Qmax, Qmin)

vtest = [(vtable[1]+vtable[2])/2.0, (vtable[3]+vtable[4])/2.0]
qtest = np.interp (vtest, vtable, qtable)

tmax = 1.0 * Tref
t = np.linspace (0, tmax, int(tmax / dt) + 1)
npts = len(t)
v0 = (vtable[1]+vtable[2])/2.0
q0 = np.interp(v0, vtable, qtable)
q = np.zeros(npts)
qpeak = 0.0
# Keep track of the initial Vref setting for 1) bShiftTable=False, and 2)␣
↪plotting the original volt-var characteristic
Vset = Vref
#print ('       t       Vpu      Vref      Verr     Qtarg         Q')
if bShiftTable: # this approach resets the V1..V4 points each time Vref␣
↪changes
  for i in range(1, npts):
    # apply the reference voltage time constant
    verr = v0 - Vref
    Vref = Vref + verr * IncRef
    #######################################################
    # calculate Q with shifted interpolation table
    vtable, qtable = set_characteristic (Vref=Vref, deadband=dB, slope=K,␣
↪qmax=Qmax, qmin=Qmin)
    qtarg = np.interp (v0, vtable, qtable)
    #######################################################
    # apply the open-loop response
    q[i] = q[i-1] + (qtarg - q[i-1]) * IncOL
    if abs(q[i]) > qpeak:
      qpeak = abs(q[i])
else: # this approach modifies the voltage entry point
  for i in range(1, npts):
    # apply the reference voltage time constant
    verr = v0 - Vref
    Vref = Vref + verr * IncRef
    #######################################################
    # calculate Q with offset voltage entry point
    qtarg = np.interp (v0 + (Vset - Vref), vtable, qtable)
    #######################################################
    # apply the open-loop response
    q[i] = q[i-1] + (qtarg - q[i-1]) * IncOL
    if abs(q[i]) > qpeak:
      qpeak = abs(q[i])
    #print ('{:8.3f} {:8.5f} {:8.5f} {:8.5f} {:8.5f} {:8.5f}'.format (t[i], v0,␣
↪vref, verr, qtarg, q[i]))

#  print ('{:d} test points, IncRef={:.6f}, IncOL={:.6f}, Qpeak={:.3f}, Qend={:.
↪3f}'.format (npts, IncRef, IncOL, qpeak, q[-1]))

  fig, ax = plt.subplots(1, 2, sharex = 'col', figsize=(10,4),␣
↪constrained_layout=True)
  fig.suptitle ('1547.1-2020 AARV test, {:s}, Tref={:.1f}, Tresponse={:.1f},␣
↪Voltage Step to Vs={:.3f}'.format (tag, Tref, Tresponse, v0))

  ax[0].set_title ('Characteristic, K={:.2f}, dB={:.2f}, Qmax={:.2f}'.format␣
↪(K, dB, Qmax))
  # remove any shift of vtable, qtable that might have been done while␣
↪simulating the test
  vtable, qtable = set_characteristic (Vset, dB, K, Qmax, Qmin)
  ax[0].plot (vtable, qtable, marker='o', color='blue', label='Points and␣
↪Sentinels')
  ax[0].plot (vtest, qtest, color='red', marker='s', linestyle='None',␣
↪label='Test Conditions')
  ax[0].set_xlabel ('V [pu]')
  ax[0].set_xlim (vtable[0]-0.01, vtable[-1]+0.01)
  ax[0].legend()

  ax[1].set_title ('Peak Q={:.4f}, Final Q={:.4f} pu'.format (qpeak, q[-1]))
  ax[1].plot (t, q, color='red', label='Dynamic')
  ax[1].plot (t, q0*np.ones(npts), color='blue', label='Qs={:.3f}'.format(q0))
  ax[1].set_xlabel ('t [s]')
  ax[1].set_xlim (t[0], t[-1])

  for i in range(2):
    ax[1].legend()
    ax[i].grid()
    ax[i].set_ylabel ('Q [pu]')

  plt.show()
```

1.4.2 Examples of AARV Tests

Run the following Python code cell to run five example AARV tests.

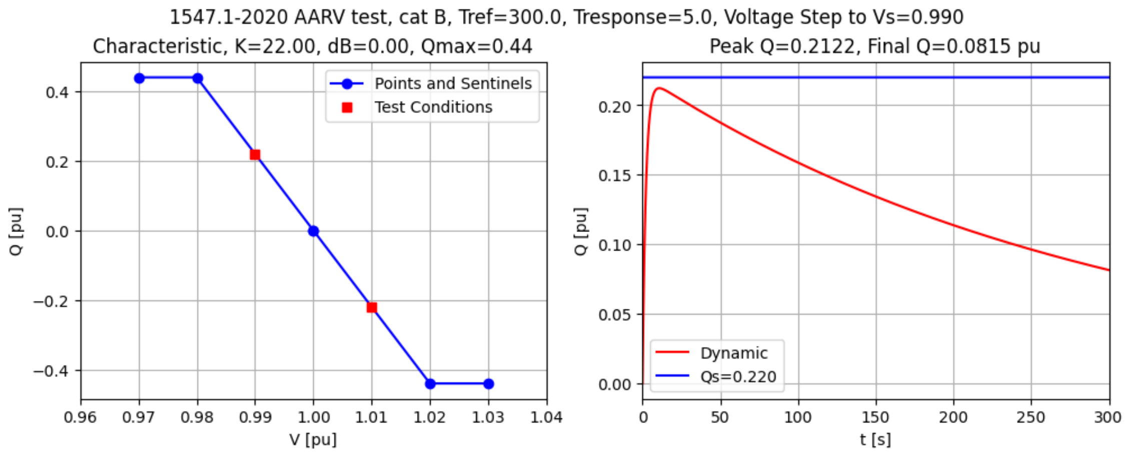
- The first example simulates one AARV test, for the step to (V1+V2)/2, and category B with maximum allowed slope and no deadband. The steady-state value of Q at this voltage would be 0.22 pu, but it only reaches 0.2122 pu due to the open-loop response time constant. The value of Q at *Tref* is 0.0815 pu, which is 18.52% of Q1. (Due to the open-loop response, this is higher than the static expected value of 18.39% at t=Tref).
- The second example repeats the first AARV test with *bShiftTable=True*, and the results

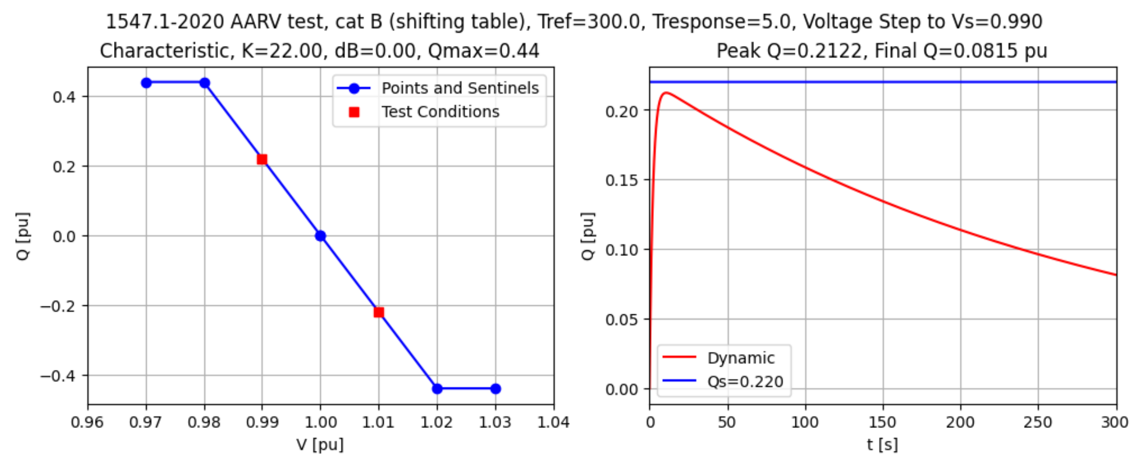
match.

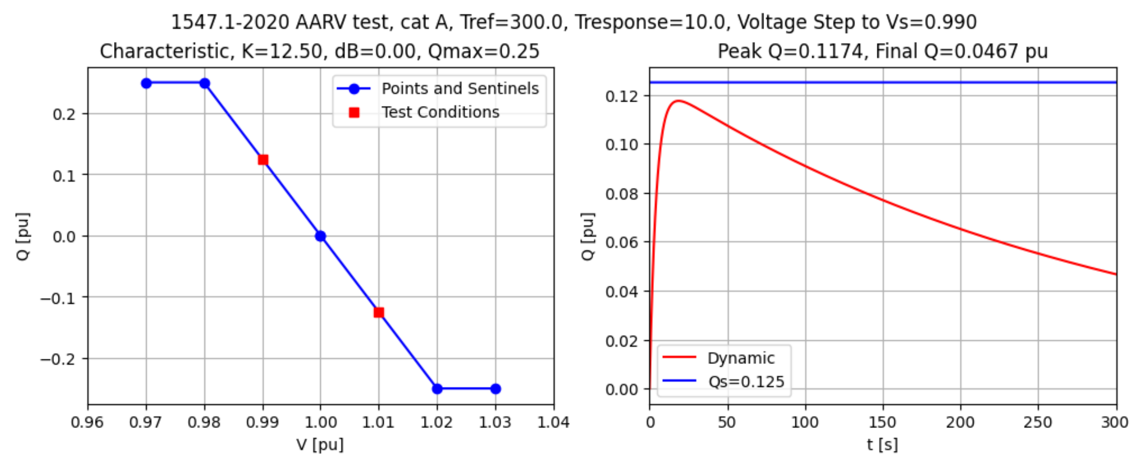
- The third example shows an AARV test for category A at maximum slope, and the longer default open-loop *Tresponse* for category A. The value of *Qpeak* is 93.92% of *Qs*, compared to 96.45% of *Qs* in the previous two examples, due to the longer open-loop response time. The final value of Q is 18.68% of Q1.
- The fourth example shows an AARV test for category B at minimum open-loop *Tresponse*. The value of *Qpeak* is 99.05% of *Qs* and the final value of Q is 18.41% of Q1.
- The fifth example shows an AARV test for category B at maximum open-loop *Tresponse*. The value of *Qpeak* is 73.68% of *Qs* and the final value of Q is 21.11% of Q1.

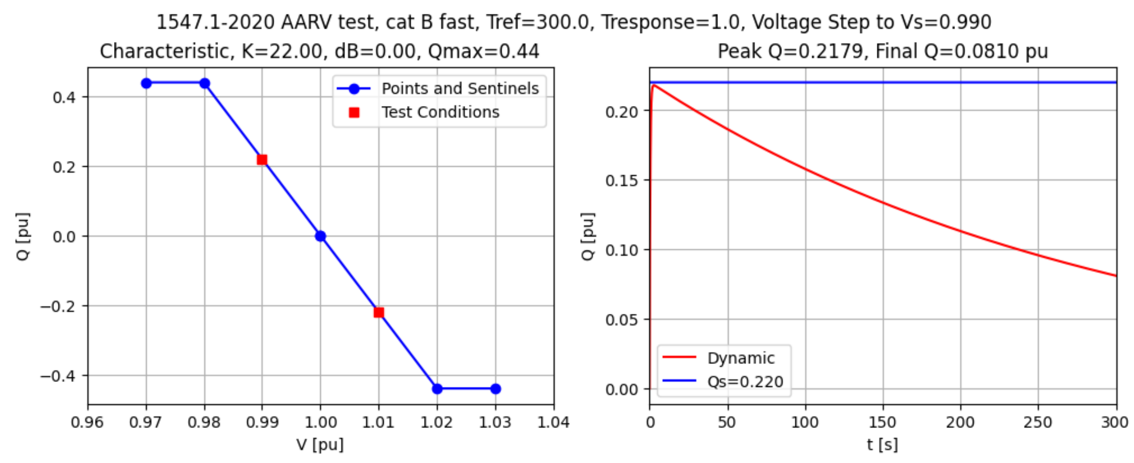
From these results, *Tresponse* in the range 1s to 10s might be preferrable to longer times.

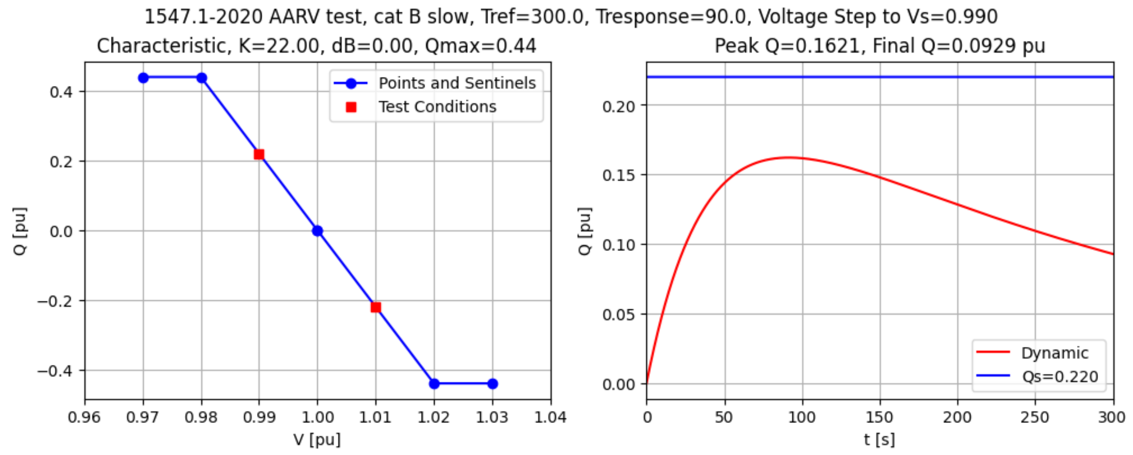
[4]:
```
show_aarv_test ('cat B', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
 ↪Tresponse=5, dt=0.1)  # for most aggressive category B
show_aarv_test ('cat B (shifting table)', Vref=1, dB=0, K=22, Qmax=0.44,␣
 ↪Qmin=-0.44, Tref=300, Tresponse=5, dt=0.1, bShiftTable=True)  # for most␣
 ↪aggressive category B
show_aarv_test ('cat A', Vref=1, dB=0, K=12.5, Qmax=0.25, Qmin=-0.25, Tref=300,␣
 ↪Tresponse=10, dt=0.1) # for most aggressive category A
show_aarv_test ('cat B fast', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44,␣
 ↪Tref=300, Tresponse=1, dt=0.1)  # category B with lowest Tresponse
show_aarv_test ('cat B slow', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44,␣
 ↪Tref=300, Tresponse=90, dt=0.1) # category B with highest Tresponse
```




1547.1-2020 AARV test, cat B (shifting table), Tref=300.0, Tresponse=5.0, Voltage Step to Vs=0.990
Characteristic, K=22.00, dB=0.00, Qmax=0.44
Points and Sentinels
Test Conditions
Q [pu]
V [pu]
Peak Q=0.2122, Final Q=0.0815 pu
Dynamic
Qs=0.220
t [s]


1547.1-2020 AARV test, cat A, Tref=300.0, Tresponse=10.0, Voltage Step to Vs=0.990
Characteristic, K=12.50, dB=0.00, Qmax=0.25
Points and Sentinels
Test Conditions
Q [pu]
V [pu]
Peak Q=0.1174, Final Q=0.0467 pu
Dynamic
Qs=0.125
t [s]


1547.1-2020 AARV test, cat B fast, Tref=300.0, Tresponse=1.0, Voltage Step to Vs=0.990
Characteristic, K=22.00, dB=0.00, Qmax=0.44
Points and Sentinels
Test Conditions
Q [pu]
V [pu]
Peak Q=0.2179, Final Q=0.0810 pu
Dynamic
Qs=0.220
t [s]



1.5 AARV Response to Storage Power-On Ramping

Run the following Python code cell to define a simulation of full power-on events of the example 6-MW BESS with AARV. Compared to the *show_aarv_test* function, new input parameters are:

- *numTrefs* determines how long to run the simulation. As the terminal voltage changes, it may take longer than expected for the DER's reactive power to settle.
- *RampTime* defines the steepness of the power ramp. In *show_aarv_test*, the source voltage changes in a single time step, e.g., in 100 ms. Because the open loop response time ranges from 1s to 90s, the BESS reactive power could not respond effectively to faster ramps in power, whether or not AARV is enabled.
- *Qbias* allows for a non-zero steady-state reactive power

The output includes the maximum value of *Verr*, i.e., the maximum difference between *Vref* and the terminal voltage. This maximum *Verr* is approximately equal to the RVC in per unit. This function enforces limits on *Vref* because grid voltage fluctuations aren't controlled as they are in the lab for IEEE 1547.1 test procedures.

[5]:
```
def show_bess_test (tag, Vref, dB, K, Qmax, Qmin, Tref, Tresponse, dt=0.1,␣
 ↪numTrefs=1, RampTime=5, bShiftTable=False, Qbias=0.0):
  TauOL = Tresponse / 2.3026
  if Tref > 0.0:
    IncRef = 1.0 - math.exp(-dt/Tref)
    tmax = numTrefs * Tref
  else: # Tref equal to zero disables AARV
    IncRef = 0.0
    tmax = 300.0
  IncOL = 1.0 - math.exp(-dt/TauOL)
  vtable, qtable = set_characteristic (Vref, dB, K, Qmax, Qmin, Qbias)

  t = np.linspace (0, tmax, int(tmax / dt) + 1)
  npts = len(t)
```

```
vsrc = np.ones(npts)
vpoc = np.ones(npts)
pstart = 1.0
# create a linear ramp in real power
pend = pstart + RampTime
p = np.interp(t, [0.0, pstart, pend, t[-1]], [0.0, 0.0, DP, DP])

q = np.zeros(npts)
vref = Vref * np.ones(npts) # array to be updated at each time step, note␣
↪that scalar Vref will not be updated
verr = np.zeros(npts)

for i in range(npts-1):
  vpoc[i], d = get_voltage (RPU, XPU, p[i], q[i], vsrc[i])
  verr[i] = vpoc[i] - vref[i]
  vref[i+1] = vref[i] + verr[i] * IncRef
  # limiting vref to allowable range
  if vref[i+1] > 1.05:
    vref[i+1] = 1.05
  elif vref[i+1] < 0.95:
    vref[i+1] = 0.95
  if bShiftTable: # this approach resets the V1..V4 points each time the␣
↪current value of reference voltage changes
    vtable, qtable = set_characteristic (vref[i], deadband=dB, slope=K,␣
↪qmax=Qmax, qmin=Qmin)
    qtarg = np.interp (vpoc[i], vtable, qtable)
  else: # this approach modifies the voltage entry point, as an offset from␣
↪the original reference voltage
    qtarg = np.interp (Vref + verr[i], vtable, qtable)
  q[i+1] = q[i] + (qtarg - q[i]) * IncQL
# solve for plotting the last voltage points in the time series
vpoc[-1], d = get_voltage (RPU, XPU, p[-1], q[-1], vsrc[-1])
verr[-1] = vpoc[-1] - vref[-1]

fig, ax = plt.subplots(1, 2, sharex = 'col', figsize=(10,4),␣
↪constrained_layout=True)
fig.suptitle ('Time Response for {:s}, Max Verr={:.4f}, Final Vref={:.4f},␣
↪Final Vpoc={:.4f}'.format (tag, np.max(verr), vref[-1], vpoc[-1]))
ax[0].plot (t, p, label='P')
ax[0].plot (t, q, label='Q')
ax[1].plot (t, vsrc, label='Vsrc')
ax[1].plot (t, vpoc, label='Vpoc')
ax[1].plot (t, vref, label='Vref')
#ax[1].plot (t, verr, label='Verr')
for i in range(2):
  ax[i].set_ylabel ('[pu]')
```

```
    ax[i].set_xlabel ('[s]')
    ax[i].set_xlim (t[0], t[-1])
    ax[i].legend()
    ax[i].grid()
  plt.show()
```

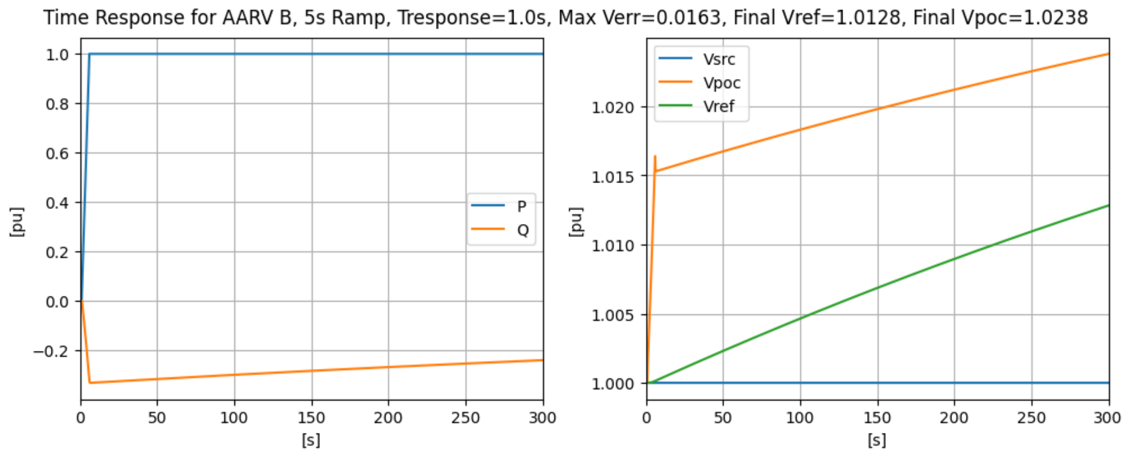
1.6 AARV Category B: Effect of Open-Loop Response

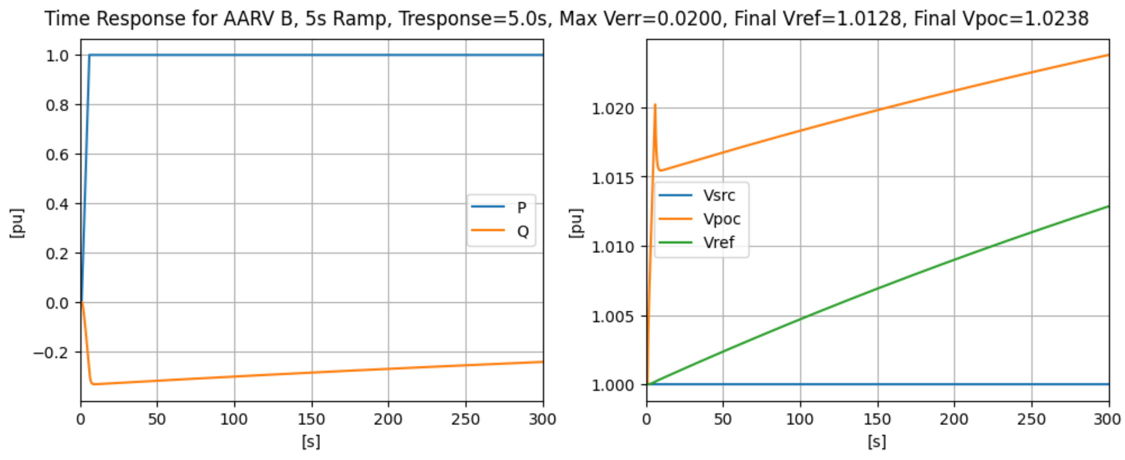
Run the following Python code cell to show these examples, with a 5s ramp in real power at the BESS. The RVC without AARV would be 4.62%, as determined above.

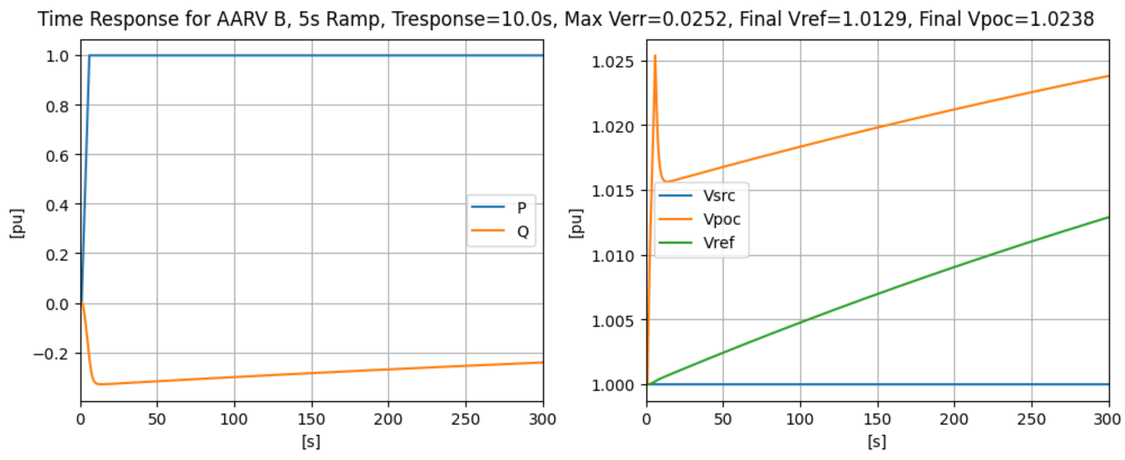
- *1s* (minimum allowed), results in 1.63% RVC, which occurs very early in the event. The terminal voltage, *Vpoc*, continues to increase, but so does *Vref*, such that the difference between them continues to decrease.
- *5s* (category B default), results in 2.00% RVC.
- *10s* (category A default), results in 2.52% RVC.

All three RVC values are within the limit of 3%, and Q does not reach the limit of -0.44 pu for category B.

```
[6]: for tr in [1.0, 5.0, 10.0]:
       tag = 'AARV B, 5s Ramp, Tresponse={:.1f}s'.format (tr)
       show_bess_test (tag, Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
     ↪Tresponse=tr, dt=0.1, numTrefs=1.0, RampTime=5.0)
```







1.7 AARV Category A: Effect of Open-Loop Response

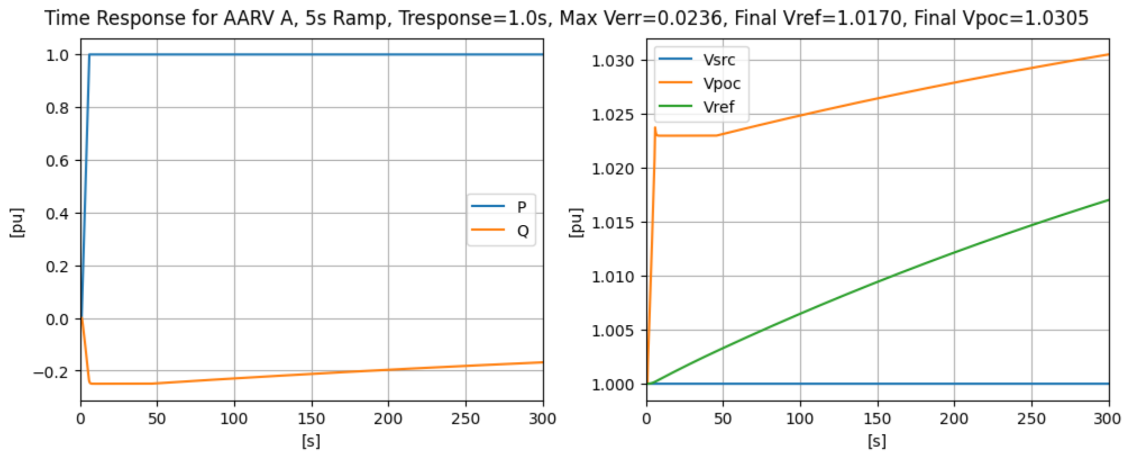
Run the following Python code cell to show these examples, with a 5s ramp in real power at the BESS. The baseline RVC is still 4.62%.

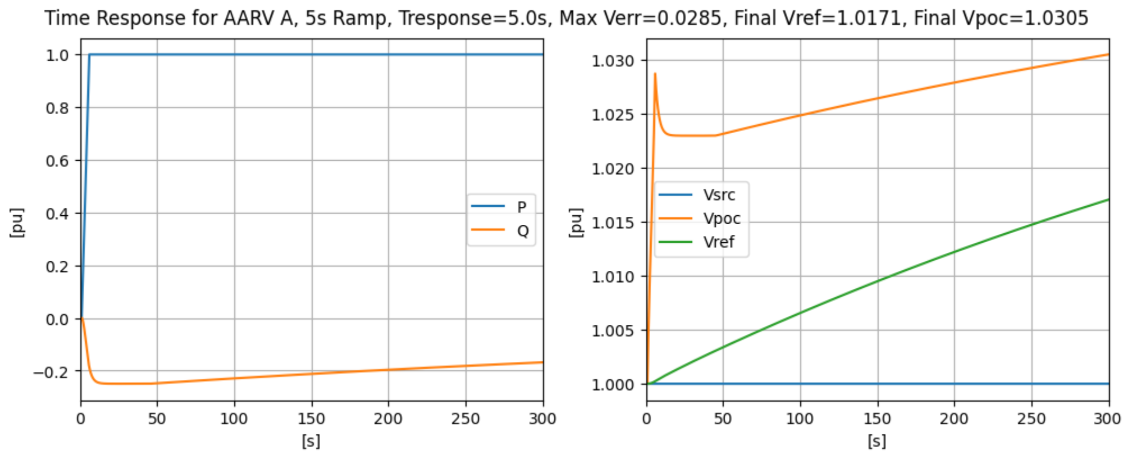
- *1s* (minimum allowed) results in 2.36% RVC.
- *5s* (category B default) results in 2.85% RVC.
- *10s* (category A default) results in 3.32% RVC, which is over the limit.

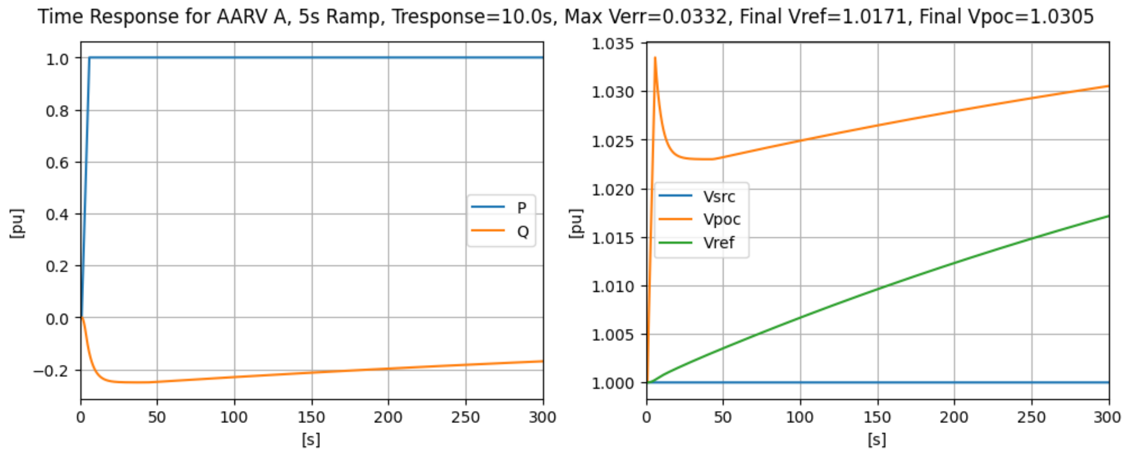
These results are not as good as with category B, because 1) the gain, *K*, is lower and 2) the magnitude of *Qmin* is also lower. From the plots, *Qmin* is reached for all three events with category A.

All of these AARV events have been simulated over a time period equal to *Tref*, but the reactive power does not decay to 37% of its peak value when t=*Tref*, as might have been expected. For example, in the third plot below (*10s* open-loop response), Q peaks at -0.25 pu and the value at *Tref* is about -0.176 pu. This is about 70% of the peak Q value. The reason is that *Vpoc* continues to increase during the event, so the difference between *Vpoc* and *Vref* does not decrease as fast. In the 1547.1 lab test, with near-zero impedance, *Vpoc* would stay constant after the initial step in source voltage.

```
[7]: for tr in [1.0, 5.0, 10.0]:
       tag = 'AARV A, 5s Ramp, Tresponse={:.1f}s'.format (tr)
       show_bess_test (tag, Vref=1, dB=0, K=12.5, Qmax=0.25, Qmin=-0.25, Tref=300,␣
      ↪Tresponse=tr, dt=0.1, numTrefs=1.0, RampTime=5.0)
```







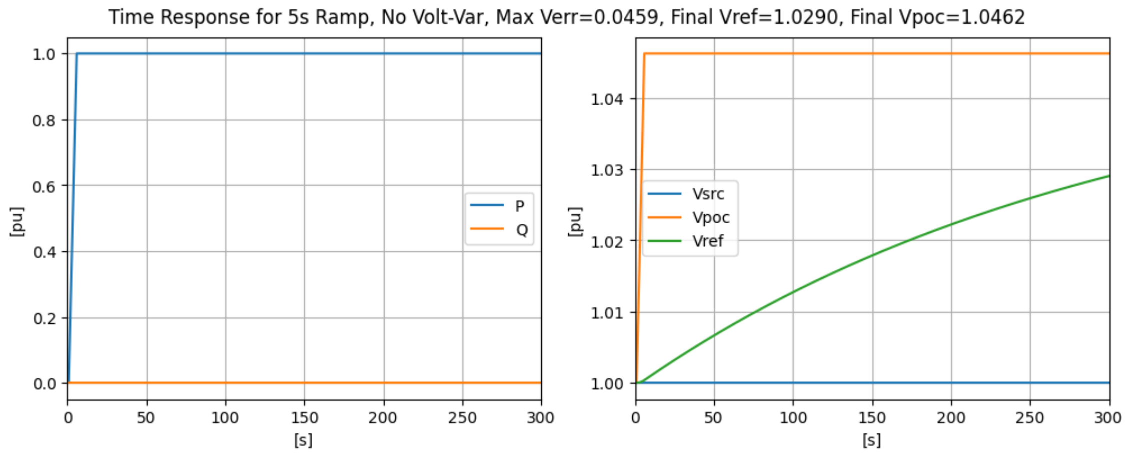
1.8 Examples with No Volt-Var Response

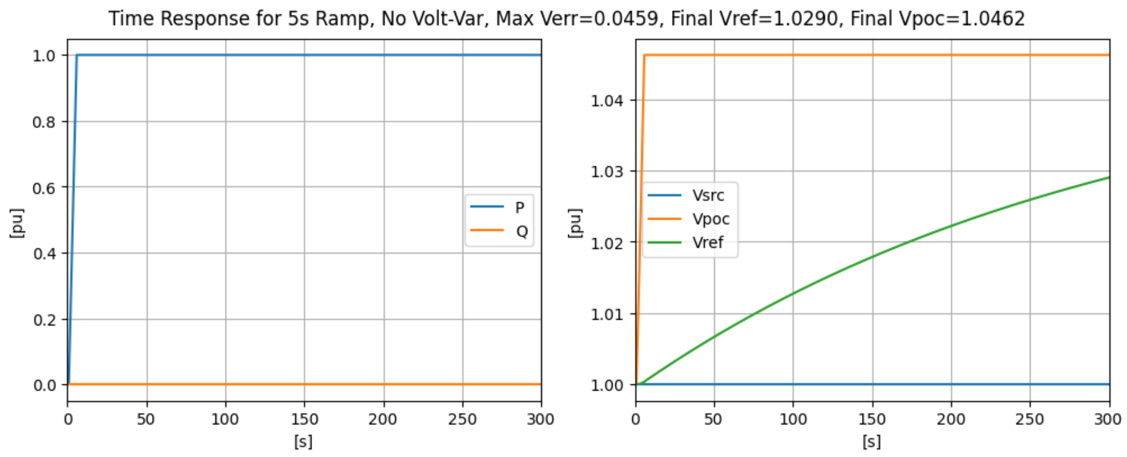
Run the following Python code cell to show the effect of setting *Qmax* and *Qmin* to zero, so the DER cannot supply or absorb reactive power.

- *With Table Shift* produces an RVC of 4.59%, which is over the limit.
- *Without Table Shift* produces an RVC of 4.59%, which is over the limit.

The results match in both cases. The RVC is 4.59% as estimated from the maximum *Verr*, which is a little less than the expected value of 4.62% due primarily to time step discretization.

[8]:
```python
show_bess_test ('5s Ramp, No Volt-Var', Vref=1, dB=0, K=22, Qmax=0, Qmin=0,␣
  ↪Tref=300, Tresponse=5, dt=0.1, numTrefs=1.0, RampTime=5.0, bShiftTable=True)
show_bess_test ('5s Ramp, No Volt-Var', Vref=1, dB=0, K=22, Qmax=0, Qmin=0,␣
  ↪Tref=300, Tresponse=5, dt=0.1, numTrefs=1.0, RampTime=5.0, bShiftTable=False)
```





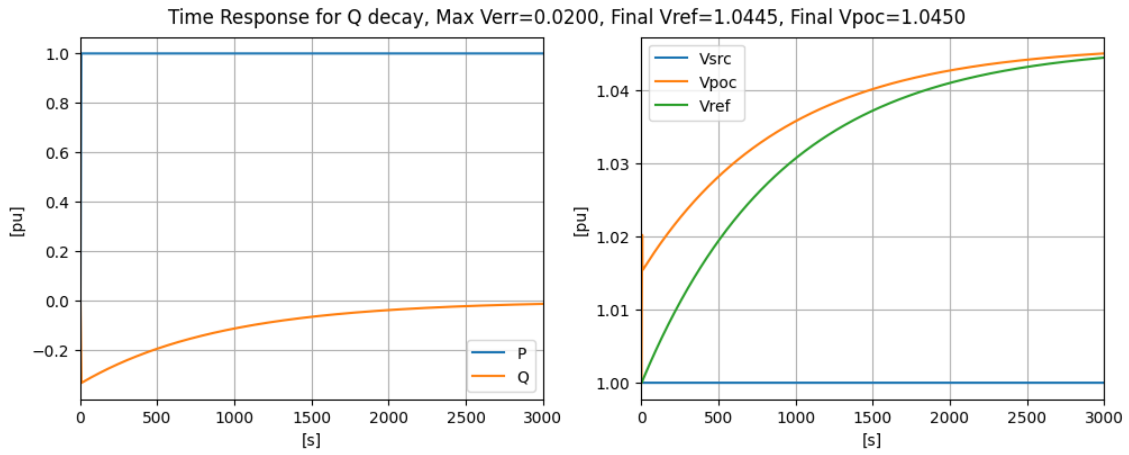
1.9 Long-Running Examples

Run the following Python code cell to show the long decay of DER reactive power when the terminal voltage is not constant, and AARV is enabled.

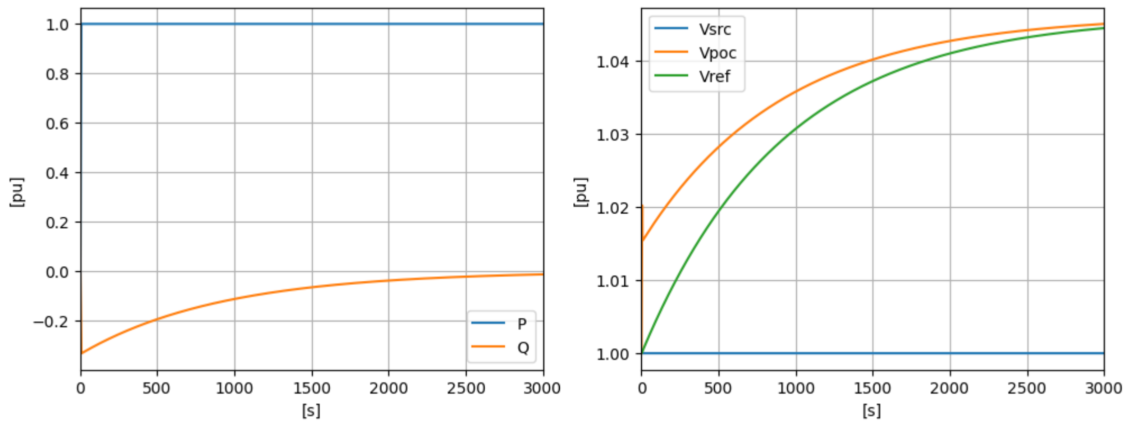
- *With Table Shift* produces an RVC of 2.0%.
- *Without Table Shift* produces an RVC of 2.0%.

The results match in both cases. After 10 time constants (*Tref*) have elapsed, the DER reactive power is close to zero, and *Vpoc* is close to *Vref*.

[9]:
```
show_bess_test ('Q decay', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
 ↪Tresponse=5, dt=0.1, numTrefs=10.0, RampTime=5.0, bShiftTable=True)
show_bess_test ('Q decay', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
 ↪Tresponse=5, dt=0.1, numTrefs=10.0, RampTime=5.0, bShiftTable=False)
```



Time Response for Q decay, Max Verr=0.0200, Final Vref=1.0445, Final Vpoc=1.0450



1.10 Effect of Power Ramp Time

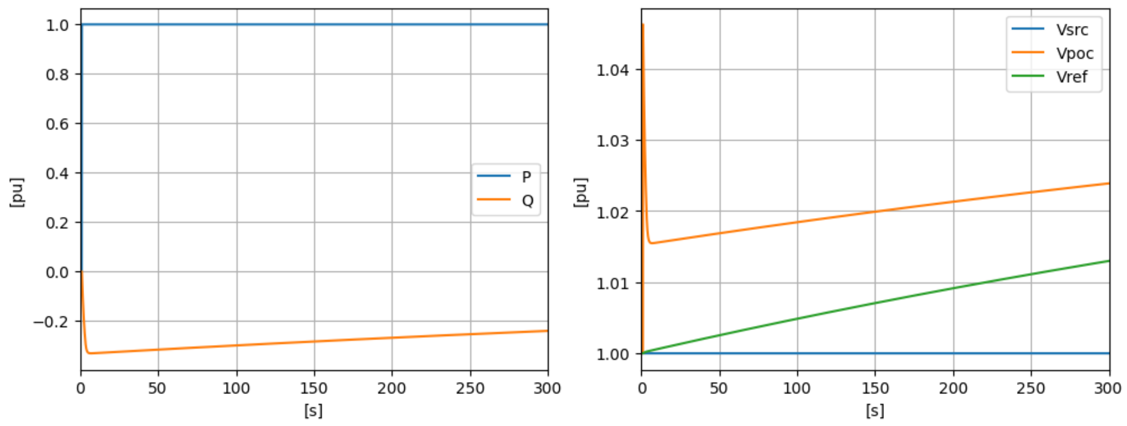
Run the following Python code cell to show the effect of Power ramping time, *RampTime*, when the open-loop response time is held constant at 5s.

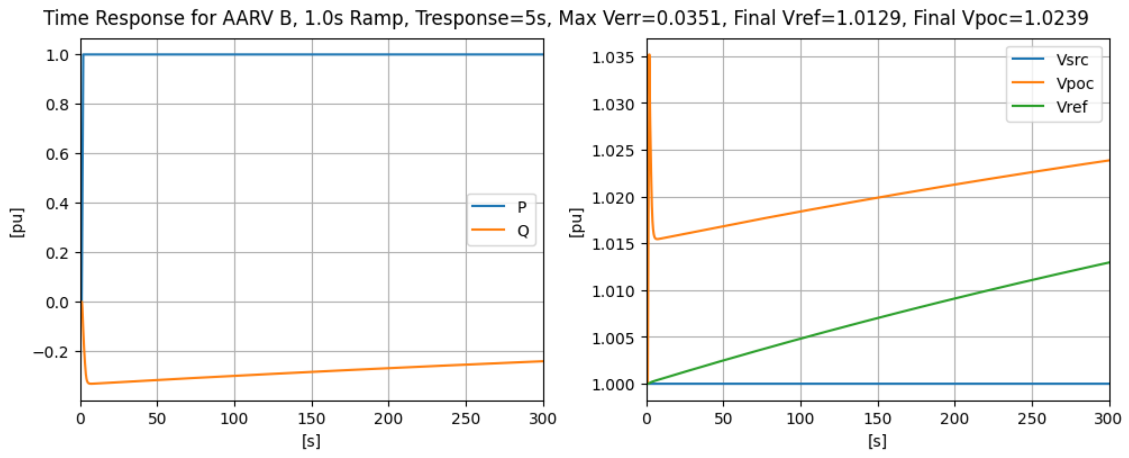
- *0.1s* produces an RVC of 4.62%, which is over the limit and equal to the expected value from static voltage change calculation. There is no time step effect on the RVC estimate, and no benefit of AARV for *RampTime* this fast.
- *1.0s* produces an RVC of 3.51%, which is reduced but still over the limit.
- *5.0s* produces an RVC of 2.00%.
- *10.0s* produces an RVC of 1.76%.

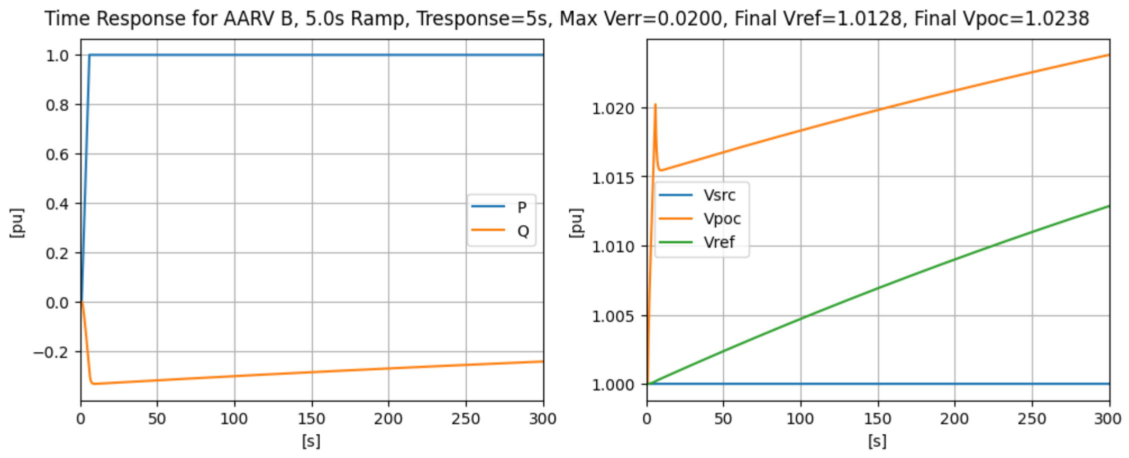
There is little effect on the values at t=*Tref*.

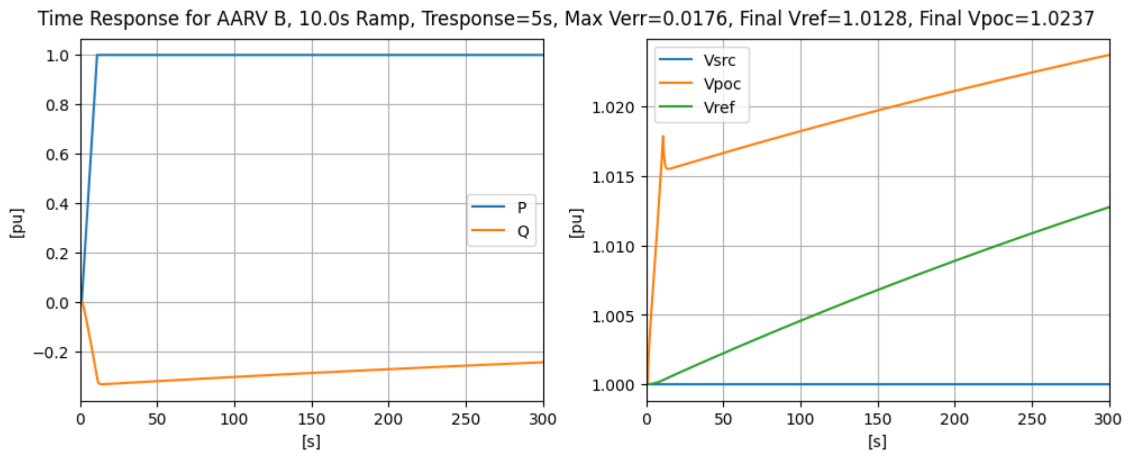
```
[10]: for tramp in [0.1, 1.0, 5.0, 10.0]:
        tag = 'AARV B, {:.1f}s Ramp, Tresponse=5s'.format (tramp)
        show_bess_test (tag, Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
      ↪Tresponse=5.0, dt=0.1, numTrefs=1.0, RampTime=tramp)
```

Time Response for AARV B, 0.1s Ramp, Tresponse=5s, Max Verr=0.0462, Final Vref=1.0130, Final Vpoc=1.0239




Time Response for AARV B, 1.0s Ramp, Tresponse=5s, Max Verr=0.0351, Final Vref=1.0129, Final Vpoc=1.0239
P
Q
Vsrc
Vpoc
Vref
[pu]
[s]


Time Response for AARV B, 5.0s Ramp, Tresponse=5s, Max Verr=0.0200, Final Vref=1.0128, Final Vpoc=1.0238
P
Q
Vsrc
Vpoc
Vref
[pu]
[s]


Time Response for AARV B, 10.0s Ramp, Tresponse=5s, Max Verr=0.0176, Final Vref=1.0128, Final Vpoc=1.0237
P
Q
Vsrc
Vpoc
Vref
[pu]
[s]

1.11 AARV Response to Grid Voltage Fluctuations

Run the following Python code cell to define a simulation of voltage source fluctuations on the example 6-MW BESS with AARV. A new input parameter is:

- *Vinit* initializes the reference point for AARV, in the case when DER real (and reactive) power might not be zero. We still typically call this function with *Vref*=1, but a different value of *Vinit* may initialize the simulation without any startup transient.

The source voltage fluctuation lasts for 12 AARV time constants, and it follows a trapezoidal shape with magnitude +/- 0.02 per-unit around the nominal value of 1.0 per-unit.

[11]:
```
def show_vsrc_test (tag, Vref, dB, K, Qmax, Qmin, Tref, Tresponse, dt=0.1,␣
↪Vinit=1.0, bShiftTable=False, Qbias=0.0):
  TauOL = Tresponse / 2.3026
  if Tref > 0.0:
    IncRef = 1.0 - math.exp(-dt/Tref)
  else:
    IncRef = 0.0
    Tref = 300.0
  IncOL = 1.0 - math.exp(-dt/TauOL)
  vtable, qtable = set_characteristic (Vref, dB, K, Qmax, Qmin, Qbias)

  tmax = 12.0 * Tref
  t = np.linspace (0, tmax, int(tmax / dt) + 1)
  npts = len(t)

  vsrc = np.interp (t, [0.0, 2*Tref, 3*Tref, 5*Tref, 7*Tref, 9*Tref, 10*Tref,␣
↪t[-1]],
                       [1.0, 1.0,    1.02,   1.02,   0.98,   0.98,   1.0,     1.
↪0])
  # determine initial condition on q and vpoc
  v0 = 1.0
  if IncRef <= 0.0: # no AARV, run 10 iterations on the static volt-var␣
↪characteristic
    q0 = 0.0
    accel = 0.5
    for i in range(10):
      v0, d0 = get_voltage (RPU, XPU, DP, q0, vsrc[0])
      q1 = np.interp (v0, vtable, qtable)
      q0 += accel * (q1 - q0)
  else: # with AARV, just use Qbias and rely on correct specification of Vinit␣
↪for Vref
    q0 = Qbias
    v0, d0 = get_voltage (RPU, XPU, DP, q0, vsrc[0])

  p = DP * np.ones(npts)
  q = q0 * np.ones(npts)
```

```
vpoc = v0 * np.ones(npts)
vref = Vinit * np.ones(npts) # array to be updated, original scalar value␣
↪(Vref) is not updated
verr = np.zeros(npts)

for i in range(npts-1):
  vpoc[i], d = get_voltage (RPU, XPU, p[i], q[i], vsrc[i])
  verr[i] = vpoc[i] - vref[i]
  vref[i+1] = vref[i] + verr[i] * IncRef
  # limiting vref to allowable range
  if vref[i+1] > 1.05:
    vref[i+1] = 1.05
  elif vref[i+1] < 0.95:
    vref[i+1] = 0.95
  if bShiftTable: # this approach resets the V1..V4 points each time Vref␣
↪changes
    vtable, qtable = set_characteristic (vref[i], deadband=dB, slope=K,␣
↪qmax=Qmax, qmin=Qmin)
    qtarg = np.interp (vpoc[i], vtable, qtable)
  else: # this approach modifies the voltage entry point with an offset␣
↪around the original Vref
    qtarg = np.interp (Vref + verr[i], vtable, qtable)
  q[i+1] = q[i] + (qtarg - q[i]) * IncOL
# for plotting the last voltage points in the time series
vpoc[-1], d = get_voltage (RPU, XPU, p[-1], q[-1], vsrc[-1])
verr[-1] = vpoc[-1] - vref[-1]

fig, ax = plt.subplots(1, 2, sharex = 'col', figsize=(10,4),␣
↪constrained_layout=True)
fig.suptitle ('Vsrc Fluctuation Response for {:s}, Max Vref={:.4f},␣
↪Vpoc[Min,Max,Final]=[{:.4f},{:.4f},{:.4f}]'.format (tag, np.max(vref), np.
↪min(vpoc), np.max(vpoc), vpoc[-1]))
ax[0].plot (t, p, label='P')
ax[0].plot (t, q, label='Q')
ax[1].plot (t, vsrc, label='Vsrc')
ax[1].plot (t, vpoc, label='Vpoc')
ax[1].plot (t, vref, label='Vref')
#ax[1].plot (t, verr, label='Verr')
for i in range(2):
  ax[i].set_ylabel ('[pu]')
  ax[i].set_xlabel ('[s]')
  ax[i].set_xlim (t[0], t[-1])
  ax[i].legend()
  ax[i].grid()
plt.show()
```

1.12 Vsrc Fluctuations at Full Power Output

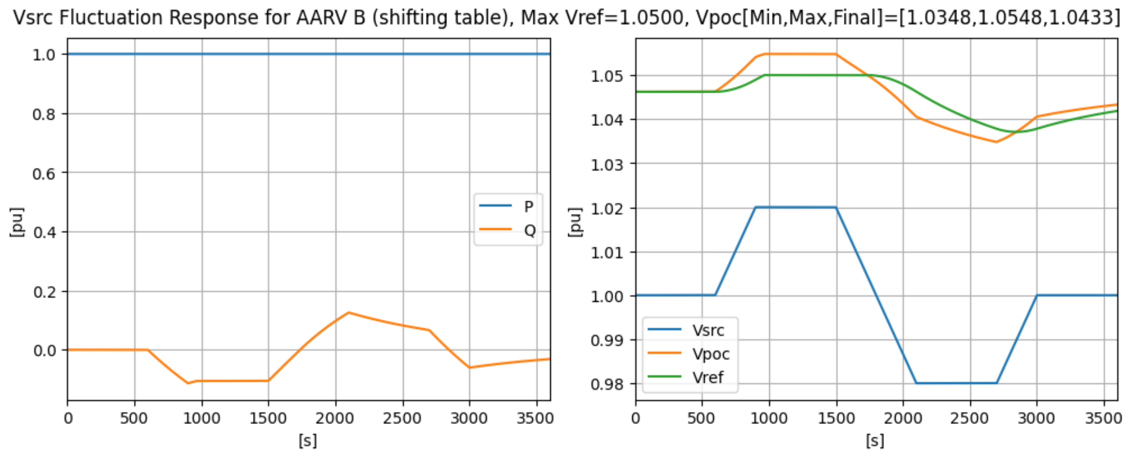
Run the following Python code cell to show the effect of voltage source fluctuations, when the steady-state terminal voltage is already 1.0459 pu, as obtained from the static voltage change calculation performed in section 1.3.

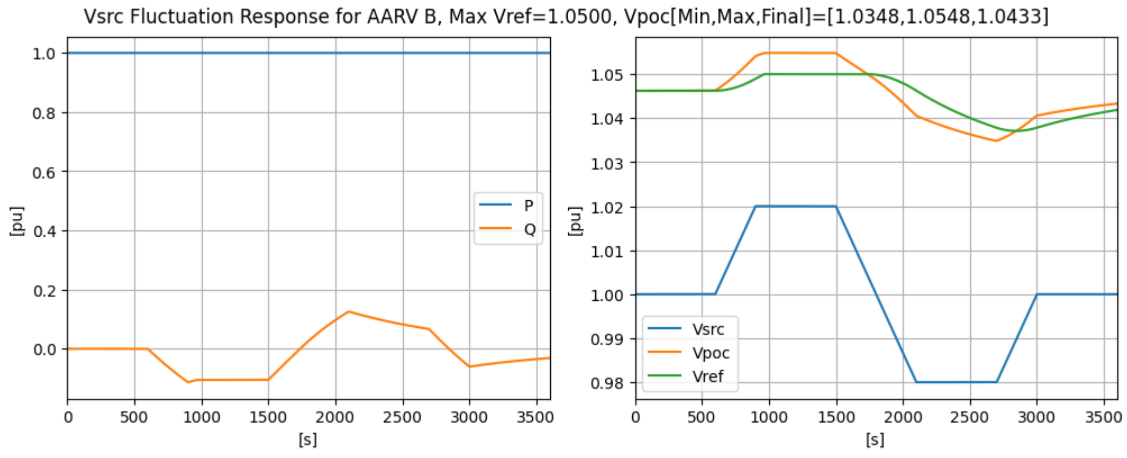
- *Category B with Table Shifting* results in a maximum steady state voltage of 1.0548 pu. The limit on Q is not reached.
- *Category B without Table Shifting* matches the result with table shifting.
- *Category A* results in a maximum steady state voltage of 1.0567 pu. The limit on Q is not reached.

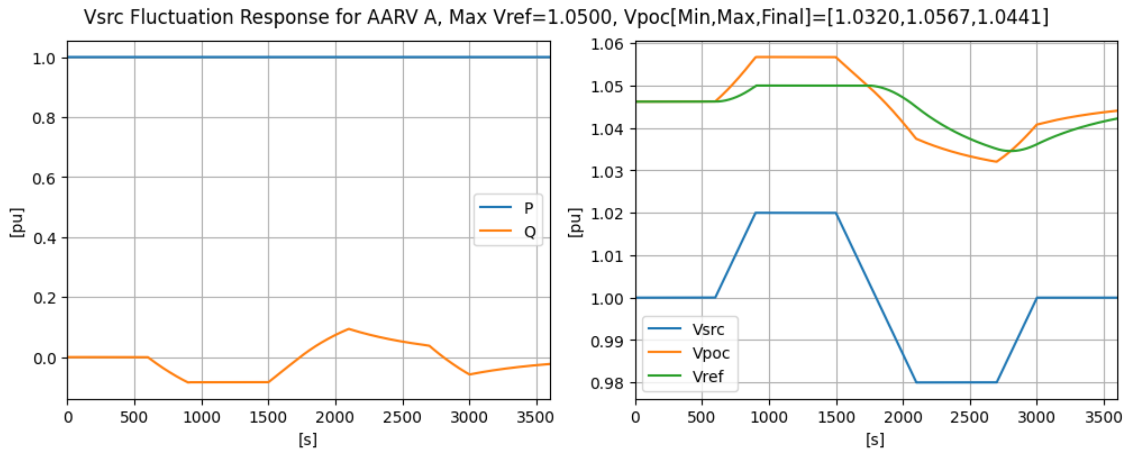
The source voltage ramp times are 300s, so RVC would be less severe than in the power ramping examples and it was not calculated for source voltage fluctuations. The graphs show that *Vpoc* varies less than *Vsrc*. In all three cases, *Vref* reaches its limit of 1.05 pu. Over the simulated interval, the positive and negative reactive power excursions average to approximately zero.

The 2019 and 2023 conference paper references proposed adjustable limits on *Vref*, e.g., the range 0.97 to 1.03 pu. This would allow the DER to inject or absorb full reactive power by the time *Vpoc* reaches 0.95 or 1.05 pu, which would further mitigate overvoltages or undervoltages. (Note: the deadband in Hawaii Rule 14H's volt-var characteristic provides a similar benefit.) However, the standard already allows a reactive power bias on AARV, which would provide a similar benefit as the adjustable limits on *Vref*.

[17]:
```
show_vsrc_test ('AARV B (shifting table)', Vref=1, dB=0, K=22, Qmax=0.44,␣
 ↪Qmin=-0.44, Tref=300, Tresponse=5, dt=0.1, Vinit=1.0462, bShiftTable=True)
show_vsrc_test ('AARV B', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44, Tref=300,␣
 ↪Tresponse=5, dt=0.1, Vinit=1.0462, bShiftTable=False)
show_vsrc_test ('AARV A', Vref=1, dB=0, K=12.5, Qmax=0.25, Qmin=-0.25,␣
 ↪Tref=300, Tresponse=5, dt=0.1, Vinit=1.0462)
```







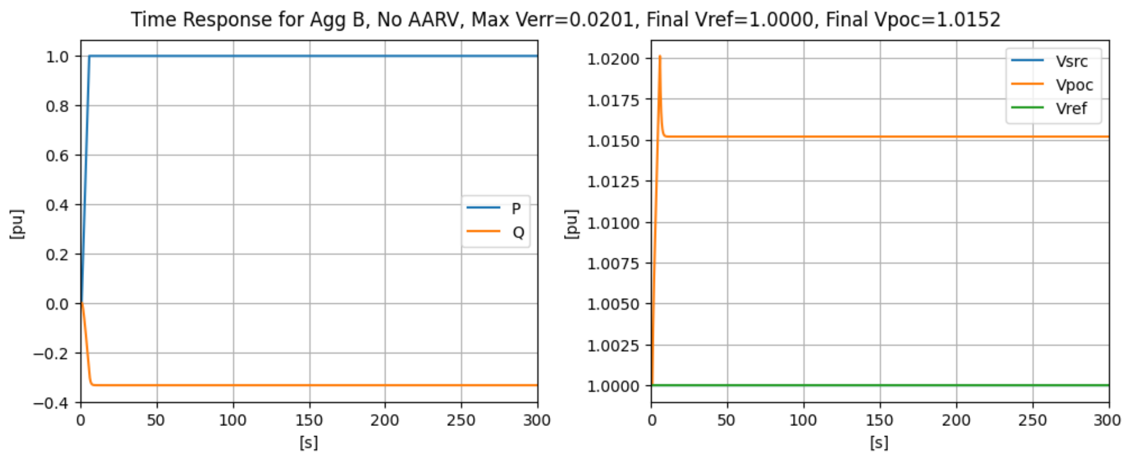
1.13 Static Volt-Var Response to Storage Power-On Ramping

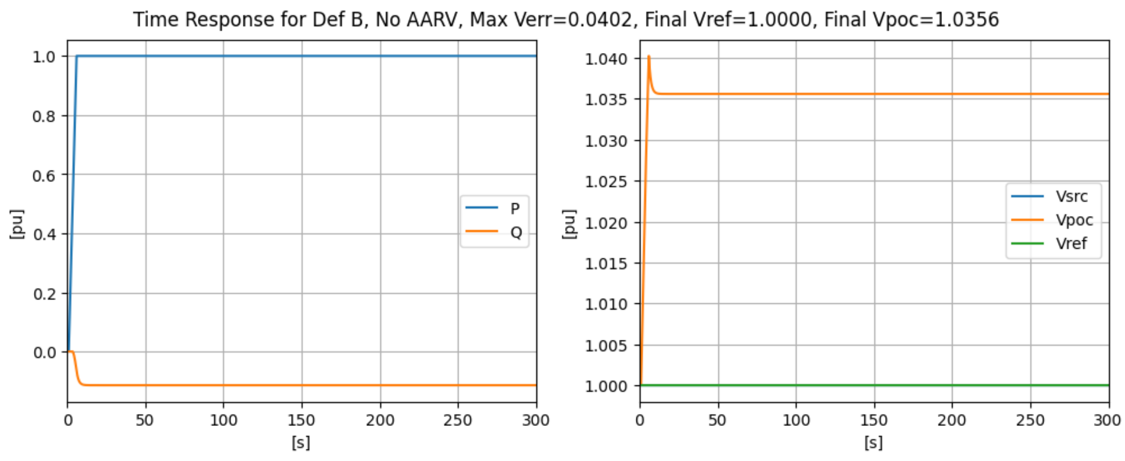
Run the following Python code cell to show the impact of BESS power-on transients, with no AARV. The open-loop response time and power ramp time are both 5s.

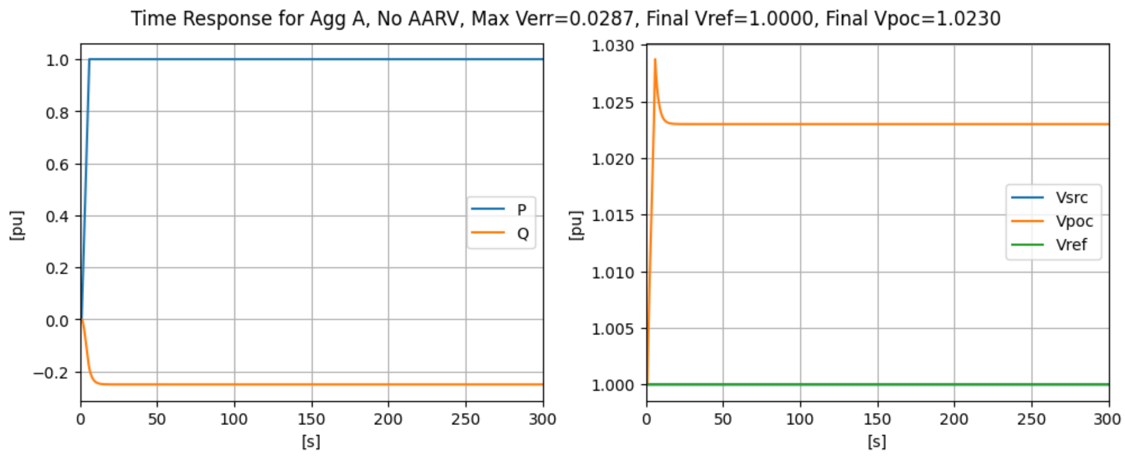
- *Aggressive Category B* produces 2.01% RVC.
- *Default Category B* produces 4.02% RVC, which is above the limit, primarily due to the non-zero deadband, *dB*.
- *Aggressive Category A* produces 2.87% RVC.
- *Default Category A* produces 4.06% RVC, which is above the limit, primary due to the low gain, *K*.
- *Hawaii Rule 14H* produces 4.15% RVC, which is above the limit, primarily due to the non-zero deadband, *dB*.

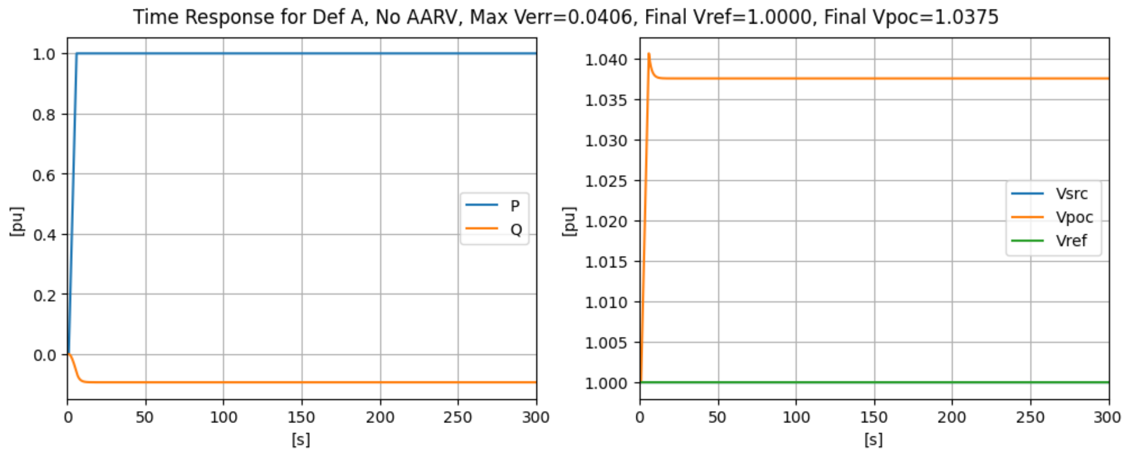
The steady-state reactive power holds steady at a non-zero value, which helps regulate steady-state voltage below 1.0462 pu, but is less effective mitigating RVC.

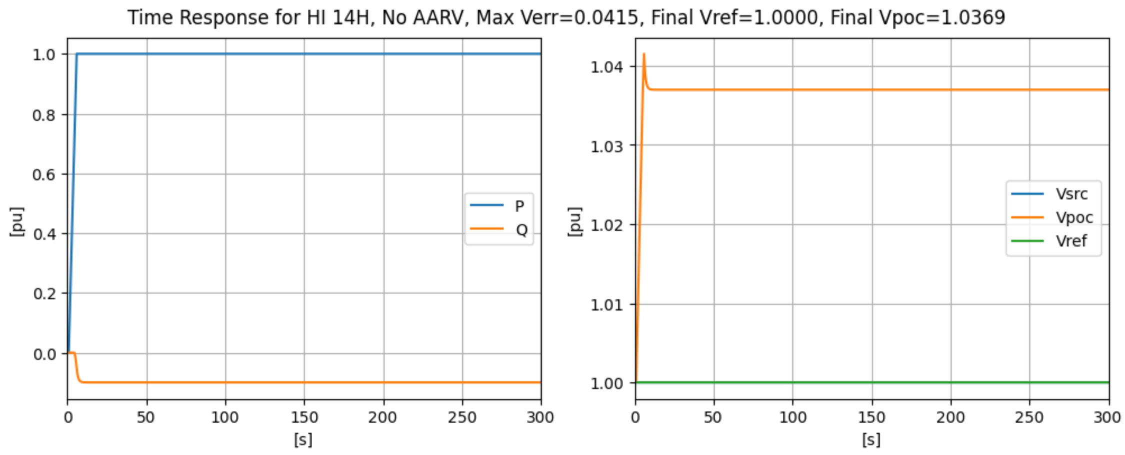
```
[13]: show_bess_test ('Agg B, No AARV', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44,␣
       ↪Tref=0, Tresponse=5)
      show_bess_test ('Def B, No AARV', Vref=1, dB=0.04, K=22.0/3.0, Qmax=0.44,␣
       ↪Qmin=-0.44, Tref=0, Tresponse=5)
      show_bess_test ('Agg A, No AARV', Vref=1, dB=0, K=12.5, Qmax=0.25, Qmin=-0.25,␣
       ↪Tref=0, Tresponse=5)
      show_bess_test ('Def A, No AARV', Vref=1, dB=0, K=2.50, Qmax=0.25, Qmin=-0.25,␣
       ↪Tref=0, Tresponse=5)
      show_bess_test ('HI 14H, No AARV', Vref=1, dB=0.06, K=43.0/3.0, Qmax=0.44,␣
       ↪Qmin=-0.44, Tref=0, Tresponse=5)
```






Time Response for Agg A, No AARV, Max Verr=0.0287, Final Vref=1.0000, Final Vpoc=1.0230
[pu]
[s]
P
Q
Vsrc
Vpoc
Vref


Time Response for Def A, No AARV, Max Verr=0.0406, Final Vref=1.0000, Final Vpoc=1.0375
[pu]
[s]
P
Q
Vsrc
Vpoc
Vref


Time Response for HI 14H, No AARV, Max Verr=0.0415, Final Vref=1.0000, Final Vpoc=1.0369
[pu]
[s]
P
Q
Vsrc
Vpoc
Vref

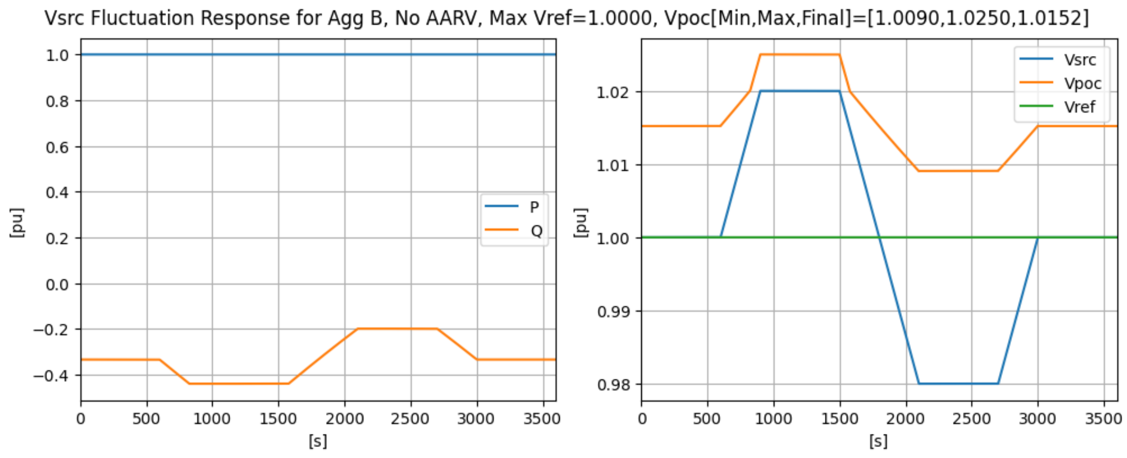
1.14 Static Volt-Var Response to Grid Voltage Fluctuations

Run the following Python code cell to show the impact of source voltage transients, with no AARV:

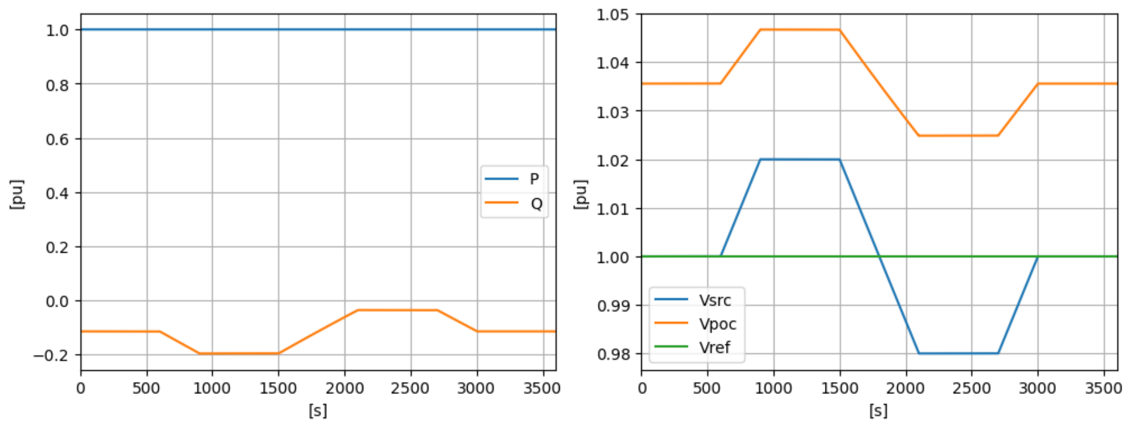
- *Aggressive Category B* limits *Vpoc* to 1.025 pu
- *Default Category B* limits *Vpoc* to 1.0467 pu
- *Aggressive Category A* limits *Vpoc* to 1.0419 pu
- *Default Category A* limits *Vpoc* to 1.0525 pu
- *Hawaii Rule 14H* limits *Vpoc* to 1.045 pu

All of these should be acceptable results.

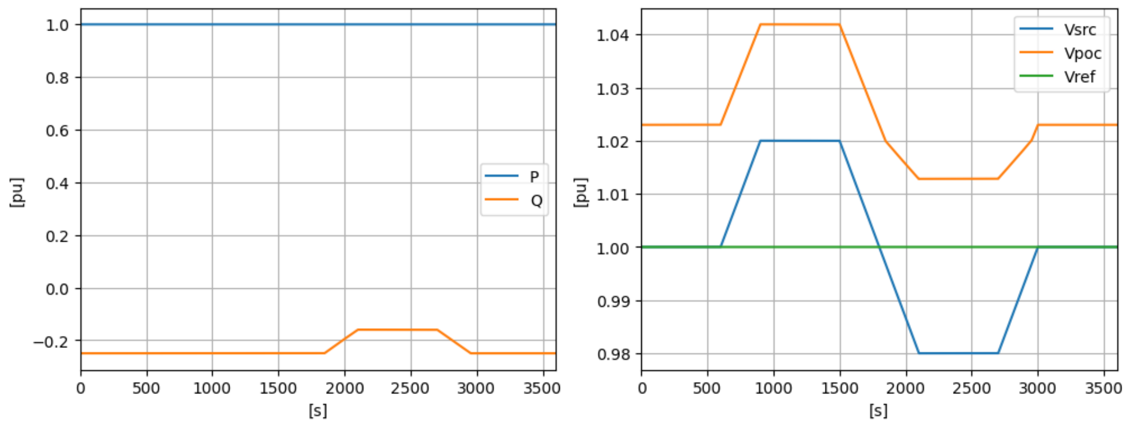
```
[14]: show_vsrc_test ('Agg B, No AARV', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44,␣
        ↪Tref=0, Tresponse=5)
      show_vsrc_test ('Def B, No AARV', Vref=1, dB=0.04, K=22.0/3.0, Qmax=0.44,␣
        ↪Qmin=-0.44, Tref=0, Tresponse=5)
      show_vsrc_test ('Agg A, No AARV', Vref=1, dB=0, K=12.5, Qmax=0.25, Qmin=-0.25,␣
        ↪Tref=0, Tresponse=5)
      show_vsrc_test ('Def A, No AARV', Vref=1, dB=0, K=2.50, Qmax=0.25, Qmin=-0.25,␣
        ↪Tref=0, Tresponse=5)
      show_vsrc_test ('HI 14H, No AARV', Vref=1, dB=0.06, K=43.0/3.0, Qmax=0.44,␣
        ↪Qmin=-0.44, Tref=0, Tresponse=5)
```



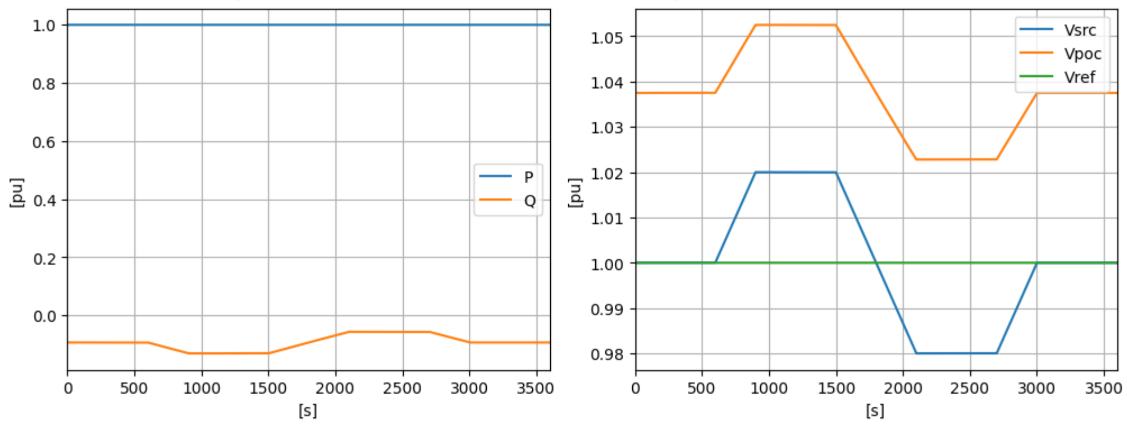
Vsrc Fluctuation Response for Def B, No AARV, Max Vref=1.0000, Vpoc[Min,Max,Final]=[1.0248,1.0467,1.0356]

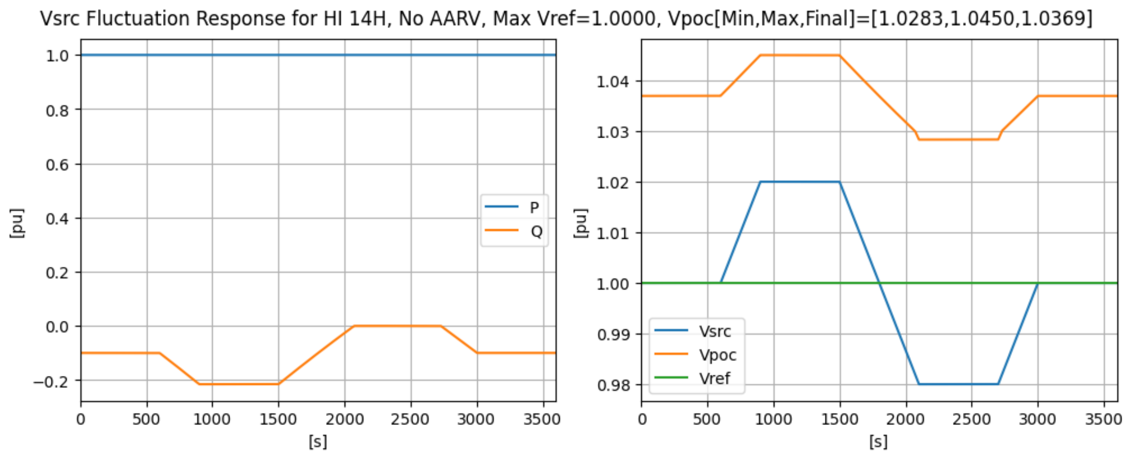


Vsrc Fluctuation Response for Agg A, No AARV, Max Vref=1.0000, Vpoc[Min,Max,Final]=[1.0128,1.0419,1.0230]



Vsrc Fluctuation Response for Def A, No AARV, Max Vref=1.0000, Vpoc[Min,Max,Final]=[1.0228,1.0525,1.0375]





1.15 Reactive Power Bias with AARV

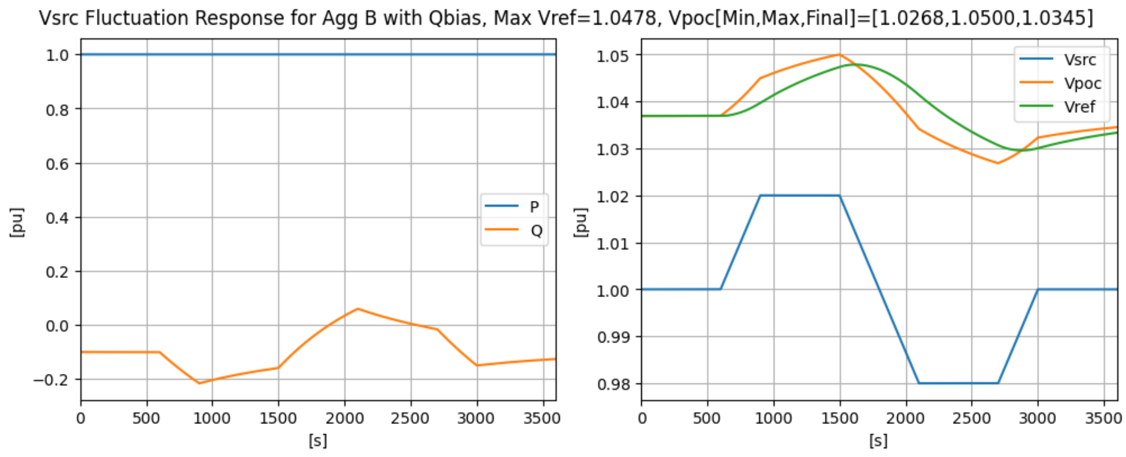
Run the following Python code cell to find the terminal voltage, when the BESS absorbs -0.1 pu reactive power. This voltage is 1.0369 pu, compared to 1.0462 pu at zero reactive power.

[15]:
```
DQ = -0.1
vpoc, d = get_voltage (RPU, XPU, DP, DQ, VSRC)
print ('Nantucket BESS example for dP={:.3f}, dQ={:.3f}, R={:.3f}, X={:.3f} pu:␣
  ↪Vpoc={:.4f}, dV={:.4f} pu'.format (DP, DQ, RPU, XPU, vpoc, d))
```

```
Nantucket BESS example for dP=1.000, dQ=-0.100, R=0.042, X=0.098 pu:
Vpoc=1.0369, dV=0.0369 pu
```

Run the following Python code cell to simulate a source voltage fluctation under this condition, specifying *Vinit*=1.0369 from the preceding output. The value of *Qbias* keeps *Vpoc* within 1.05 pu, but the DER reactive power fluctuations no longer average to zero.

[16]:
```
show_vsrc_test ('Agg B with Qbias', Vref=1, dB=0, K=22, Qmax=0.44, Qmin=-0.44,␣
  ↪Tref=300, Tresponse=5, Vinit=1.0369, Qbias=-0.1)
```



1.16 Conclusions

1. Shorter open-loop response times, *Tresponse*, improve the mitigation of RVC. A specified default of 1-5s should be considered.
2. Shorter AARV time constants, *Tref*, improve the mitigation of RVC. A specified default of 300s should be considered.
3. Paragraph 2 of clause 5.14.5.3 in 1547.1-2020 should be reviewed for accuracy, i.e., the cited 10% should be 20%.
4. The mitigation that AARV provides for RVC should be effective for the normal output fluctuations of solar, wind, and controlled power ramping. It's less effective for sudden changes due to DER switching, which would also be true of static volt-var characteristics.
5. Options for mitigating steady-state undervoltage and overvoltage may include:
    - Static volt-var, with supplemental volt-watt
    - AARV with non-zero *Qbias*
    - AARV with tighter limits on *Vref*, which would require a change to Table 8 in IEEE 1547-2018
    - Reliance on utility-controlled tap changers, capacitor banks, etc.

1.17 BSD 3-Clause License