

Updated to fix typos and include feedback  
from discussion during IEC/IEEE 60802 call  
on Monday 5<sup>th</sup> May 2025

See slides 10 and 60

IEC/IEEE 60802

# Continuous vs. Discrete Implementations of Endpoint Filter - Considerations

Geoff Garner (Self Affiliated – [gmgarner@alum.mit.edu](mailto:gmgarner@alum.mit.edu))

David McCall (Intel Corporation – [david.mccall@intel.com](mailto:david.mccall@intel.com))

v2 - April 2025

# References

- [1] Python is a high-level, general-purpose programming language. For more information go to [Python.org](https://python.org)
- [2] The code in each attached Python file was developed using code supplied by Kilian Brunner as a starting point (March 11, 2025)
- [3] David McCall, “[60802 Time Synchronisation – Monte Carlo Analysis: 100-hop Model, “Linear” Clock Drift, NRR Accumulation Overview & Details, Including Equations – v2](#)”, contribution to IEC/IEEE 60802, September 2022

# Acknowledgement

The authors would like to thank Kilian Brunner for raising these points for consideration and supplying the initial Python model and code.

# Content

- Background
- ClockTimeReceiver – Unfiltered & Filtered Outputs
  - ClockTimeReceiver Unfiltered Output – Possible Behaviours
- Endpoint Filter – Annex C – Continuous 2<sup>nd</sup> Order Implementation
  - Related Normative Requirements
- Endpoint Filter – Discrete Implementations Considerations
  - Varying Sample Rate
  - Higher Order Filters
- Conclusion & Proposed Addition to Annex C

# Background

# Background – 1

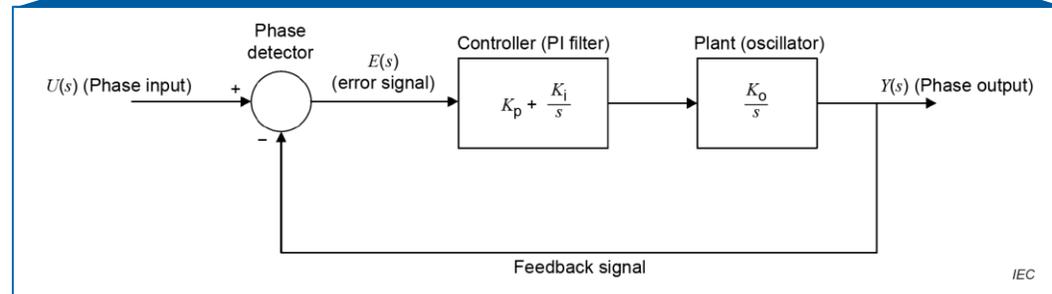
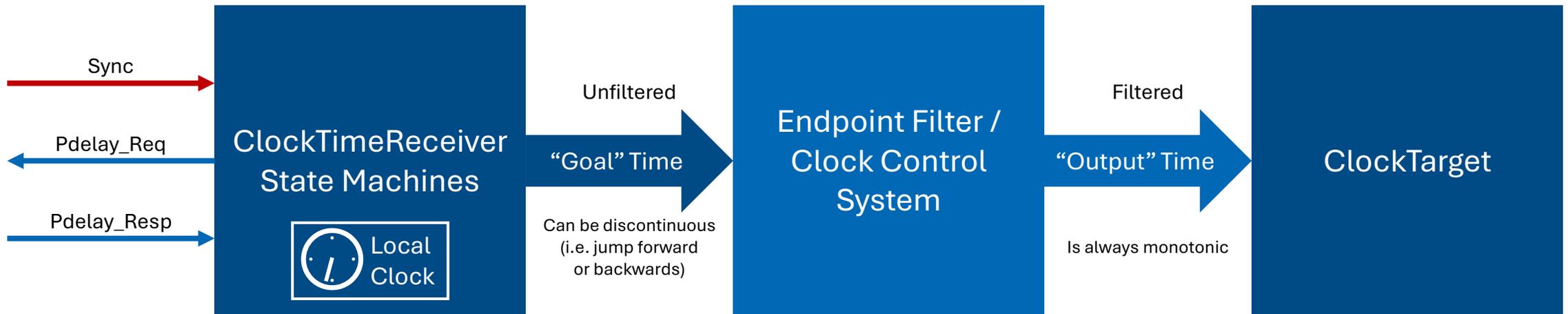
- In IEEE 802.1AS, the PTP End Instance typically includes an “endpoint filter” or “clock control system” that filters the output of the ClockTimeReceiver
  - The behaviour of this filter alters the characteristics of the ClockTarget
- IEC/IEEE 60802 requires specific behaviour from the filter to meet the normative requirements
  - Table 11 – Clock control system requirements
  - Table 14 – Error generation limits for PTP End Instance
  - Any filter that meets the minimum requirements (which eliminate sufficient errors of one type) without introducing excessive errors (of a different type) is acceptable.
- 60802 does not specify an exact filter, but does provide an example in Annex C
  - The example is a continuous 2<sup>nd</sup> order filter, with behaviour similar to the one used in the Time Series simulations (note: the Monte Carlo simulations did not include an endpoint filter)

# Background – 2

- Kilian Brunner, a research associate & lecturer at the Zurich University of Applied Sciences, built one of the first implementations of IEC/IEEE 60802.
  - Simple “Grandmaster – 1 Relay – End Instance” setup
  - In short: in works as expected.
    - “...shown significant improvements in accuracy—especially during startup when switches are warming up.”
    - “...allowed me to demonstrate that the Drift Tracking TLV seems to be the only effective method for compensating [for] high [oscillator] drift rates of 1 ppm/s.”
- But it has also raised concerns that a simple translation of the continuous 2<sup>nd</sup> order filter in Annex C to a discrete 2<sup>nd</sup> order filter implementation may make the normative requirements difficult or impossible to achieve
  - Need to verify these concerns and potentially make readers of the specification aware of them
  - Also need to check that other implementations will address any concerns and are viable to implement (if not...maybe change the normative requirements, but only if system performance can still be assured?)

# ClockTimeReceiver Unfiltered & Filtered Outputs

# ClockTimeReceiver



2<sup>nd</sup> Order Continuous Filter provided as an example in Annex C

# Additional Information

From discussions during IEC/IEEE 60802  
call on Monday 5<sup>th</sup> May 2025

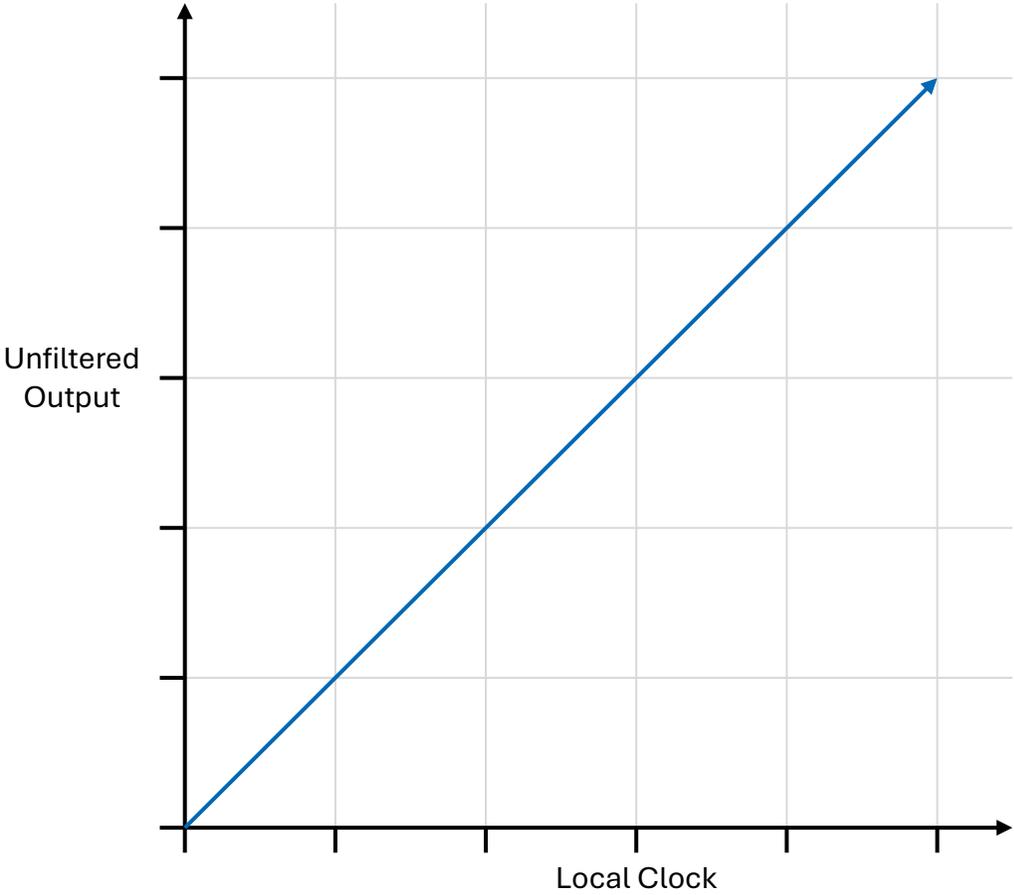
- In some implementations (e.g. Linux) the most computationally expensive part of the endpoint filter process is the “cross-timestamping” necessary for comparing the phase input  $U(s)$  with the phase output  $Y(s)$  in the phase detector
  - The complication is that the OS can’t actually read both values at exactly the same time, so it repeatedly reads one then the other and derives a matched pair of values at the same instance via an “averaging” algorithm. This can take up to 1 ms.
- Other implementations do not suffer from this complexity
- It is probably feasible to limit the number of times this cross-timestamping must be done to only the arrival of a new Sync message.
  - More work is required to verify this

# ClockTimeReceiver – Unfiltered Output

- Time is updated to latest “best estimate” on arrival of new Sync message
  - $\text{preciseOriginTimestamp} + \text{correctionField} + \text{meanLinkDelay}$
- Time is progressed until arrival of next Sync message based on Local Clock x Rate Ratio
  - “Rate Ratio” is the Rate Ratio in the incoming Sync message modified by the Neighbor Rate Ratio measured by the End Instance

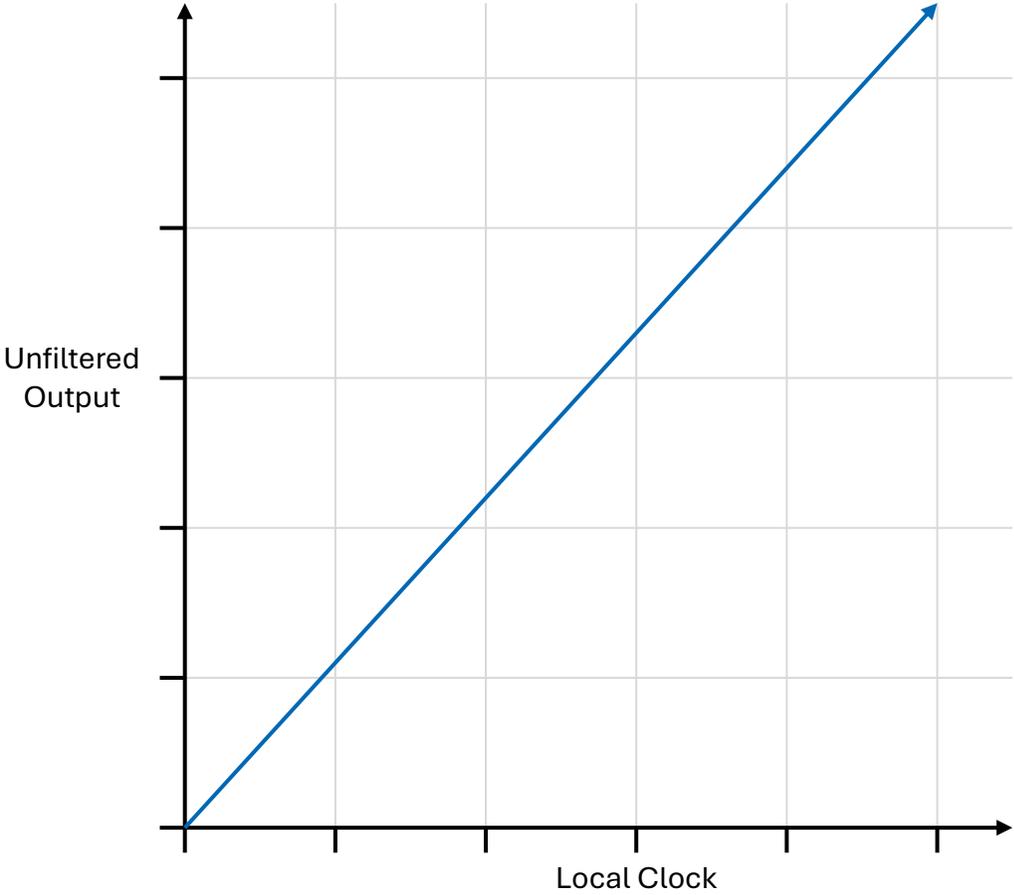
The following slides illustrate some principles of operation and makes them visible by exaggerating the Rate Ratios and errors involved, e.g.  $RR = 1.1$  when in IEC/IEEE 60802 the maximum is 1.0001. Also, don't worry about the time interval; things like drift are just X drift per time interval.

# ClockTimeReceiver – Unfiltered Output



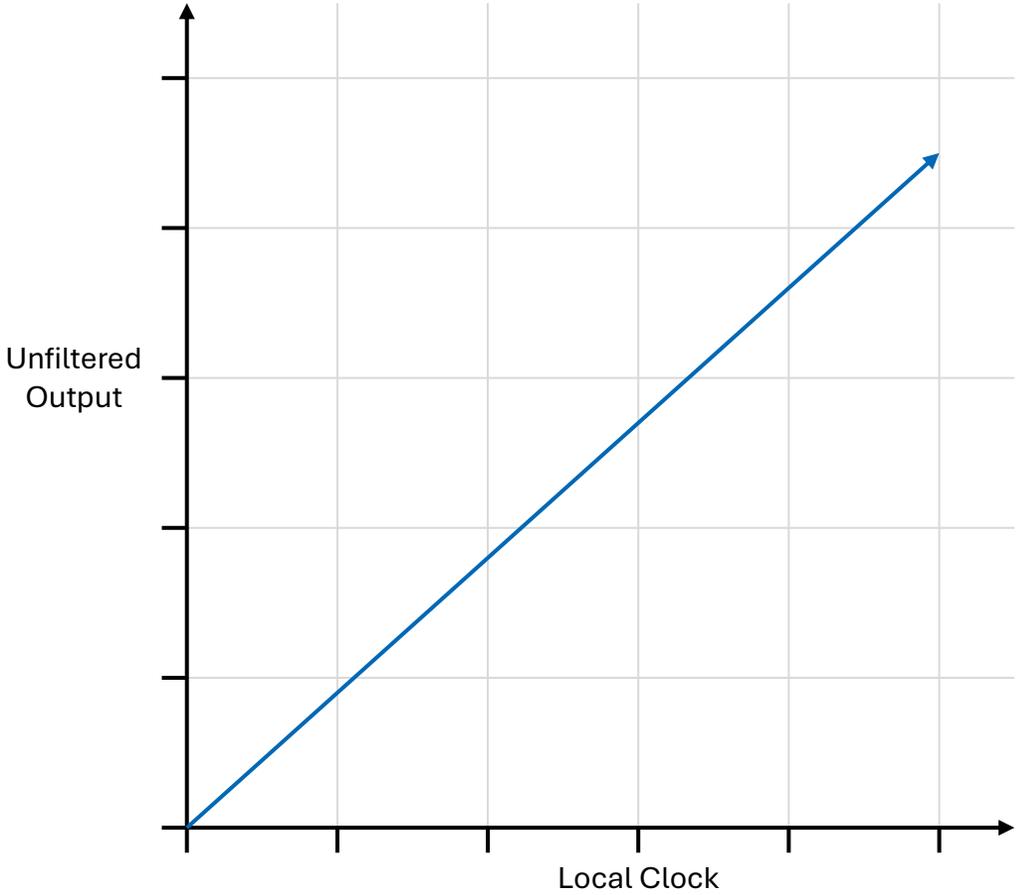
**RR = 1**

# ClockTimeReceiver – Unfiltered Output



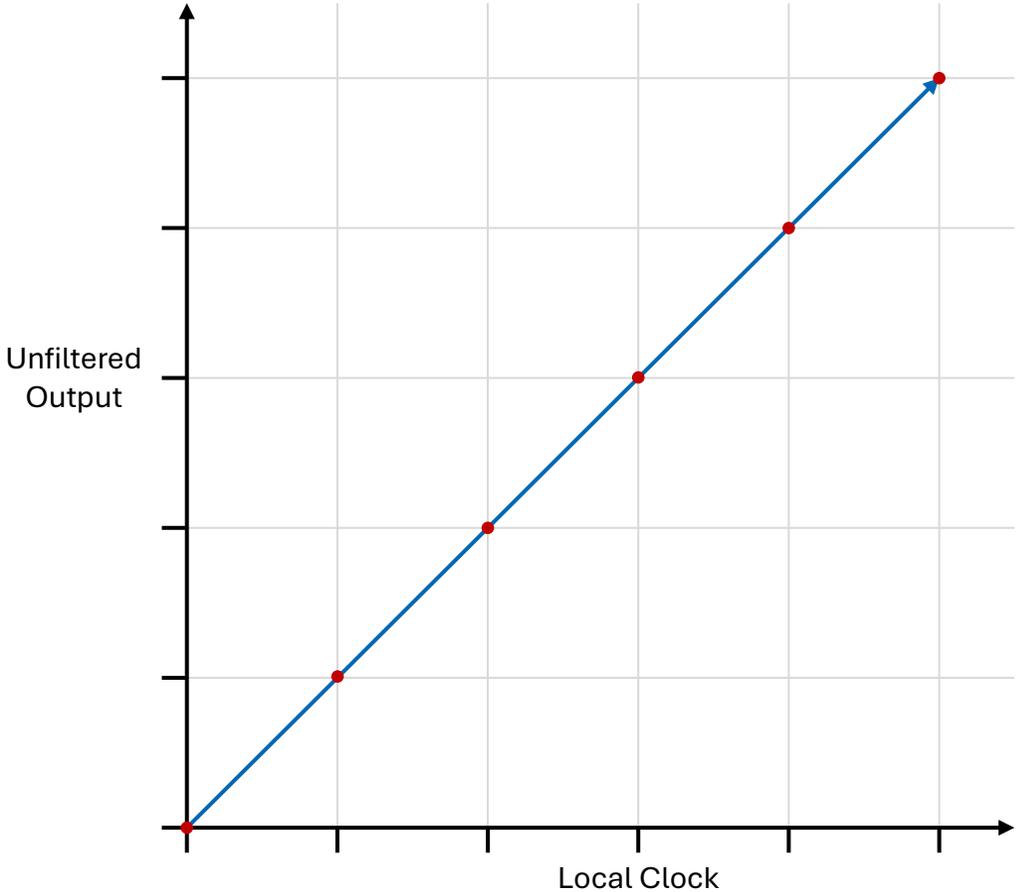
**RR = 1.1**

# ClockTimeReceiver – Unfiltered Output



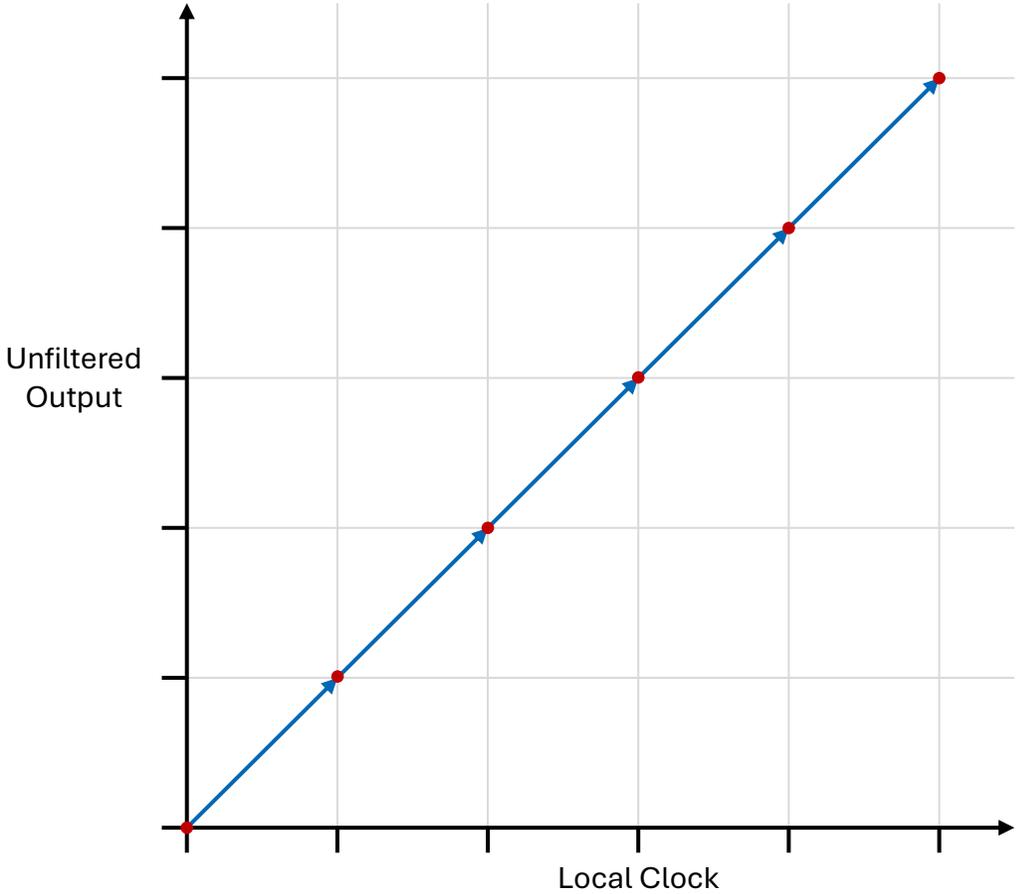
**RR = 0.9**

# ClockTimeReceiver – Unfiltered Output



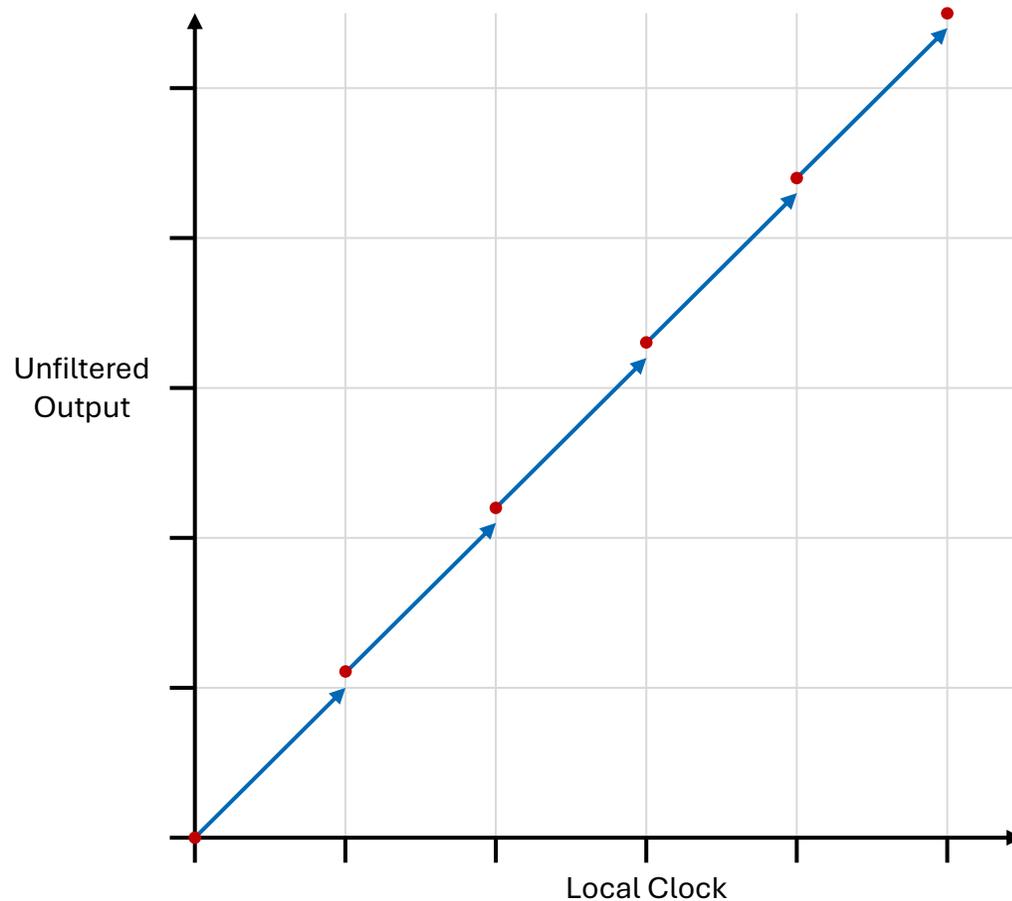
**RR = 1**

# ClockTimeReceiver – Unfiltered Output



**RR = 1**

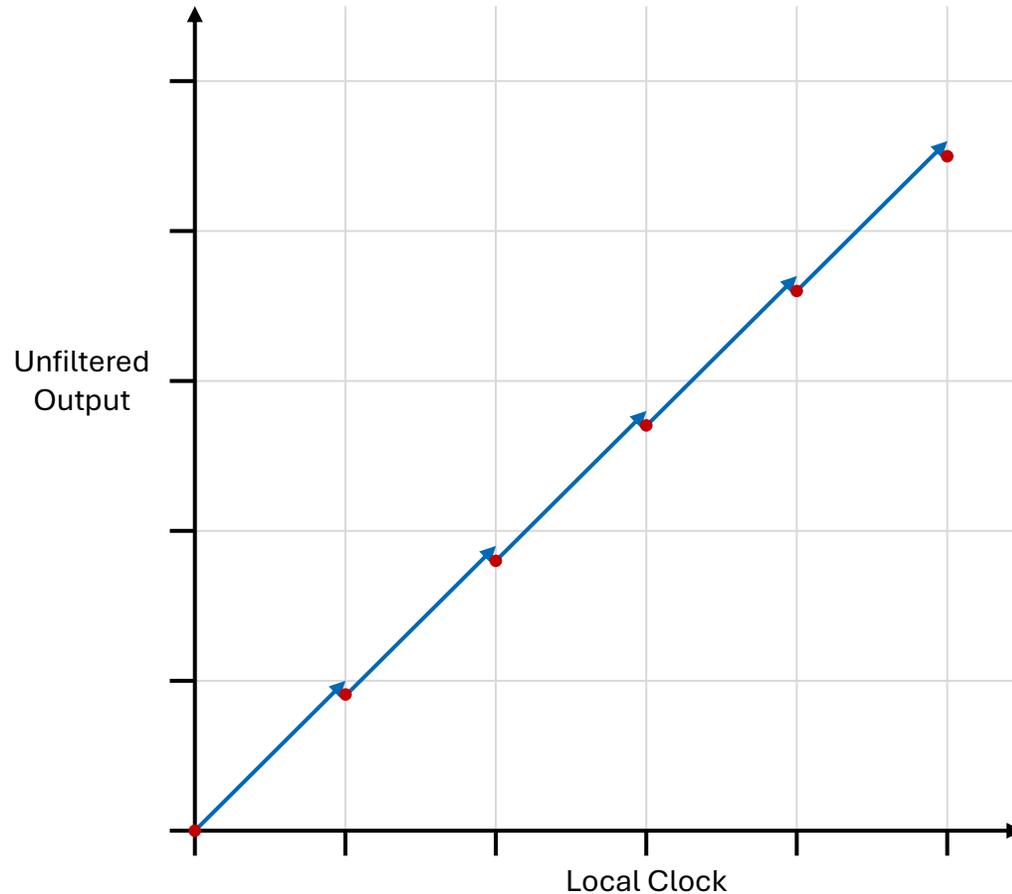
# ClockTimeReceiver – Unfiltered Output



**RR = 1**

(but should be 1.1)

# ClockTimeReceiver – Unfiltered Output



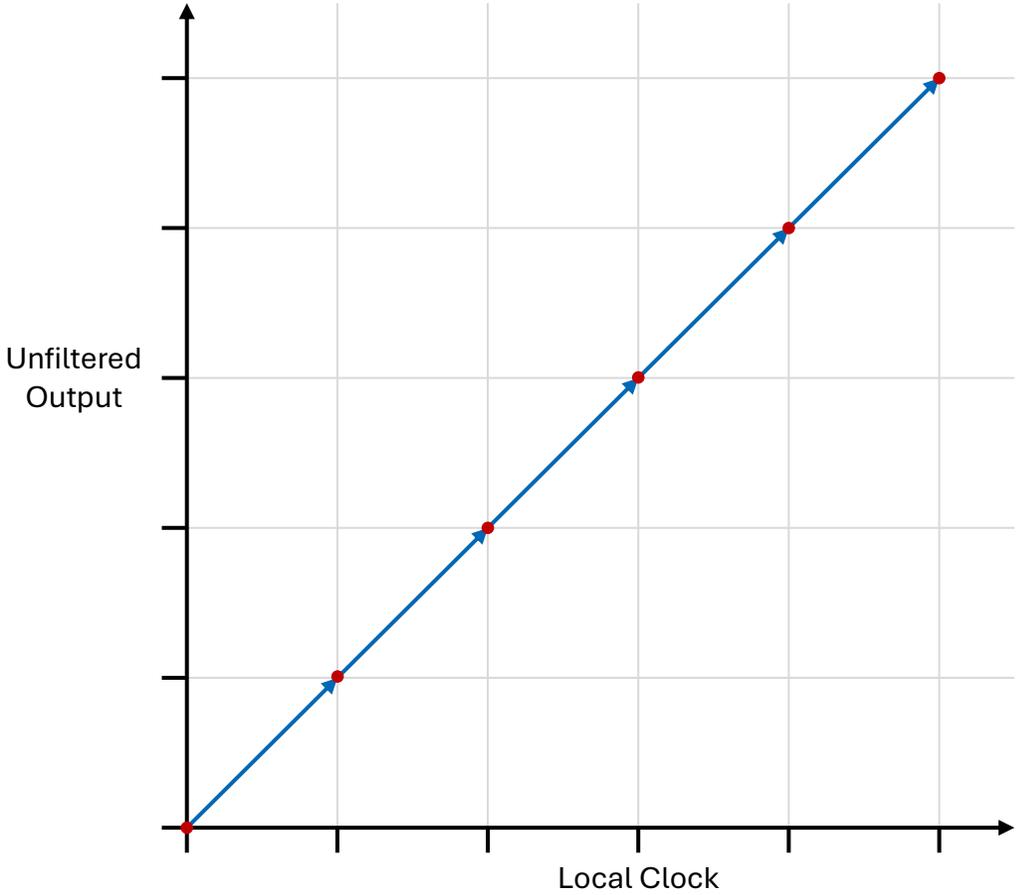
**RR = 1**

(but should be 0.9)

# ClockTimeReceiver – Unfiltered Output: Sources of Error

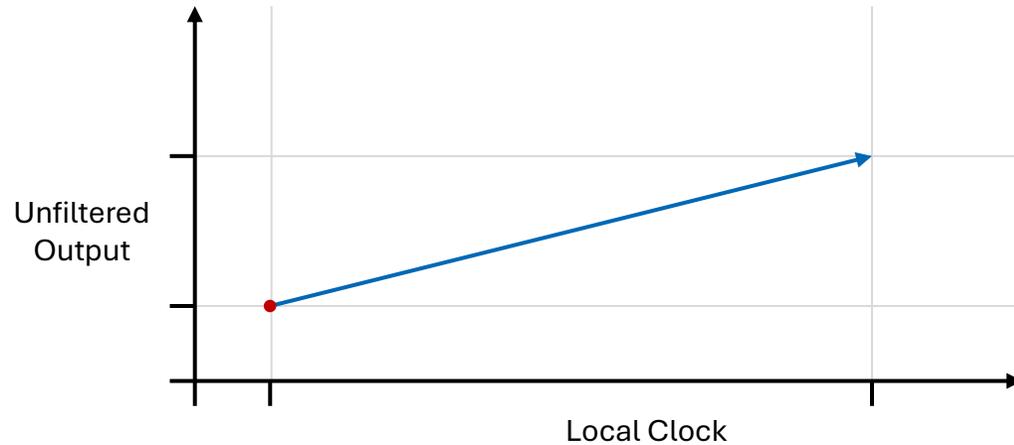
- Error in preciseOriginTimestamp + correction Field + meanLinkDelay
  - Error on arrival of Sync Message
- Error in Rate Ratio
  - Increasing error before arrival of next Sync message
- Change in Rate Ratio, i.e. Rate Ratio Drift
  - Even if the two items above are 100% correct on arrival of the Sync message, by the time the next Sync message arrives...there will be an error.
  - But, with the addition of Rate Ratio Drift measurement, there is an opportunity to address this source of error
    - Pushes the source of error “one layer down”, i.e. if the Rate Ratio Drift is itself changing over time (in addition to any error in the Rate Ratio Drift)

# ClockTimeReceiver – Unfiltered Output

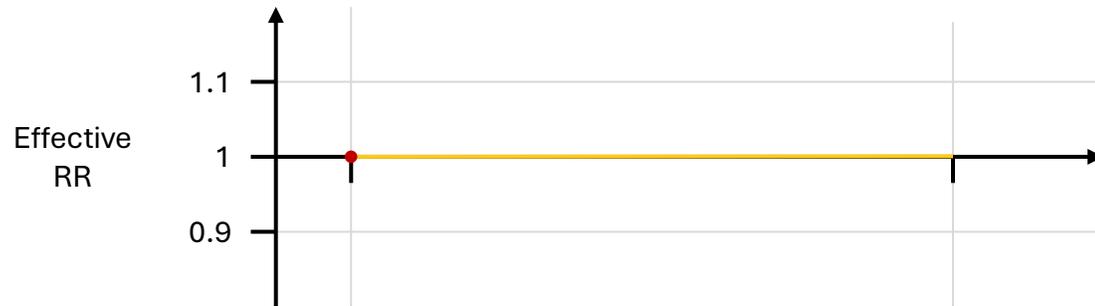


**RR = 1**

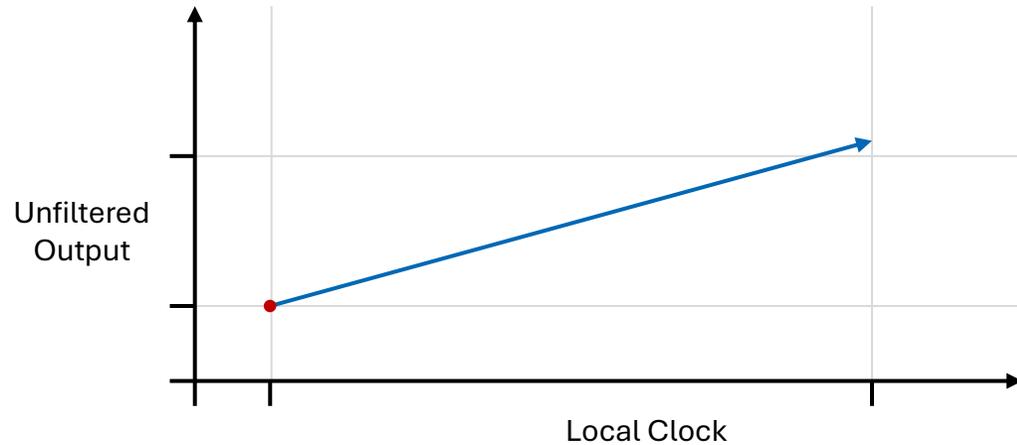
# ClockTimeReceiver – Unfiltered Output



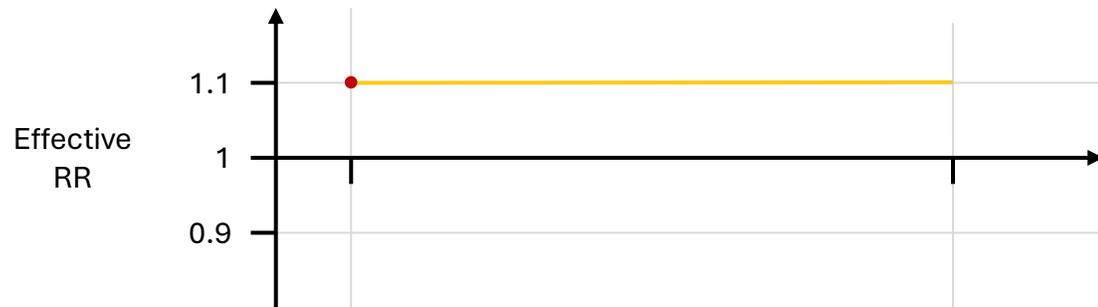
**RR = 1**



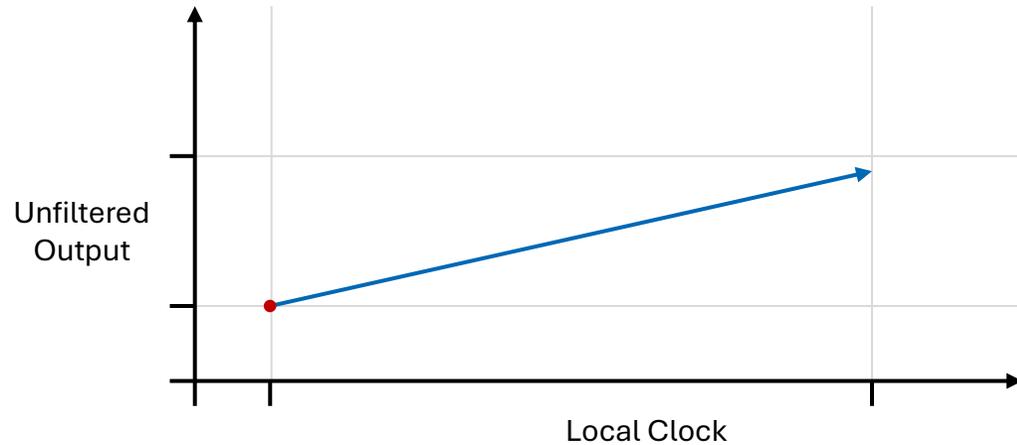
# ClockTimeReceiver – Unfiltered Output



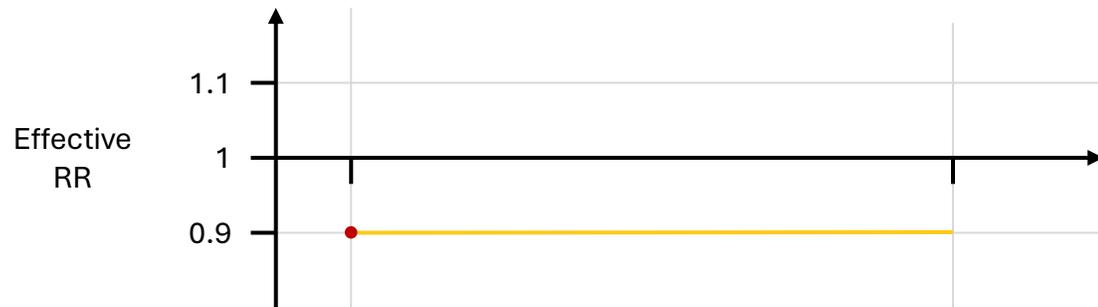
**RR = 1.1**



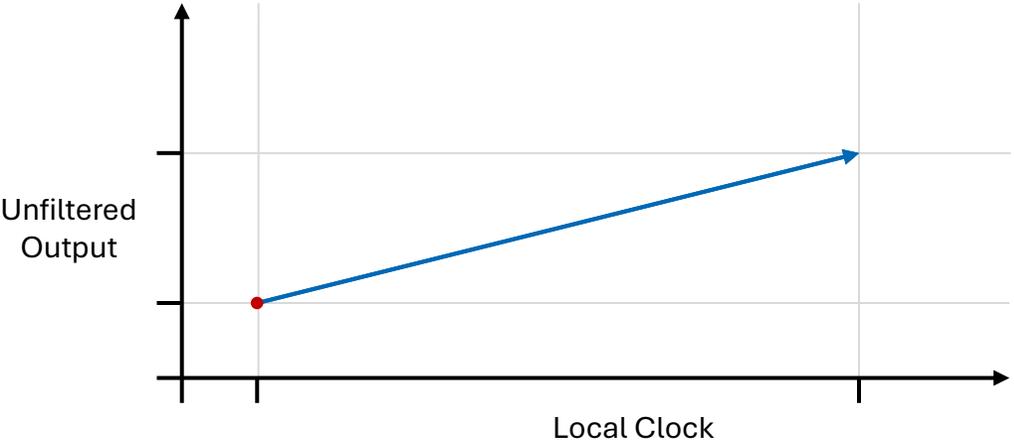
# ClockTimeReceiver – Unfiltered Output



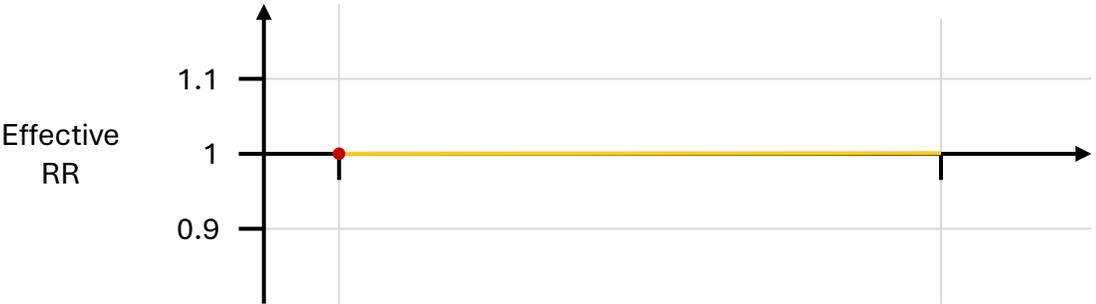
**RR = 0.9**



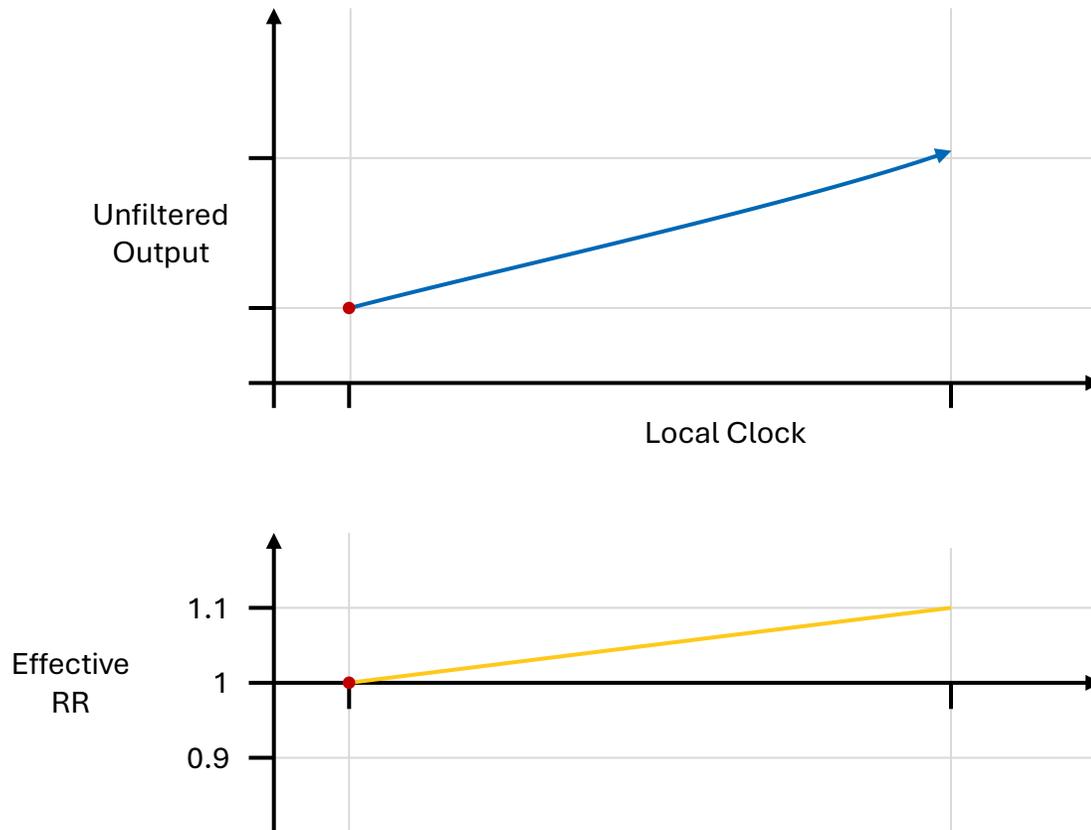
# ClockTimeReceiver – Unfiltered Output



**RR = 1**



# ClockTimeReceiver – Unfiltered Output



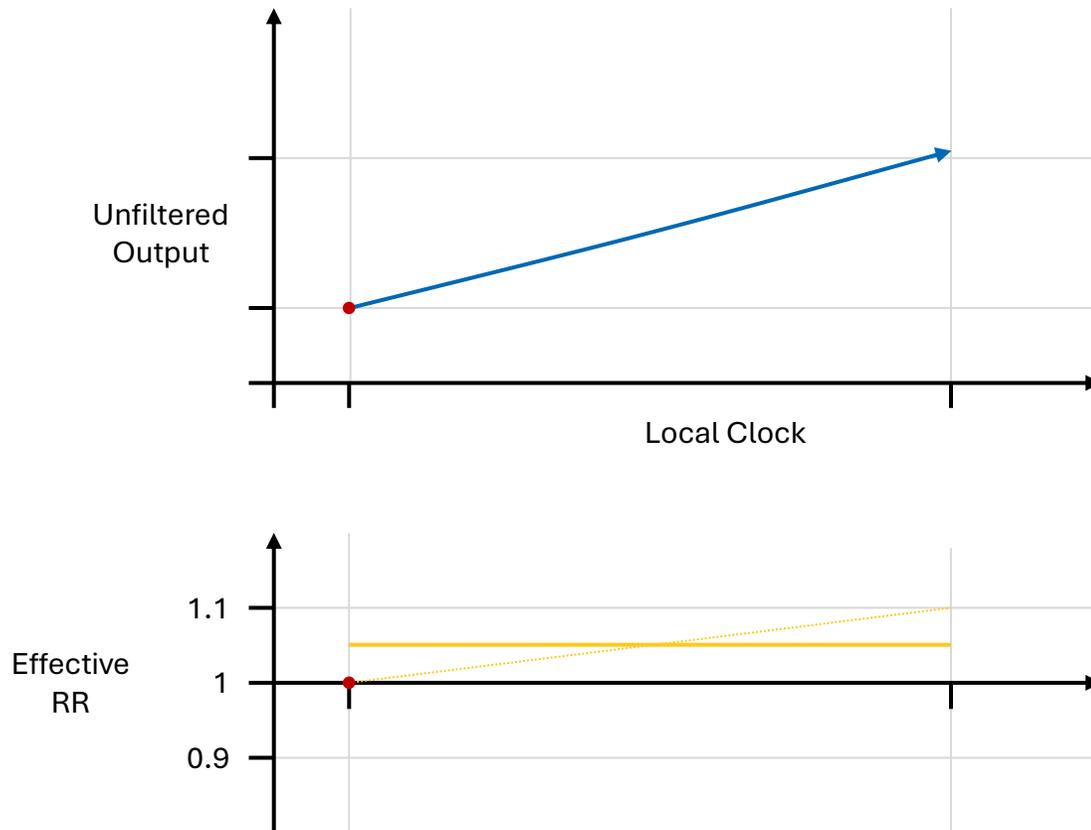
**RR Drifting +0.1**

**RR Start = 1**

**RR End = 1.1**

Taking account of RR Drift improves accuracy but is more computationally expensive.

# ClockTimeReceiver – Unfiltered Output



**RR Drifting +0.1**

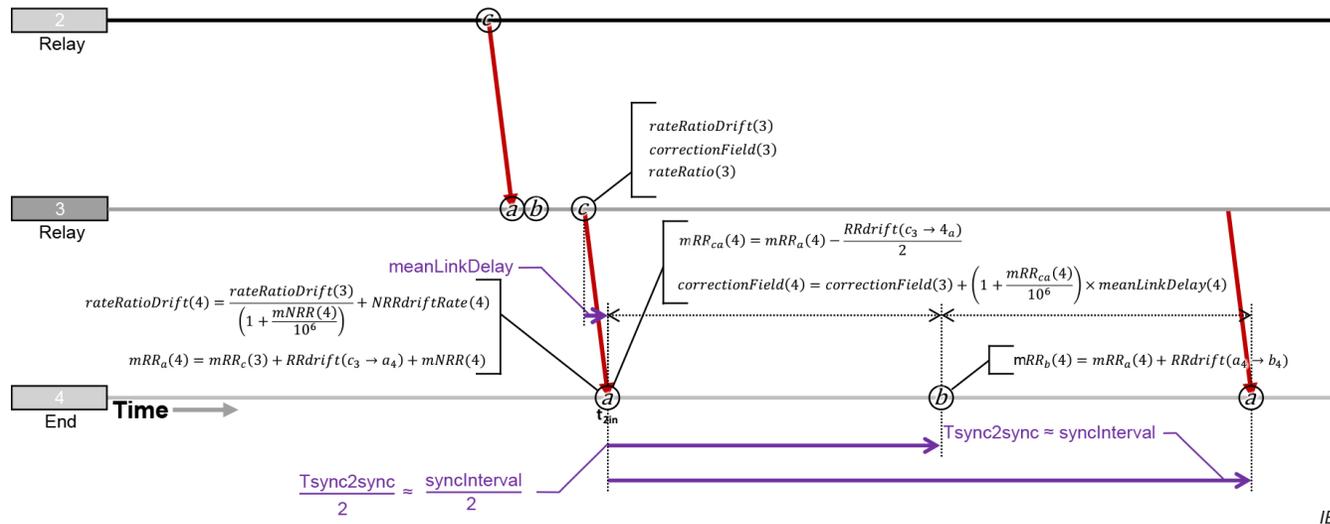
**RR Start = 1**

**RR End = 1.1**

An approximation can be made by calculating an average RR (in this example: too high initially; too low at the end).

# ClockTimeReceiver – Unfiltered Output

- Annex D describes an algorithm that takes the “approximation” approach to including RR Drift information



$$mRR_b(n) = mRR_a(n) + RRdrift_n(a \rightarrow b)$$

$$= mRR_a(n) + \left( rateRatioDrift(n) \times \frac{syncInterval}{2} \right)$$

$$= mRR_a(n) + (rateRatioDrift(n) \times 0,0625)$$

“Of course, the exact interval until the next Sync message's arrival (Tsync2sync in Figure D.7) can't be known before it happens, but the Rate Ratio value is required as soon as possible after arrival of the most recent Sync message. The solution is to use the nominal value of the interval, i.e. syncInterval, which is 125 ms.”

# Endpoint Filter – Annex C: Continuous 2nd Order Implementation

# IEC/IEEE 60802 Approach

- IEC/IEEE does not require a specific design or implementation of the Clock Control System (Endpoint Filter), but...
- Without a filtered output, some normative requirements would be hard or impossible to meet
  - Including the requirement that the Clock Target increases monotonically (6.2.2)
- The latest Time Series Simulations implemented a 2<sup>nd</sup> Order Discrete Filter running at 1 kHz sample rate
  - ClockTimeReceiver Unfiltered Output was adjusted to account for RR drift, adjusted at each sample (i.e. not using the “averaging” approach from the example in Annex D).
- Annex C describes, as an example of a filter that can be used to meet the normative requirements, a 2<sup>nd</sup> Order Continuous Filter
  - Relatively easy to translate this implementation into various other implementations (discrete or otherwise), which is why it was chosen as the example.

# Normative Requirements – 1

- 6.2.2 PTP Instance requirements
  - c) During operation, the Working Clock and Global Time at Grandmaster PTP Instances and PTP End Instances shall increase **monotonically**, where monotonic means that for a time  $y$  that occurs after time  $x$ , the ClockTarget's timestamp of  $y$  is greater than or equal to the ClockTarget's timestamp of  $x$
  - d) The Working Clock and Global Time at a PTP End Instance can be controlled by applying a **frequency change** over a period of time. The frequency applied can have a fine resolution to **speed up or slow down the clock smoothly**, and it has a total range of frequency adjustment.

# Normative Requirements – 2

- 6.2.4 Clock Control System Requirements

Topic	Value
Maximum Bandwidth	1,0 Hz
Minimum Bandwidth	0,9 Hz
Maximum Gain Peaking	2,2 dB
Minimum absolute value of Roll-off	20 db/decade
NOTE 1 For more information regarding the clock control system, see Annex C	
NOTE 2 The values contained in this table apply to both the Working Clock and Global Time	

# Normative Requirements – 3

- 6.2.5 Error generation limits

Table 14 – Error generation limits for PTP End Instance

Topic	Value
Accuracy of recovered clock (Rate Ratio drift):  Working Clock (acting as ClockTarget) at PTP End Instance  minus  Working Clock (acting as Clock Source) at Grandmaster  while... <ul style="list-style-type: none"> <li>WorkingClock (acting as ClockSource) at Grandmaster PTP Instance, fractional frequency offset with respect to the nominal frequency is increasing at 1 ppm/s</li> <li>Local Clock at upstream PTP Instance is stable.</li> <li>meanLinkDelay between upstream PTP Instance and PTP Relay Instance is negligible</li> </ul>	Allowable range of cTE: - 10 ns to + 10 ns  Allowable range of dTE: - 145 ns to + 15 ns

Asymmetric range is due to the delay inherent in using a 2<sup>nd</sup> order filter, i.e. the offset allow the use of a 2<sup>nd</sup> order filter, but only with certain performance, i.e. too much delay and the implementation will be non-compliant.

There is a similar asymmetry for accuracy with Rate Ratio Drift and Neighbor Rate Ratio Drift.

Annex D includes **D.3.7 Explanation for the asymmetric normative requirements for the allowable range of dTE in Table 14, rows 2 and 3**

# 2<sup>nd</sup> Order Filter – Design Constraints – 1

- There is very little leeway to change the parameters  $K_p K_o$  and  $K_i K_o$
- Decreasing  $K_i$  would cause  $\omega_n$  to decrease (Eq. (C.6)/60802), which would then increase the steady-state response to a frequency drift (Eq. (D.26)/60802)
- Increasing  $K_i$  would decrease the steady-state response to a frequency drift, which would be ok; however, this would result in a decrease in the damping ratio  $\zeta$  (Eq. (C.7)/60802), which would increase the gain peaking

# 2<sup>nd</sup> Order Filter – Design Constraints – 2

- In addition, even if  $K_i$  were increased, the amount of the increase would be limited because the resulting increase in  $\omega_n$  would tend to increase the 3 dB bandwidth
  - On this point, it might be thought that even though  $\omega_n$  increases with increasing  $K_i$ , since  $\zeta$  decreases the two effects would tend to cancel (Eq. (C.13)/60802)
  - It is true that, for  $\zeta \gg 1$ , 3 dB bandwidth is approximately equal to  $2\zeta\omega_n$  (Eqs. (C.6) and (C.7)); however, in the case here  $\zeta = 0.682$ ). In fact, increasing  $K_i$  increases the 3 dB bandwidth.
- Note that  $\omega_n$  is given by the same expression for both the continuous-time and discrete-time filters (Eq. (C.19)/60802). The expression for  $\zeta$  for the discrete-time filter is more complicated than for the continuous-time filter (it has an additional term compared to Eq. (C.7)); however, it is still true that increasing  $K_i$  increases the 3 dB bandwidth

# Endpoint Filter: Discrete Implementation Considerations

# Discrete Filter Implementation Considerations

- Translation from a continuous filter (as in Annex C) to a discrete filter implementation is well understood, but sample rate has an impact on performance.
- An assumption that filter sample point will be on arrival of Sync message is not accurate
  - Filter should operate with a consistent sample rate, but can't assume arrival of a Sync message every 125 ms (nominal)
    - Actual interval is permitted to be 119ms to 131 ms, so alignment with endpoint filter sample instance can be highly variable
- No matter what sample rate is used, the variability of the Sync Interval means the samples (input to the filter) must take RR (and, preferably, RR Drift) into consideration
  - There is no guarantee that an 8 Hz sample rate will align with arrival of Sync messages

# Concerns from Kilian Brunner's Implementation

- Implementation was a 2<sup>nd</sup> order discrete filter with an 8 Hz sample rate
- Could not meet -145 ns normative requirement for allowable range of dTE with Rate Ratio drift (or Rate Ratio + Neighbor Rate Rate drift)
  - Indicates phase performance of filter was not sufficient
  - Would require relaxing requirement to -200 ns or lower for this implementation to pass

# Resulting Questions

- Would raising the sample rate address the issue?
- Is raising the sample rate viable?
  - i.e. affordable for most implementations?
- If not...what other options are available?
  - Maybe higher order filters? (Which have their own computational cost.)
- Either way: should we make any changes to the specification?
  - Change normative requirements?
    - Not ideal: would require extensive simulation work to verify that goal are still met.
  - Add informative text?

# Raising the Sample Rate – Effectiveness? - 1

- The discrete-time filter model is obtained from the continuous-time filter model (Figure C.1/60802) by replacing  $1/s$  by  $1/(1 - z^{-1})$ ,  $K_i$  by  $K_i T_s$ , and  $K_o$  by  $K_o T_s$ , where  $T_s$  is the sampling time
- Geoff Garner modified Kilian Brunner's Python script characterizing the filters
  - 2<sup>nd</sup> order filter with sample rates: 8 Hz, 16 Hz, 32 Hz, 64 Hz, 100 Hz
- Note that this implementation (Python script) does not take account of Rate Ratio Drift between arrival of Sync Messages
  - Taking account of Rate Ratio Drift would probably improve performance

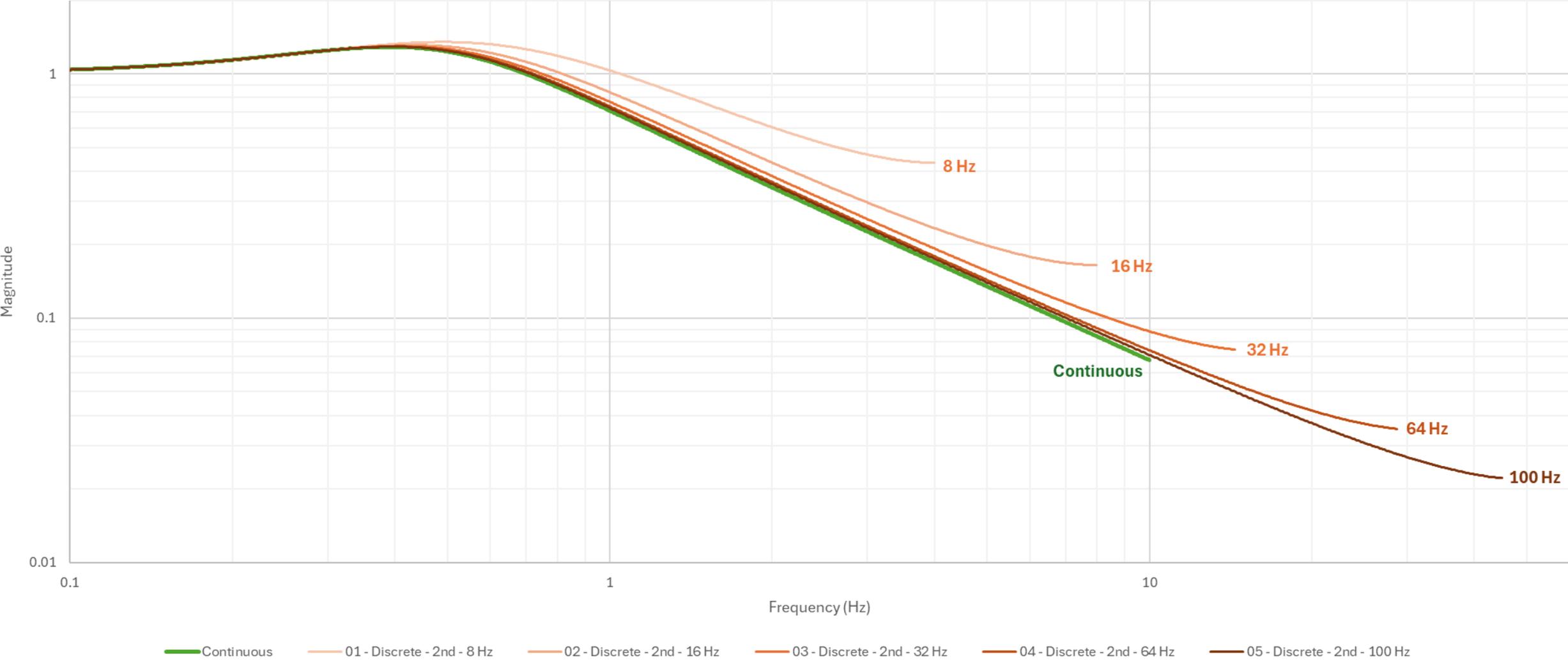
# Raising the Sample Rate – Effectiveness? – 2

- The script generated...
  - Bode plots (magnitude (frequency response) and phase) for the discrete-time system (curve labeled Td), continuous-time system with Annex C.4/60802 parameter values (curve labeled T), and continuous-time system error response (curve labeled S)
    - The discrete-time system error response (Sd) is not shown because this plot caused the frequency scale to extend to frequencies many orders of magnitude smaller and caused the plots for T and S to be obscured)
  - The continuous system transfer function is given by Eqs. (C.4) and (C.5) of Annex C/60802
  - The continuous system error transfer function is given by the negative of Eq. (D.25) of Annex D/60802
  - Error responses of the continuous-time and discrete-time systems (S and Sd, respectively) to a frequency drift of 1 ppm/s (i.e., 1000 ns/s<sup>2</sup>)
    - This is the response of each error transfer function to a frequency drift

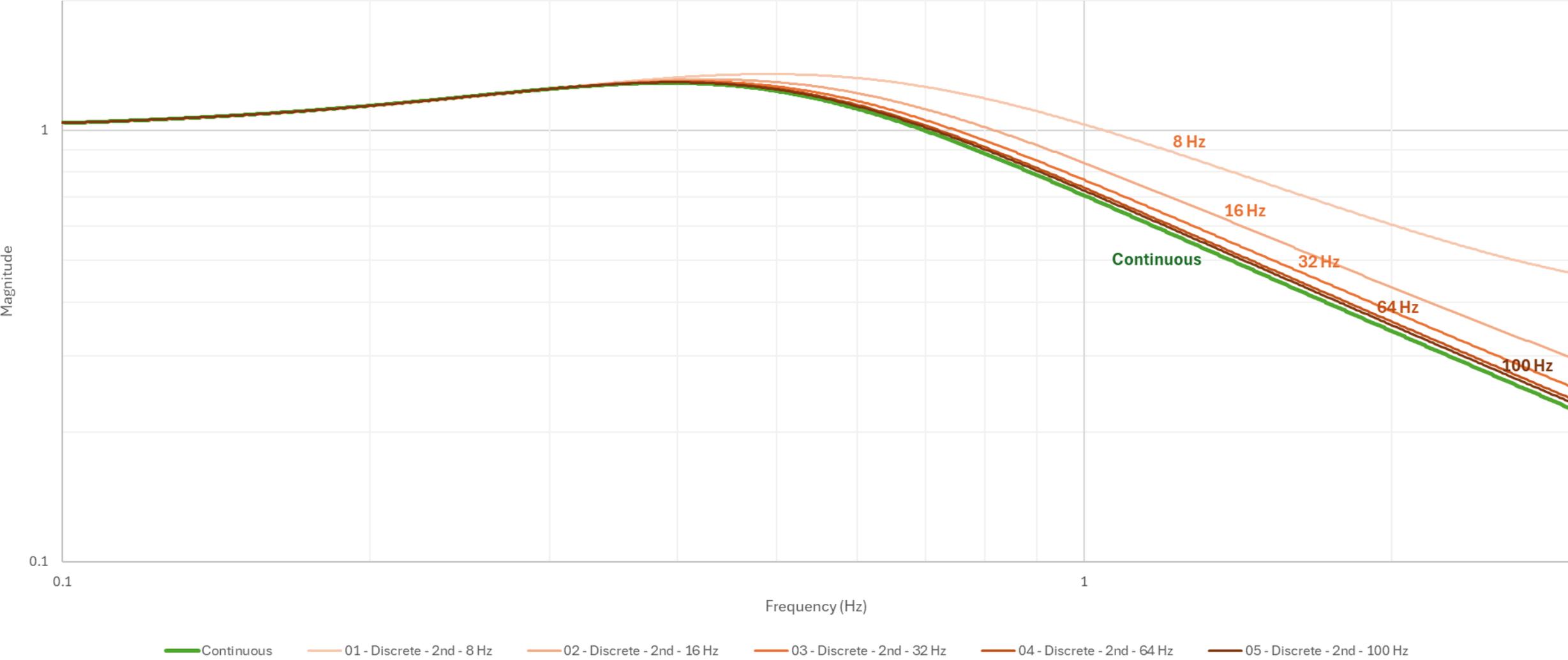
# Raising the Sample Rate – Effectiveness? – 3

- David McCall further modified the scripts to output the Magnitude and Phase data for each sample rate then combined them into single charts for easier comparison.
  - Original charts from the Python script are available in backup

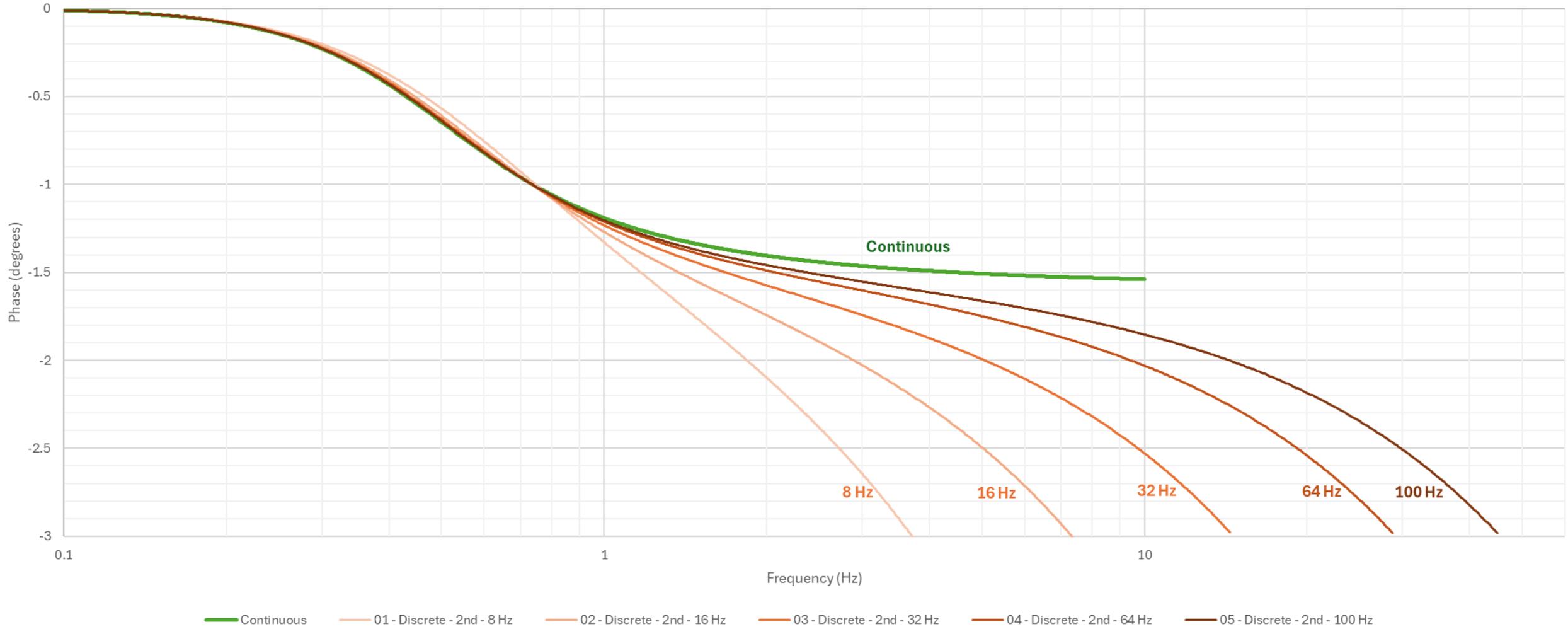
### Continuous vs. 2<sup>nd</sup> Order Discrete Filters



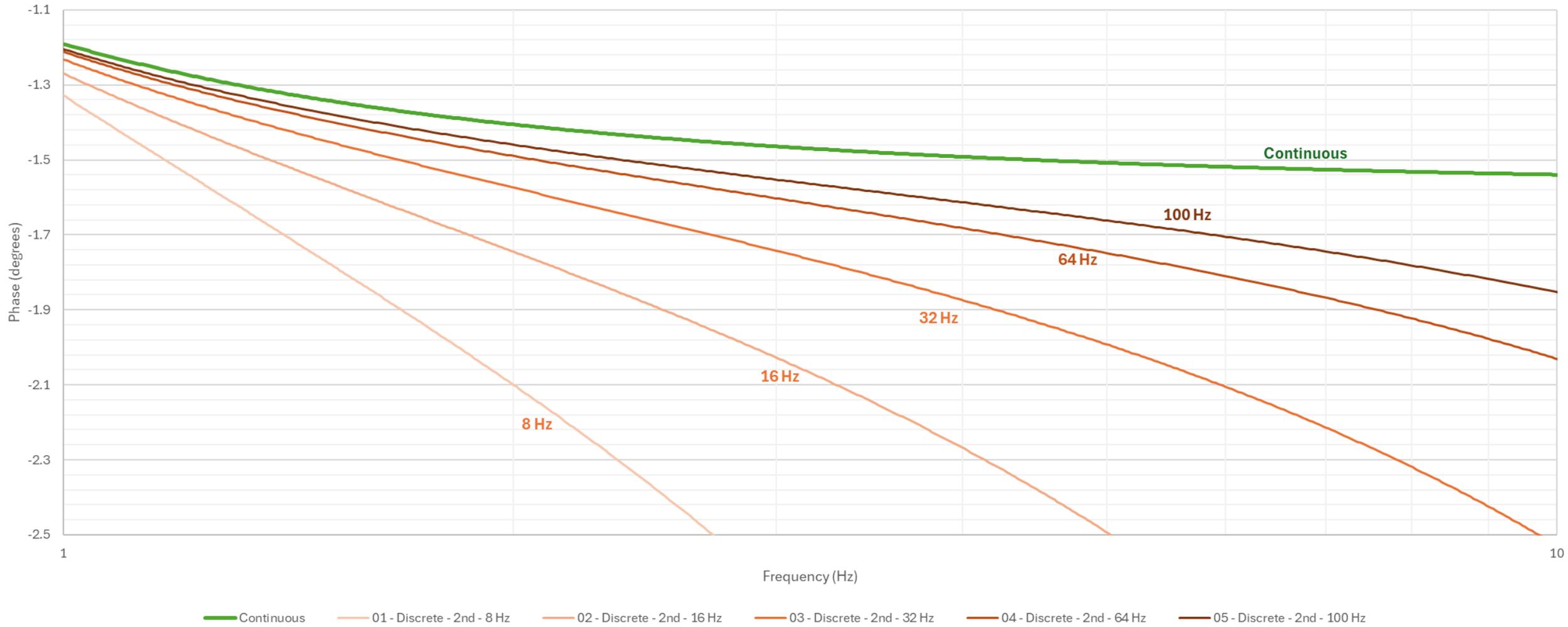
### Continuous vs. 2<sup>nd</sup> Order Discrete Filters



## Continuous vs. 2<sup>nd</sup> Order Discrete Filters



### Continuous vs. 2<sup>nd</sup> Order Discrete Filters



# Raising the Sample Rate – Effectiveness? – 4

- Raising the Sample Rate **does** address the concern
  - 100 Hz sample rate delivers performance very close to the continuous implementation.
  - Bode plots indicate that the discrete-time and continuous-time frequency responses are almost the same up to a reasonable fraction of the Nyquist frequency of 50 Hz. This is to be expected, since the sampling rate is large compared to the 3 dB bandwidth (1 Hz)
- Phase drop-off at higher frequencies is inherent to a discrete implementation but not an issue for this application

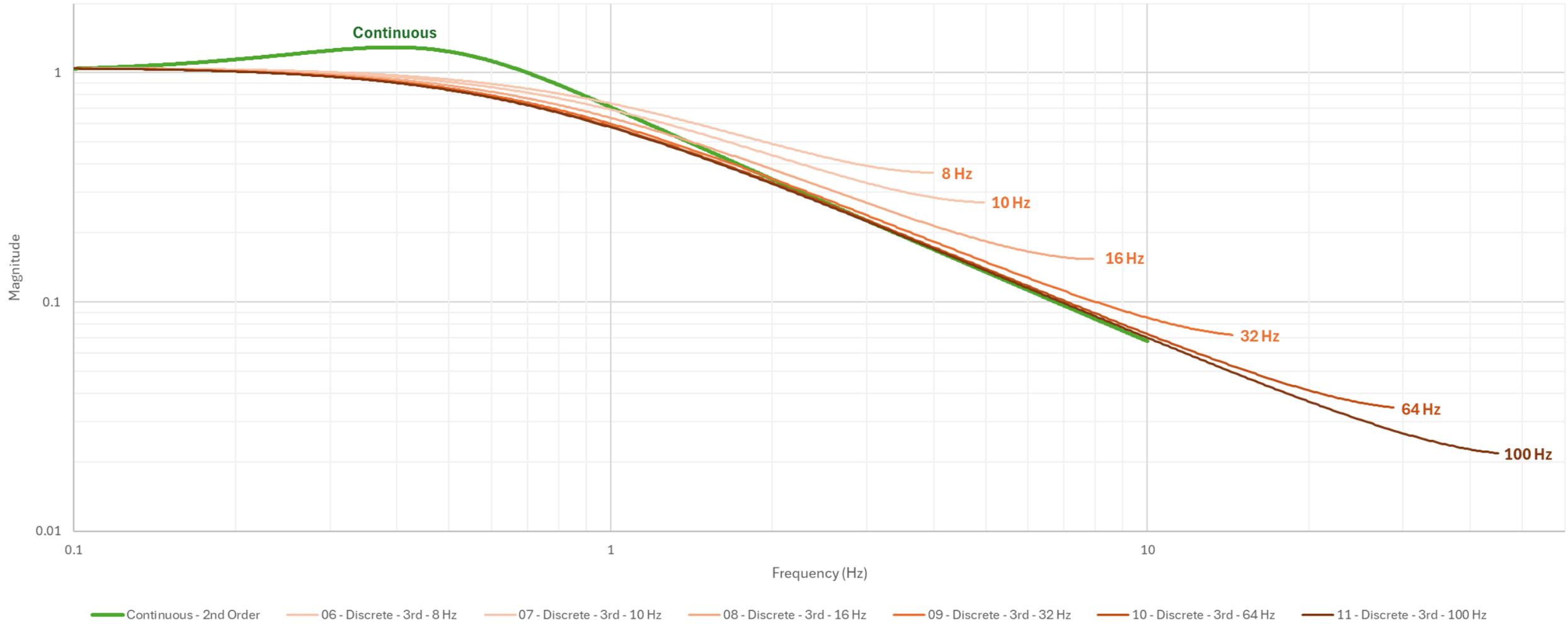
# Raising the Sample Rate – Feasibility

- Informal discussion with implementation experts at various device manufacturers have not identified any concerns with operating the endpoint filter at sample rates of 100 Hz or higher
  - Additional feedback is always appreciated

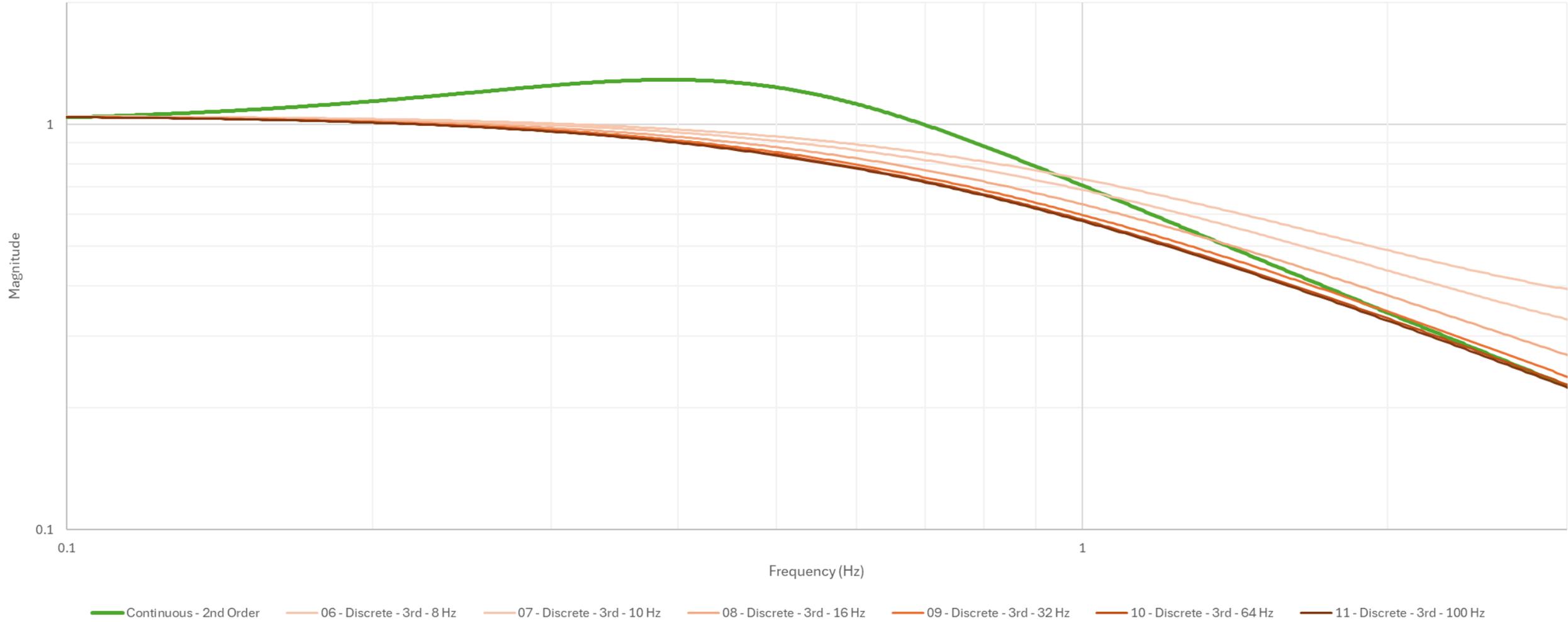
# Higher Order Filters? – 1

- Geoff Garner further modified the Python Script to characterize a 3<sup>rd</sup> order discrete implementation
  - Added to the feed-forward path, after the PI block, an additional PI block of the form  $1+aT_s/(1-z^{-1})$ , where  $a$  is a new parameter (in all cases,  $K_p$  and  $K_o$  are as in Annex C.4/60802,  $K_i = 0.5 \text{ s}^{-1}$ , and  $a = 0.1 \text{ s}^{-1}$ )
  - Sample Rates: 8 Hz, 10Hz, 16 Hz, 32 Hz, 64 Hz, 100 Hz
- David McCall again modified the scripts to output the magnitude and phase data so single comparison plots could be generated
  - Original charts from the Python script are available in backup

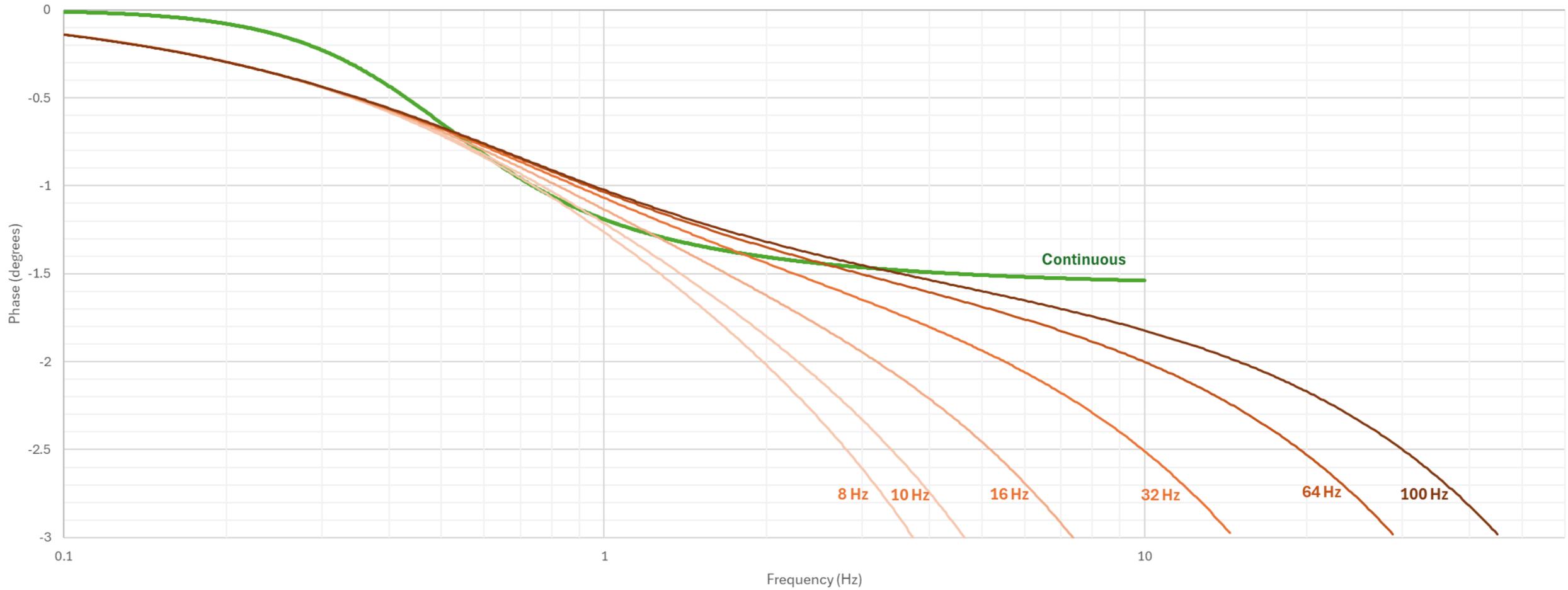
## 2<sup>nd</sup> Order Continuous vs. 3<sup>rd</sup> Order Discrete Filters



## 2<sup>nd</sup> Continuous vs. 3<sup>rd</sup> Order Discrete Filters

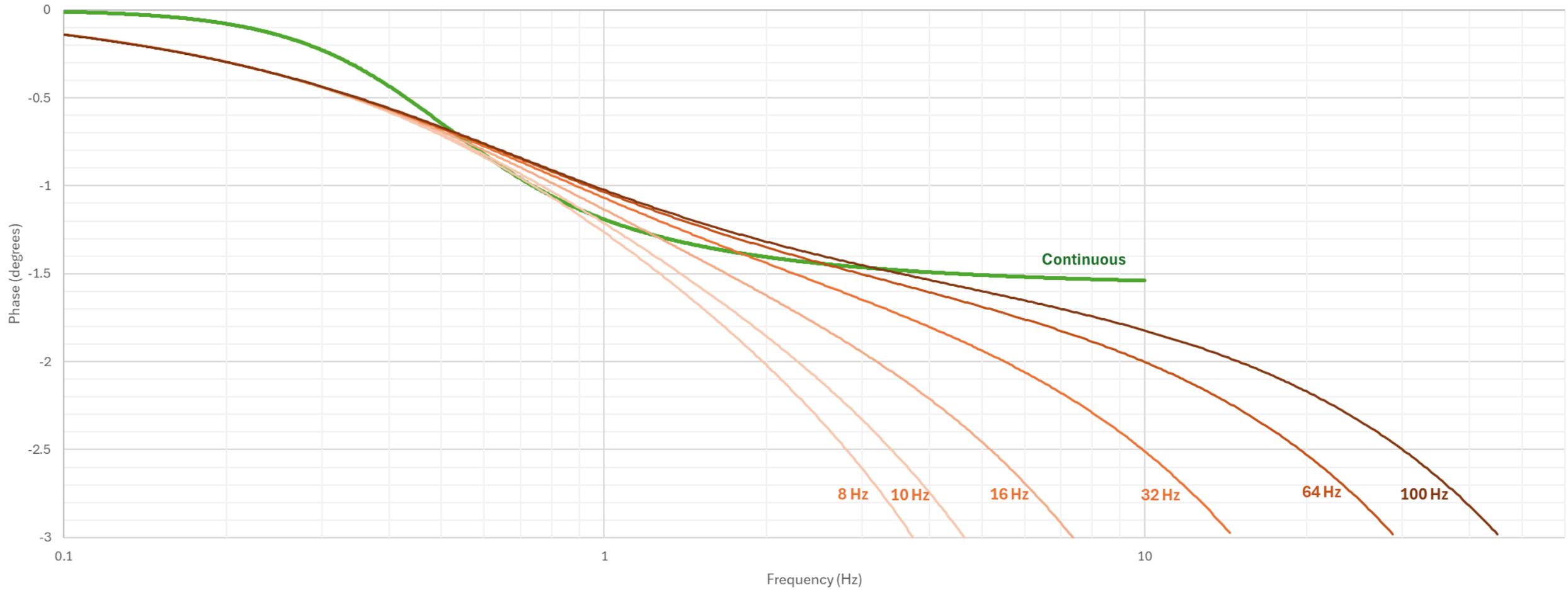


## 2<sup>nd</sup> Order Continuous vs. 3<sup>rd</sup> Order Discrete Filters



Continuous - 2nd Order    06 - Discrete - 3rd - 8 Hz    07 - Discrete - 3rd - 10 Hz    08 - Discrete - 3rd - 16 Hz    09 - Discrete - 3rd - 32 Hz    10 - Discrete - 3rd - 64 Hz    11 - Discrete - 3rd - 100 Hz

## 2<sup>nd</sup> Order Continuous vs. 3<sup>rd</sup> Order Discrete Filters

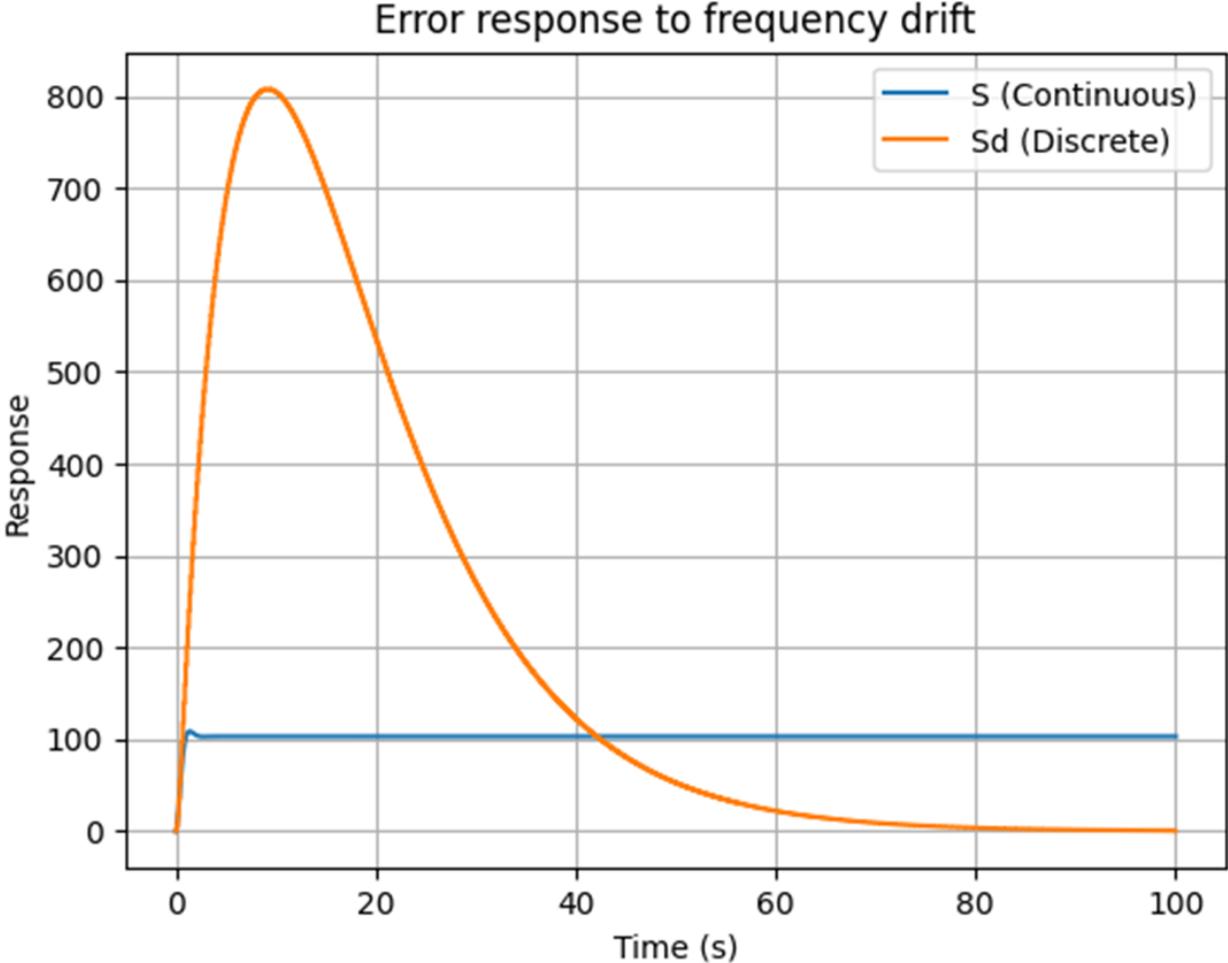


Continuous - 2nd Order    06 - Discrete - 3rd - 8 Hz    07 - Discrete - 3rd - 10 Hz    08 - Discrete - 3rd - 16 Hz    09 - Discrete - 3rd - 32 Hz    10 - Discrete - 3rd - 64 Hz    11 - Discrete - 3rd - 100 Hz

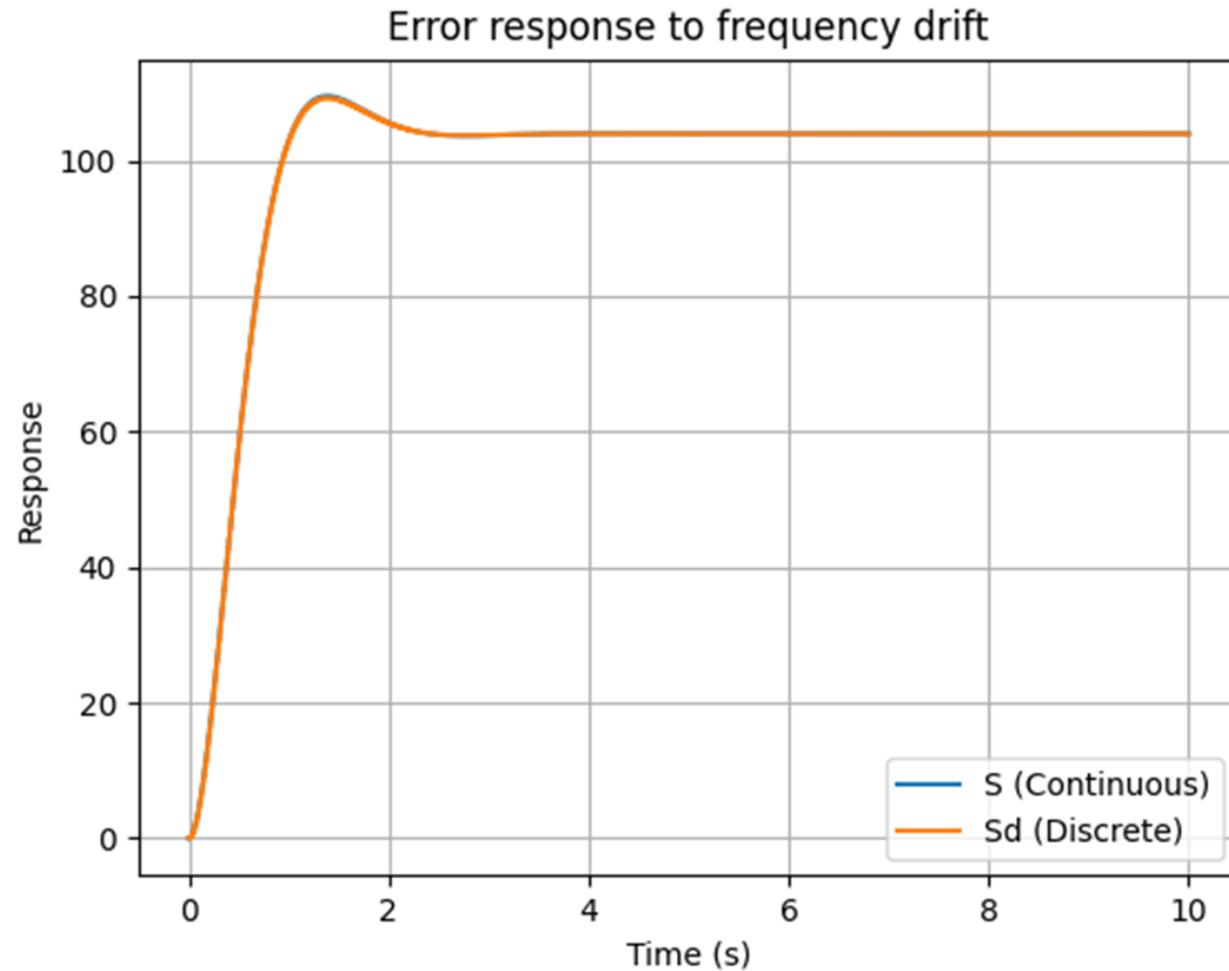
# Higher Order Filters? – 2

- A 3<sup>rd</sup> order discrete implementation allows the normative requirements to be met with a lower sampling rate
- Meets the 3 dB bandwidth requirement (0.9 – 1 Hz) for 10 Hz sampling rate
  - The 3 dB bandwidth is slightly more than 1 Hz for 8 Hz sampling rate, and very slightly less than 0.9 Hz for 16 Hz sampling rate
- The third-order filter has no gain peaking for all the sampling rates chosen. For the model chosen, i.e., addition of a PI filter in the feed-forward path, third-order filter is a Butterworth filter
- Also...

# 3<sup>rd</sup> Order – 10 Hz – Response to 1 ppm/s Frequency Drift



# 2<sup>nd</sup> Order – 100 Hz – Response to 1 ppm/s Frequency Drift



# Higher Order Filters? – 3

- As expected, the steady-state response of the third-order filter to a 1 ppm/s frequency drift approaches zero after a transient.
- The duration of the transient is approximately 80 s.
- Some concerns were raised about the feasibility of implementing higher order filters, but mainly from a lack of data (i.e. respondents had not tried it and were reluctant to commit to the feasibility)

# Conclusion & Proposed Addition to Annex C

# Conclusion

- No change is required to the normative requirements
  - Normative requirements can be met with discrete 2<sup>nd</sup> order filter implementations running at 100 Hz or higher sample rate
  - This is feasible to implement on current hardware
- Other implementation are possible but may carry some risks.
  - More work would be required to characterise the risks; work that is outside the scope of the current effort
  - Impact of any risk is limited to the End Instance implementation
- Some guidance regarding considerations for discrete implementation of the endpoint filter would be a good idea
  - In particular, engineers with a process control background might assume that a sample rate of 8 Hz – based on the 125 ms Sync Interval – may be sufficient

# Recommended Addition to Annex C

(If agreed, sample text can be contributed)

- C.5 Discrete Filter Implementation Considerations
- Covers the following topics...
  - Conversion from continuous implementation to discrete implementation
  - Sample rate considerations
    - Regardless of sample rate, most samples will not align with arrival of a Sync message
    - Behaviour of unfiltered output (filter input) between arrival of sample messages
      - Inclusion of Rate Ratio Drift
      - Approximation algorithm in Annex D
    - Analysis indicates...
      - 8 Hz sample rate is too low
      - 100 Hz or higher can meet all normative requirements

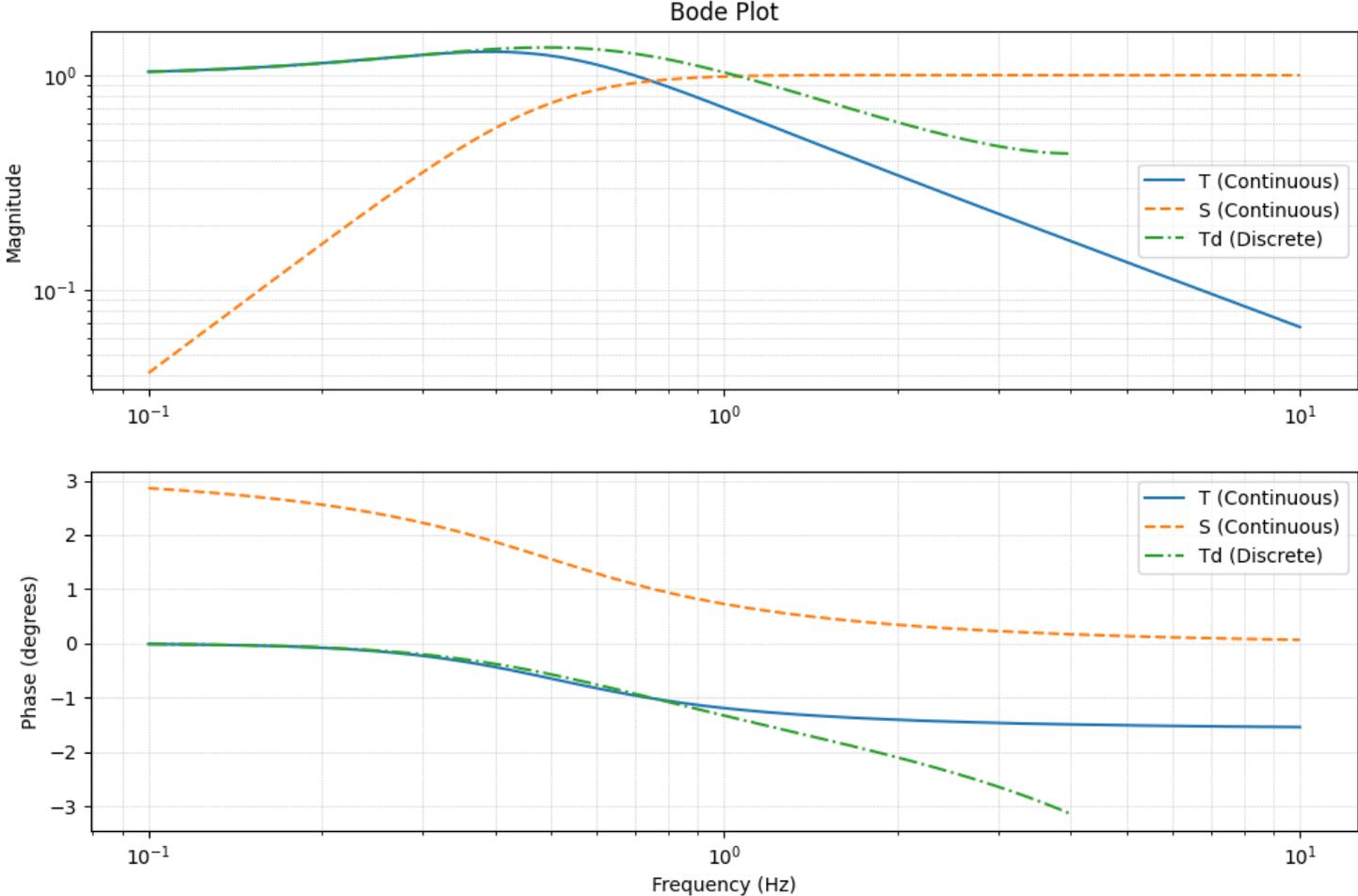
From discussions during IEC/IEEE 60802  
call on Monday 5<sup>th</sup> May 2025

# Comment Resolution

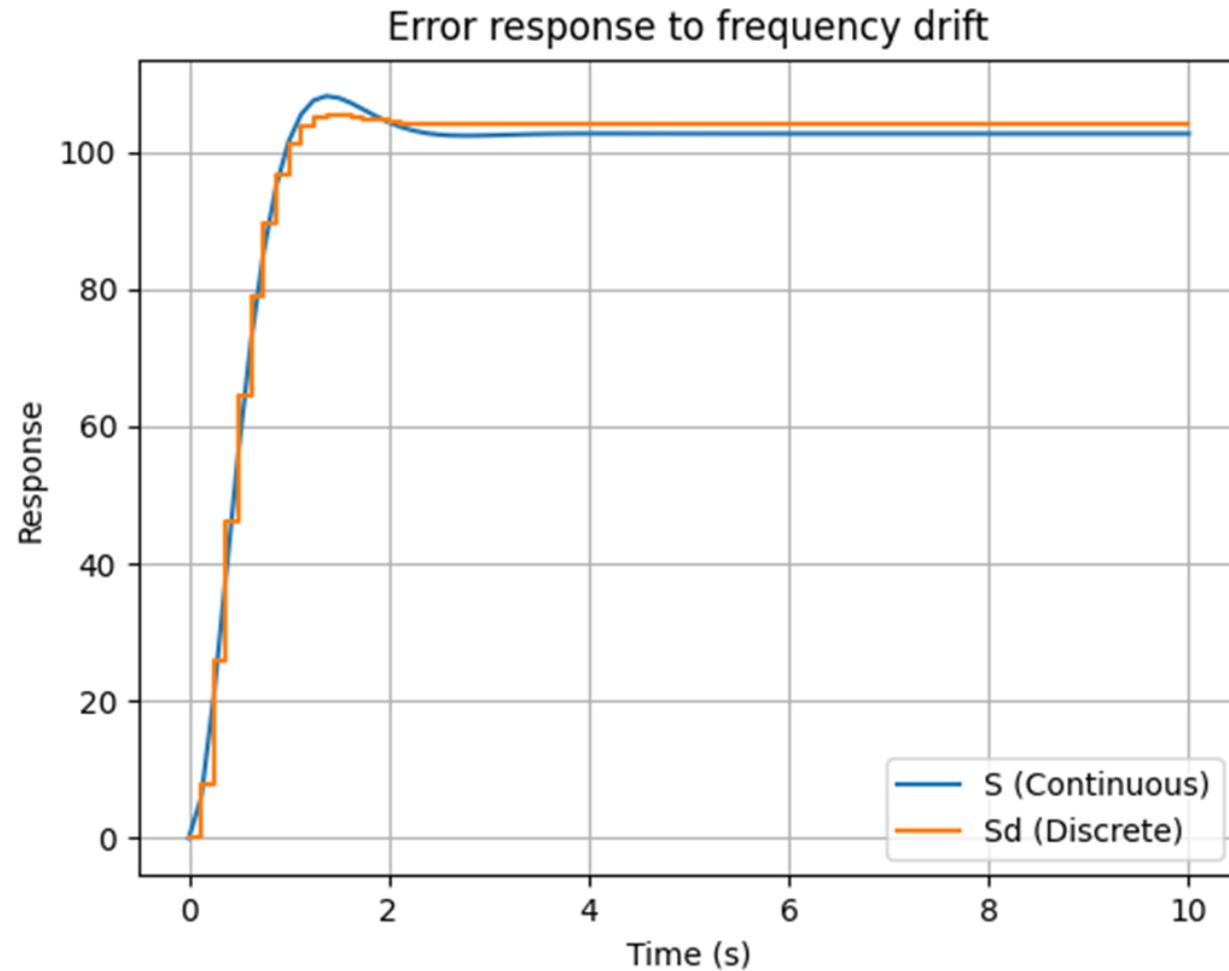
- The related comment that this presentation addresses (# R2-15) was REJECTED by group consensus following discussion.
  - Formal reason was “This comment is out of scope as it is on an unchanged portion of the draft.”
  - Informally, the group felt that...
    - Any change would be to informative text; no normative requirements would change.
    - Any change would open up the potential for new comments, putting the schedule for publication of IEC/IEEE 60802 at risk.
    - Any change would either...
      - Be short, and not terribly informative
      - Be long, further increasing the potential for new comments.
    - There are other paths to educate implementers about relevant considerations for the endpoint filter.
    - Overall, the cost / benefit of making a change to the specification at this stage was in favour of **not** making a change to this edition of the specification, but possibly reconsider as a Maintenance item or for inclusion in an amendment. Hence: REJECTED.

# Backup

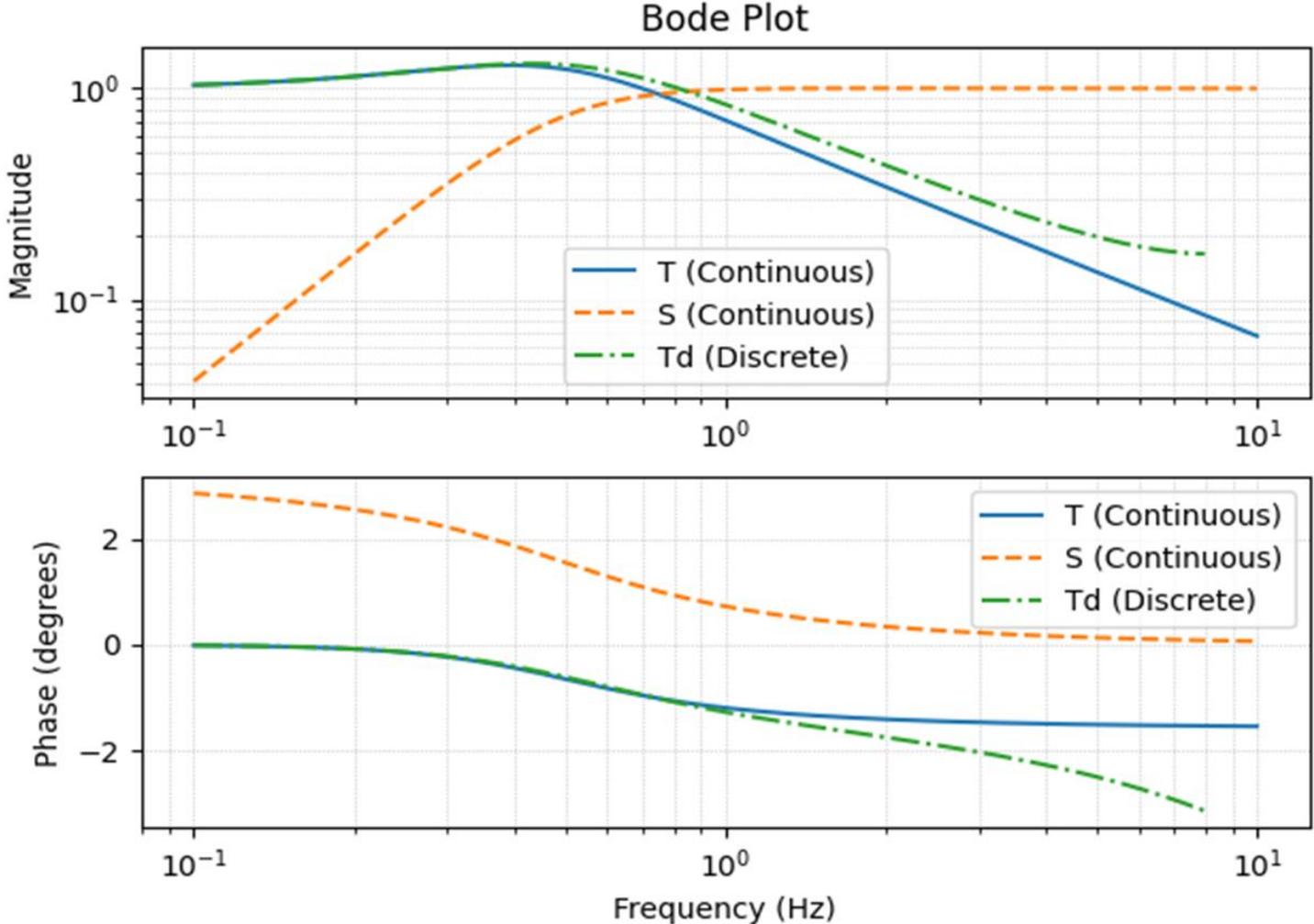
# 2<sup>nd</sup> Order – 8 Hz – Bode Plots



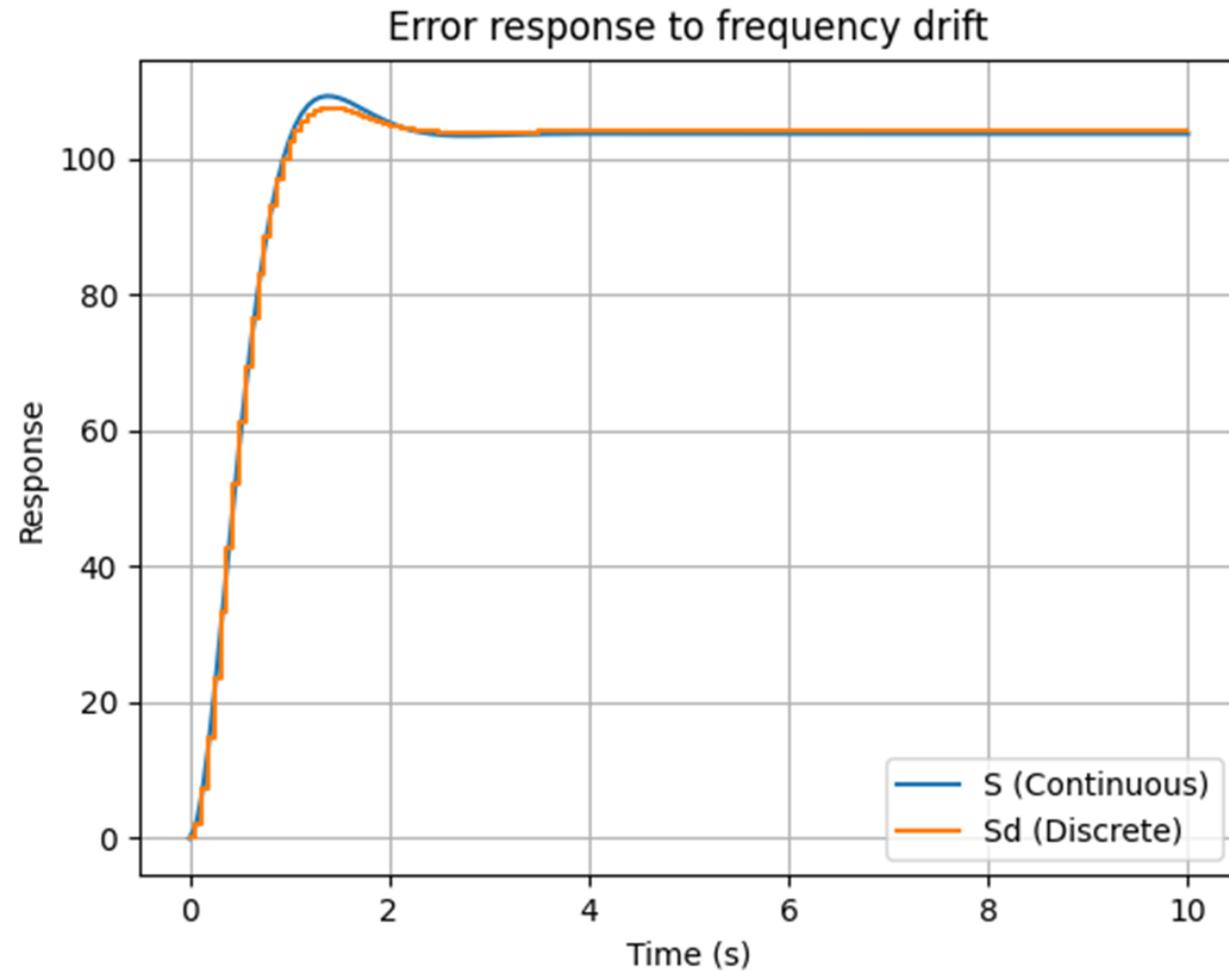
# 2<sup>nd</sup> Order – 8 Hz – Response to 1 ppm/s Frequency Drift



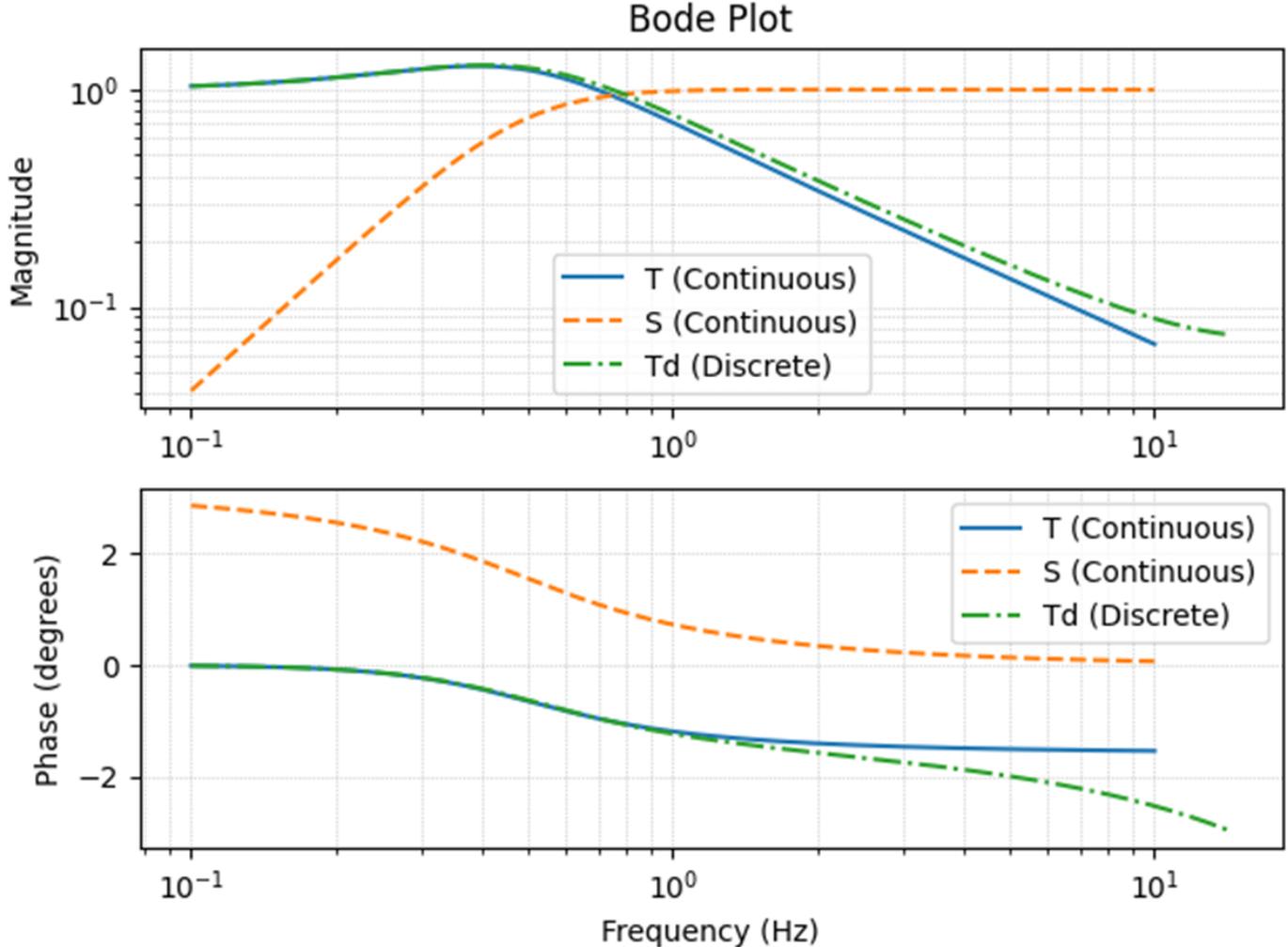
# 2<sup>nd</sup> Order – 16 Hz – Bode Plots



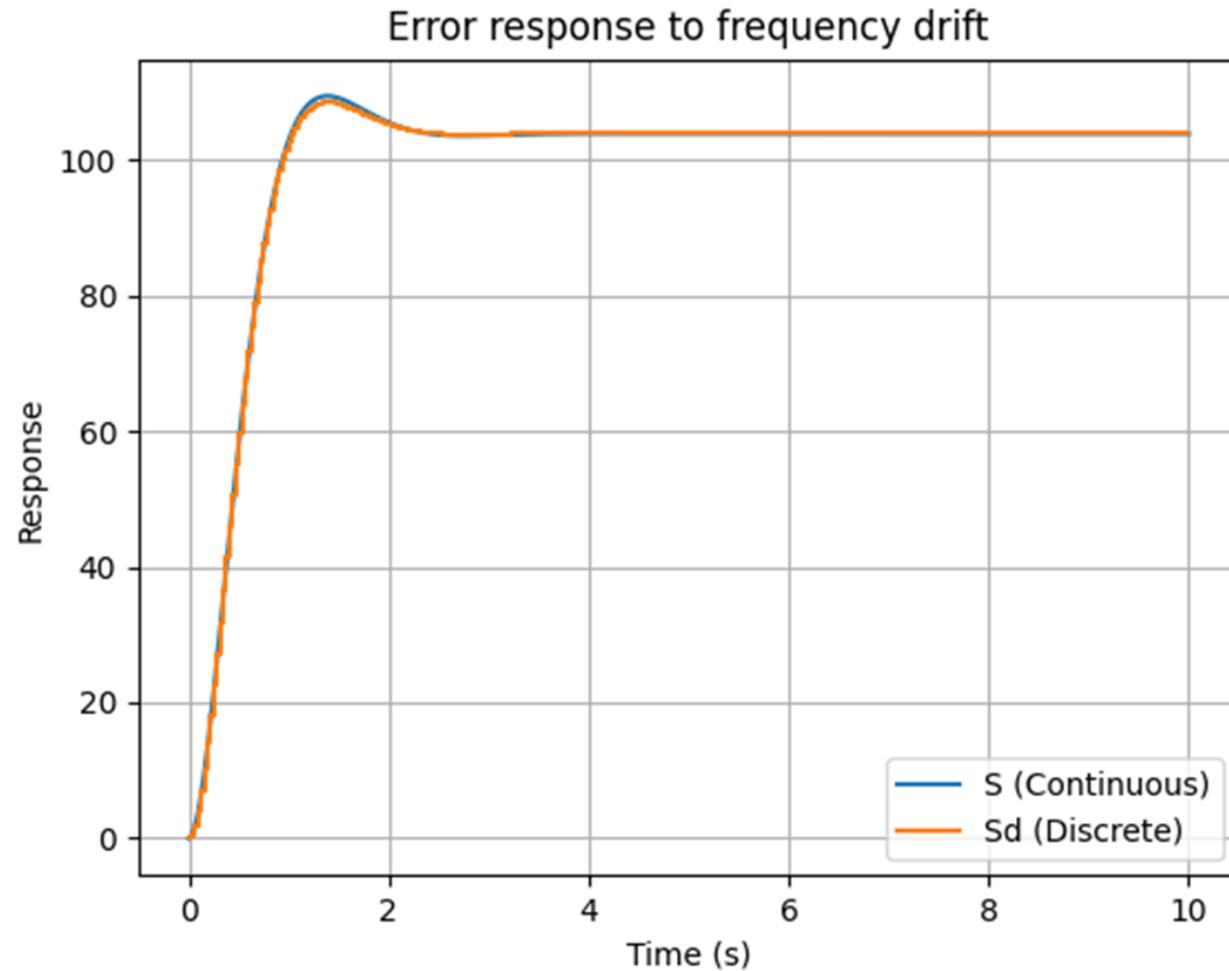
# 2<sup>nd</sup> Order – 16 Hz – Response to 1 ppm/s Frequency Drift



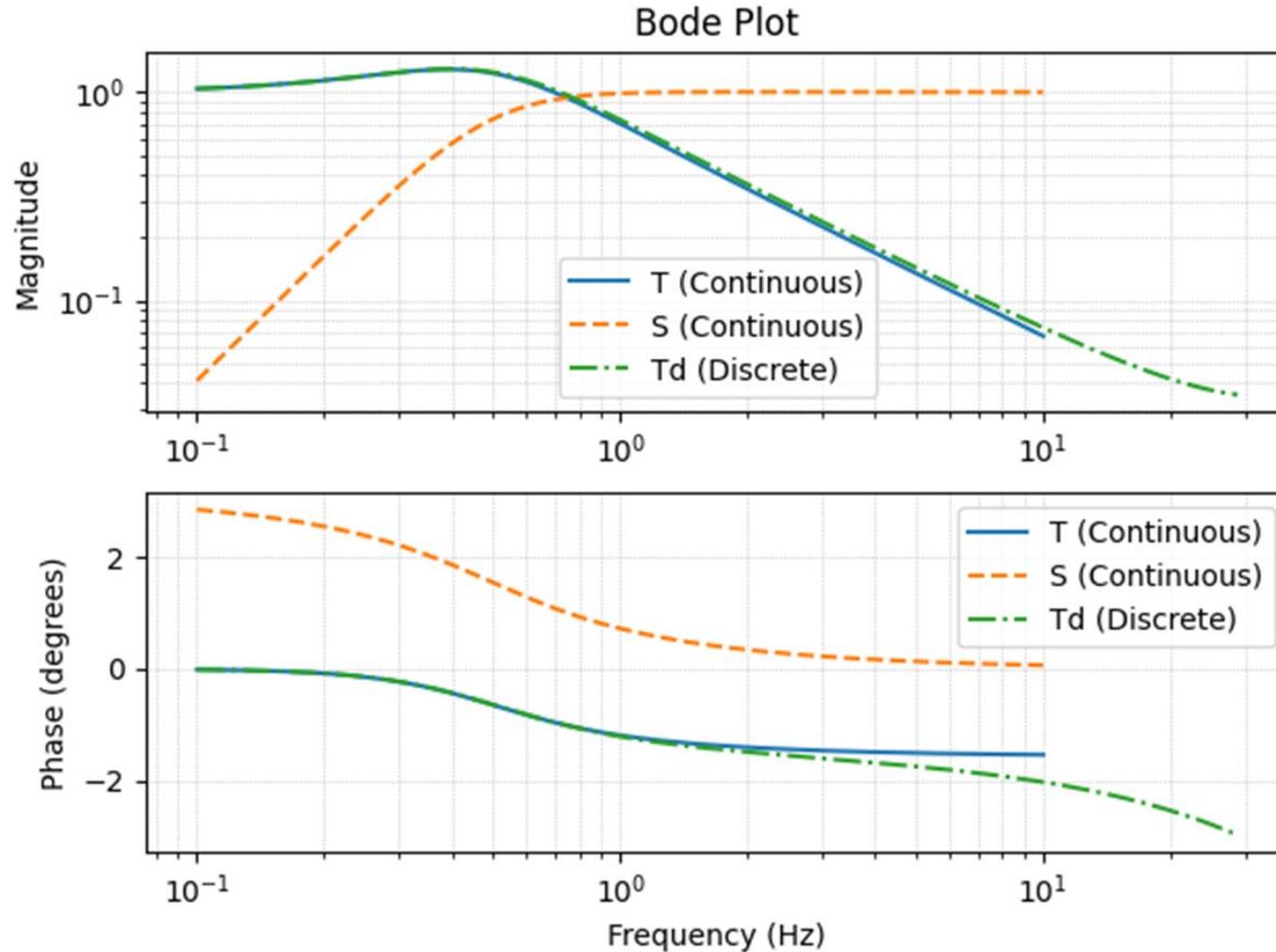
# 2<sup>nd</sup> Order – 32 Hz – Bode Plots



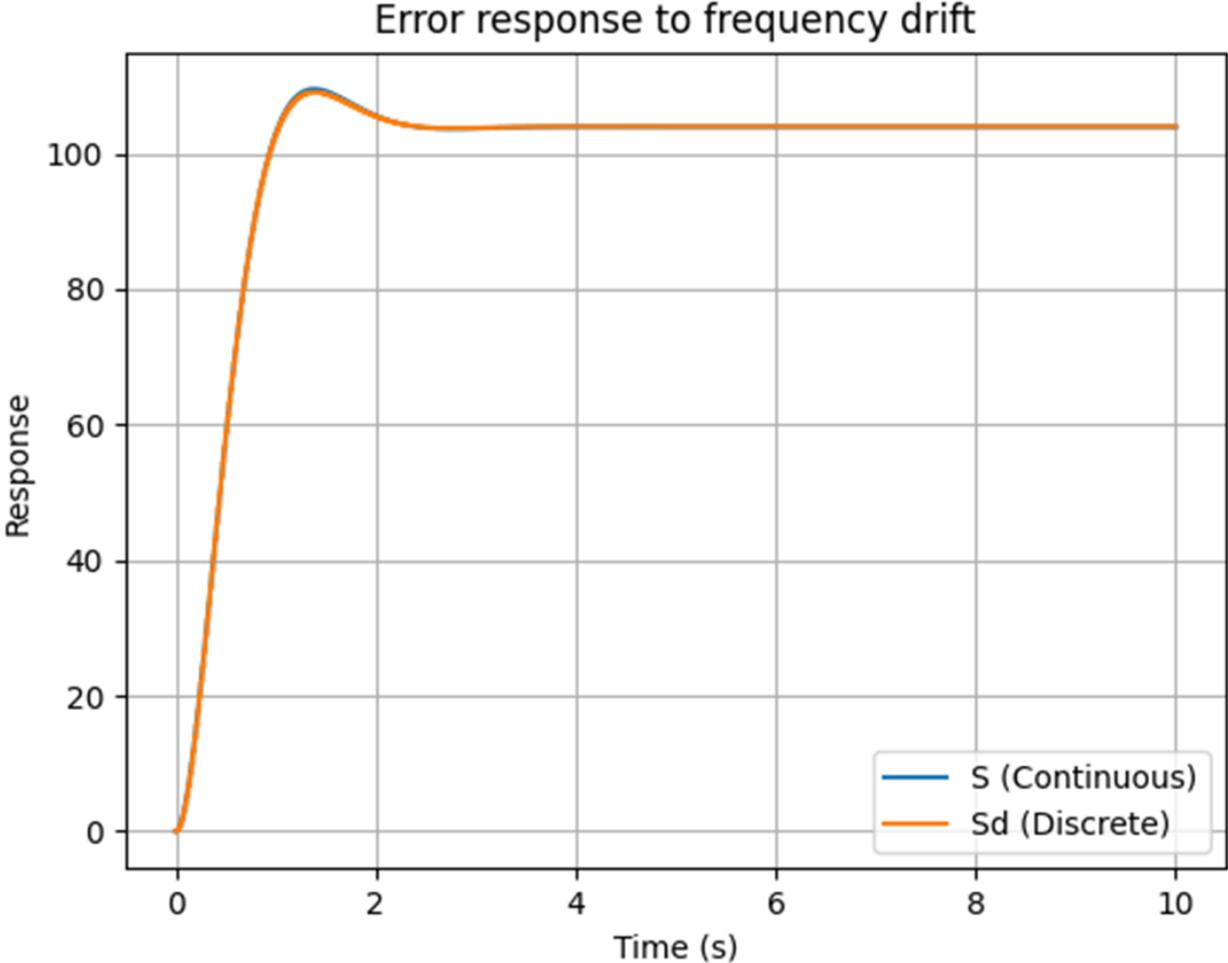
# 2<sup>nd</sup> Order – 32 Hz – Response to 1 ppm/s Frequency Drift



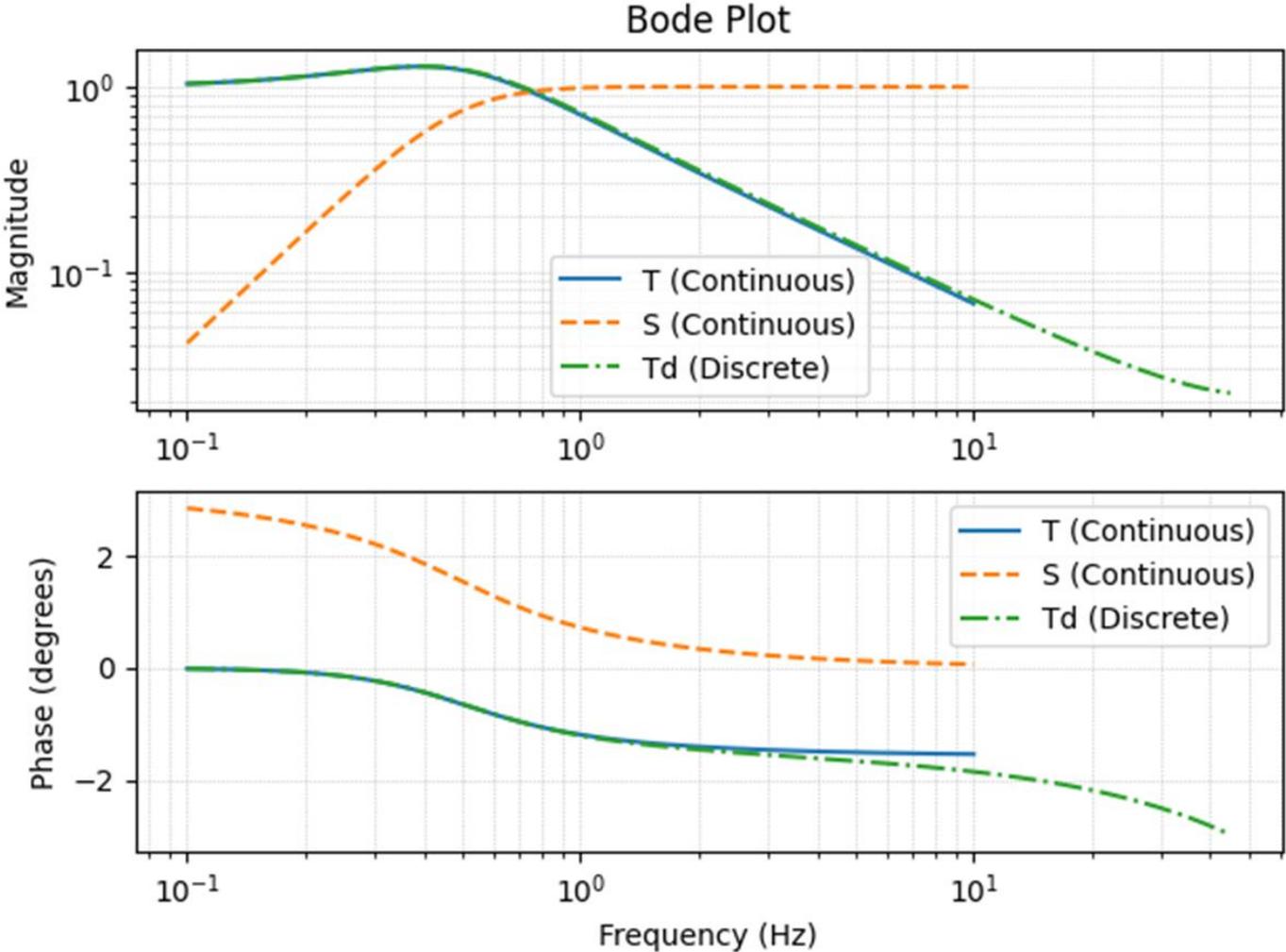
# 2<sup>nd</sup> Order – 64 Hz – Bode Plots



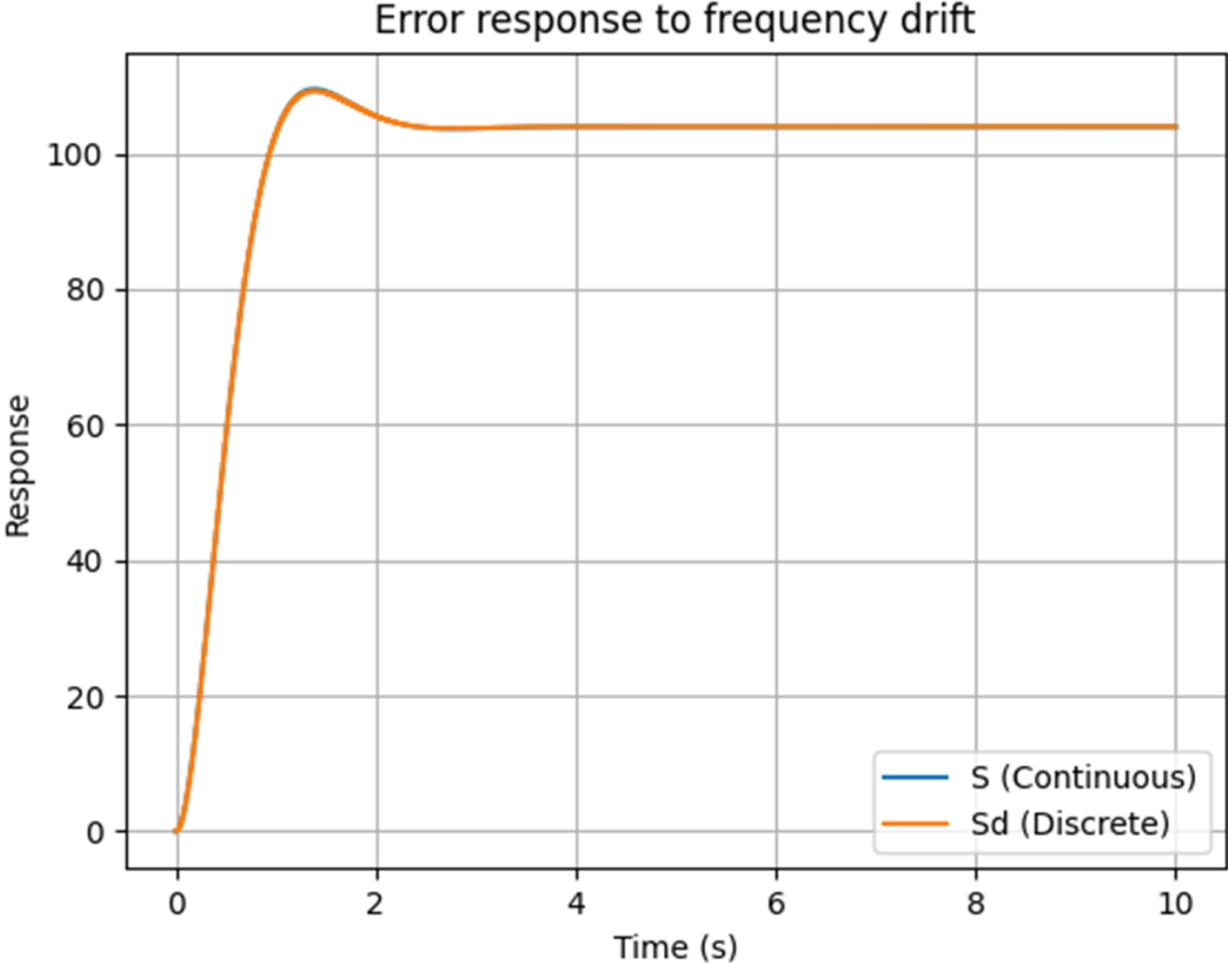
# 2<sup>nd</sup> Order – 64 Hz – Response to 1 ppm/s Frequency Drift



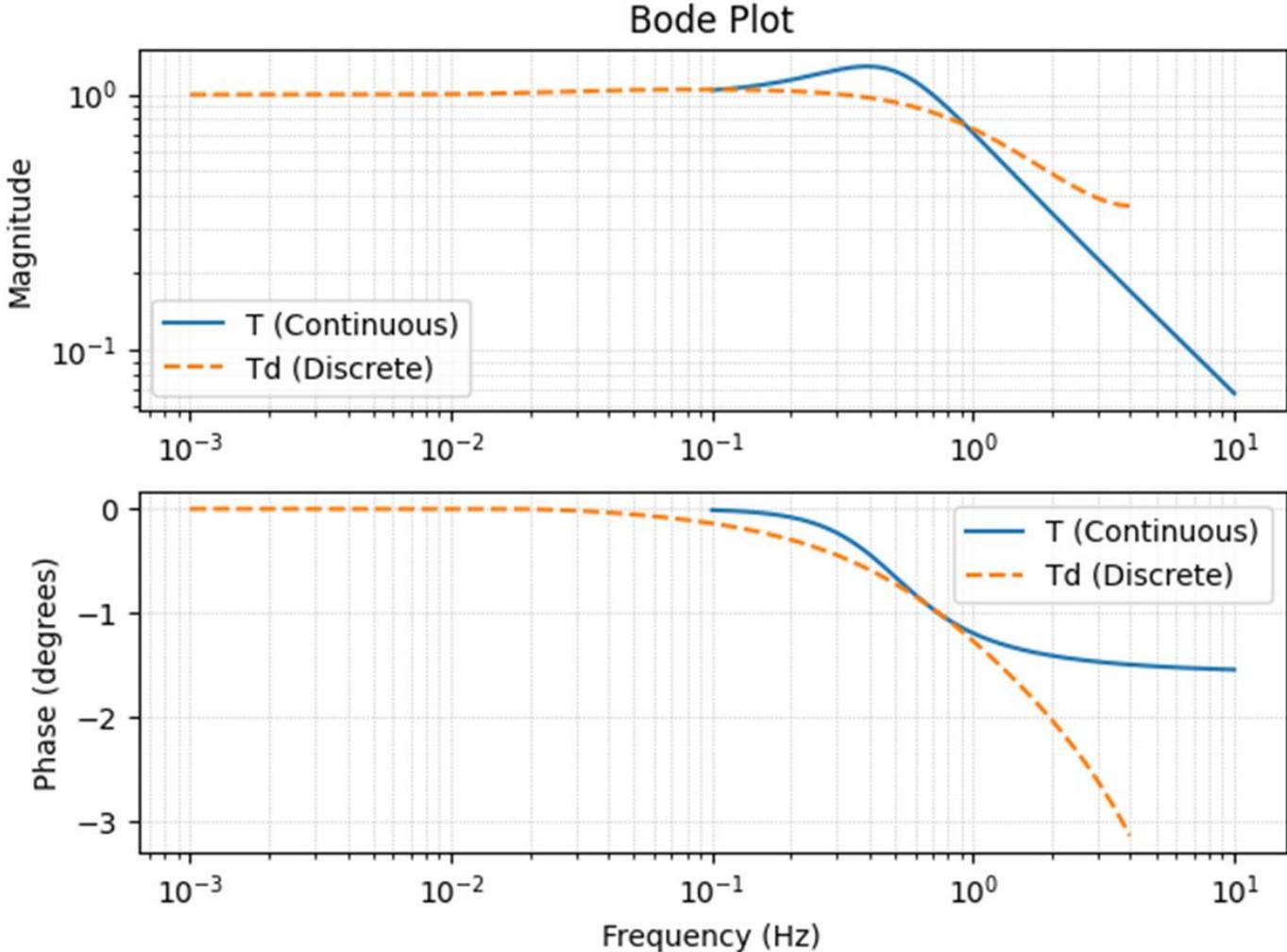
# 2<sup>nd</sup> Order – 100 Hz – Bode Plots



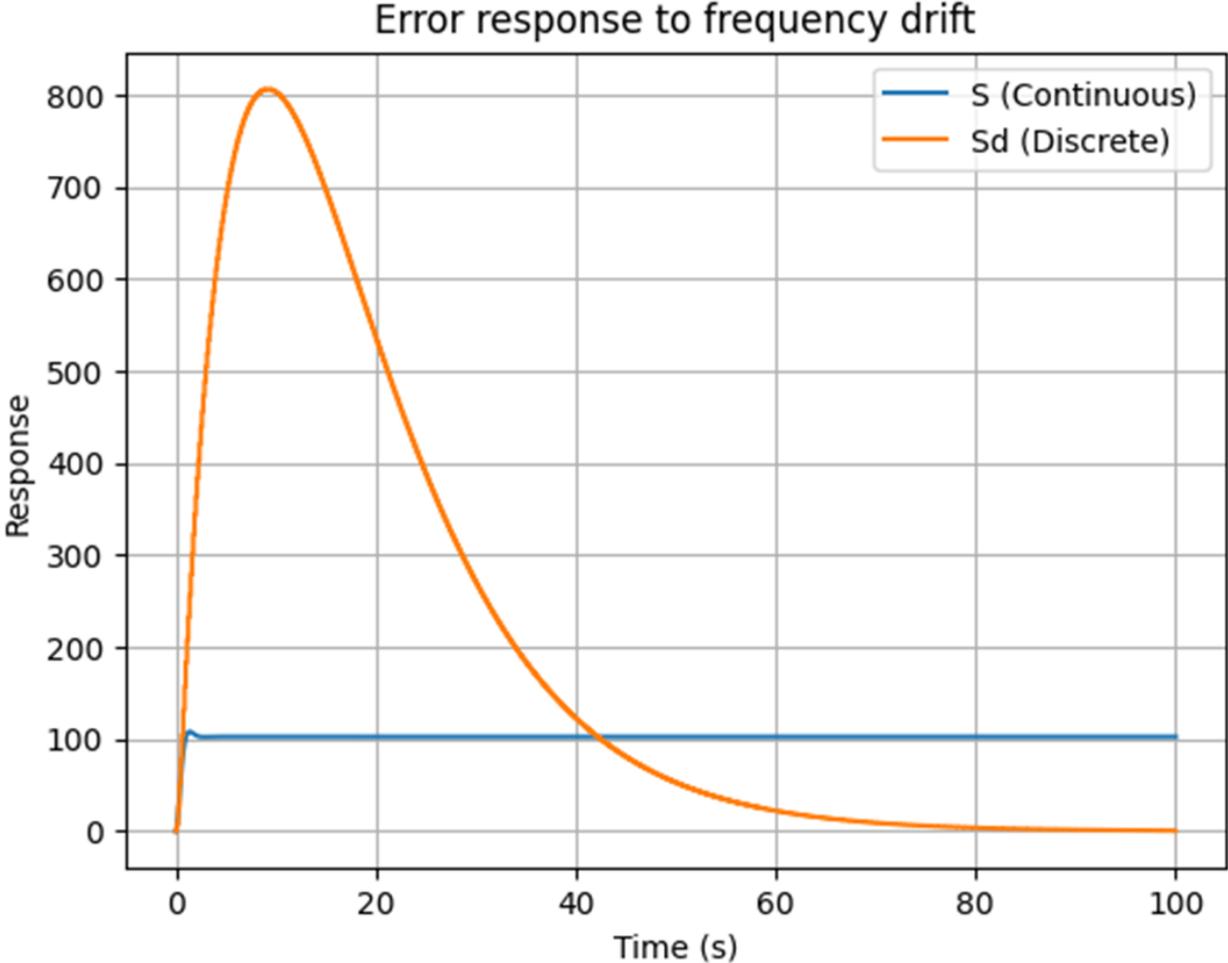
# 2<sup>nd</sup> Order – 100 Hz – Response to 1 ppm/s Frequency Drift



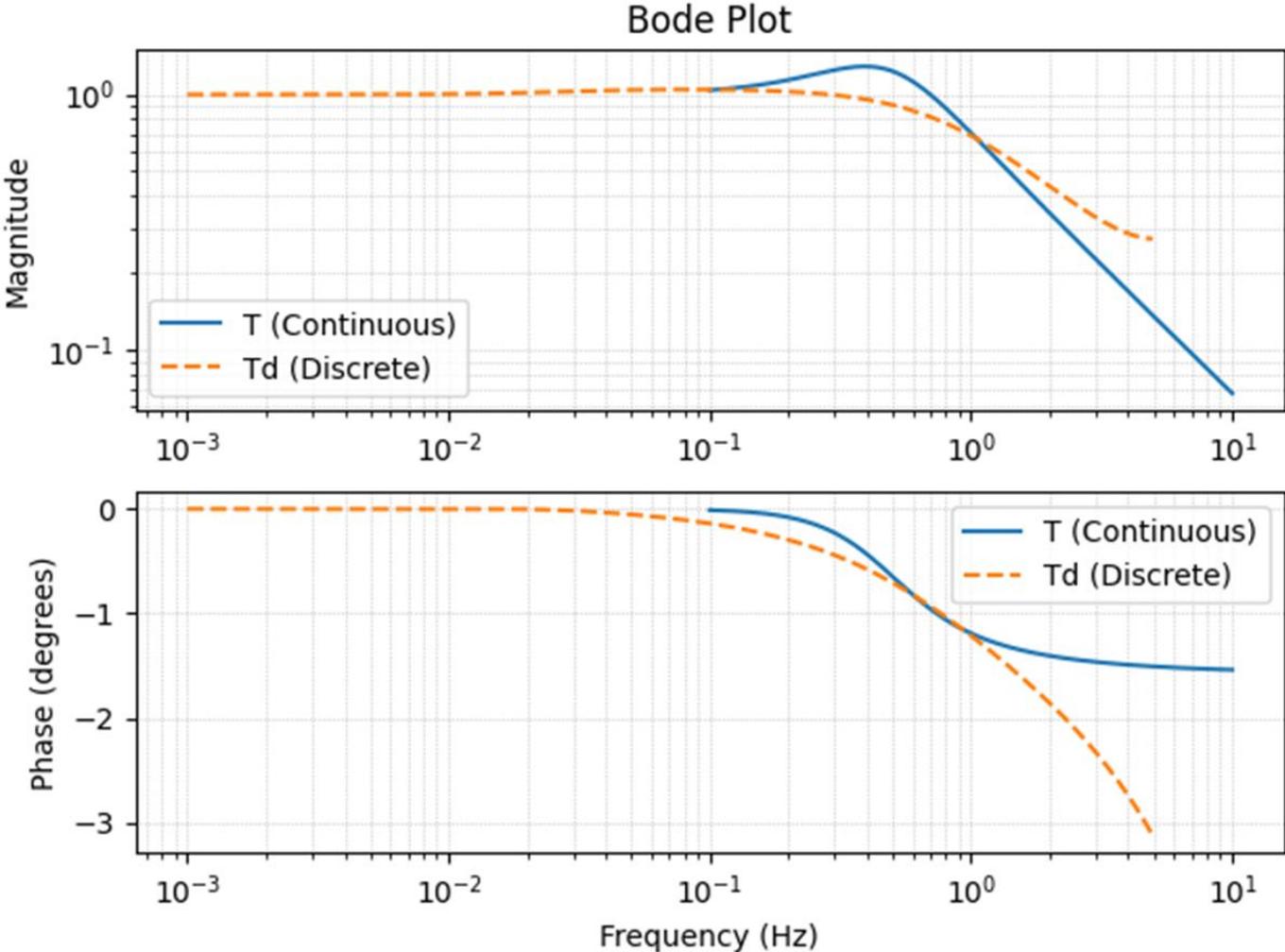
# 3<sup>rd</sup> Order – 8 Hz – Bode Plots



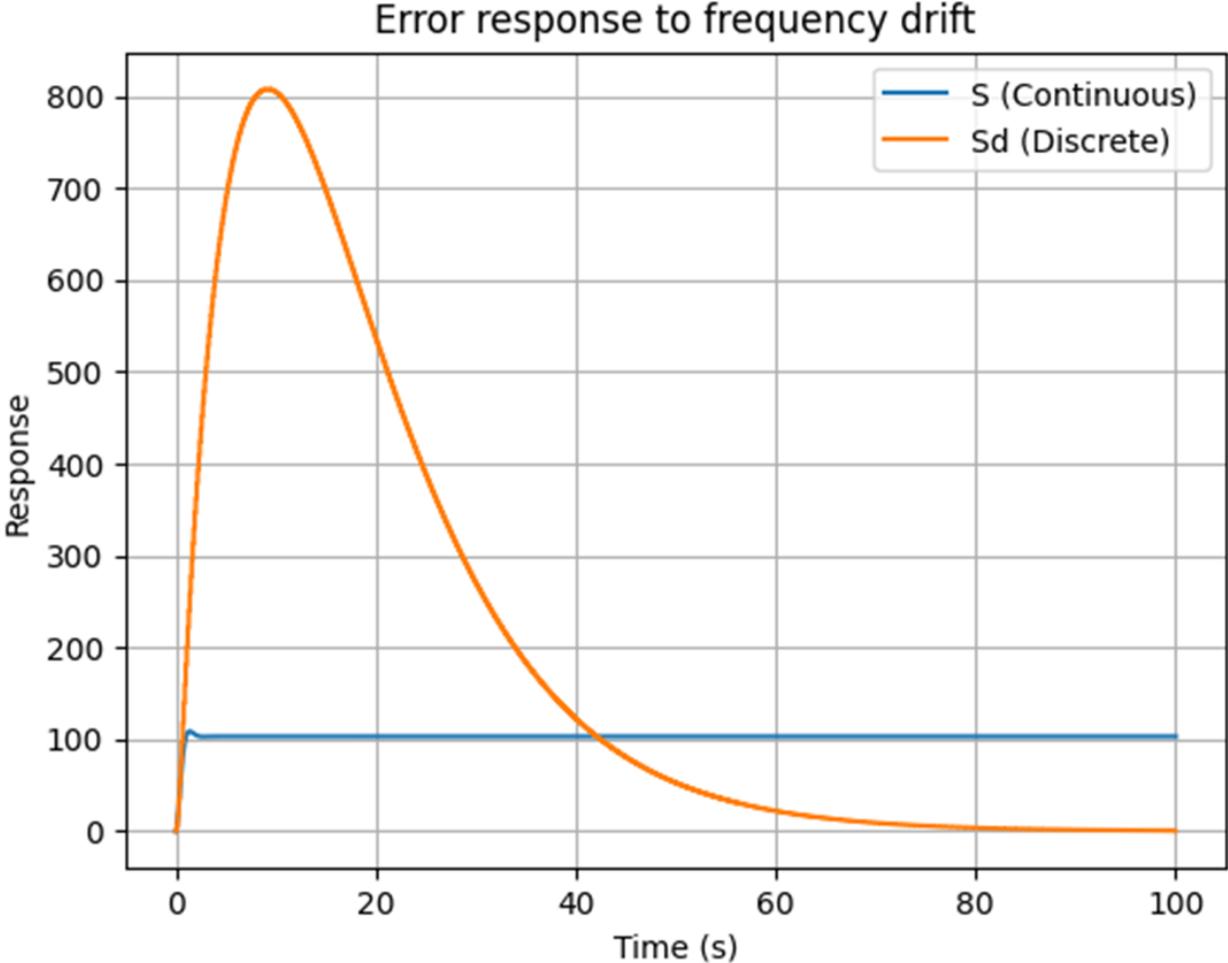
# 3<sup>rd</sup> Order – 8 Hz – Response to 1 ppm/s Frequency Drift



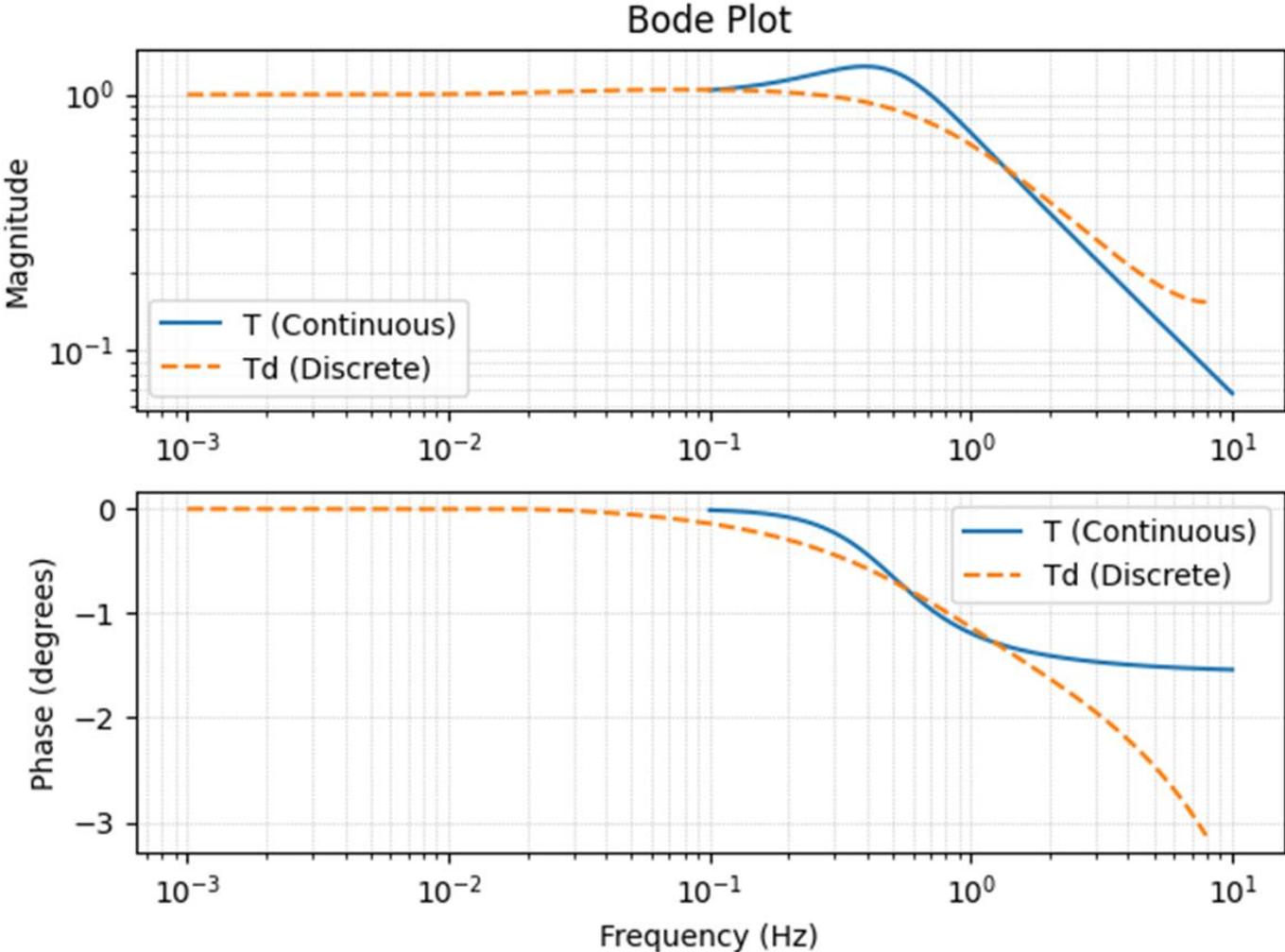
# 3<sup>rd</sup> Order – 10 Hz – Bode Plots



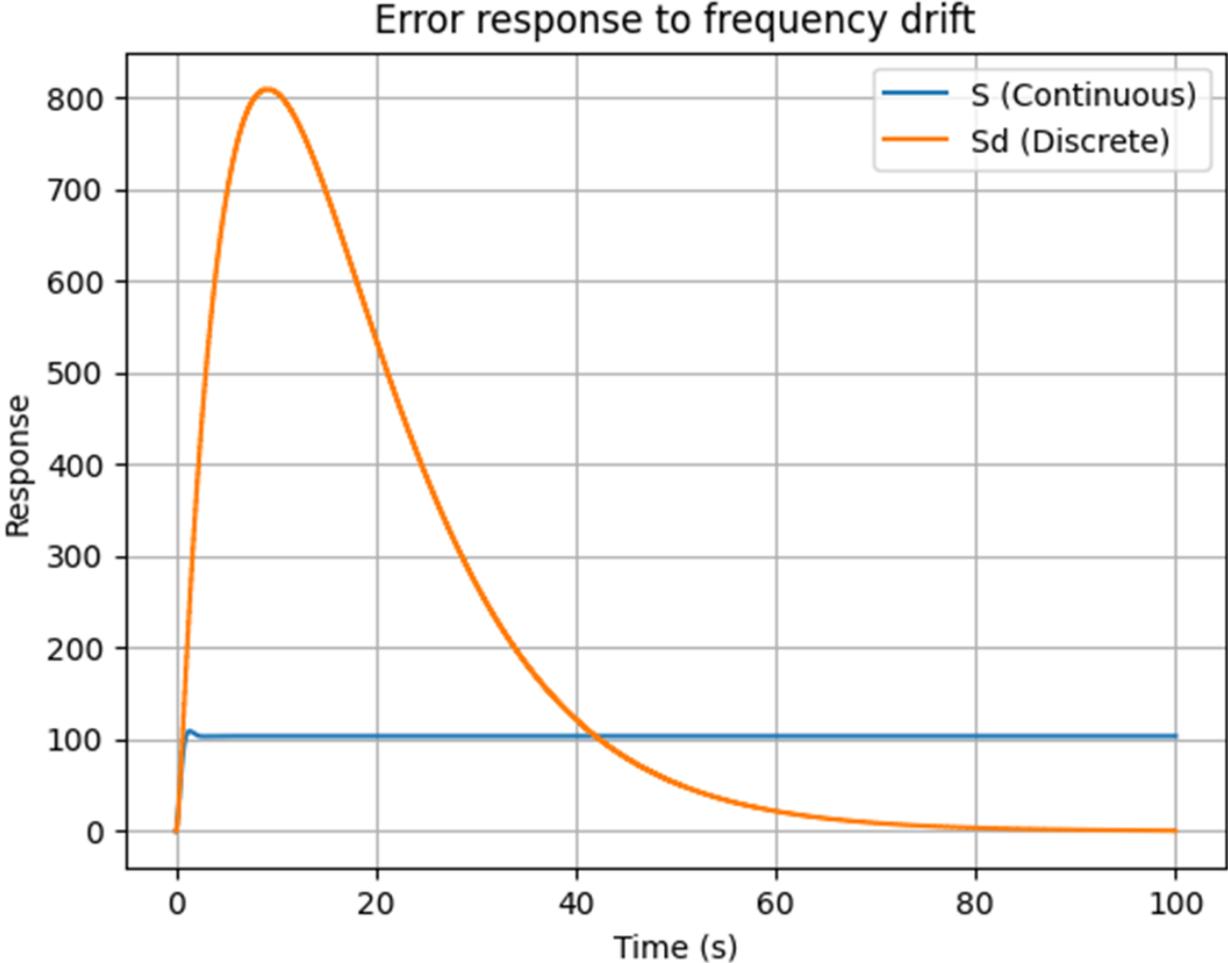
# 3<sup>rd</sup> Order – 10 Hz – Response to 1 ppm/s Frequency Drift



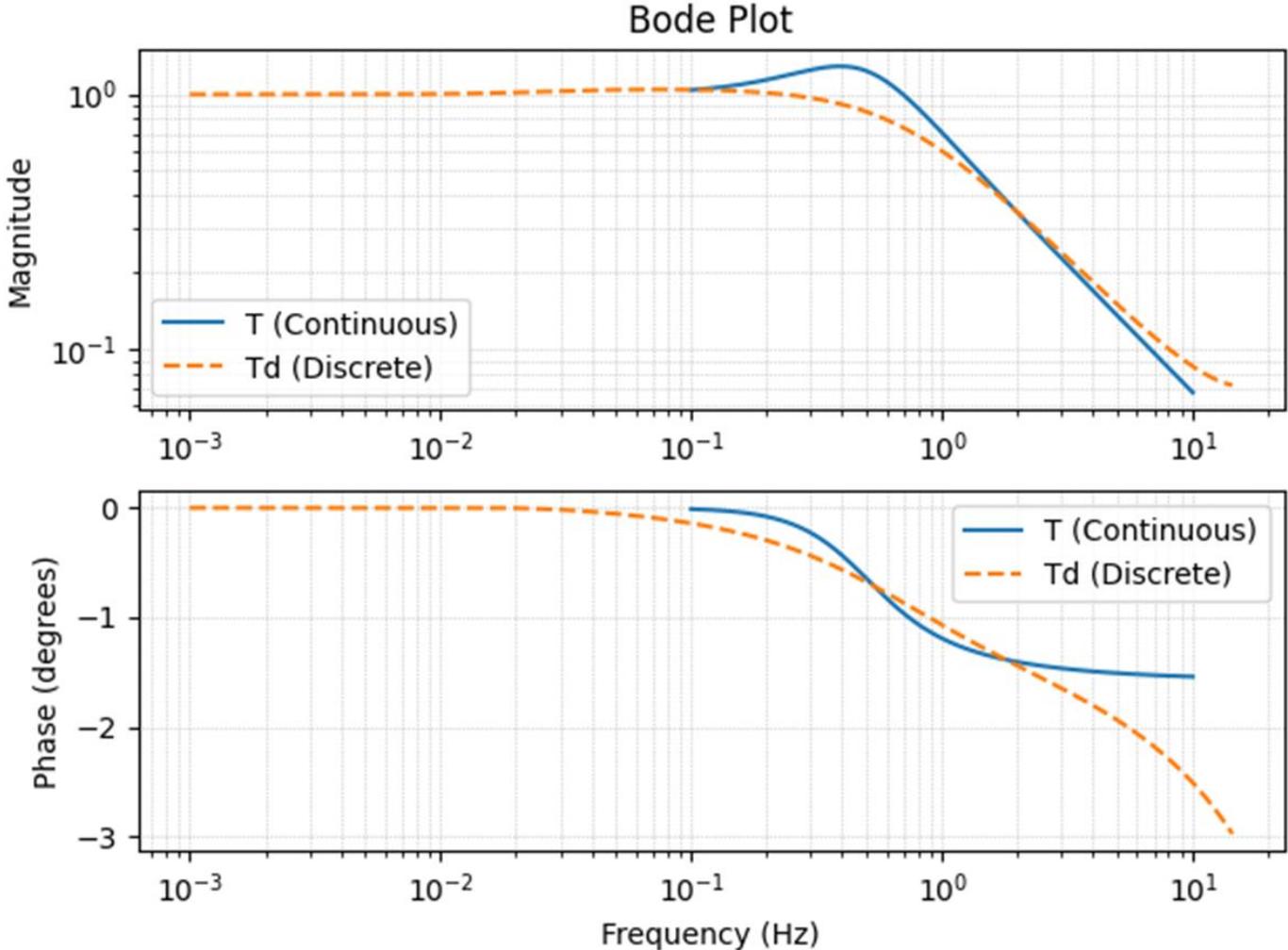
# 3<sup>rd</sup> Order – 16 Hz – Bode Plots



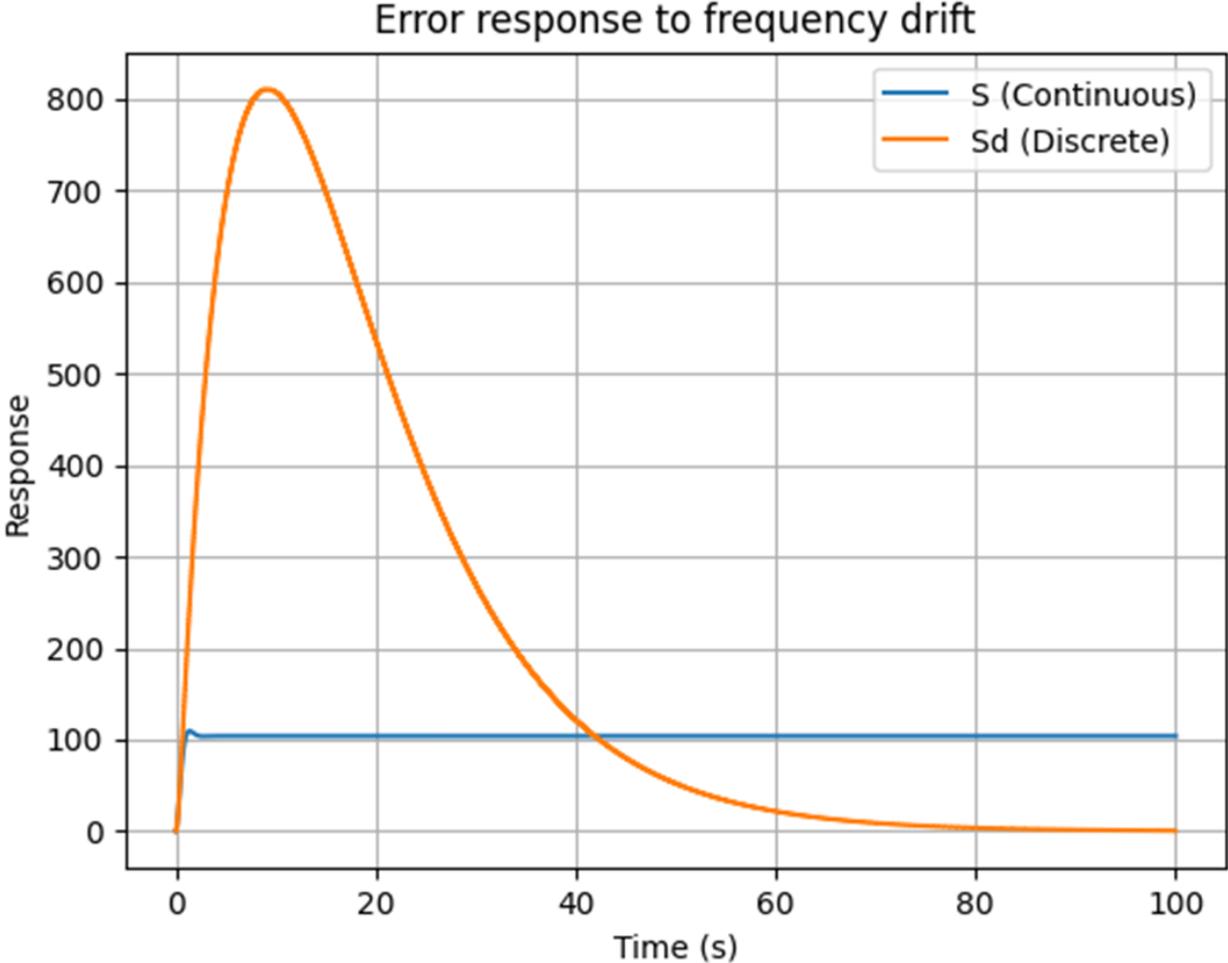
# 3<sup>rd</sup> Order – 16 Hz – Response to 1 ppm/s Frequency Drift



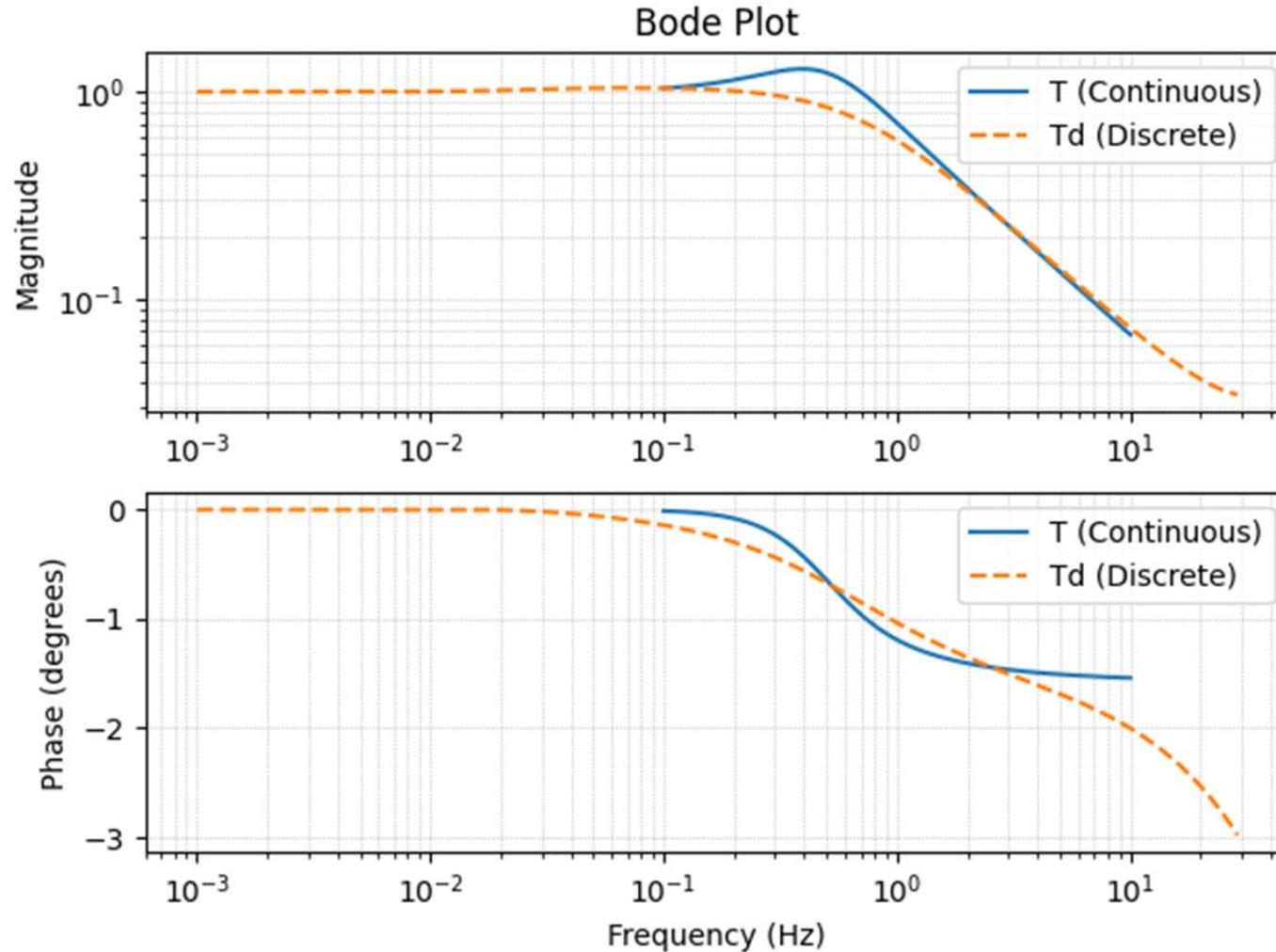
# 3<sup>rd</sup> Order – 32 Hz – Bode Plots



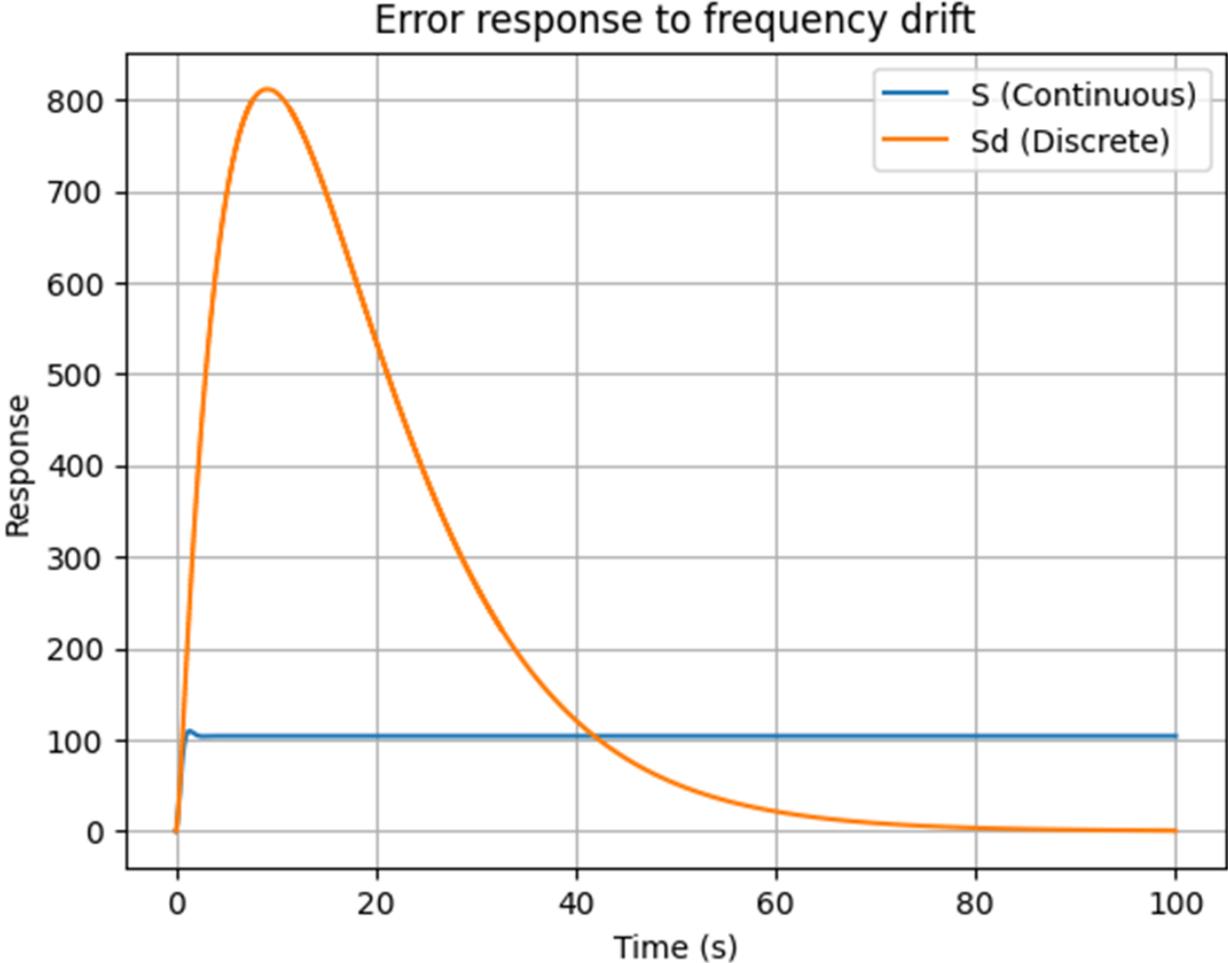
# 3<sup>rd</sup> Order – 32 Hz – Response to 1 ppm/s Frequency Drift



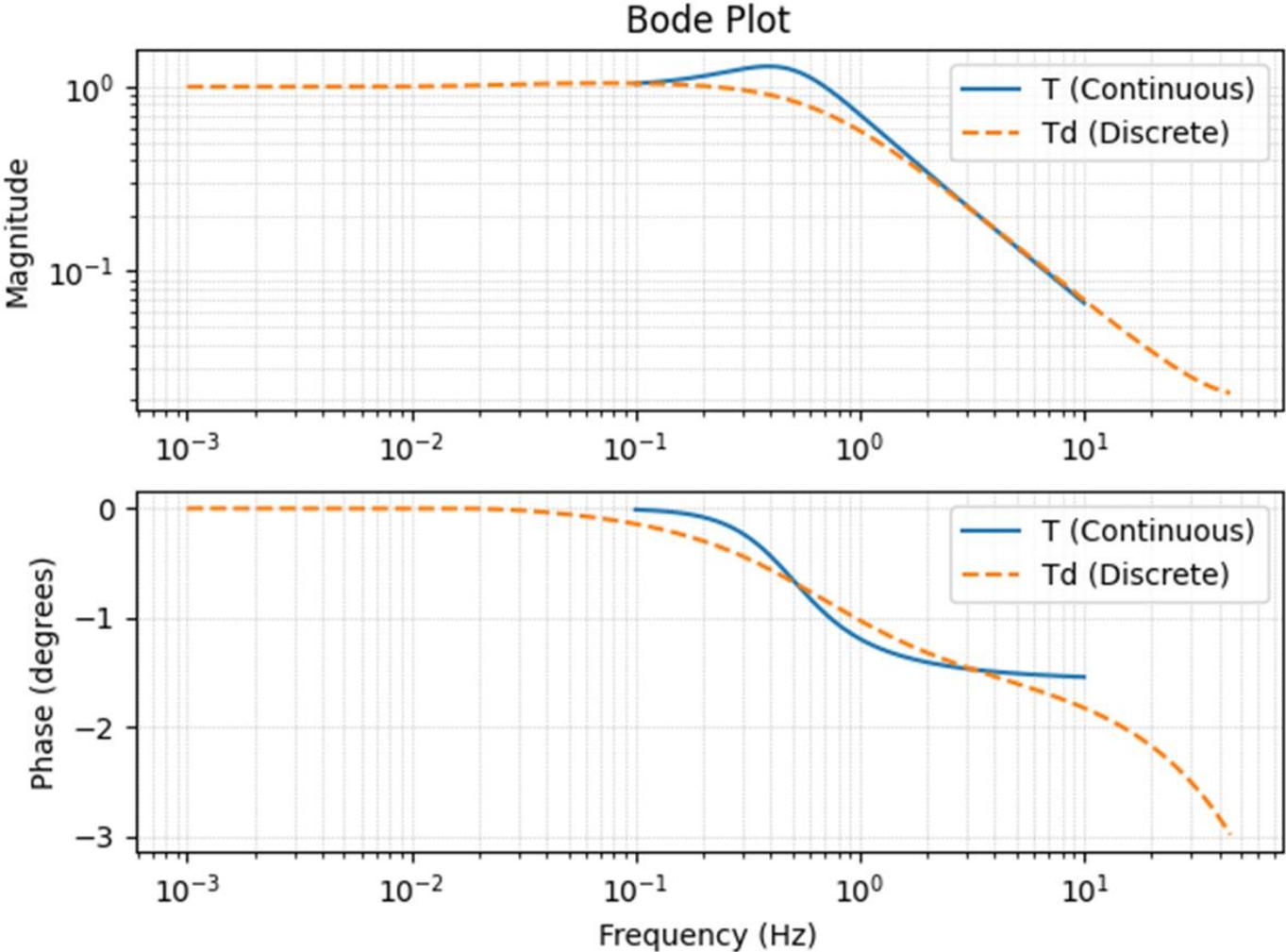
# 3<sup>rd</sup> Order – 64 Hz – Bode Plots



# 3<sup>rd</sup> Order – 64 Hz – Response to 1 ppm/s Frequency Drift



# 3<sup>rd</sup> Order – 100 Hz – Bode Plots



# 3<sup>rd</sup> Order – 100 Hz – Response to 1 ppm/s Frequency Drift

