`*.pid & *.spo`

The *.pid (PSV) and *.spo CSV (pipe- and comma-separated-value) formatted files are suggested file types or extensions only and may be inputted, loaded or imported with the include frame's attribute DATAFRAME to read PID retuning related data required for the routines SISOARX() and RETUNEPID() as well as SIMULATEPID() i.e., identification, estimation (estimization), optimization and simulation (simulization) of the process model and the PID settings respectively. The "pid" data-vector found inside *.pid must be consistent with the PID data-vector argument required by the RETUNEPID() and SIMULATEPID() datadata functions where each field may itself be a calc-scalar expression or formula. The *.spo stands for setpoint variable value ("sp"), process output variable value ("pv") and controller output / process input variable value ("op") related to single-loop proportional, integral (re-set) and derivative (pre-act) PID controllers where the sp,pv,op tuple of data-sets, -lists or -vectors, with lower case names only, may be arranged in any order, permutation or sequence and other data-sets, -lists or -vectors may also be present in the *.spo file such as an integer or real number for the time-stamp, time-step, time-interval, time-index, time-period (ts, ti, tp), etc.

An example *.pid file in IMPL's pipe-separated-value (PSV) format is provided below with symbols and nomenclature description for more clarity on its configuration.

```
pid, Kp     | KpL    | KpLs   | KpU    | KpUs   |
   , Ti     | TiL    | TiLs   | TiU    | TiUs   |
   , Td     | TdL    | TdLs   | TdU    | TdUs   |

   , Ts     |
   , PIDeq  |
   , opL    | opU    |
   , pvL    | pvU    |

   , Unt    | Unl    |

   , mL     | mU     |
   , nL     | nU     |
   , kL     | kU     |
```

The `Kp`, `KpL`, `KpLs`, `KpU` and `KpUs` are the proportional gain, proportional gain lower bound plus step-size and proportional gain upper bound plus step-size where `Kp` acts as the center-point or centroid for the brute-force or grid search inside the RETUNREPID() datadata function i.e., `KpL <= Kp <= KpU`. Similarly, `Ti`, etc. and `Td`, etc. are the integral-time and derivative-time with lower, upper and steps /

strides. The Ts represents the sampling- or scan-interval time and must be in the same time unit-of-measure as Ti and Td. The PIDeq currently supports the three well-known PID equation types A = 0, B = 1 and C = 2 as described further below. The opL, opU, pvL and pvU are the lower and upper bounds for the process input or controller output (op) and process output (pv). The Unt and Unl are the upper norm type (1, 2 or 3 for infinity) and the upper norm limit required by RETUNEPID() to constrain the controller output, manipulated variable or process input minus its previous or immediate past value (i.e., input-move-error) variance for **self-regulating processes** (stable) whereby the setpoint (reference or target) minus its process output (i.e., output-error) is minimized. And if negative (-ve), then the output-error variance is constrained instead for **integrating processes** (marginally stable) whereby the input-move-error variance is minimized known as "level-flow smoothing" (LFS)) or "averaging level control" (ALC) and typically employed for buffer or surge vessel "loose" versus "tight" level or volume control loops – see also the NORMOE() and NORMIME() datacalc functions. It should also be mentioned that even for self-regulating processes, the upper norm limit (Unl) may also be negative (-ve) whereby the input-move-error variance norm is minimized and is constrained or subject to the output-error norm upper limit implying "loose" versus "tight" control performance for the self-regulating process under PID feedback control.

The mL, mU, nL, nU, kL and kU are the lower and upper degree bounds for the brute-force or grid search performed with the SISOARX() datadata function in order to determine the best numerator (m), denominator (n) and dead-time / time-delay (k) ARX (Auto-Regressive eXogenous) linear parametric dynamic (transfer function) model for the process from closed- and/or open-loop operating data found in the *.spo file. *Interestingly, even though the single-input and single-output linear dynamic process model is most likely structurally over-parameterized (i.e., estimate, fit or learn more parameters than necessary), the phenomenon of self-regularization helps to inherently prevent over-fitting. That is, redundant parameters tend to converge to zero (0.0) as more data becomes available and the remaining parameters tend to converge to their true low-order system parameter or coefficient values without the requirement for explicit parameter regularization via 1- (absolute, Manhattan) or 2- (squared, Euclidean) norm penalty-errors or -elastic (artificial) variables cf. Du, Liu, Weitze and Ozay, "Sample complexity analysis and self-regularization in identification of over-parameterized ARX models", IEEE 61st Conference on Decision and Control (CDC), 2022.*

In terms of the theoretical number of PID retuning brute-force or grid-search simulations required to be performed by RETUNEPID(), we can easily calculate this assuming all steps or strides (KpLs, etc.) are positive (+ve) and non-zero as:

```
(nKpl + nKpU) * (nTil + nTiU) * (nTdl + nTdU)
```

where

```
nKpL = INT((Kp – KpL) / KpLs) + 1,
nKpU = INT((KpU – Kp) / KpUs) + 1,
nTiL = INT((Ti – TiL) / TiLs) + 1,
nTiU = INT((TiU – Ti) / TiUs) + 1,
nTdL = INT((Td – TdL) / TdLs) + 1, and
nTdU = INT((TdU – Td) / TdUs) + 1.
```

In order to elucidate some guidance and clarity on how to configure the brute-force or grid search for the PID retuning, suggested values for KpL, KpU, TiL, TiU and TdL, TdU are provided below based on the default or existing PID parameter settings multiplied by a user, modeler or analyst supplied factor or multiplier (e.g., 2.0):

```
KpL =      0.0,
KpU = 2.0 * Kp,
TiL =       Ts,
TiU = 2.0 * Ti,
TdL =      0.0, and
TdU = 2.0 * Td.
```

Suggested step or stride values for KpLs, KpUs, TiLs, TiUs and TdLs, TdUs may be also chosen using a user, modeler or analyst supplied factor or multiplier (e.g., 0.1 = 10^-1) which is directly related to the inverse of the suggested number of steps, strides, increments or discrete elements within its lower and and/or upper bound sub-ranges or -domains (e.g., 10):

```
KpLs = 0.1 * (Kp – KpL),
```

```
KpUs = 0.1 * (KpU – Kp),
TiLs = 0.1 * (Ti – TiL),
TiUs = 0.1 * (TiU – Ti),
TdLs = 0.1 * (Td – TdL), and
TdUs = 0.1 * (TdU – Td).
```

Therefore, the theoretical number of PID retuning brute-force or grid-search simulations performed by RETUNEPID() is proportional to the given factors or mulitpliers discussed above.

And for completeness, we provide the three (3) most popular PID equations A, B and C as implemented in IMPL below in **velocity-form** where `t` and `t-1` indicate the current / present and the immediate past / previous values respectively:

PIDeq = 0 (A)

```
op,t = op,t-1 + Kp*       ((sp,t – pv,t) – (sp,t-1 - pv,t-1)) +
                Kp*Ts/Ti* (sp,t – pv,t) +
                Kp*Td/Ts*((sp,t – pv,t) –
                           2*(sp,t-1 - pv,t-1) + (sp,t-2 - pv,t-2))
```

PIDeq = 1 (B)

```
op,t = op,t-1 + Kp*       ((sp,t – pv,t) – (sp,t-1 - pv,t-1)) +
                Kp*Ts/Ti* (sp,t – pv,t) +
                Kp*Td/Ts* (pv,t – 2*pv,t-1 + pv,t-2)
```

PIDeq = 2 (C)

```
op,t = op,t-1 + Kp*       (pv,t – pv,t-1) +
                Kp*Ts/Ti*(sp,t – pv,t)   +
                Kp*Td/Ts*(pv,t – 2*pv,t-1 + pv,t-2)
```

Finally, it should be mentioned that the sign of the controller gain `Kp` may be either positive (+ve) or negative (-ve) depending on the sign of the process gain where it is important to understand that the product of the process gain times the controller gain must its be positive (-ve). *Or stated more simply, the sign of the process gain and the sign of the controller gain must be the same i.e., if the process gain is*

negative (-ve), then the controller gain (K_p*) must also be negative (-ve).* This is also related to whether the actuator or final control element is direct- (DEV) or reverse-acting (REV) whereby a direct-acting controller increases its output (OP) as the process variable (PV) increases (i.e., they move in the same direction) and a reverse-acting controller increases its output (OP) as the process variable (PV) decreases (i.e., they move in opposite directions).