PIDABS, PIDINC and INCOUT Functions for Closed Loop Control

PIDABS S34
PIDINC S35
INCOUT S36

DESCRIPTION

Both PIDABS and PIDINC are functions intended for the control of closed loop systems. They are both three-term functions, incorporating proportional, integral, and derivative action.

The difference between PIDABS and PIDINC lies in the type of output signal which the GEM 80 controller is required to provide to the plant.

PIDABS is the more commonly required function, and PIDINC is only required in a limited number of applications. See the Application Notes below for examples.

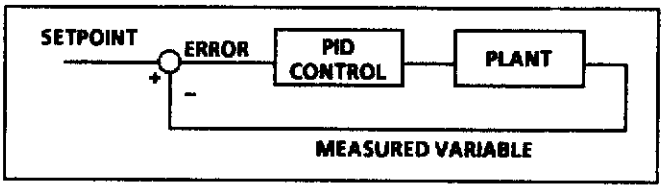
The mnemonics PIDABS and PIDINC stand for Proportional, Integral and Derivative control with ABSolute or with INCremental outputs respectively.

INCOUT is a special function sometimes needing to be used with PIDINC to ensure correct reset action. It provides an output signal whose integral with time is equal to its input, but limited to a preset maximum value. It also incorporates a deadband to prevent signals being output below a preset threshold. See the Application Notes below for examples of its use. The mnemonic INCOUT stands for INCremental OUTput.

FEATURES

For both PIDABS and PIDINC:

- Independently programmable proportional, integral and derivative gains
- Independent positive and negative output limits
- Control input to suicide the output (subject to rate limits with PIDABS)



Additional features with PIDABS only:

- Separate rate limits for output moving away from and towards zero
- Control input to hold output constant
- Output pre-settable, e.g. for bumpless changeover from manual to auto control

For INCOUT:

- Programmable limits and deadzone
- Flag indication when output within dead-zone or limited at positive or negative limits
- Provision for initialisation

SPECIFICATION

For PIDABS:

Control language :
```
        PIDABS
X----SPEC----VALUE----Q
        S34
```

Quantity of table locations : 15

Equivalent equation when not in limit : $Q = K_1X + K_2\int X\,dt + K_3\frac{dX}{dt}$

Equation when suicide on : $Q = 0$

Equation when output held : Q = constant

For PIDINC:

Control language :

```
          PIDINC
X----SPEC----VALUE----Q
          S35
```

Quantity of table locations : 9

Equivalent equation when not in limit : $Q = \frac{d}{dt}(K_1X + K_2\int X\,dt + K_3\frac{dX}{dt})$

Equation when suicide on : $Q = 0$

For INCOUT:

Control language :

```
          INCOUT
X----SPEC----VALUE----Q
          S36
```

Quantity of table locations : 5

Equivalent : $\int Q\,dt = \int X\,dt$

subject to:

$|Q| \leq QMAX$

$Q = 0$ if $|Q|$ would otherwise be $< QMIN$

Note: 1. Both PIDABS and PIDINC must be executed at regular time intervals to achieve constant derivative and integral gains.

2. The outputs of PIDABS and PIDINC are calculated by the following difference equations:

PIDABS:

$$Q_n = \frac{P(X_n - X_{n-1}) + IX_n + D(X_n - 2X_{n-1} + X_{n-2})}{100} + Q_{n-1}$$

PIDINC:-

$$Q_n = \frac{P(X_n - X_{n-1}) + IX_n + D(X_n - 2X_{n-1} + X_{n-2})}{100}$$

where: suffix n means value on current scan
suffix n–1 means value on previous scan

$P = 100K_1$
$I = 100K_2T$
$D = 100K_3/T$

and T = program repetition interval

APPLICATION NOTES

CONTENTS

1. PRINCIPLES OF CLOSED LOOP CONTROL

ACCURACY

Steady state accuracy of a closed loop control is the sum of the accuracies of the following:

- Transducer giving the measured variable
- Measured variable signal conditioning
- Analogue to digital conversion of the measured variable
- Any scale factor calculations on the measured variable
- Subtraction of the measured variable from the setpoint

Note that inaccuracies in the control signal to the plant do not enter into the system accuracy calculation. What can be of importance to the system operation is the resolution of the control signal, i.e. what the smallest change is that can be made in it. This topic is dealt with later.

RESPONSE

Depends on:

- Characteristics of the plant
- Response of the measuring transducer
- Placement of the measuring transducer
- Analogue and digital filtering

Note, for instance, that if a temperature transducer is placed a long distance downstream of where the heat is fed in on a temperature control, then any change of heat input may not be sensed for several seconds by the transducer, and a fast control loop becomes impossible.

2. PIDABS OR PIDINC - WHICH ONE?

PIDABS must be used where, under steady state conditions, the control signal to the plant is higher when the setpoint is higher, and lower when the setpoint is lower (or vice versa); in other words, use PIDABS where the control signal is roughly proportional to the setpoint.

PIDINC must be used where the control signal to the plant is always of the same value under steady state conditions, regardless of the value of the setpoint. We will refer to this as the "home" value of the control signal. With this type of system, the control signal will move away from its home value transiently when the setpoint is changed, but settle back at its home value when steady conditions have been achieved.

Example of a Flow Control Loop

Consider the closed loop control of fluid flow, but with valves having different types of actuators.

Example 1:

Valve actuator input in the range 4 - 20mA.

4mA - Valve fully closed
20mA - Valve fully open

For a larger flow, we need a larger control signal, i.e. control signal must increase as the flow setpoint is increased. PIDABS is therefore the required function.

Example 2:

Valve actuator input in the range 4 - 20mA.

4mA - Valve closing at maximum rate
20mA - Valve opening at maximum rate
12mA - Valve stationary (stays put at whatever opening it has reached)

Whatever the flow required, under steady state conditions the valve must remain stationary; valve movements are only needed when the flow setpoint is changed, but once the change is complete, the valve must stay put at its new opening. For this to happen, the control signal must always return to its "home" value of 12mA, regardless of the flow setpoint. PIDINC is therefore the required function.

For most applications of PIDINC, the "home" output value will probably be zero, e.g. where the plant signal swings between −10V and +10V. The example above is given to make clear that this is not always so, and not a requirement.

3. SIGNAL LEVELS

The control signal to the plant, and the measured variable, will normally both be analogue signals. Within the GEM 80 controller, however, both signals will be treated as numbers.

When programming a GEM 80 controller, there is a natural temptation for the user to convert signals into numbers which are directly meaningful for monitoring, e.g. to convert signals from a thermocouple into numbers equal to the temperature in °C or °F. However, this may reduce the resolution obtainable with the control loop, and may also cause a dither type of fluctuation.

If, for instance, a temperature in the range 0 to 100°C is converted into a number in the range 0 to 100, then the system will only be able to set the temperature in discrete steps of 1°C. If it is converted to a number in the range 0 to 10,000, then the temperature can be controlled to 0.01% steps.

Similarly, if the control signal to a valve actuator with a positioner comes from a three-term control function where the range of output number is only 0 to 100, then the valve can only be set to discrete positions 1% apart.

It may well be that, to maintain the measured variable at a value equal to the setpoint, the valve should be at a position halfway between two steps, in which case the controller will cause the valve to dither up and down 1% so as to obtain the correct mean setting.

The numbers used for the setpoint, measured variable and control signal should therefore be sufficient to give the required resolution. Where it is necessary to provide "real values" for monitoring purposes, simple multiples (e.g. 10 or 100 times) should be used for preference; if this is not possible, then separate rungs in the program for monitoring only should be used.

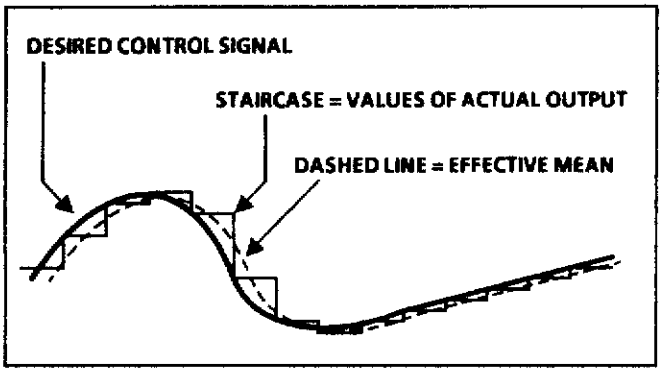
Both PIDABS and PIDINC have facilities for suiciding their outputs, where application of a logic signal from a program rung causes their outputs to be driven down to 0 (zero). For this condition, however, a non-zero value may be required for the control signal to the plant. In such cases, the PIDABS or PIDINC function should be followed in its program rung by LINCON special function S11 so that zero out of the three-term function provides the appropriate number as the control signal out of LINCON.

4. PROGRAM REPETITION INTERVAL

To achieve the fastest response, the user could arrange his GEM 80 program so that the three-term function updated the value of the control signal to the plant on every cycle through the program.

However, it is usually undesirable to update the control signal more frequently than necessary, because the three-term control functions involve several calculations inside them, and consequently take longer to perform than most other functions. Refer to the execution times given in the Technical Manual for the particular model of GEM 80 controller.

It is therefore recommended that the sections of program containing PIDABS or PIDINC are included within BLOCKs, initiated at regular intervals, where the time interval is such as to make only a marginal difference to the closed loop response. The effect of the control signal only being updated at the repetition interval can be visualised as below:



where, approximately, the control signal to the plant is delayed by half the repetition interval.
In addition, there is a delay between measuring the input to the PID Function (the Error signal) and setting the output (the Control Signal). This additional delay equals the cycle time of the GEM 80 Controller set by the P-table location P1. The total delay between the Error and the Control Signal is therfore:

$$\text{Effective controller time delay} = \frac{T_b}{2} + T_s$$

where : T_b = BLOCK repetition interval
T_s = Controller cycletime set by P1

If the PID function is NOT operated in a BLOCK, but allowed to operate on every program cycle, this reduces to :

$$\text{Effective controller time delay} = \frac{3T_s}{2}$$

TIME DEPENDENT BLOCK EXAMPLE

Using the DELAY function, a three-term control can be put in a BLOCK as in the example below:

```
|                                                    |
+--]/[---DELAY---VALUE-----------( )--+
| G50.0   G51     20                G50.0 |
|                                                    |
+--] [--------------------------OBEY--+
| G50.0                             BLOCK |
|                                     *              |
********** START OF BLOCK *********                  |
|                                                    |
| SETPT    M.V.   PIDABS          CTRLSIG|
+-<AND>---<SUB>---SPEC---VALUE---<OUT>-+
|  G52     G53     S34   W100     D32   |
|                                                    |
+************ END OF BLOCK ************+
```

In the opposite example, the coil G50.0 is operated after a delay of 2 seconds (20 units of 0.1s) for one program cycle only, as the contact of G50.0 in the same rung resets the DELAY, and the operation
then repeats. The normally open contact of G50.0 in the next rung causes the BLOCK containing PIDABS to be operated after 2 seconds and, as the operation of the first rung repeats, this means that the BLOCK will be initiated at 2 second intervals and skipped on other program cycles (while the DELAY is timing out).

As the repetition interval for the BLOCK depends just on the VALUE set for the DELAY, it does not matter if the user subsequently alters the configuration data in the GEM 80 controller P-table for the program scan time; the BLOCK will still be initiated at 2 second intervals.

PROGRAMS CONTAINING SEVERAL 3-TERM CONTROLS

If the user has a system containing several three term controls, it is desirable, in order not to slow down the program cycle time too much, for different control loop BLOCKs to be initiated on different cycles through the program. This is programmed easily using the SEQR (Sequencer) function. If all the control loops need to operate at the same repetition interval, this is straightforward, but, if loops need to operate at different intervals, the user is recommended to draw himself a timing program.

Example:

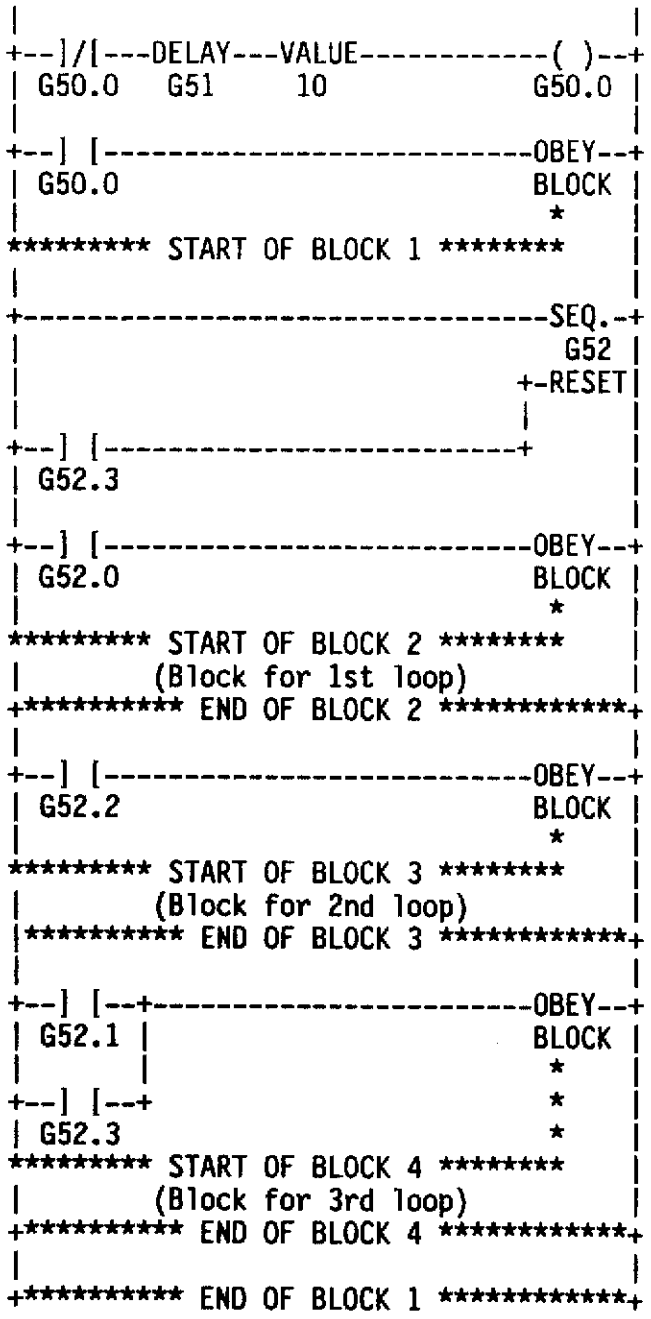
The user has 3 loops, 2 of which can operate at a 4 second repetition interval, but the remaining loop needs to operate at 2 second intervals.

One method of doing this would be to arrange the operation of the loops as follows:

Seconds	0	1	2	3	4	5	6	7	8	9	10
	11										
1st loop	X				X				X		
2nd loop			X				X				X
3rd loop		X		X		X		X		X	
					X						

$$\text{Effective controller time delay} = \frac{T_b}{2} + T_s$$

which the sequencer must be reset so that the cycle of operations can repeat. The ladder diagram would be:



The above example is not the only possible method of programming the given set of loops, but probably about the neatest. Note that it is essential that each loop is operated at a regular interval and that the following alternative timing diagram would be wrong because it would invalidate this rule for the 3rd loop:

```
Seconds      0 1 2 3 4 5 6 7 8 9 10 11
1st loop     X     X     X
2nd loop       X     X     X
3rd loop         X X   X X   X X
             0 1 2 3 4 5 6 7 8 9 10 11
```

where, instead of being operated at regular 2 second intervals as in the first diagram, here the 3rd loop would be operated at intervals of alternately 1 second and 3 seconds, giving unpredictable performance.

Note: 1. Other methods of initiating loops at regular intervals are possible using the timing markers E0.0 or E0.1. However, problems can occur with trying to count timing markers depending on what the program cycle time is. The method above, using DELAY, works generally, regardless of the program cycle time.

2. The same techniques for time-dependent blocks can, of course, be used for other types of functions besides three-term controls. The user may find it useful to draw timing diagrams including other functions which can be operated at regular intervals, to ensure that no program scan is more heavily loaded (and therefore takes longer) than other program scans.

5. DETERMINATION OF CONTROLLER SETTINGS

When PIDABS or PIDINC are incorporated into a program, values must be entered to set the proportional gain, the integral time, and the derivative time. Also, the length of program repetition interval needs to be established.

These values can be determined by one of four methods:

1. Measurement

   Run the plant open loop and measure its response to a small step change of input. From this response, find settings from standard formulae.

2. Calculation

   From design information about the plant, calculate its transfer functions, and then use standard analytical techniques (e.g. Bode or Nyquist diagrams) to find the required controller settings.

3. Knowledge

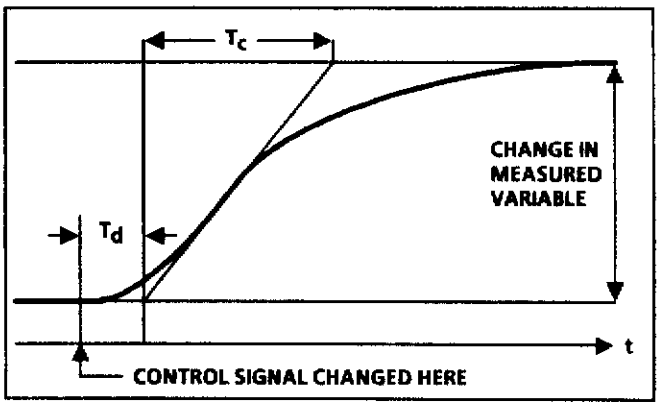
   Where the user is aware of a similar plant, he may be able to use similar settings, at least for initial trials.

4. Cut-and-Try

   The user starts from a low gain, long integral time condition, and winds the gains up until the response starts to get oscillatory.

5.1 MEASUREMENT METHOD

The response of the measured variable to a small step change of control signal can be recorded or plotted as shown below:



Note: When conducting such a test, the control signal to the plant should be sufficient for the plant to be operating somewhere in its normal working range prior to the step change. For example, with a valve actuator, the valve should already be open to some extent when the step change of control signal is applied, and not starting from the closed position. See, for instance, the worked example in Section 7 below. This is to avoid non-linearities which are often found in plant characteristics at the extremes of the control range.

From this plot, three values should be measured as follows:

1. The gain of the plant:

$$A = \frac{\text{change in measured variable}}{\text{change in control signal}}$$

2. The effective deadtime. Obtain this by drawing a tangent as shown, and find where it cuts the initial value of the measured variable.

   T_d = Effective deadtime

3. The effective time constant. Obtain this by extending the tangent to where it cuts the final value of the measured variable, and find the difference in time between the two ends of the tangent as shown.

   T_c = Effective time constant

These three values will be made use of in determining the three-term controller settings and the repetition interval.

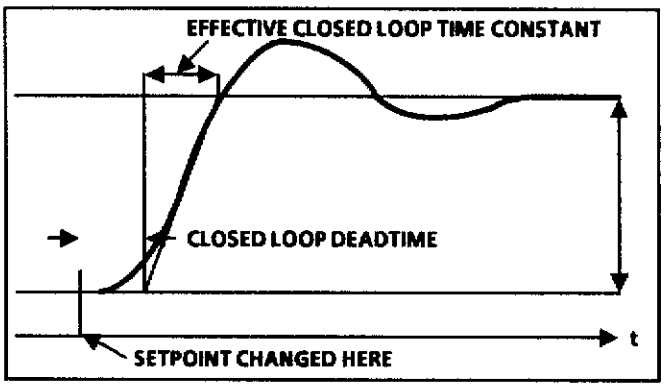
If the plant is one where PIDINC must be used, then it will not be possible to obtain a response like that shown, because the plant output will continue to change until the control signal is dropped back to its home value. In this case to conduct a test the control signal should be changed for a short time (e.g. 1 second) and then brought back to its home value. Denoting the length of time the control signal change is applied by T_t, the formula for the gain of the plant is:

$$A = \frac{\text{change in measured variable}}{\text{change in control signal}} \times \frac{1}{T_t}$$

If the measured deadtime T_d looks to be close to the test time T_t, then the test should be re-done with a larger change of control signal for a shorter test time.

CLOSED LOOP RESPONSE

If this plant is now closed loop controlled, using a three-term control function, and with a very short update time which we can ignore, the fastest response which can normally be obtained for a small step change in setpoint is as below:



The following approximations apply:

- The deadtime of the closed loop response is about the same as that of the open loop response
- The effective time constant of the closed loop response is about the same as the deadtime
- If the three-term control function is tuned up any more, little improvement in response speed can be obtained; what happens is that the response becomes more oscillatory or unstable

PROPORTIONAL, INTEGRAL AND DERIVATIVE GAINS

Note: PIDABS and PIDINC require that there is always an integral term. For systems where only a proportional term is required, the user should use LINCON (special function S11), maybe in conjunction with RAMP (special function S33).

Noting the values of plant gain A, plant dead-time T_d and plant time constant T_c measured from the open loop response plot, and the program cycle time T_s, settings can be determined from the following rules of thumb:

Repetition interval:

$$T \leq T_d/2$$

To allow for the delay effect of the repetition interval on updating, we add on the effective controller deadtime, i.e.

$$\text{Effective plant deadtime } T_d' = T_d + (T/2) + T_s$$

For 2-term (PI) control:

$$\text{Proportional gain} = \frac{1}{A} \times \frac{T_c}{T_d'} \times 0.9$$

$$\text{Integral time } T_I = \frac{T_d'}{0.3}$$

For 3-term (PID) control:

$$\text{Proportional gain} = \frac{1}{A} \times \frac{T_c}{T_d'} \times 1.2$$

$$\text{Integral time } T_I = \frac{T_d'}{0.5}$$

Derivative time $T_D = \frac{Td'}{0.5} \times 0.5$

(Note: proportional band % = $\frac{100}{\text{Proportional gain}}$)

5.2 CALCULATION METHOD

If the user has sufficient design data about his plant to be able to calculate its transfer function, then he may analyse the closed loop system by whatever methods he is most familiar with (Nyquist, Bode etc.).

Compared with a pure analogue control system, the only extra factors to be considered are the time delay effects of the BLOCK repetition interval and program cycle time. As shown previously, a BLOCK repetition interval of T effectively introduces a deadtime (transport lag) of $(T/2 + T_s)$.

If the closed loop system is to have a bandwidth of ω radians/second, then the choice of $T = 1/(2\omega)$ would introduce an extra 14° phase shift at the bandwidth frequency. Shortening the repetition interval would reduce the phase shift pro rata. (This is assuming that T_s is negligible compared with T, otherwise ω would have to be reduced or T_s shortened).

From his analysis, the user should determine the values of proportional gain, integral time (and derivative time, if derivative action is needed) and program repetition interval which will give the response he desires.

5.3 KNOWLEDGE METHOD

Where a similar plant is GEM 80 controlled, all values can be copied directly.

Where a similar plant is analogue controlled, or where an analogue control loop is to be replaced by a GEM 80 system, the same proportional gain, integral and derivative times can be used, but it is still necessary to establish a suitable program repetition interval.

Rules of thumb:

For 2-term (PI) control:

$T \leq 0.15\, T_I$

where T_I is the integral time

For 3-term (PID) control:

$T \leq 0.25\, T_I$

where T_I is the integral time

Note: These formulae assume that the integral action has been wound up to give more or less the quickest reset action to a setpoint change which gives a satisfactory response. If this is not so, then the repetition interval may have to be made somewhat shorter than implied by the above rules of thumb.

5.4 CUT-AND-TRY METHOD

This is the least methodical of the four methods. Where the user is inexperienced in setting up closed loop controls, it is recommended that he should employ the measurement method in Section 5.1.

The experienced user who has a "feel" for the plant being controlled may, however, attempt to short cut open-loop testing of the plant and go direct to entering first-try table values. How these table values relate to the proportional gain, integral and derivative times, and the program repetition interval, are given in Section 6 below. Further details are given in the Worked Examples, Section 7.

5.5 RESPONSE NOT FAST ENOUGH

On a few occasions, it may be found that the response obtainable for a closed loop system obtained using PIDABS or PIDINC is not as fast as the application requires.

It must be remembered that the response obtainable depends on the system deadtime. This is, in most cases, determined by the plant (see the Measurement Method in Section 5.1 above) but in a few cases could be primarily caused by the GEM 80 program cycle time. In either case, the only way in which the response can be speeded up is by reducing the deadtime. The improvement in response obtainable will be proportional to the reduction in deadtime.

Where the deadtime of the plant is predominant, reduction in deadtime can only be effected by modifying the plant itself. This may involve physically moving measuring transducers so that they respond to changes earlier. In other cases it may be necessary to redesign the system such that a single loop is replaced by two loops in cascade. For example, with a temperature control, it might be necessary to have a fast inner loop with a temperature measurement close to where the heat input is adjusted, and a slower outer loop with a temperature measurement further downstream where the user really wants to control the temperature. The fast inner loop then does most of the work, while the outer loop acts as a fine trim, having to cater for heat losses between the two transducers only.

Where the program cycle time of the GEM 80 Controller represents the predominant portion of the deadtime, then to begin with the PIDABS or PIDINC function should be operated on every program cycle and not put in a BLOCK; secondly, the program cycle time itself should be reduced as far as practicable, by putting other portions of program which do not have to be executed on every program cycle in BLOCKs, and by the other techniques for reducing program cycle time with which the user should be familiar.

Where extremely fast response loop are required, it may be necessary to use dedicated hardware, e.g. slave micro controllers, handling only one or two control loops each.

6. TABLE VALUES

The values to be programmed into the GEM 80 controller program for the integral and derivative times are both related to the program repetition interval. If, at any time, the program repetition interval for the block of program containing PIDABS or PIDINC is altered, then the constants defining these two terms in the control function must be altered as well.

For both PIDABS and PIDINC, having worked out the required values of proportional gain, integral time and derivitive time, proceed as follows:

Proportional gain:

Table value P = proportional gain in units of 0.01 (e.g. value required is 5000 for a proportional gain of 50)

Integral time:

$$\text{Table value I} = \text{Table value P} \times \frac{T}{T_I}$$

where T is the repetition interval, and T_I is the integral time.

Derivative time:

$$\text{Table value D} = \text{Table value P} \times \frac{T_D}{T}$$

where T is the repetition interval, and T_D is the derivative time

Note that both the table values I and D depend on the repetition interval T. If T is made very short, then I can work out to be very small and D very large, maybe larger than the maximum value of 32767 which can be entered into the table.

If the repetition interval has been chosen in accordance with the rules of thumb previously given, it is to be expected that I will be of the order of $0.1P < I < 0.25P$, and D will be of the order of $P < D < 2.5P$ for three-term control; for 2-term PI control, D = 0 and I will be slightly larger than for 3-term control.

Note: There are two minor restrictions on P, I and D table values as follows:

1. $|P + I + D|$ must not exceed 65535

2. $|P + 2D|$ must not exceed 65535

These restrictions are hardly ever met in practice, but in exceptional circumstances it might be necessary to reduce the value of D to slightly less than the desired value. If the above inequalities are not complied with, both PIDABS and PIDINC will give zero rung outputs.

7. WORKED EXAMPLES

EXAMPLE - MEASUREMENT METHOD

Temperature control. Output from GEM 80 feeds a valve in a steam heating pipe. Measured variable from RTD transducer.

Both values are scaled using LINCON special function S11 to be in the range 0 to 32000.

A test is conducted of running the valve with a control signal value of 12000, which is then increased as a step change to 13500. The corresponding change in the measured variable is from 12375 to 12975. The response shows an effective deadtime of 12s and an effective time constant of 140s.

Therefore:

$$A = \frac{12975 - 12375}{13500 - 12000} = \frac{600}{1500} = 0.400$$

$T_d = 12s$

$T_c = 140s$

Rule of thumb for repetition interval:

$T \leq T_d/2 = 6s$

Let's use $T = 5s$

Then $T_d' = 12 + (5/2) = 14.5s$

(The program cycle time has been ingnored as being small compared with this).

Using three-term control, the PID terms can now be worked out.

Rules of thumb:

$$\text{Proportional gain} = \frac{1}{A} \times \frac{T_c}{T_d'} \times 1.2$$

$$= \frac{1}{0.400} \times \frac{140}{14.5} \times 1.2$$

$$= 28.97$$

$$\text{Integral time } T_I = \frac{T_d'}{0.5}$$

$$= \frac{14.5}{0.5} = 29s$$

$$\text{Derivative time } T_D = T_d' \times 0.5$$

$$= 14.5 \times 0.5 = 7.25s$$

Lastly, calculate the required table values:

P = Proportional gain in units of 0.01

= 2897

$$I = P \times \frac{T}{T_I} = 2897 \times \frac{5}{29} = 499.48$$

= 499 to nearest integer

$$D = P \times \frac{T_D}{T} = 2897 \times \frac{7.25}{5} = 4200.65$$

= 4200 to nearest integer

Cross check:

I = 0.17P and D = 1.45P, which look O.K.

EXAMPLE - CALCULATION METHOD

No example is given here, as analytical methods for the design of closed loop systems and worked examples can be found in many textbooks.

However, an elementary example appears in the exercises in Section 11 below.

EXAMPLE - KNOWLEDGE METHOD

Again, no example is given here, as the method simply involves copying known data into the GEM 80 controller, following the formulae in Section 6.

EXAMPLE - CUT-AND-TRY METHOD

Make a first stab at the repetition interval, bearing in mind that this should be at least half the plant deadtime. Suppose we try 1 sec.

To start with, we want no derivative action, so the table value D is left set to zero. Because of the way PIDABS and PIDINC perform their calculations, it is not possible to remove the integral action entirely, and the best that can be done is to set the integral action to be very slow. Because the table value I is inversely proportional to the integral time T_I, we will get the longest integral time by setting I to be 1 (unity).

Now set in some low, arbitrary value for P so as to give an initial low proportional gain. If the control signal and measured variable have been scaled using LINCON so that their span utilises most of the permissible number range, it is unlikely that P need be less than 100 as a first try. With a 1 second program repetition interval and I set to 1, this would also give an initial integral time T_I of 100 secs. (see the formula in Section 6).

Now increase P by stages. This will speed up the plant response to setpoint changes. Eventually, an optimum setting may be found beyond which the plant response becomes too oscillatory to be acceptable. (Note that increasing P makes the integral time T_I longer at the same time). Suppose that we find the optimum setting for P is 3520.

We now want to increase I to shorten the integral time and make the reset action quicker.

However, making the integral time shorter will make the system less stable, so P should first be reduced to 90% of its previous setting. In our example, we set P to 3520 x 0.9 = 3168. Now we can start increasing I by stages until a new optimum response is found. One should not be too "enthusiastic" about increasing I; it may turn out to be a fairly small figure. For each new trial setting of I, don't increase it to more than twice its previous setting. Suppose that we find the optimum value is 98.

If we only require two-term control, that may be all we need to do. However, it is worth checking whether the repetition interval, which we set initially to 1 second, could be made longer. This is something we should always check before turning the control into a three-term control, as otherwise the size of D could turn out larger than the maximum number (32767) which can be entered into the data table.

From the formula for integral time in Section 6, with a repetition interval I of 1 second, then the integral time will be:

$$T_I = \frac{\text{Table value for P x T}}{\text{Table value for I}}$$

$$= \frac{3168 \times 1}{98} = 32.3 \text{ seconds}$$

For 2-term control with a well-tuned controller, it should be possible to use a repetition interval approaching $0.15T_I$ (see Section 5.3), or 0.15 x 32.3 = 4.85 seconds.

Allowing for the fact that we may not have tuned the two-term control precisely to optimum, let's increase the repetition interval to 4 seconds. This will require modification to program rungs, maybe by altering VALUEs which define DELAY times (see Section 4). However, if we increase the repetition interval by four times (from 1s to 4s), then the table value for I must also be increased four times to keep the integral time the same (see the formula above), so we set I to 4 x 98 = 392.

Now we can start increasing D to bring in derivative action. Unlike reset action, this tends to make the control more stable, and it should be possible to increase P and I after it has been added. If D is increased too far, however, the system will again become oscillatory, but at a faster frequency than when P or I were set too high. It should be possible to wind D up to a figure around ten times I, i.e. around 3920, without any problems. Suppose we find a value around 8400.

It should now be possible to increase P by about 30% and I by about 100%, giving us settings of about P = 4120 and I = 784, with D = 8400.

Note that, if the repetition interval has not been changed from 1s to 4s prior to adding the derivative action, D would have worked out four times bigger, which would have been 33600 and too big to be entered in data table.

Note also that, if the repetition interval is roughly right (i.e. not unnecessarily quick), then the checks given at the end of Section 6 will apply. In this example, I = 0.19P and D = 2.04P, which fall within the bands given.

8 LIMITS

8.1 LIMITS WITH PIDABS

A closed loop system only remains in closed loop mode operationally as long as the plant responds to what the control signal demands of it. However, it is possible that, with a well tuned three-term controller, the controller may demand more from the plant than it is capable of.

For instance, suppose that signal levels have been chosen so that the output from a PIDABS function is in the range of 0 to 10000 (giving numbers which are easily interpreted when monitoring to give % output). Suppose that the PIDABS output is then scaled by LINCON and fed to a signal conditioning transmitter such that a 4 to 20mA output is provided to the plant. It is possible that, on setpoint changes, the output of PIDABS could go up to 12000 or 15000, say. However, it is unlikely that the signal conditioning transmitter could provide 30mA and, even if it could, the increased signal might have no effect on, say, a valve actuator - once the valve is fully open, no further increase in signal makes any difference.

In such cases, if no limits are used, large changes in setpoint can cause undesirable overshoots in the measured variable. This is because of the length of time it takes for the output of PIDABS to come back down into the working range of values; until this happens, the control signal to the plant will remain at its maximum value or above, and the measured variable will be driven too high.

Similar effects can be caused, but to a lesser extent, if rate limits are not used with PIDABS.

Limits should therefore be set according to the following rules:

- Limits must be set on the three-term control function itself, and not on any subsequent function operating on the control signal (such as LINCON).

- Limit values should be chosen such that no following hardware or software saturates, i.e. any hardware may be driven flat out but no harder; any software program function must not be driven to maximum or limit number values.

With PIDABS, rate limits should be set such that the rates of change of control signal are no faster than the plant can follow. The Table values L1 and L2 (See Program Rung and Value data, Table 1) define the maximum change of number which can occur on any one scan. The data values to be set up are therefore dependant on the repetition interval; if this is ever changed during programming, then L1 and L2 will need altering just as with the integral and derivative times.

8.2 LIMITS WITH PIDINC

PIDINC, as was explained in Section 2, is intended for use with actuators where the control signal always returns to a "home" value under steady state conditions. Such actuators, in fact, integrate the difference between the control signal and its "home" value. PIDINC is normally used where its "home" output value is zero, and any required offset obtained by using LINCON special function S11 following it.

At first sight, it might be thought that the same rules for limit settings should be applied as for PIDABS, i.e. that limits should be set so as to prevent the control signal trying to drive the actuator faster than it will respond. This may be all right in practice with an actuator which has very little friction and stiction.

However, with many actuators, if the setpoint to the control loop is changed or if there is a disturbance affecting the measured variable, the system may come to a steady state condition where PIDINC gives an output too small to overcome friction, with the actuator having stopped short. Proper reset action is then lost. Also, the actuator could overheat if left standing with a steady current through it.

To overcome any such problem, the user can add the special function INCOUT in his ladder diagram following PIDINC, e.g.

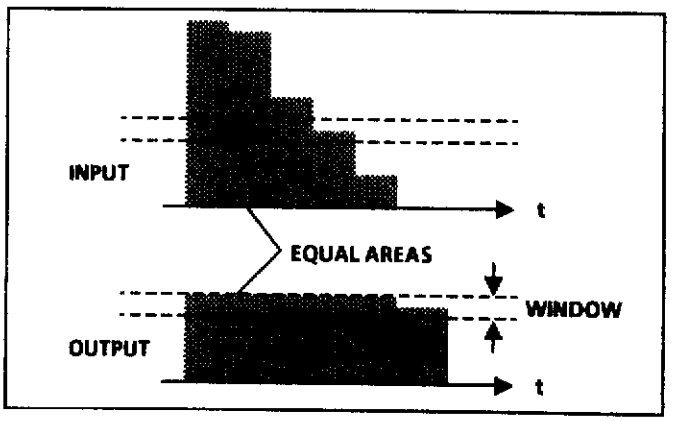
```
    PIDINC              INCOUT
----SPEC----VALUE----SPEC----VALUE----
    S35      W100       S36      W110
```

When INCOUT is used as above, then the limits on PIDINC should be set to the maximum numbers permissible (±32767), and the limits set to match the actuator on INCOUT only.

8.3 CHARACTERISTICS OF INCOUT

The function INCOUT will only give outputs which lie between upper and lower limits of windows, symmetrical about zero. The upper limits operate as normal limits; the lower limits provide a deadzone. The deadzone should be set such that signals smaller than what the actuator can respond to are blocked.

The chief property of INCOUT, however, is that it gives an output whose integral is equal to that of its input from PIDINC (subject to the output being within the limit window). This is best explained by the diagram below:



Note that, if the input area had been fractionally less, the last output value on the last program scan shown might have fallen below the lower limit window, in which case the actual output on this scan would have been zero; the value not outputted would be held over until PIDINC provided some further input to INCOUT.

The user can check from the difference equation for PIDINC given in the Specification at the beginning of this data sheet that, on a step change of setpoint, PIDINC will normally give a large positive output on the first program repetition interval, followed by a large negative output on the second interval. INCOUT enables these two "spikes" to be made use of in the output to the plant without being chopped off by limits on PIDINC or being disregarded by the actuator.

9 MISCELLANEOUS APPLICATION NOTES

9.1 SETPOINT SHAPING

One difference between GEM 80 three-term control functions PIDABS and PIDINC and many types of hardware three-term controllers is that PIDABS and PIDINC operate on the error signal. This has to be formed separately by subtraction of the measured variable from the setpoint elsewhere in the program, even though this may only be earlier within the same rung.

As a result, any derivative action programmed into PIDABS or PIDINC operates for changes both to the setpoint and to the measured variable, whereas many hardware three-term controllers have the derivative action operating on the measured variable only. A faster action is therefore obtained with PIDABS or PIDINC in response to setpoint changes than for such hardware three-term controllers.

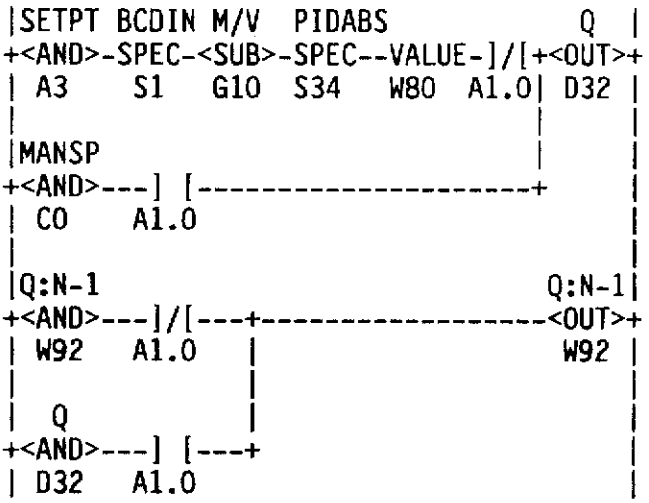
If the user finds that the response is too fast for the application, then he should slug the setpoint signal, rather than altering the three-term settings. Since the setpoint signal comes outside the closed loop, any such slugging has no effect on stability, and can be made whatever the user wants, without delaying the response of the closed loop to disturbances.

Two alternative methods are to use a time constant, or to use a ramp generator. A time constant can be provided using the ANALAG special function S37. If the time constant is made equal to the Derivative Time set for PIDABS or PIDINC, a response will be obtained similar to that for hardware controllers with derivative action on the measured variable only. A similar response can be obtained, with initial transients smoothed out, but with quicker settling, by using RAMP special function S33 instead of ANALAG.

9.2 BUMPLESS CHANGEOVER

With closed loop controls using PIDINC, there is no problem in changing over from automatic closed loop control to manual control, as PIDINC will under steady state conditions be giving zero output, with any offset required if the "home" value of the control signal is not zero being provided by a following LINCON special function.

With closed loop control using PIDABS, however, under steady state conditions the output from PIDABS must be of the appropriate size and will not generally be zero. When there is a changeover from manual to automatic control, it is therefore necessary to pre-load the PIDABS output to be equal to the manual control output. This is most simply achieved by arranging for the table location normally used by PIDABS for storing its previous output to track the manual control output. An example is given below:



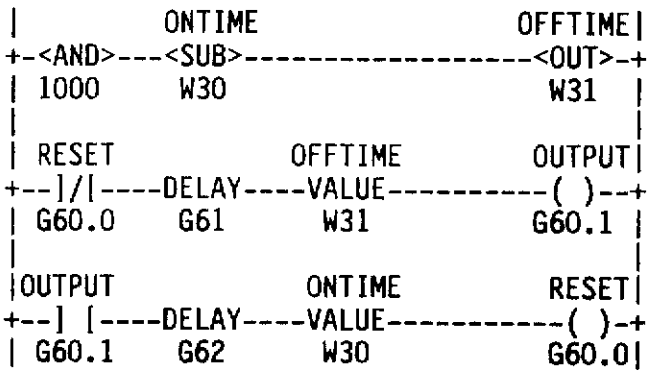
This would ensure bumpless control from manual (with A1.0 closed) to auto (with A1.0 open), but not the other way round. If bumpless control were required from auto to manual, then the manual setpoint MANSP could be fed via a RAMP special function, and the output of the RAMP preset by a similar rung to the second rung in the example above.

Note that the table location W92 is the 12th one after the one below VALUE (W80) after PIDABS. See Table 1 for the usage of the various table locations.

9.3 CONTACTOR OUTPUTS

In some applications, the output of a closed loop control needs to switch contactors on and off with a variable on-to-off time ratio, e.g. for controlling electric heating elements.

For such applications, PIDABS is the correct three-term function to use, but with its output feeding a variable mark-space generator. The example below shows a possible realisation. In this section of program, the output from PIDABS is taken into W30, and has been arranged to lie in the range 0 to 1000.



The first rung is designed so that there is a constant cycle time of 100 seconds. The output from PIDABS into W30 sets the on-time in units of 0.1 second; subtracting this from 1000 gives the required off-time. The second rung, with G60.0 initially de-energised, provides an output after a delay equal to the off-time. The output then initiates the on-time delay in the third rung, and the output from G60.1 remains on until this delay has timed out, after which G60.0 resets the off-time delay in the second rung, G60.1 de-energising resets the on-time delay in the third rung, and the sequence recycles.

Note that a section of program such as the above should operate on every program scan, and should not be put in a BLOCK operating at a slower repetition interval, even though the PIDABS function determining the on-time may well be put in such a BLOCK.

10. EXERCISES

A user having a GEM 80 Controller and an 8902 simulation unit may like to try out PIDABS and PIDINC using the following simple exercises, in which the plant is simulated within the GEM 80 program itself. It is left to the user, if he so wishes, to alter the characteristics of the simulated plant.

In both exercises, the set point is taken from the thumbwheel switches (A3), and the response of the measured variable is displayed on the LCD numeric display (B2).

PIDABS EXERCISE

Set the controller program repetition interval to 100ms by making P1 = 100.

Simulate the plant with the following rung:

```
|CTRLSIG MULT  ANALAG       ANALG
+-<AND>--SPEC--SPEC---VALUE-SPEC--VALUE-
|  W10   T10    S37    W20   S37   W30
|       +<  >                (continued
|       |                       below)
+-<AND>-+
|   2
```

```
    ANALAG                          M/V |
-----SPEC------VALUE-------------<OUT>+
     S37        W40                 W11 |
```

Set up data for the ANALAG functions as follows:

W21 = 5 W31 = 100 W41 = 100
W24 = -1 W34 = -1 W44 = -1

Explanation: What the rung does is to represent the plant as a gain of 2 (via the MULT), and three time constants in cascade, the first of 20 seconds, and the other two both of 1 second.

To be able to display the measured variable on the LCD display, insert the following rung:

```
|  M/V    BCDOUT                  DISPLAY|
+-<AND>---SPEC-----------------<OUT>-+
|  W11     S2                      B2  |
```

This rung simply takes the output of the previous rung and converts it into suitable form to drive the display using BCDOUT.

Before trying to put a closed loop control round the "plant", you may like to get the feel of the plant response by running it open loop, using the thumbwheel switches to drive directly into W10 using the following rung:

```
| INPUT   BCDIN                   CTRLSIG|
+-<AND>---SPEC.-----------------<OUT>-+
|  A3     S1                       W10  |
```

For a large change of input signal (e.g. if you alter the thumbwheel from 0000 to 1000), it will be found that the number displayed changes slowly for the first second or two, then speeds up to a nearly constant rate, after which its rate tails off, and it seems to take ages to move the last figure. Since the plant has a gain of 2, the final figure should be twice the figure set on the thumbwheel (as long as the former is less than 9999, the largest number the LCD can display).

If you want to try the measurement method, as given in Section 5.1 above, then you will need to make a plot of the response of the plant to find the deadtime and the effective time constant. One possible way of doing this would be to route the measured variable output from W11 to an analogue output (e.g. D32) and using a pen recorder. Another way is to write a section of program in the GEM 80 which copies the contents of W11 every 5 seconds, say, into a sequence of table locations which can be read off as a data-list on the Programmer screen, which can then be plotted out manually as a graph.

If you want to use the cut-and-try method of Section 5.4, then you can put in the following rungs:

```
|                                       |
+--]/[---DELAY---VALUE----------( )--+
| G0.0    G1       5              G0.0  |
|                                       |
+--] [-------------------------OBEY--+
| G0.0                          BLOCK |
|                                  *    |
********** START OF BLOCK *********
|                                       |
| INPUT BCDIN  M/V  PIDABS      CTRLSIG|
+-<AND>-SPEC--<SUB>-SPEC---VALUE-<OUT>-+
|  A3    S1    W11   S34   W100   W10  |
|                                       |
+************ END OF BLOCK ************+
```

Before anything else, delete the previous rung used for driving the plant control signal W10 direct from the input thumbwheel A3.

The rungs above, firstly, include a rung so that the following block is executed not on every scan through the program but only at roughly 0.5 sec. intervals. The working rung with the three-term control PIDABS in it forms the setpoint from the thumbwheel input A3, converts it to decimal form with BCDIN, and then the measured variable is subtracted from the setpoint to produce the error signal input to PIDABS.

We now need to set up the data table values for PIDABS as given in Table 1 at the end of this data sheet:

W101 = P; try 100 initially
W102 = I; try 50 initially
W103 = D; try 1000 initially
W104 = 32000
W105 = -32000
W106 = 32000
W107 = 32000
W108 = -1
W109 = 0

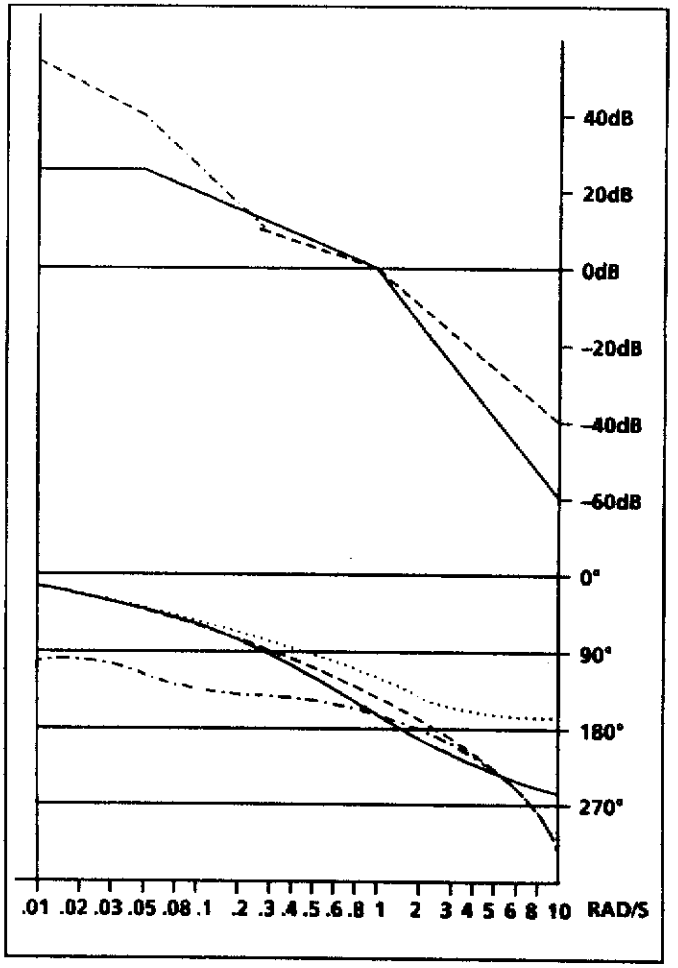
These table values, apart from the first three, set the limits and rate limits to large values so that they will not interfere with the system (W104 and W107) while W108 and W109 set suicide off and hold off.

Try the effect of changing the setpoint on A3 and see the plant response on B2. Try altering the values of P, I and D in W101 to W103; you should not have much difficulty in making the system unstable if you make the values too high.

If you want to use the analysis method of Section 5.2, then you can use whichever analysis method you are most familiar with. In the example below, we use the Bode diagram. Firstly we note that the transfer function of the plant is:

$$\frac{2}{(1 + 20p)(1 + p)^2}$$

for which the amplitude and phase shift plots are as below:



If we make the Controller gain 10, then the phase shift at crossover is approximately 177°, which is too close to 180° for comfort. Putting in a derivative term with derivative time of 1 second will reduce this to approximately 132° (see dotted line). We have got to choose the repetition rate for the 3-term control to operate at; if we make this twice/second, this will mean operating the PIDABS at 1/2 second intervals, giving an effective time delay of 1/4 second, which, together with the 100ms cycle time set by P1, introduces an additional 19° phase shift at crossover (see dashed line) but makes no difference to the amplitude plot. The phase shift at crossover is then 151°. Putting in an integral time of 4 seconds (chain line) will introduce another 14° phase shift at crossover, making a total of 165°. The system is then

stable, although somewhat wobbly with only a 15° phase margin. For these values, the corresponding table values will be:

W101 = P in units of 0.01 = 1000 for a gain of 10.

W102 = I = P x T/T_I = 1000 x 0.5/4 = 125

W103 = D = P x T_D/T = 1000 x 1/0.5 = 2000

and the 1/2 second repetition interval can be set as in the cut-and-try example with a block executed every 1/2 second.

PIDINC EXERCISE

Simulate the plant by the same rung as before, but insert, as the final element before the OUT to W11, and ADD with location W11. Replace the PIDABS by PIDINC. Change the W table values to suit PIDINC as follows:

W104 = -32000
W105 = 32000
W106 = 0 } will get changed subsequently by
W107 = 0 } PIDINC operating
W108 = -1

Then try running similarly to the PIDABS exercise.

Explanation: The additional ADD in the "plant" simulation run means that, under steady state conditions, the control signal in W10 must be zero. For this condition, the rung adds nothing to W11 and puts this value back into W11, so that W11 stays put at whatever value it has reached with no control signal input. This is the sort of plant characteristic which PIDINC is intended to control, where, after a change of setpoint, the control signal always comes back to its "home" value (in this case, zero value).

Table 1 Program rung and value data (PIDABS)

Sym	Location	Use	Range
F	Zm	Fault code and flags set by PIDABS	See Table 2
P	Zm+1	Proportional gain. Set by user.	Each ±32767; see "Table Values" in application notes for how to calculate them.
I	Zm+2	Integral gain. Set by user.	
D	Zm+3	Derivative gain. Set by user.	
L+	Zm+4	Limit on output excursion, positive. Set by user.	0 to +32767
L-	Zm+5	Limit on output excursion, negative. Set by user.	0 to -32767
L1	Zm+6	Limit on output rate of change for Q moving away from zero. Set by user.	Each 0 to +32767; see "Limits" in application notes for how to set them.
L2	Zm+7	Limit on output rate of change for Q moving towards zero. Set by user.	

Table 1 (continued)

Sym	Location	Use	Range
S	Zm + 8	Suicide on/off switch. Set by user, either preset to suicide off, or else controlled by separate rung.	0: Suicide on; Q reduces to zero at L2 rate. Any non-zero value: Suicide off (normal operation).
H	Zm + 9	Hold on/off switch. Set by user, either preset to hold off; or else controlled by separate rung. NOTE: Suicide on (S = zero) overrides Hold on (H = non-zero)	0: Hold off (normal operation) Any non-zero value: Hold on; Q remains at current value.
X_{n-1}	Zm + 10	Previous input, stored by PIDABS. Used in next two scans.	±32767
X_{n-2}	Zm + 11	Previous input, stored by PIDABS. Used in next scan.	±32767

Table 1 (continued)

Sym	Location	Use	Range
Q_{n-1}	Zm + 12	Previous output, stored by PIDABS. Used in next scan.	±32767, but subject to L+ and L– limits
R1	Zm + 13	Integration remainder. Stored by PIDABS. Used in next scan.	
R2	Zm + 14	Rate limit remainder. Stored by PIDABS. Used in next scan.	
X	–	Input to PIDABS from rung.	±32767
Q	–	Output from PIDABS	±32767, but subject to L+ and L– limits

Table 2 Fault conditions and flags (PIDABS)

Condition	Result	Code in Zm	
		Value	Bits Set
PROGRAMMING ERRORS			
No VALUE after SPEC.	Q = 0	-	-
VALUE given number instead of address	Q = 0	-	-

Table 2 (continued)

Condition	Result	Code in Zm	
		Value	Bits Set
VALUE address Zm in write-protected memory	Q=0	-	-
VALUE address Zm in table not usable by special functions	Q=0	-	-
Fifteenth address Zm+14 beyond end of data tables memory area	Q=0	2	Zm.1=1
DATA ERRORS			
Invalid positive limit: L+>0	Q=0	5	Zm.0=1 Zm.2=1
Invalid negative limit: L–>0 or = –32768	Q=0	9	Zm.0=1 Zm.3=1
Invalid rate limit: L1<0	Q=0	17	Zm.0=1 Zm.4=1
Invalid rate limit: L2<0	Q=0	33	Zm.0=1 Zm.5=1
Invalid gain values: |P+I+D| > 65535 or |P+2D| > 65535	Q=0	65	Zm.0=1 Zm.6=1
STATUS			
Output limited at positive limit	Q=L+	–28672	Zm.12=1 Zm.15=1
Output limited at negative limit	Q=L–	+20480	Zm.12=1 Zm.14=1

Table 3 Program rung and value data (PIDINC)

Sym	Location	Use	Range
F	Zm	Fault code and flags set by PIDINC	See Table 4
P	Zm+1	Proportional gain. Set by user.	Each ±32767; see "Table Values" in application notes for how to calculate them.
I	Zm+2	Integral gain. Set by user.	
D	Zm+3	Derivative gain. Set by user.	
L–	Zm+4	Limit on output excursion, negative. Set by user.	0 to –32767
L+	Zm+5	Limit on output excursion, positive. Set by user.	0 to +32767
X_{n-1}	Zm+6	Previous input, stored by PIDINC. Used in next two scans.	±32767
X_{n-2}	Zm+7	Previous input, stored by PIDINC. Used in next scan.	±32767

Table 3 (continued)

Sym	Location	Use	Range
S	Zm + 8	Suicide on/off switch. Set by user, either preset to suicide off, or else controlled by separate rung.	0: Suicide on; Q reduces to zero. Any non-zero value: Suicide off (normal operation).
X	-	Input to PIDINC from rung.	±32767
Q	-	Output from PIDINC	±32767, but subject to L+ and L- limits

Table 4 Fault conditions and flags (PIDINC)

Condition	Result	Code in Zm	
		Value	Bits Set
PROGRAMMING ERRORS			
No VALUE after SPEC.	Q=0	-	-
VALUE given number instead of address	Q=0	-	-
VALUE address Zm in write-protected memory	Q=0	-	-
VALUE address Zm in table not usable by special functions	Q=0	-	-

Table 4 (continued)

Condition	Result	Code in Zm	
		Value	Bits Set
Ninth address Zm + 8 beyond end of data tables memory area	Q=0	-	-
DATA ERRORS			
Invalid positive limit: L+ < 0	Q=0	5	Zm.0=1 Zm.2=1
Invalid negative limit: L- > 0 or = -32768	Q=0	5	Zm.0=1 Zm.2=1
Invalid gain values: |P+I+D| > 65535 or |P+2D| > 65535	Q=0	65	Zm.0=1 Zm.6=1
STATUS			
Output limited at positive limit	Q=L+	-28672	Zm.12=1 Zm.15=1
Output limited at negative limit	Q=L-	+20480	Zm.12=1 Zm.14=1
Suicide on (S=0)	Q=0	9	Zm.0=1 Zm.3=1

Table 5 Program rung and value data (INCOUT)

Sym	Location	Use	Range
F	Zm	Fault code and flags set by INCOUT	See Table 6
QMIN	Zm+1	Lower limit of output window (deadzone). Set by user.	0 to +32767 with QMIN ≤ QMAX
QMAX	Zm+2	Upper limit of output window (excursion limit). Set by user.	
D	Zm+3	Σ Q − Σ X Stored by INCOUT. Used in next scan.	Automatically limited to ±32767
M	Zm+4	Mode control switch. Set by user, either preset to non-zero value for normal operation, or else controlled by separate rung.	0: Sets Q=0, and SUM=X Any non-zero value: Normal operation
X	–	Input to INCOUT from PIDINC via rung.	±32767

Table 5 (continued)

Sym	Location	Use	Range
Q	–	Output from INCOUT	0, or in the range QMIN to QMAX, or in the range –QMIN to –QMAX

Table 6 Fault conditions and flags (INCOUT)

Condition	Result	Code in Zm	
		Value	Bits Set
PROGRAMMING ERRORS			
No VALUE after SPEC.	Q=0	-	-
VALUE given number instead of address	Q=0	-	-
VALUE address Zm in write-protected memory	Q=0	-	-
VALUE address Zm in table not usable by special functions	Q=0	-	-
Fifth address Zm+4 beyond end of data tables memory area	Q=0	-	-
DATA ERRORS			
Invalid upper limit on output window, QMAX<0	Q=0 SUM=0	5	Zm.0=1 Zm.2=1

Table 6 (continued)

Condition	Result	Code in Zm	
		Value	Bits Set
Invalid lower limit on output window, QMIN<0 or QMIN > QMAX	Q=0 SUM=0	5	Zm.0=1 Zm.2=1
STATUS			
Mode control input, M=0	Q=0 SUM=X	9	Zm.0=1 Zm.3=1
Output zero because of deadzone, |SUM+X| <QMIN	Q=0	8192	Zm.13=1
Output limited at -QMAX (SUM+X) <-QMAX	Q= -QMAX	16384	Zm.14=1
Output limited at +QMAX (SUM+X) > +QMAX	Q= +QMAX	-32768	Zm.15=1
(SUM+X)<-32767	Q= -QMAX SUM=-32767	16448	Zm.6=1 Zm.14=1
(SUM+X)> +32767	Q= +QMAX SUM= +32767	-32640	Zm.7=1 Zm.15=1

CEGELEC
INDUSTRIAL CONTROLS

Kidsgrove, Stoke-on-Trent, ST7 1TW, England
Tel: (0782) 783511 Fax: (0782) 776329 Telex: 36293/4 CICKID G
Registered in England No. 2259095