NEW L2-DYNAMIC-RANGE-SCALING CONSTRAINTS FOR LOW PARAMETRIC SENSITIVITY REALIZATIONS

Thibault Hilaire

Vienna University of Technology, Austria
Institute of Communications and Radio-Frequency Engineering
thibault.hilaire@nt.tuwien.ac.at

ABSTRACT

This paper presents a new dynamic-range scaling for the implementation of filters/controllers in state-space form. Specific fixed-point considerations allow us to relax the classical L2-scaling constraints while still preserving the implementation from overflows. It gives more degrees of freedom for the optimal L2-parametric sensitivity problem. The underlying constrained problem is converted into an unconstrained problem for which a solution can be provided. This leads to realizations which are still scaled but less sensitive.

1. INTRODUCTION

The majority of signal processing (or control) systems is implemented in digital general purpose processors, DSPs\(^1\), FPGAs\(^2\), etc. Since these devices cannot compute with infinite precision and approximate real-number parameters with a finite binary representation, the numerical implementation of controllers (filters) leads to deterioration in characteristics and performance. This has two separate origins, corresponding to the quantization of the embedded coefficients and the roundoff errors occurring during the computations. They can be formalized as parametric errors and numerical ones. They can be formalized as parametric errors and numerical noise contributions, respectively. The focus of this paper are parametric errors, coefficients and the roundoff errors occurring during the computation.

These constraints have to be considered in the work has been extended with a more natural and reasonable measure to prevent overflows and thus it can be slightly relaxed constraint is solved. A numerical example illustrates that the proposed

2. L2-SENSITIVITY ANALYSIS

Let \((A, b, c, d)\) be a stable, controllable and observable linear discrete-time SISO\(^3\) state-space system, i.e.

\[
\begin{align*}
x(k+1) &= Ax(k) + bu(k) \\
y(k) &= cx(k) + du(k)
\end{align*}
\]

where \(A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^{n \times 1}, c \in \mathbb{R}^{1 \times n} \) and \(d \in \mathbb{R}^1. u(k)\) is the scalar input, \(y(k)\) is the scalar output and \(x(k) \in \mathbb{R}^{n \times 1}\) is the state vector. Its input-output relationship is given by the scalar transfer function \(h : \mathbb{C} \to \mathbb{C}\) defined by:

\[
h : z \mapsto c(zI_n - A)^{-1}b + d.
\]

The quantization of the coefficients introduces some uncertainty to \(A, b, c, d\) leading to \(A + \Delta A, b + \Delta b, c + \Delta c\) and \(d + \Delta d\) respectively. It is of interest to consider the sensitivity of the transfer function with respect to the coefficients, based on the following definitions.

Definition 1 (Transfer function sensitivity) Consider \(X \in \mathbb{R}^{m \times n}\) a matrix and \(f : \mathbb{R}^{m \times n} \to \mathbb{C}\) a scalar complex function, differentiable with respect to all the entries of \(X\).

The sensitivity of \(f\) with respect to \(X\) is defined by the matrix \(S_X \in \mathbb{R}^{m \times n}:

\[
\frac{\partial f}{\partial X} \triangleq S_X \quad \text{with} \quad (S_X)_{i,j} \triangleq \frac{\partial f}{\partial X_{i,j}}
\]

Definition 2 (L2-Norm) Let \(H : \mathbb{C} \to \mathbb{C}^{k \times l}\) be a function of the scalar complex variable \(z\). \(\|H\|_p\) is the L\(_p\)-norm of \(H\) defined by:

\[
\|H\|_p \triangleq \left( \int_0^{2\pi} \| H(e^{i\omega}) \|_p^p \, d\omega \right)^{\frac{1}{p}}
\]

where \(\| \cdot \|_p\) is the Frobenius norm.

Gevers and Li [3] have proposed the L2-sensitivity measure to evaluate the coefficient roundoff errors. It is defined by

\[
M_{L2} \triangleq \left\| \frac{\partial h}{\partial A} \right\|_2^2 + \left\| \frac{\partial h}{\partial b} \right\|_2^2 + \left\| \frac{\partial h}{\partial c} \right\|_2^2 + \left\| \frac{\partial h}{\partial d} \right\|_2^2
\]

and can be computed by \(\frac{\partial h}{\partial A}(z) = G^T(z)F(z), \frac{\partial h}{\partial b}(z) = G(z), \frac{\partial h}{\partial c}(z) = F(z)\) and \(\frac{\partial h}{\partial d}(z) = 1\), with

\[
F(z) \triangleq (zI_n - A)^{-1}b, \quad G(z) \triangleq c(zI_n - A)^{-1}.
\]

This measure is an extension of the more tractable but less natural \(L_1/L_2\) sensitivity measure proposed by V. Tavşanoğlu and L. Thiele [16] \(M_{L2} \left( \left\| \frac{\partial h}{\partial A} \right\|_1^2 \right.\) instead of \(M_{L2} \left( \left\| \frac{\partial h}{\partial A} \right\|_2^2 \right.\) in (5)).

\(^1\)Digital Signal Processors
\(^2\)Field Programmable Gate-Array
\(^3\)Single Input Single Output
Applying a coordinate transformation, defined by \( \bar{x}(k) = T^{-1}x(k) \) to the state-space system \((A, b, c, d)\), leads to a new equivalent realization \((T^{-1}AT, T^{-1}b, cT, d)\).

Since these two realizations are equivalent in infinite precision but are no more equivalent in finite precision (fixed point arithmetic, floating-point arithmetic, etc.), the \(L_2\)-sensitivity then depends on \(T\), and is denoted \(M_{L_2}(T)\). In this case, it is natural to define the following problem:

**Problem 1 (optimal \(L_2\)-sensitivity problem)** Considering a state-space realization \((A, b, c, d)\), the optimal \(L_2\)-sensitivity problem consists of finding the coordinate transformation \(T_{opt}\) that minimizes \(M_{L_2}\):

\[
T_{opt} = \arg \min_{T} M_{L_2}(T). \tag{7}
\]

[3] shows that the problem has one unique solution. Hence, for example, a gradient method can be used to solve it.

### 3. \(L_p\)-DYNAMIC-RANGE SCALING

The \(L_p\)-dynamic-range-scaling constraints have been introduced by Jackson in [11] and Hwang in [9]. It consists in scaling the state-variable vector such that overflows or underflows during its evaluation are prevented.

**Definition 3 (\(L_p\)-scaling)** A state-space realization \((A, b, c, d)\) is said to be \(L_p\)-scaled if the \(L_p\)-norms of the transfer functions from the input to each state are set to 1, i.e.:

\[
\left\|e_i^T(zI_n - A)^{-1}b\right\|_p = 1, \quad \forall 1 \leq i \leq n \tag{8}
\]

where \(e_i\) is the column vector of appropriate dimension and with all elements being 0 except from the \(i\)th element which is 1.

Let \(\max u\) denote the maximum value of the input \(u\):

\[
\max u = \max_{k \in \mathbb{N}} |u(k)|. \tag{9}
\]

The \(L_1\)-sensitivity guarantees that the dynamic of each state \(x_i\) is lower than \(\max u\), whereas the \(L_2\)-sensitivity guarantees that the variance of each state is unitary for a unit-variance centered white noise input. \(L_2\)-scaling doesn’t completely prevent overflow as does \(L_1\), but it is less conservative and more realistic, so it is widely used [15].

**Proposition 1** The \(L_2\)-norms of the transfer functions from the input to each state are given by

\[
\left\|e_i^T(zI_n - A)^{-1}b\right\|_2 = \sqrt{(W_c)_{i,j}} \tag{10}
\]

where \(W_c\) is the controllability Gramian of the state-space system \((A, b, c, d)\). This matrix is the solution of the Lyapunov equation:

\[
W_c = AW_cA^\top + bb^\top. \tag{11}
\]

**Proof:**
A classical result on \(L_2\)-norm gives the \(L_2\)-norm of a state-space system \(H := (K, L, M, N)\):

\[
\|H\|_2 = \text{tr}(NN^\top + MW_cM^\top) \tag{12}
\]

where \(\text{tr}(.)\) is the trace operator and \(W_c\) is the controllability Gramian of the system.

This is here applied to the system \((A, b, e_i^\top, o)\).

**Problem 2 (sensitivity problem with \(L_2\)-scaling constraints)** The optimal \(L_2\)-sensitivity problem with \(L_2\)-norm dynamic-range-scaling constraints can be formulated as the optimization problem 1, subject to the constraints

\[
(W_c)_{i,j} = 1, \quad \forall 1 \leq i \leq n. \tag{13}
\]

This constrained problem can be transformed into an unconstrained problem, and an optimization algorithm can be used to solve it [7].

### 4. FIXED-POINT IMPLEMENTATION

#### 4.1 Fixed-point representation

In this paper, the notation \((\beta, \gamma)\) is used for the fixed-point representation of a variable or coefficient (2’s complement scheme), according to Figure 1. \(\beta\) is the total wordlength of the representation in bits, whereas \(\gamma\) is the wordlength of the fractional part (it determines the position of the binary-point). They are fixed for each variable (input, states, output) and each coefficient, and implicit (unlike the floating-point representation). \(\beta\) and \(\gamma\) will be suffixed by the variable/coefficient they refer to.

![Fixed-point representation](image)

To represent a value \(x\) without overflow, a fixed-point representation \((\beta_x, \gamma_x)\) may satisfy:

\[
\beta_x - \gamma_x - 1 \geq \left\lfloor \log_2 |x| \right\rfloor + 1 \tag{14}
\]

where the \(\lfloor\cdot\rfloor\) operation rounds \(a\) to the nearest integer less or equal to \(a\) (for positive numbers \(\lfloor a \rfloor\) is the integer part).

An important fixed-point issue is to find a valid fixed-point representation, such that (14) is satisfied for all values which \(x\) can assume during the execution of the algorithm.

#### 4.2 State-overflow

**Definition 4 (State-overflow)** The overflow of the state variables \((x_i)_{1 \leq i \leq n}\) can be strictly avoided if all the values \(x_i(k)\) are within the range allowed by the fixed-point representations, i.e. \((1 \leq i \leq n)\)

\[
\forall k, \quad -2_\beta x_i - \gamma_x - 1 \leq x_i(k) < 2_\beta x_i - \gamma_x - 1. \tag{15}
\]

The overflows are avoided if the binary-point position of each state is carefully chosen, such that

\[
\gamma_x = \beta_x - 2 - \left\lfloor \log_2 \max_{k \in \mathbb{N}} |x_i(k)| \right\rfloor, \tag{16}
\]

where \(x_i\) is the maximum magnitude for the \(i\)th state:

\[
\max_{k \in \mathbb{N}} |x_i(k)| \leq \max_{k \in \mathbb{N}} \gamma_x. \tag{17}
\]

**Remark 1** At least, \(\gamma_x\) should satisfy \(\gamma_x \leq \beta_x - 2 - \left\lfloor \log_2 \max_{k \in \mathbb{N}} |x_i(k)| \right\rfloor\), but the greater \(\gamma_x\) is, the more accurate is the fixed-point format.
However only upper bounds can be computed. A first upper bound \( \| x_i \|_1 \) can be obtained by an \( L_1 \)-norm:
\[
\| x_i \|_1 \leq \left\| e_i^T (zI_n - A)^{-1} b \right\|_1 \leq \max_{u} \| x_i \|_1.
\]
and a second one can be estimated by an \( L_2 \)-norm [15]:
\[
\| x_i \|_2 \leq \left\| e_i^T (zI_n - A)^{-1} b \right\|_2 \leq \max_{u} \| x_i \|_2
\]
Here, the parameter \( \delta \) can be interpreted as a representation of the number of standard deviations of \( x_i \), if the input is unit-variance white centered noise (\( \delta \approx 1 \)). Since the \( L_2 \)-norm in (19) doesn’t give a strict bound (contrary to (18)), \( \delta \) can be seen as a safety parameter [15].
Finally, these upper bounds are used to define the binary-point positions:
\[
\gamma_{x_i} = \beta_{x_i} - 2 - \left\lfloor \log_2 \| x_i \|_{\text{up}} \right\rfloor.
\]

In general, the \( L_1 \) and \( L_2 \) estimations of \( \| x_i \|_{\text{up}} \) approximately leads to the same binary-point position, with 1 or 2 bits deviation. However, since the \( L_2 \)-norm is more tractable (with proposition 1) and the \( L_1 \)-norm too conservative (\( \| x_i \|_{\text{up}} \ll \| x_i \|_1 \)), in practice (19) is used, with \( \delta = 1 \). After implementation, a simulation-based estimation like in [11] or [12] can also be used to verify in situ the peak values and the binary point positions, according to the inputs.

### 4.3 Computational scheme

In order to implement a realization without overflows, two equivalent choices are possible:
- set the binary-point position for each state, according to (20), to make sure that the fixed-point representation of the states avoids state-overflows;
- or define a binary-point position for each state, and apply a scaling to them in order to adapt the peak values of each state to the chosen binary-point position.

Here, we here focus on the 2\textsuperscript{nd} choice, referring to dynamic-range-scaling constraints.

Let us consider in detail the fixed-point implementation of the system given in (1). It leads to \((n+1)\) scalar products to be evaluated, of the form:
\[
S = \sum_{i=1}^{N} p_i q_i
\]
where the \((p_i)\) are given coefficients and \((q_i)\) are bounded variables. To avoid bit-shift operations between each addition in the evaluation of eq. (21), the binary-point positions of each partial product of the sum should be equal.

Then, two computational schemes are possible: the Roundoff After Multiplication scheme, where shifts are added after each product to align the operands of the sum \((p_i q_i) \) is implemented as \((p'_i * q'_i) \geq d_i\) and the Roundoff Before Multiplication scheme, where the required shifts are reported into the coefficients \((p_i q_i) \) is implemented as \((p'_i \geq d_i * q'_i)\). See Figure 2.

![Figure 2: Scalar product with Roundoff After Multiplication scheme](image)

The main idea of the scaling is to scale each variable \((q_i)\) such that the shifts \((d_i = 0, \forall i)\) are prevented. In fixed-point representation, the scaling only implies that all the \((q_i)\) have a common format, and so have the \((p_i)\). See [2, 6] for more details on implementation schemes.

Applied to the state-space realization (1), this yields that all the states must have the same binary-point position as the input and the coefficients \(A, b, c\) and \(d\).

Besides, since they have the same fractional part \(\gamma\), their quantization’s errors \(\Delta A, \Delta b, \Delta c\) and \(\Delta d\) have the same magnitude \(2^{-\gamma - 1}\), and the \(L_2\)-sensitivity measure represents a meaningful bound on the transfer function error \(\Delta h\) (considering that \(h\) is shifted in \(h + \Delta h\) by the coefficients’ quantization):
\[
\| \Delta h \|_2 \leq \frac{\partial h}{\partial A} \times \| \Delta A \|_2 + \frac{\partial h}{\partial b} \times \| \Delta b \|_2
\]
\[
\leq 2^{-2(\gamma + 1)} L_{H,2}
\]

#### 4.4 New \(L_2\)-scaling constraints

Taken into consideration, the overflows will be avoided by setting the same binary-point position for the states and the input, and by applying an appropriate scaling on the states such that the constraints (16) are satisfied.

Compared to strict \(L_2\)-scaling where the states must satisfy \(x_i = x_{\text{up}}\), here, the constraints are relaxed (but still restrictive enough to guarantee the protection against overflow) and replaced by \(x_{\text{up}} = x_{\text{up}}\).

**Proposition 2 (relaxed-\(L_2\)-scaling constraints) Since the input and the states may have the same binary-point position, the \(L_2\)-scaling constraints (13) are now transformed into**
\[
\frac{2^{2\alpha_i}}{\delta} \leq (W_i)_{ii} < 4 \frac{2^{2\alpha_i}}{\delta}, \quad \forall 1 \leq i \leq n
\]
where
\[
\alpha_i \triangleq \beta_{x_i} - 2 - \left\lfloor \log_2 \| x_i \|_{\text{up}} \right\rfloor
\]

and \(F_2(x)\) is defined as the fractional value of \(\log_2(x)\):
\[
F_2(x) \triangleq \log_2(x) - \left\lfloor \log_2(x) \right\rfloor
\]

**Proof:**
The binary-point position of the input is set to \(\gamma_u = \beta_u - 2 - \left\lfloor \log_2 \| u \|_{\text{up}} \right\rfloor\). Hence, with (20), the constraints \(\gamma_u = \gamma_{x_i}\) lead to
\[
\beta_u - \left\lfloor \log_2 \| u \|_{\text{up}} \right\rfloor = \beta_{x_i} - \left\lfloor \log_2 \left( \delta \left\| e_i^T (zI_n - A)^{-1} b \right\|_2 \right) \right\rfloor
\]

and
\[
\log_2 \left( \delta \left\| (W_i)_{ii} \right\| + F_2 \left( \frac{\text{max}}{u} \right) \right) = \beta_{x_i} - \beta_u
\]

And finally
\[
2^{2\alpha_i} \leq \delta \left\| (W_i)_{ii} \right\| < 2^{\alpha_i + 1}
\]

**Remark 2** For microcontroller or DSP implementations (contrary to FPGA or some ASIC implementations), the wordlength of all variables is mostly of the time equal, i.e. \(\beta_i = \beta_{x_i}\) (1 \(\leq i \leq n\)), \(\delta\) is set to unity (as for classical \(L_2\)-scaling constraints).

In best case, \(\| u \|_{\text{up}}\) is equal to a power of 2 \(2^p, p \in \mathbb{Z}\), and the relaxed-\(L_2\)-scaling constraints (24) become:
\[
1 \leq (W_i)_{ii} \leq 4, \quad \forall 1 \leq i \leq n.
\]
and in worst case, $\max u$ is equal to the representable value immediately lower than a power of 2 ($2^p - 2\beta - 2^{-p}$), and the constraints become:

$$\frac{1}{4} \leq (W_c)_{ij} < 1, \quad \forall 1 \leq i \leq n$$

(30)

(in the first case, $p + 1$ bits are used for the integer part of the states, whereas $p$ bits only are used in the second case).

It is important to remark that these new constraints allow more freedom for the scaling and introduce a new degree of freedom for the search for optimal realizations. Even though not considered in this paper, moreover it could give more freedom for the minimization of the roundoff noise power.

5. OPTIMAL $L_2$-SENSITIVITY REALIZATION WITH RELAXED $L_2$-NORM DYNAMIC-RANGE-SCALING CONSTRAINTS

Then, these relaxed constraints can be applied to a new sensitivity problem:

**Problem 3 (relaxed sensitivity problem)** The optimal $L_2$-sensitivity problem with relaxed $L_2$-norm dynamic-range-scaling constraints can be expressed in the form of the constrained problem 2 subject to constraints in (24).

This constrained problem can be solved by scaling the system after each transformation in order to ensure that the constraints are met:

**Proposition 3 (a posteriori relaxed scaling)** Considering a state-space realization, it is possible to a posteriori scale it with a diagonal transformation matrix $T$ given by

$$T_{ij} = \delta \sqrt{(W_c)_{ij} 2^{-p}} (\delta \sqrt{|W_c|})^{-\alpha},$$

(31)

such that the constraints (24) are satisfied. Moreover, it is possible to build all the transformation matrices that meet the constraints (24): Let us consider an invertible matrix $U \in \mathbb{R}^{n \times p}$, then the transformation matrix $T = UV$ with $V$ diagonal such that:

$$V_{ij} = \delta \sqrt{(U^+ W U^+)_{ij} 2^{-p}} (\delta \sqrt{|U^+ W U^+|})^{-\alpha},$$

(32)

produces the relaxed-$L_2$-scaling.

**Proof:**

$\mathcal{F}_2$ acts as a modulo operator. For $x \in \mathbb{R}$, $\bar{x} \triangleq 2^{p_1}(x)+a$ is such that $2^{d} \leq \bar{x} < 2^{d+1}$. Since the constraints (24) are equal to

$$2^{\alpha_i} \leq \delta \sqrt{|W_c|_{ij} 2^{-p}} < 2^{\alpha_i+1},$$

(33)

and $T$ transforms $(W_c)_{ij}$ into $T_{ij}^{-2}$ $(W_c)_{ij}$, then $T_{ij}$ has to be of the form:

$$\delta T_{ij} = 2^{\alpha_i} (\delta \sqrt{|W_c|_{ij} 2^{-p}} < 2^{\alpha_i+1}),$$

(34)

Thus, the optimization problem is given by

$$U_{opt} = \arg \min \quad M_{L_2}(UV).$$

(35)

This was implemented in the FWR toolbox\(^4\) for Matlab, with fminsearch, and fminunc functions, and they both give same results with similar numbers of iterations.

Of course, the use of the matrix $V$, that is merely used to eliminate the constraints and solve an unconstrained minimization problem, increases the degree of non-linearity for the objective function to minimize. However, this seems not to be a problem, since in our tests, the optimal realizations found seem to be global optima.

6. EXAMPLE

Let us consider the following digital filter, given by its transfer function:\(^5\)

$$H(z) = \frac{4.297e - 3z^4 - 8.595e - 3z^2 + 4.297e - 3}{z^4 - 3.803z^3 + 5.550z^2 - 3.678z + 0.9355}$$

(36)

and its multiple equivalent (in infinite precision) realizations:

- $\mathcal{F}_1$ is the Direct Form II given by (37).
- $\mathcal{F}_2$ is the optimal $L_2$-scaled realization (solution of problem 2).
- $\mathcal{F}_3$ is the optimal relaxed-$L_2$-scaled realization (problem 3), with $\max u$ a power of 2, and $\delta = 1$. It is obtained with Proposition 3. The numerical values are given by (39).

The balanced realization (known to be close to the unscaled $M_{L_2}$-optimal realization) was chosen as the starting point for the optimization. The numerical results are presented in the appendix and the pseudo-code associated to the realization $\mathcal{F}_3$ is presented in algorithm 1 (16 bits are considered for the input, output, states and coefficients, 32 bits for the accumulator (no guard bits), and $\max u = 16$).

The following table gives the $M_{L_2}$ sensitivities of these different realizations:

<table>
<thead>
<tr>
<th>realization</th>
<th>$M_{L_2}$ sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}_1$</td>
<td>1.690e+09</td>
</tr>
<tr>
<td>$\mathcal{F}_2$</td>
<td>5274.9</td>
</tr>
<tr>
<td>$\mathcal{F}_3$</td>
<td>5222.1</td>
</tr>
</tbody>
</table>

In this example, the relaxed $L_2$-scaled realization $\mathcal{F}_3$ achieves lower sensitivity than the strict $L_2$-scaled optimal realization $\mathcal{F}_2$ while protecting implementation from overflows.

But it is not always the case : if we consider the example in [7], the optimal relaxed-$L_2$-scaled realization satisfies $(W_c)_{ij} = 1$ and is then also a strict $L_2$-scaled realization. This depends on the diagonal terms of the controllability Gramians of the (non scaled) optimal realization.

It is also interesting to notice that a good estimation of $\max u$ (if it is not a power of 2) can allow to achieve lower sensitivity by moving the constraints (it could also be the case for the example in [7]).

7. CONCLUSION

This paper has presented the $L_2$-sensitivity minimization problem and the associated $L_2$-scaling constraints. These constraints that prevent from overflows have been considered with concrete fixed-point implementation schemes. Novel $L_2$-dynamic-range constraints have been exhibited.

Even if the goal of this paper is not a detailed optimization algorithm like in [7], a proposition to solve the constrained optimization problem has been exhibited and applied on a numerical example.

These relaxed constraints could also be very important for some other realizations, like $\delta$-operator state-space, the $p$-Direct Form II transposed [13] and also classical filtering structures. For these realizations where a parameter $\Delta$ should be used to achieve the $L_2$-scalling, a relaxed-$L_2$-scalling permits to fix this parameter as a power of 2, in order to decrease the amount of computations.

To apply this work to other classical structures, it will be soon extended to the Specialized Implicit Framework [5] that allows to encompass existing structures in an implicit state-space form.

REFERENCES


\(^4\)Due to a lack of space, only 4 digits are given, but more may be required to completely define the system.

\(^5\)sources available at http://fwrtoolbox.gforge.inria.fr/
A. NUMERICAL RESULTS

\[
A_1 = \begin{pmatrix}
3.8031 & -1.3875 & 0.9196 & -0.4678 \\
4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0.5 & 0
\end{pmatrix}, \quad b_1 = \begin{pmatrix}
0.1250 \\
0 \\
0 \\
0
\end{pmatrix},
\]
\[c_1 = (0.1307 -0.0649 0.0316 0.0011), \quad d_1 = 4.2974e-3
\]

\[
A_2 = \begin{pmatrix}
0.9386 & -0.2477 & -0.0427 & 0.0077 \\
0.2525 & 0.9675 & -0.0010 & -0.0003 \\
0.0228 & -0.0031 & 0.9301 & -0.2524 \\
0.0039 & -3.071e-05 & 0.2517 & 0.9669
\end{pmatrix}, \quad b_2 = \begin{pmatrix}
-0.0652 \\
-0.0193 \\
0.3130 \\
0.0471
\end{pmatrix},
\]
\[c_2 = (-1.1151 0.2002 -0.1789 0.0729), \quad d_2 = 4.2974e-3
\]

\[
A_3 = \begin{pmatrix}
0.9300 & -0.2504 & 0.0290 & -0.0030 \\
0.2528 & 0.9671 & 0.0027 & -0.0059 \\
-0.0483 & 0.0091 & 0.9385 & -0.2466 \\
0.0001 & 0.0056 & 0.2527 & 0.9674
\end{pmatrix}, \quad b_3 = \begin{pmatrix}
0.6289 \\
0.0665 \\
-0.1326 \\
-0.0440
\end{pmatrix},
\]
\[c_3 = (-0.0872 0.0335 -0.5567 0.1104), \quad d_3 = 4.2974e-3
\]

**Input:** xn: 16 bits integer  
**Output:** y: 16 bits integer  
**Data:** xn, xnp: array [1..4] of 16 bits integers  
**Data:** Acc: 32 bits integer

```plaintext
begin
    // Intermediate variables
    Acc ← (xn(1)*31209);  
    Acc ← Acc + (xn(2) * -8303);  
    Acc ← Acc + (xn(3) * -600);  
    Acc ← Acc + (xn(4) * 78);  
    Acc ← Acc + (u * -806);  
    xnp(1) ← Acc >> 15;  
    Acc ← (xn(1) * 8304);  
    Acc ← Acc + (xn(2) * 31690);  
    Acc ← Acc + (xn(3) * -77);  
    Acc ← Acc + (xn(4) * 10);  
    Acc ← Acc + (u * -104);  
    xnp(2) ← Acc >> 15;  
    Acc ← (xn(1) * 2398);  
    Acc ← Acc + (xn(2) * -315);  
    Acc ← Acc + (xn(3) * 30009);  
    Acc ← Acc + (xn(4) * -8149);  
    Acc ← Acc + (u * 1523);  
    xnp(3) ← Acc >> 15;  
    Acc ← (xn(1) * 313);  
    Acc ← Acc + (xn(2) * -41);  
    Acc ← Acc + (xn(3) * 8147);  
    Acc ← Acc + (xn(4) * 31711);  
    Acc ← Acc + (u * 198);  
    xnp(4) ← Acc >> 15;  
    // Outputs
    Acc ← (xn(1) * -25782);  
    Acc ← Acc + (xn(2) * 3381);  
    Acc ← Acc + (xn(3) * -12182);  
    Acc ← Acc + (xn(4) * 1575);  
    Acc ← Acc + (u + 18);  
    y ← Acc >> 15;  
    // Permutations
    x ← xnp;
end
```

**Algorithm 1:** Numerical fixed-point algorithm of realization \( \mathcal{H}_3 \)