Quantum algorithms

Daniel J. Bernstein

"Quantum algorithm"
means an algorithm that
a quantum computer can run.

i.e. a sequence of instructions,
where each instruction is
in a quantum computer's
supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “T gate”.

Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “*T* gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

Quantum computer type 1 (QC1): contains many “qubits”; can efficiently perform “NOT gate”, “Hadamard gate”, “controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions to compute “Toffoli gate”; . . . “Simon’s algorithm”; . . . “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.

Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;

A note on D-Wave

Apparent scientific consensus:
Current “quantum computers” from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.
e.g.: $(0,1,1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0,1\}$.
e.g.: $(1,1,1,1,1,0,0,0,1,$
$0,0,0,0,0,0,1,1,0,0,0,$
$0,1,0,0,1,0,0,0,0,0,1,$
$1,0,1,0,0,0,1,0,0,0,1,$
$0,0,1,1,1,0,0,1,0,0,0,$
$1,1,0,1,1,0,0,1,0,0,1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list
of numbers stored in n qubits.

Measuring a quantum computer

Can simply look at a bit.

Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

e.g.: Say 3 qubits have state $(1,1,1,1,1,1,1,1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state

$(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state

$(0, 0, 0, 0, 0, 1, 0, 0)$.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

$000 = 0$ with probability 0;

$001 = 1$ with probability 0;

$010 = 2$ with probability 0;

$011 = 3$ with probability 0;

$100 = 4$ with probability 0;

$101 = 5$ with probability 1;

$110 = 6$ with probability 0;

$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$

$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT gates

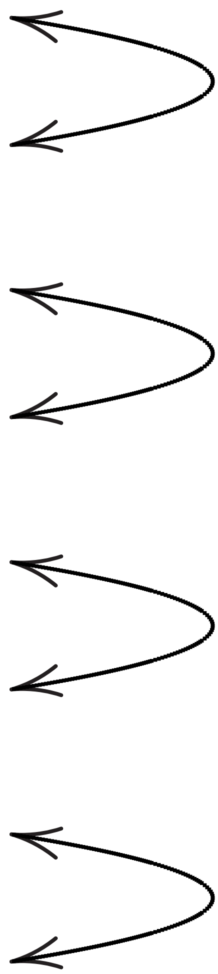
NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111



Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$

$(3, 1, 1, 4, 5, 9, 6, 2)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,9,5,6,2)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. C_0NOT_2:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $C_0C_1NOT_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

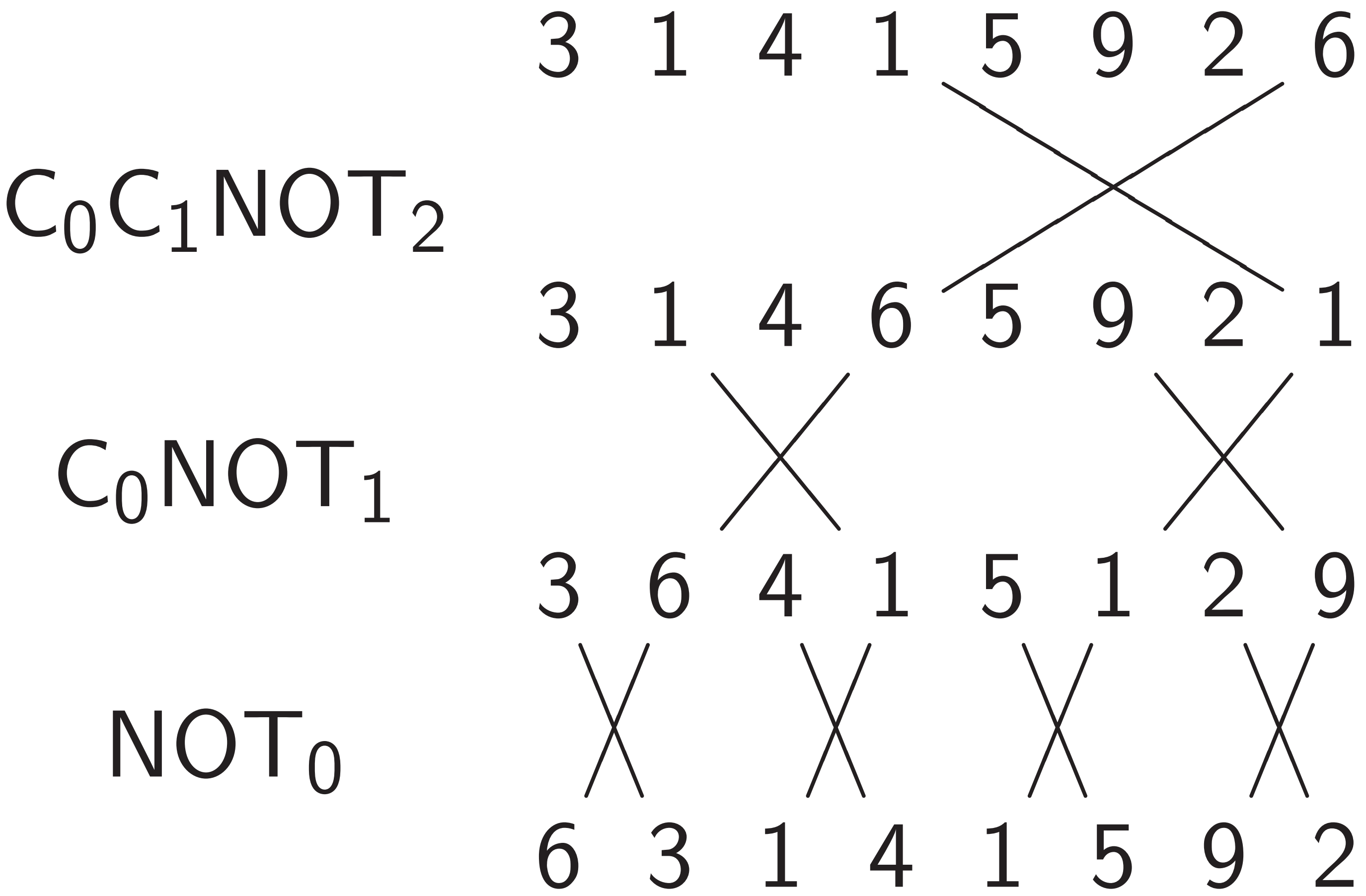
More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

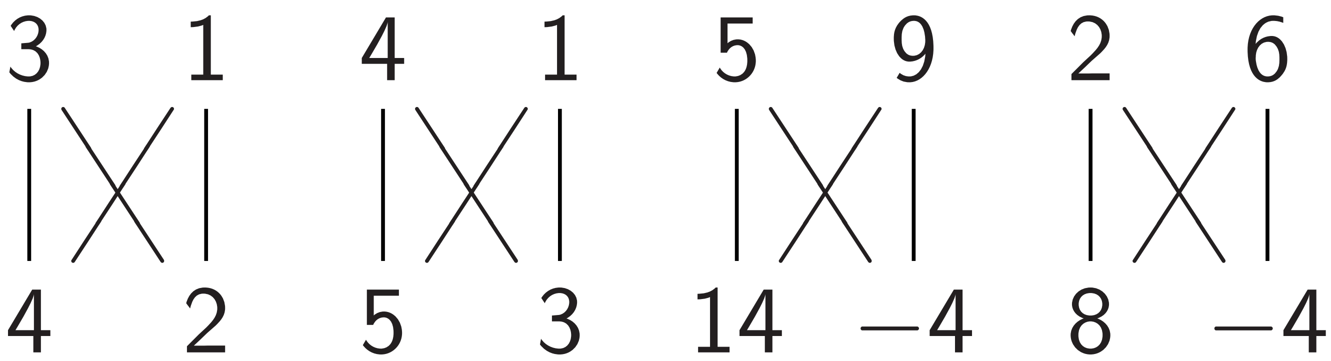
e.g. series of gates to rotate 8 positions by distance 1:



Hadamard gates

Hadamard$_0$:

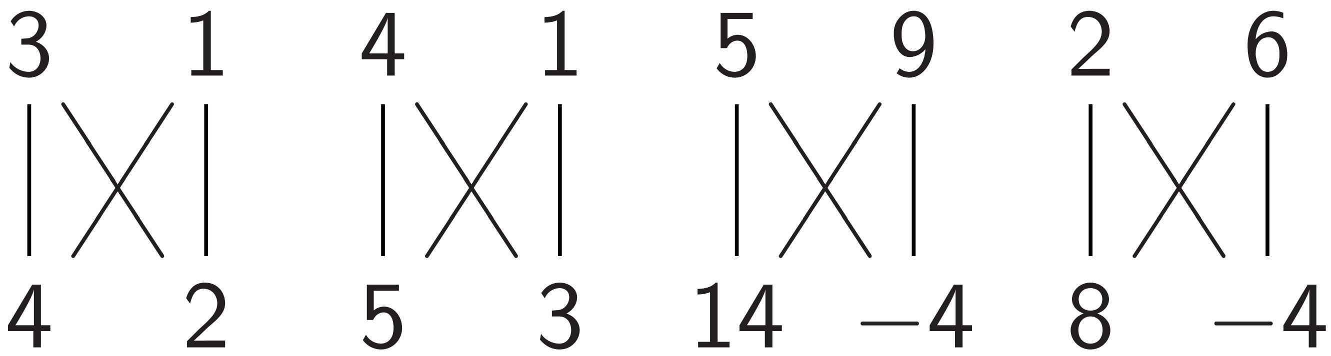
$(a, b) \mapsto (a + b, a - b)$.



Hadamard gates

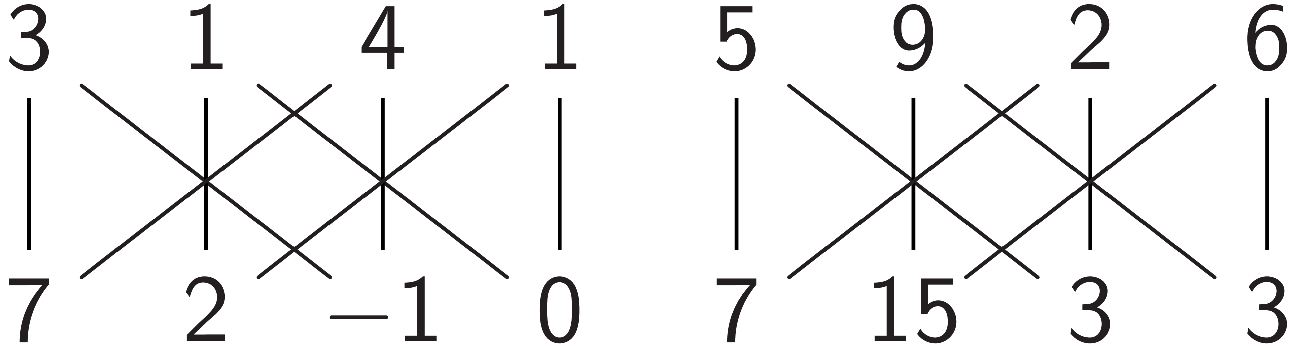
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



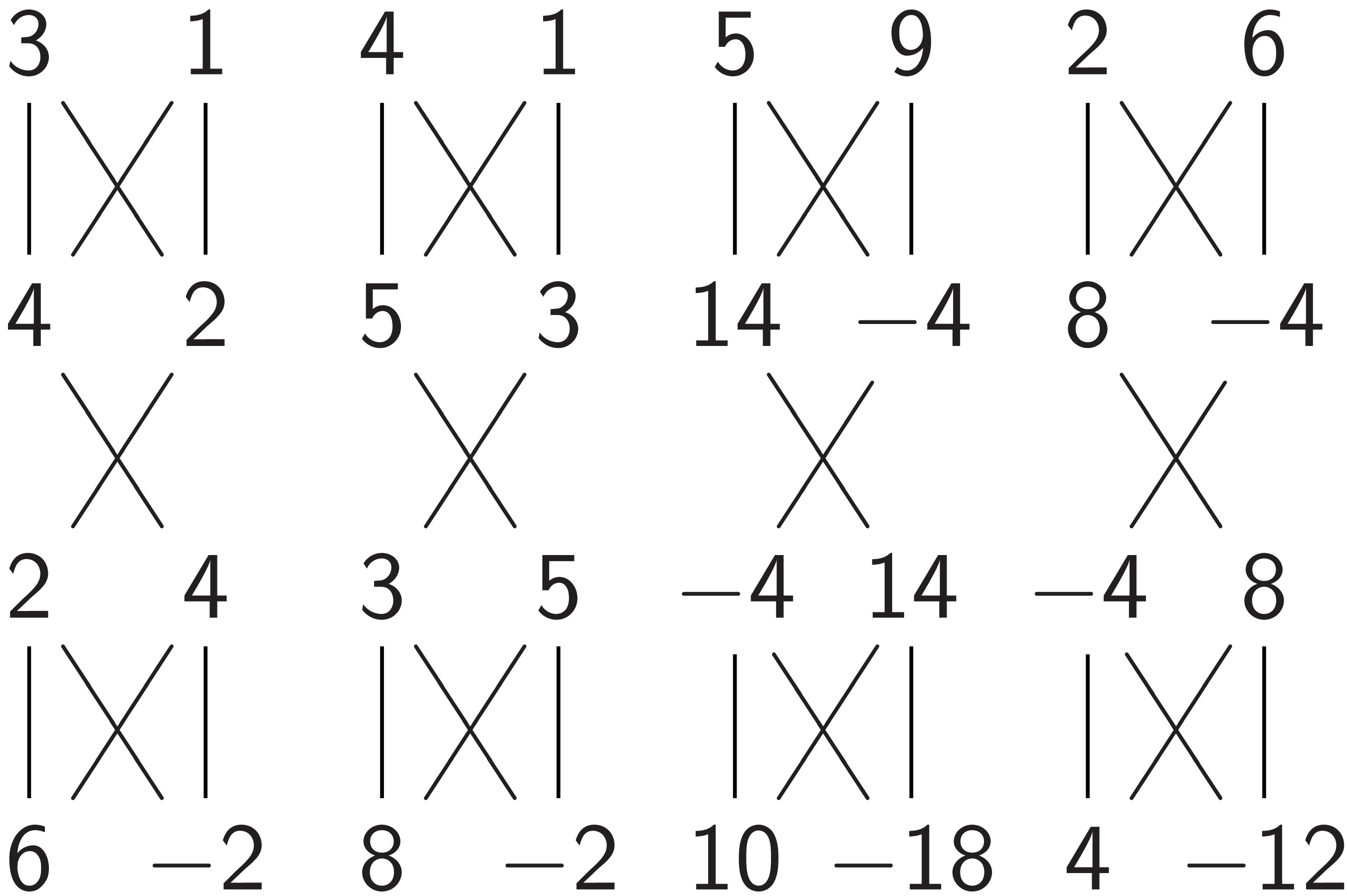
Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



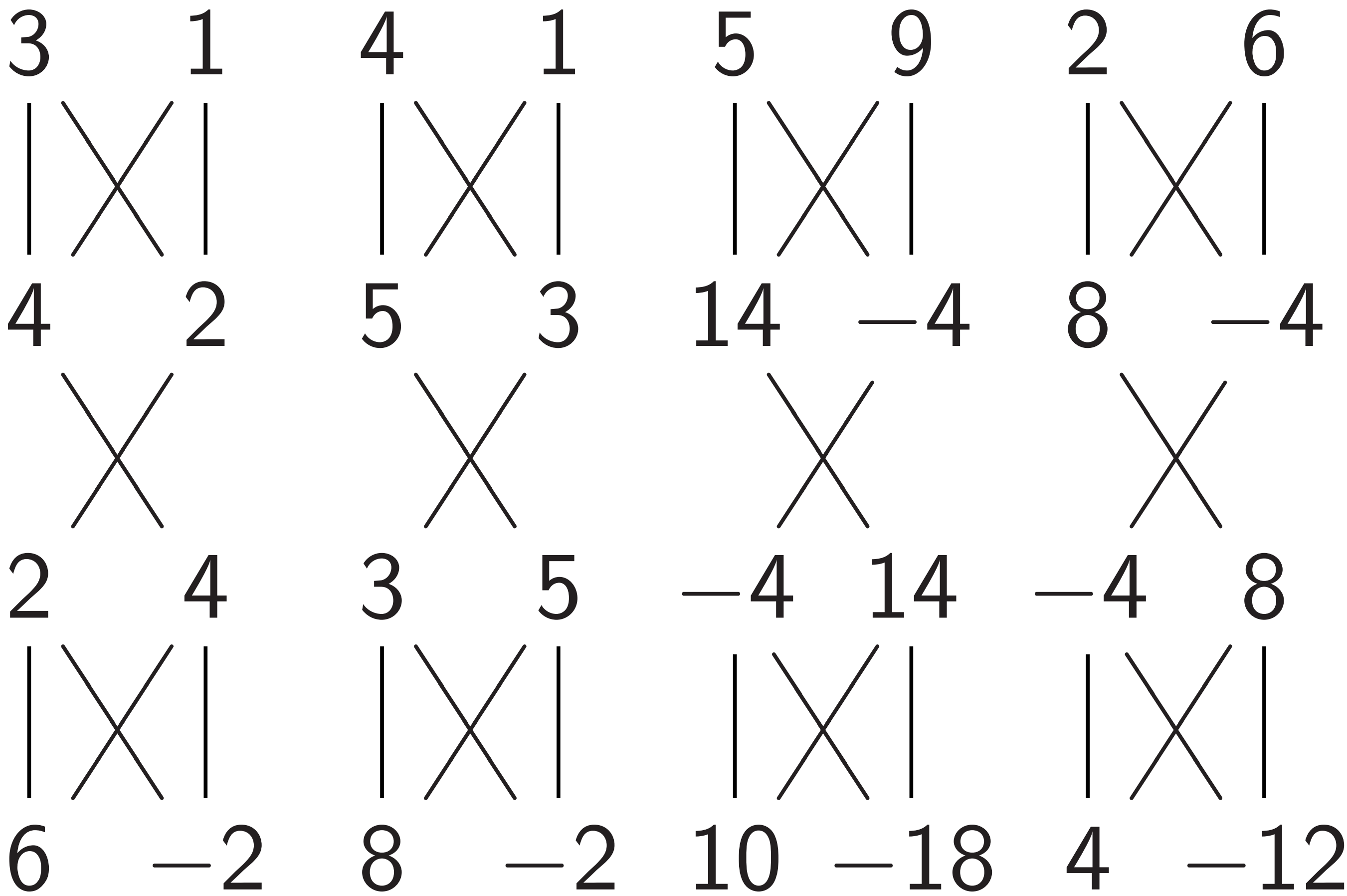
Some uses of Hadamard gates

Hadamard_0, NOT_0, Hadamard_0:



Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:

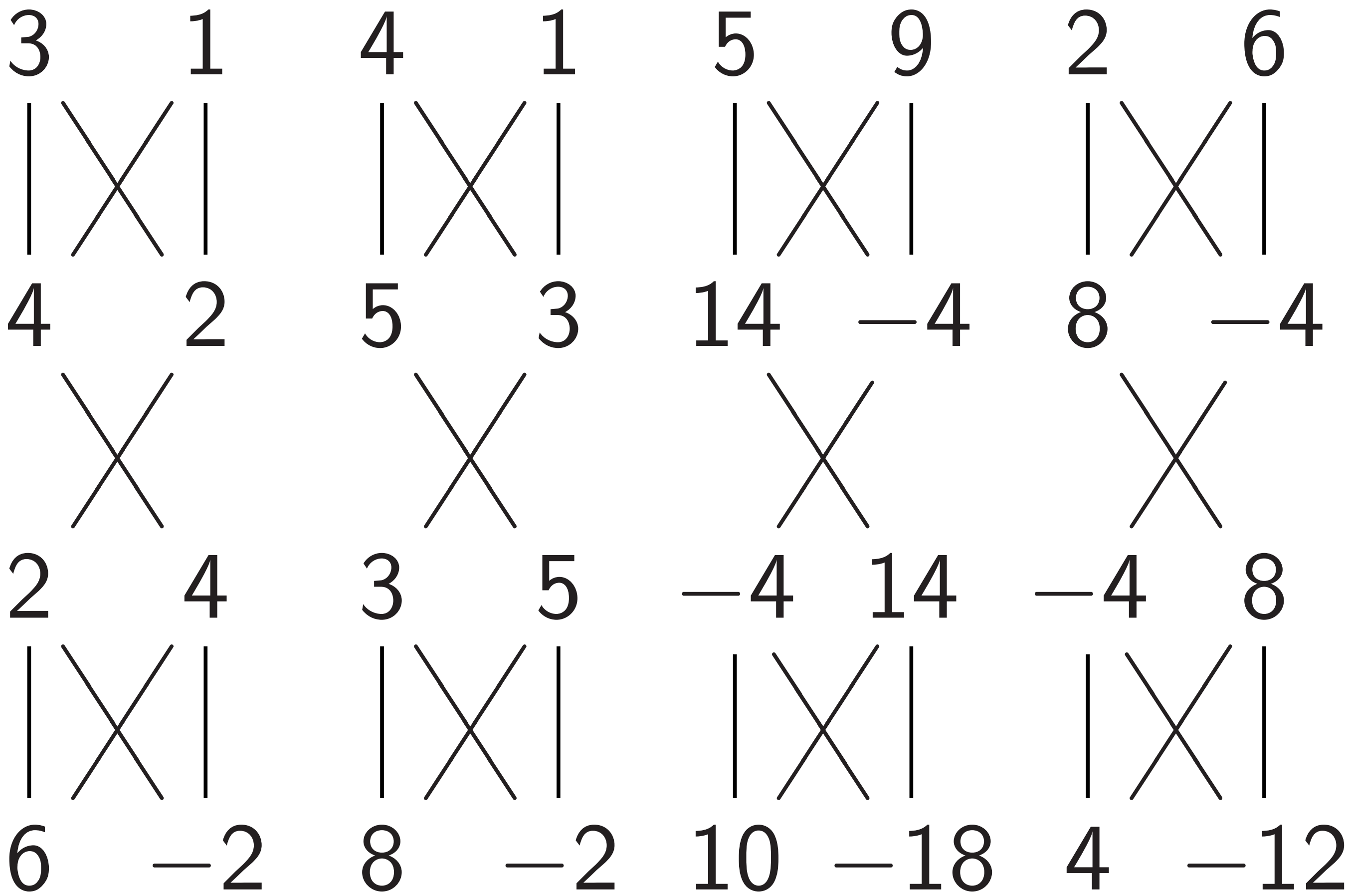


"Multiply each amplitude by 2."

This is not physically observable.

Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:



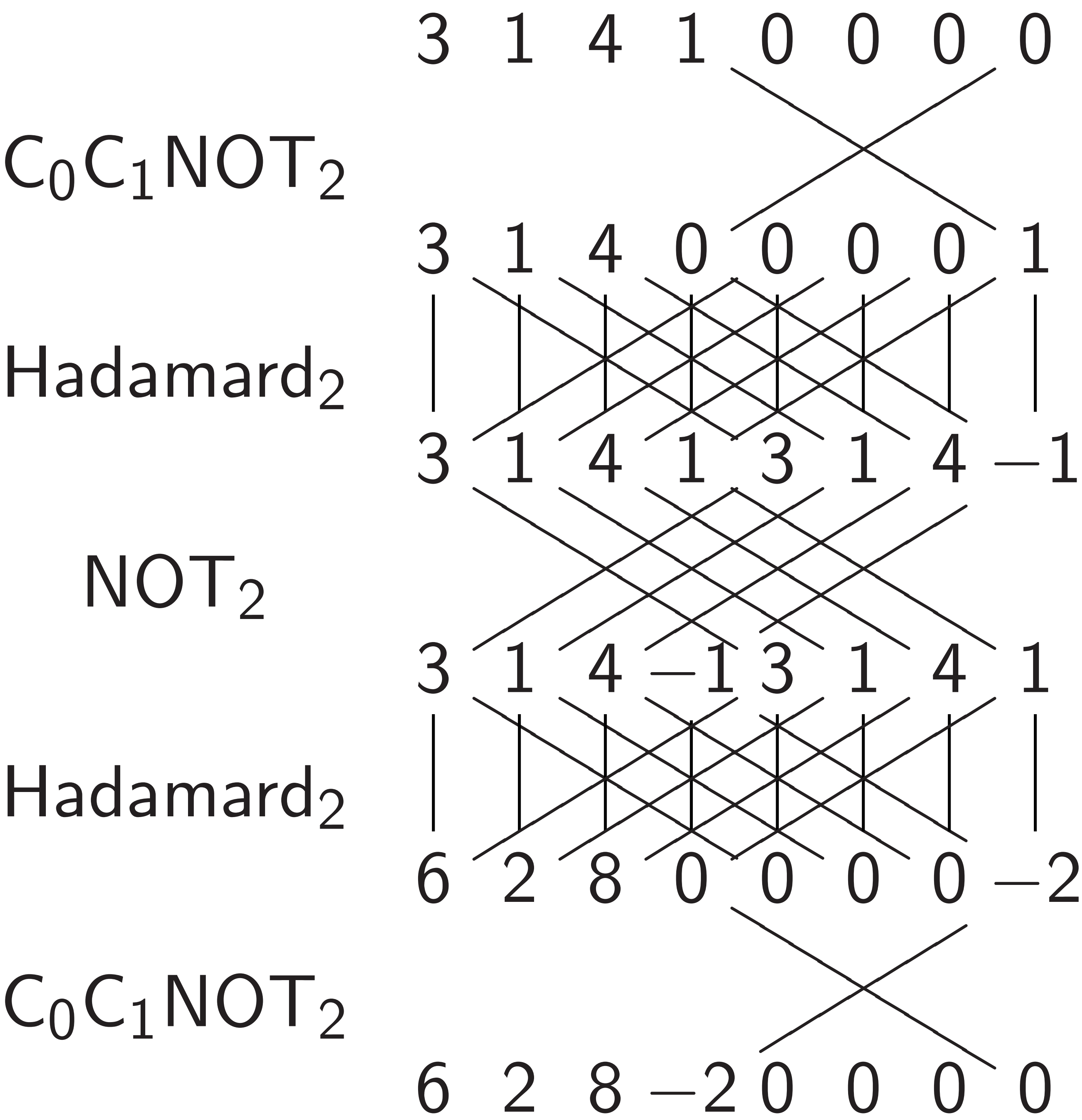
"Multiply each amplitude by 2."
This is not physically observable.

"Negate amplitude if q_0 is set."
No effect on measuring *now*.

Fancier example:

"Negate amplitude if $q_0 q_1$ is set."

Assumes $q_2 = 0$: "ancilla" qubit.

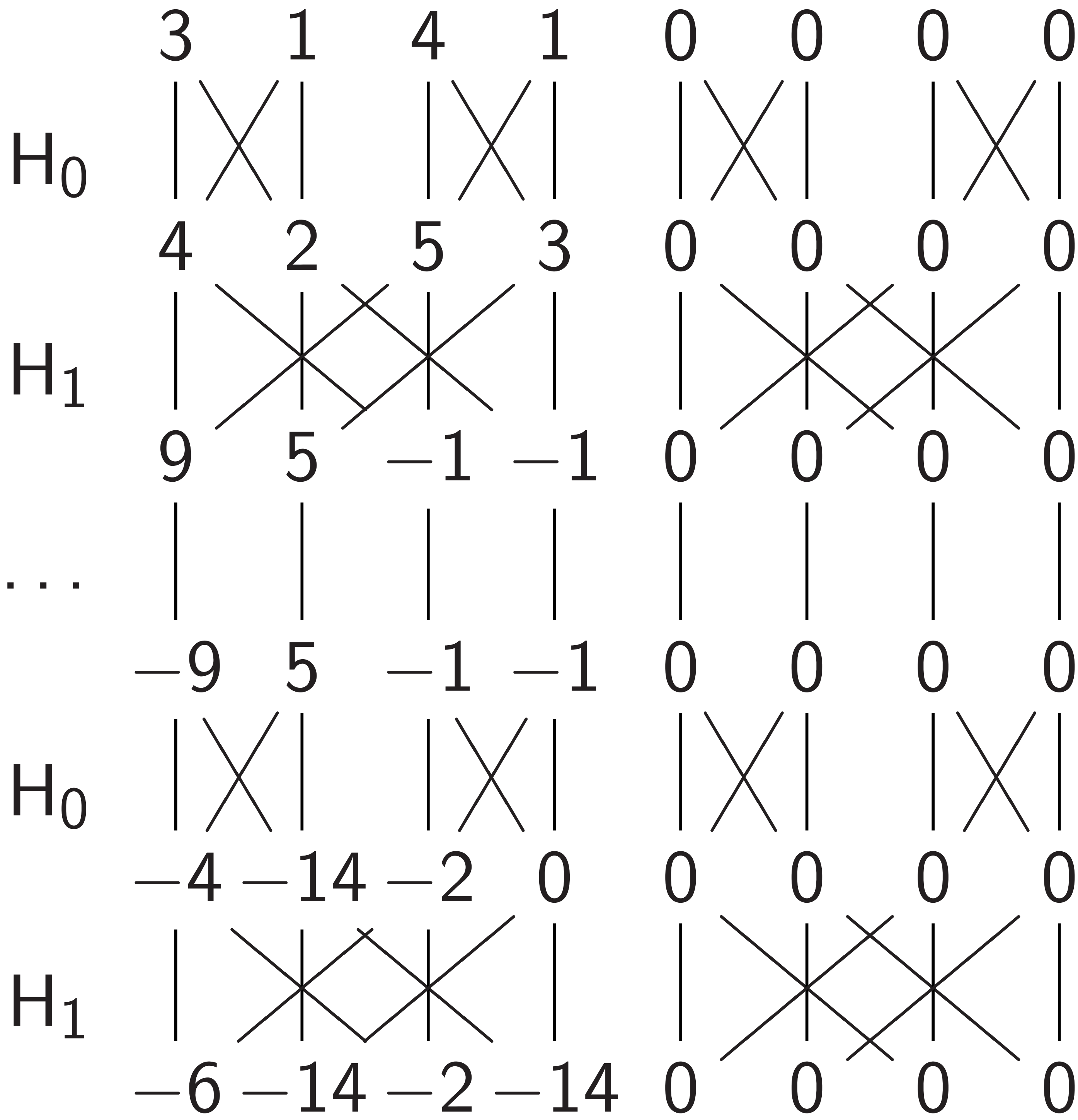


Affects measurements: "Negate amplitude around its average."

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

Affects measurements: "Negate amplitude around its average."

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.



Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 1. Set up pure zero state:

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Example of Simon's algorithm

Step 2. Hadamard$_0$:

1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Example of Simon's algorithm

Step 3. Hadamard$_1$:

1, 1, 1, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0.

Example of Simon's algorithm

Step 4. Hadamard_2:

1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe.

Example of Simon's algorithm

Step 5. C_0NOT_3:

1, 0, 1, 0, 1, 0, 1, 0,
0, 1, 0, 1, 0, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5b. More shuffling:

1, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5c. More shuffling:

1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5d. More shuffling:

1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0,

0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 1, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5e. More shuffling:

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5f. More shuffling:

0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5g. More shuffling:

0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5h. More shuffling:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5i. More shuffling:

0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 1, 0, 0, 0, 0, 0,

0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5j. Final shuffling:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0.

Each column is a parallel universe performing its own computations.

Example of Simon's algorithm

Step 5j. Final shuffling:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0.

Each column is a parallel universe performing its own computations.

Surprise: u and $u \oplus 101$ match.

Example of Simon's algorithm

Step 6. Hadamard$_0$:

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, $\overline{1}$, 0, 0, 1, 1,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 1, 0, 0, 1, $\overline{1}$,

1, $\overline{1}$, 0, 0, 1, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 0, 0, 1, $\overline{1}$, 0, 0.

Notation: $\overline{1}$ means -1.

Example of Simon's algorithm

Step 7. Hadamard$_1$:

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, \bar{1}, \bar{1}, 1, 1, 1, \bar{1}, \bar{1},$

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, 1, \bar{1}, \bar{1}, 1, \bar{1}, \bar{1}, 1,$

$1, \bar{1}, 1, \bar{1}, 1, 1, 1, 1,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, 1, 1, 1, 1, \bar{1}, 1, \bar{1}.$

Example of Simon's algorithm

Step 8. Hadamard$_2$:

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, $\overline{2}$, 0, 0, $\overline{2}$, 0, 2,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, $\overline{2}$, 0, 0, 2, 0, $\overline{2}$,

2, 0, 2, 0, 0, $\overline{2}$, 0, $\overline{2}$,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, 2, 0, 0, 2, 0, 2.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, $\overline{2}$, 0, 0, $\overline{2}$, 0, 2,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, $\overline{2}$, 0, 0, 2, 0, $\overline{2}$,

2, 0, 2, 0, 0, $\overline{2}$, 0, $\overline{2}$,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, 2, 0, 0, 2, 0, 2.

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$.
"Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation.
Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$.
Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$.
Easy to compute discrete logs.

Grover's algorithm

Assume: unique $s \in \{0,1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0. Success probability is very low until #inputs approaches 2^n.

Grover's algorithm

Assume: unique $s \in \{0,1\}^n$
has $f(s) = 0$.

Traditional algorithm to find s:
compute f for many inputs,
hope to find output 0.
Success probability is very low
until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$
reversible computations of f.
Typically: reversibility overhead
is small enough that this
easily beats traditional algorithm.

Start from uniform superposition over all n-bit strings u.

Start from uniform superposition over all n-bit strings u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

Start from uniform superposition over all n-bit strings u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Start from uniform superposition over all n-bit strings u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Start from uniform superposition over all n-bit strings u.

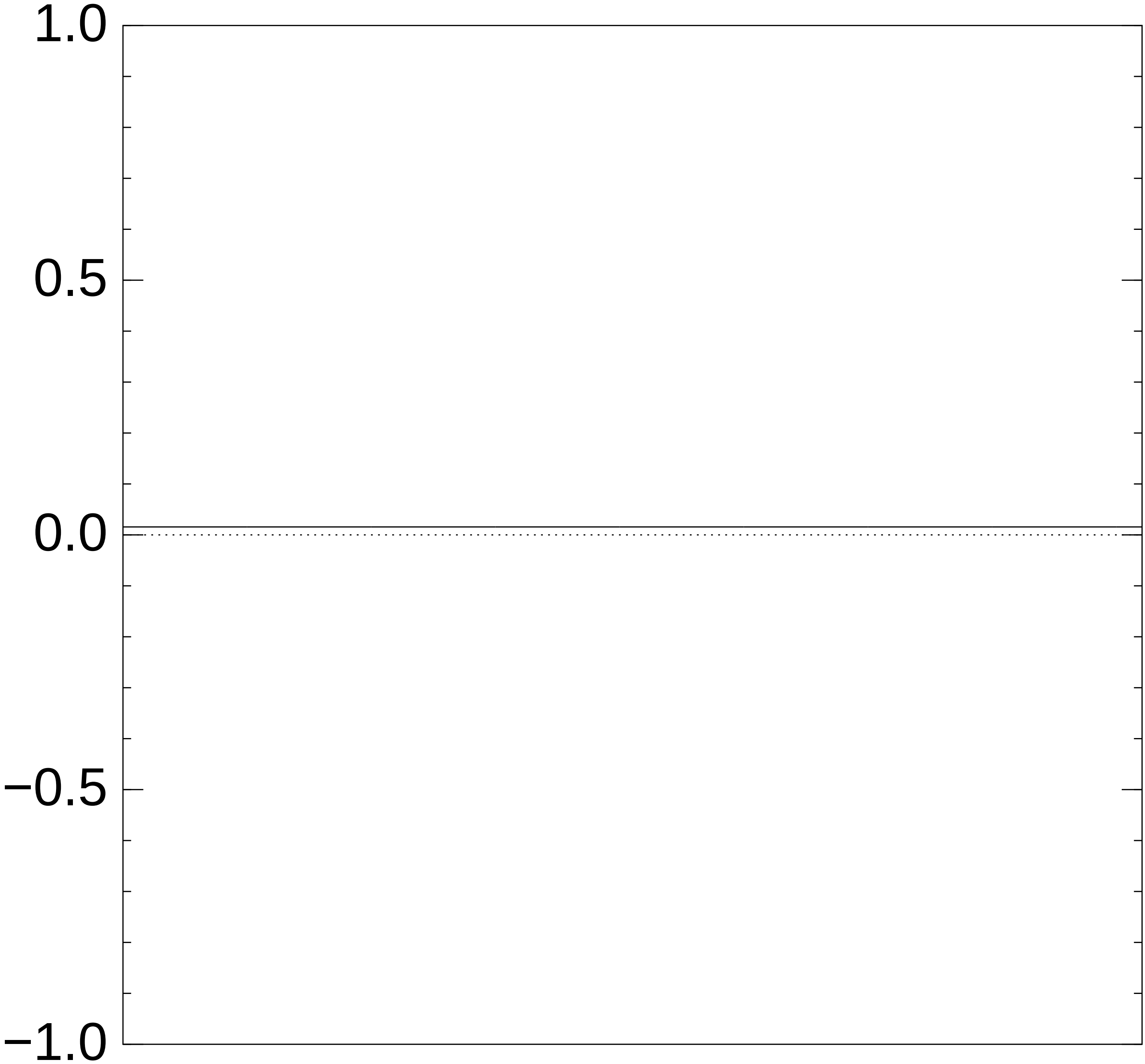
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

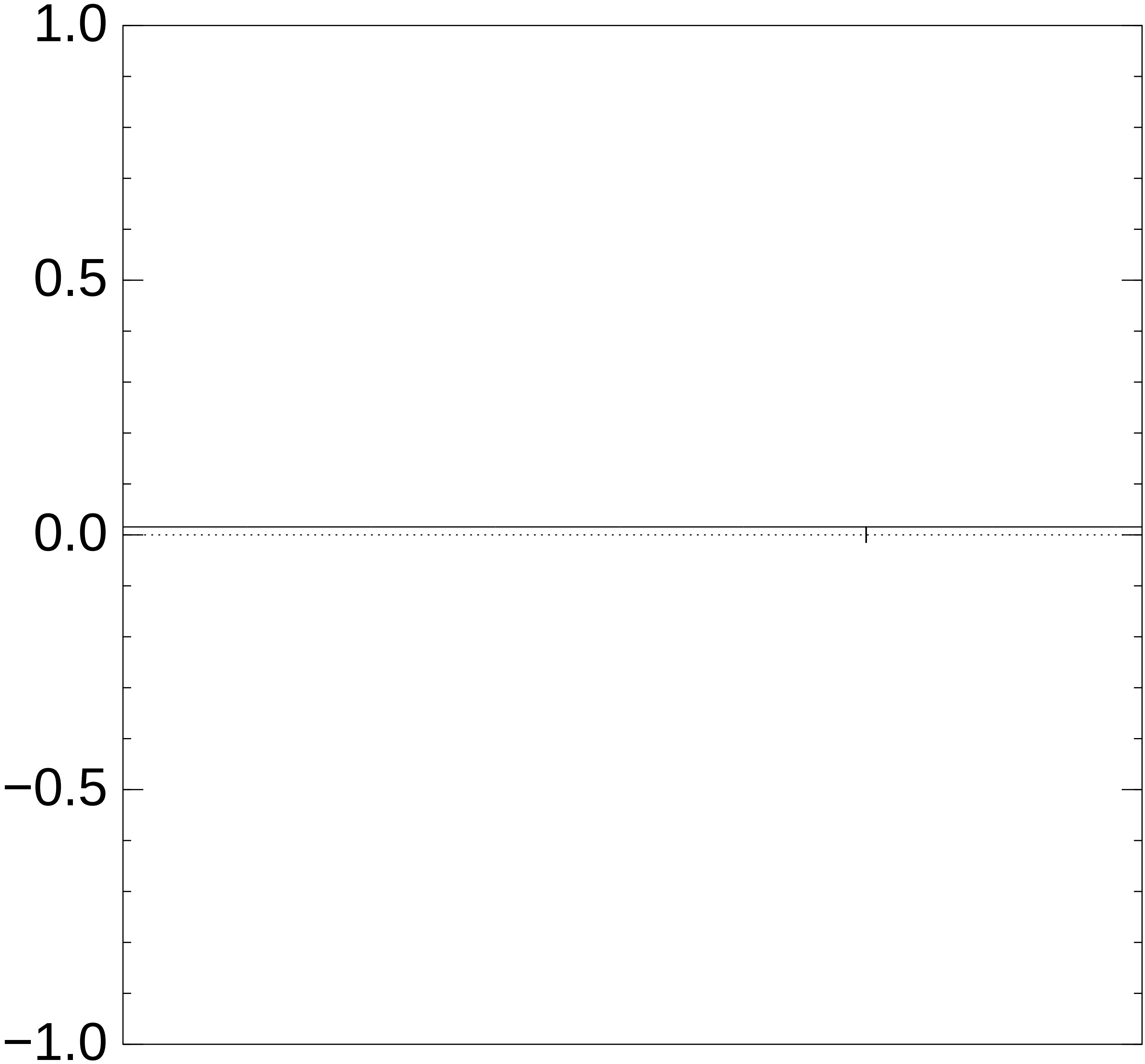
Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

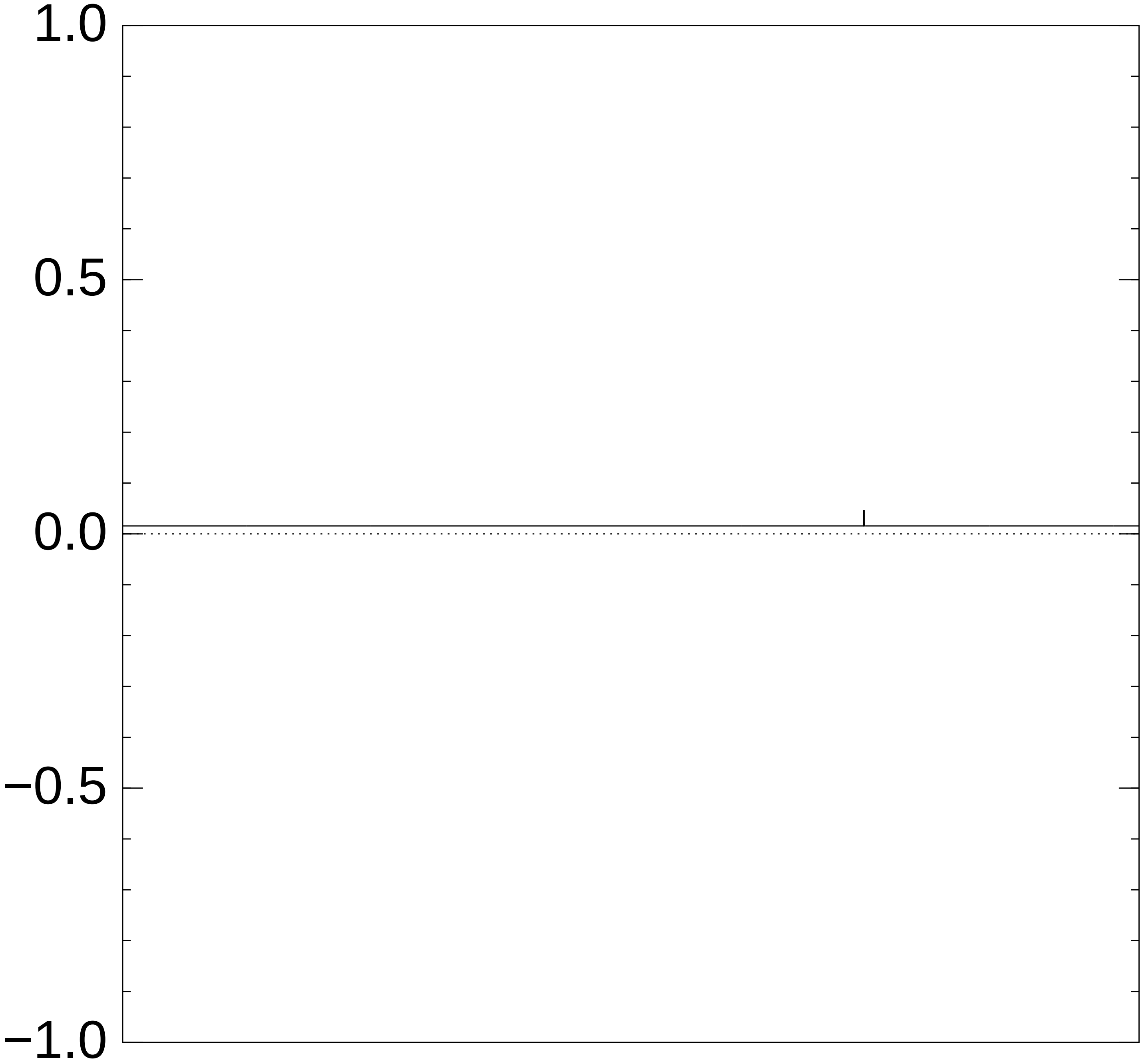
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after 0 steps:



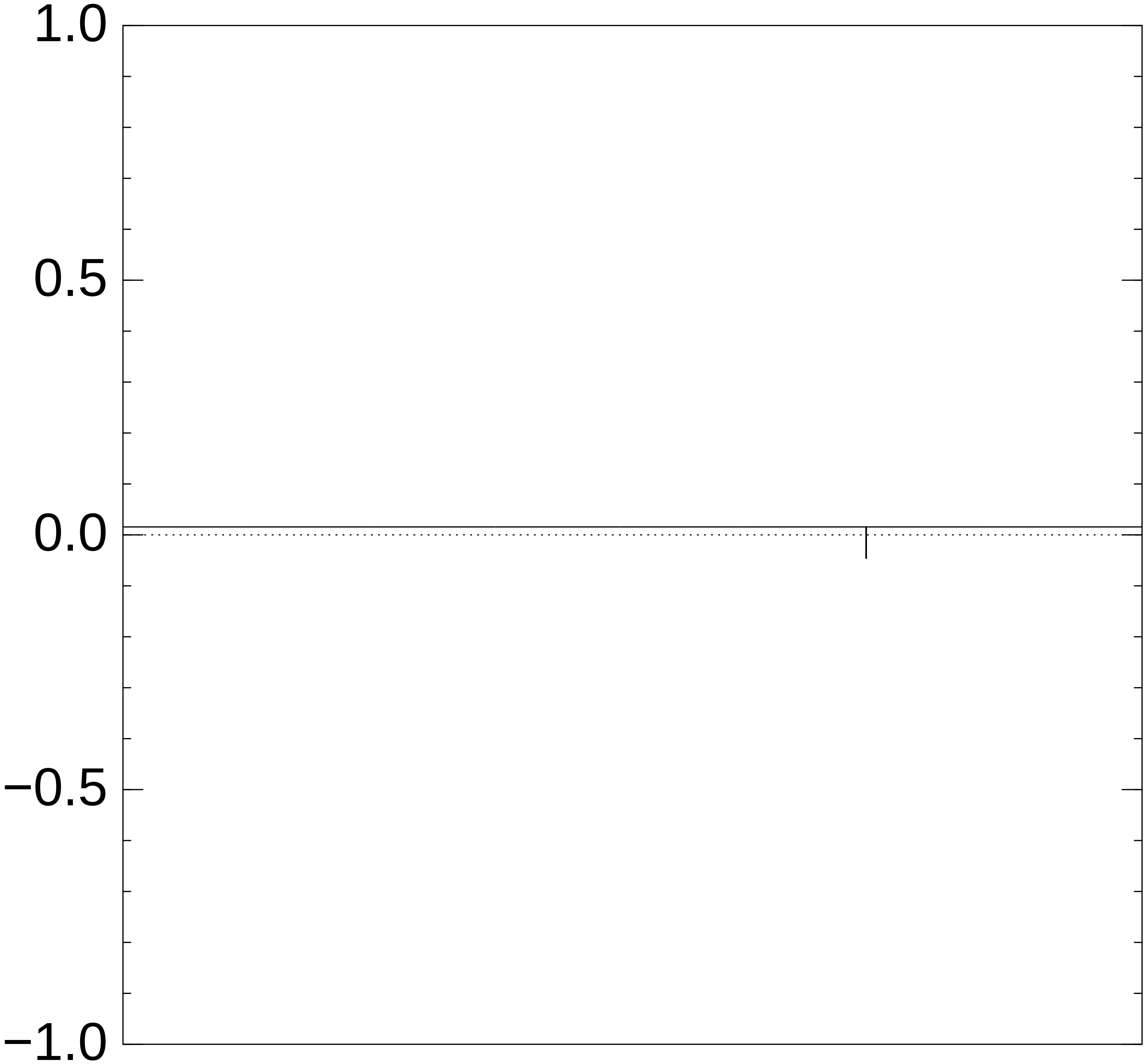
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after Step 1:



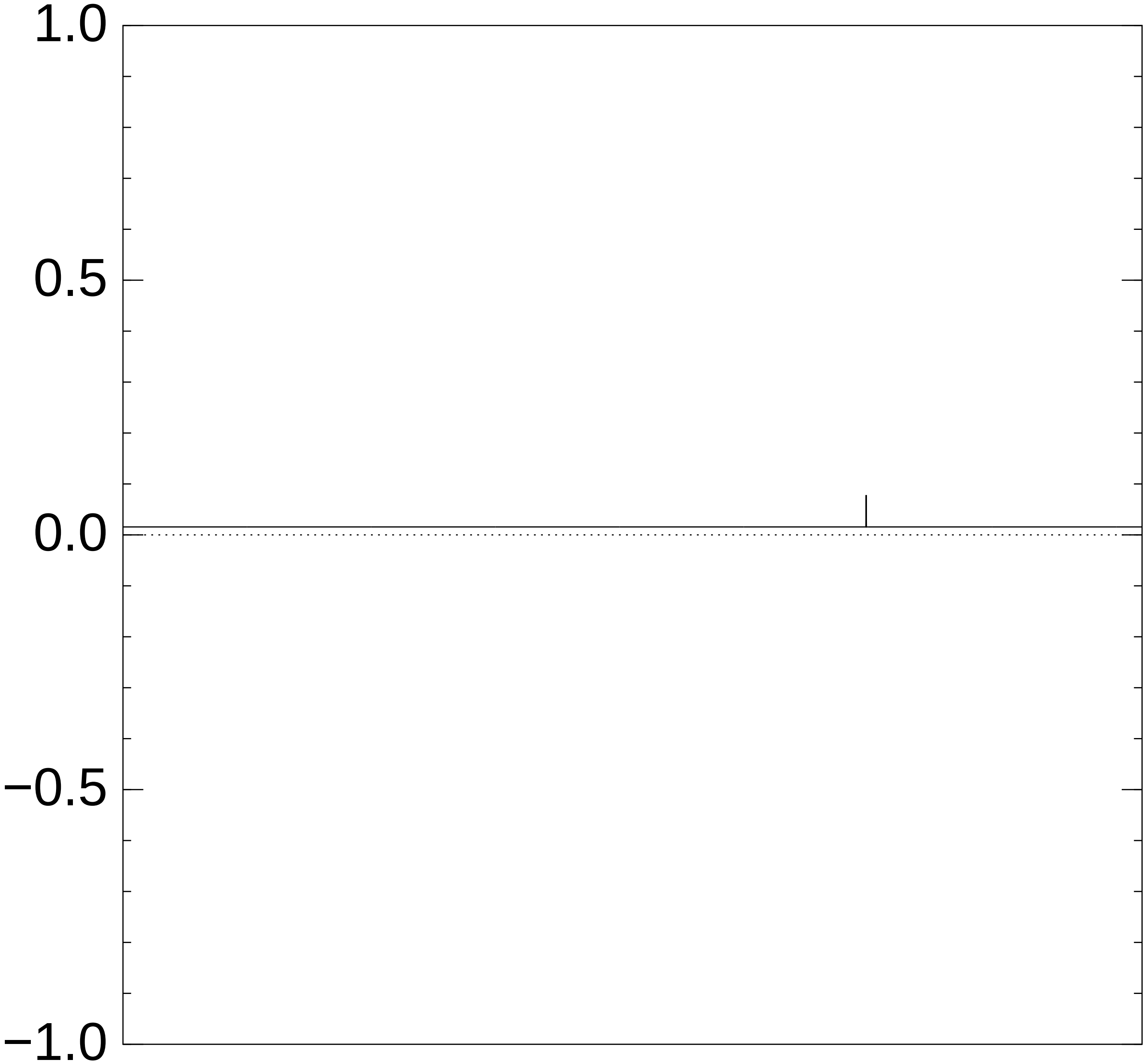
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after Step 1 + Step 2:



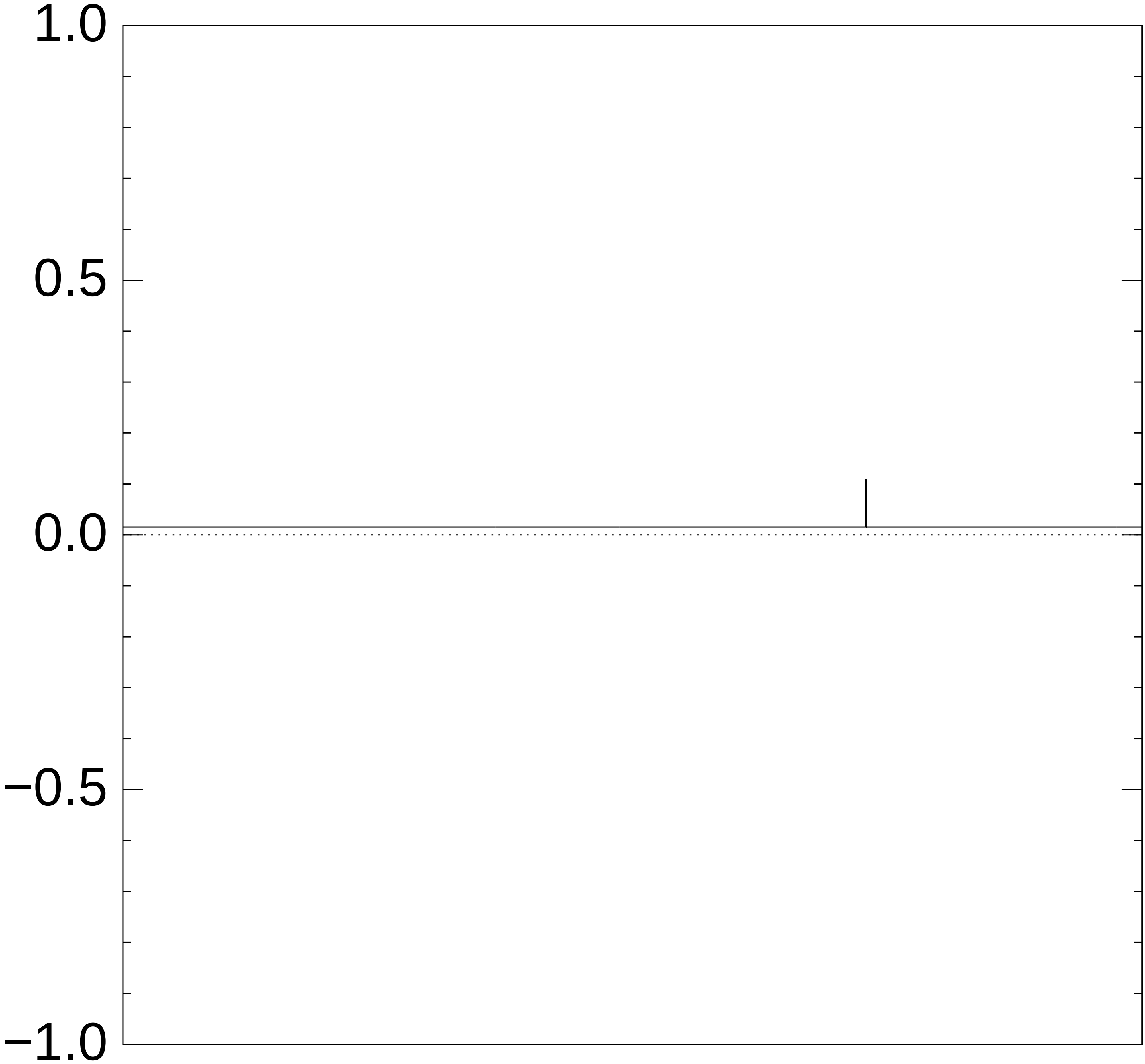
Normalized graph of $u \mapsto a_u$ for an example with $n = 12$ after Step $1 +$ Step $2 +$ Step 1:



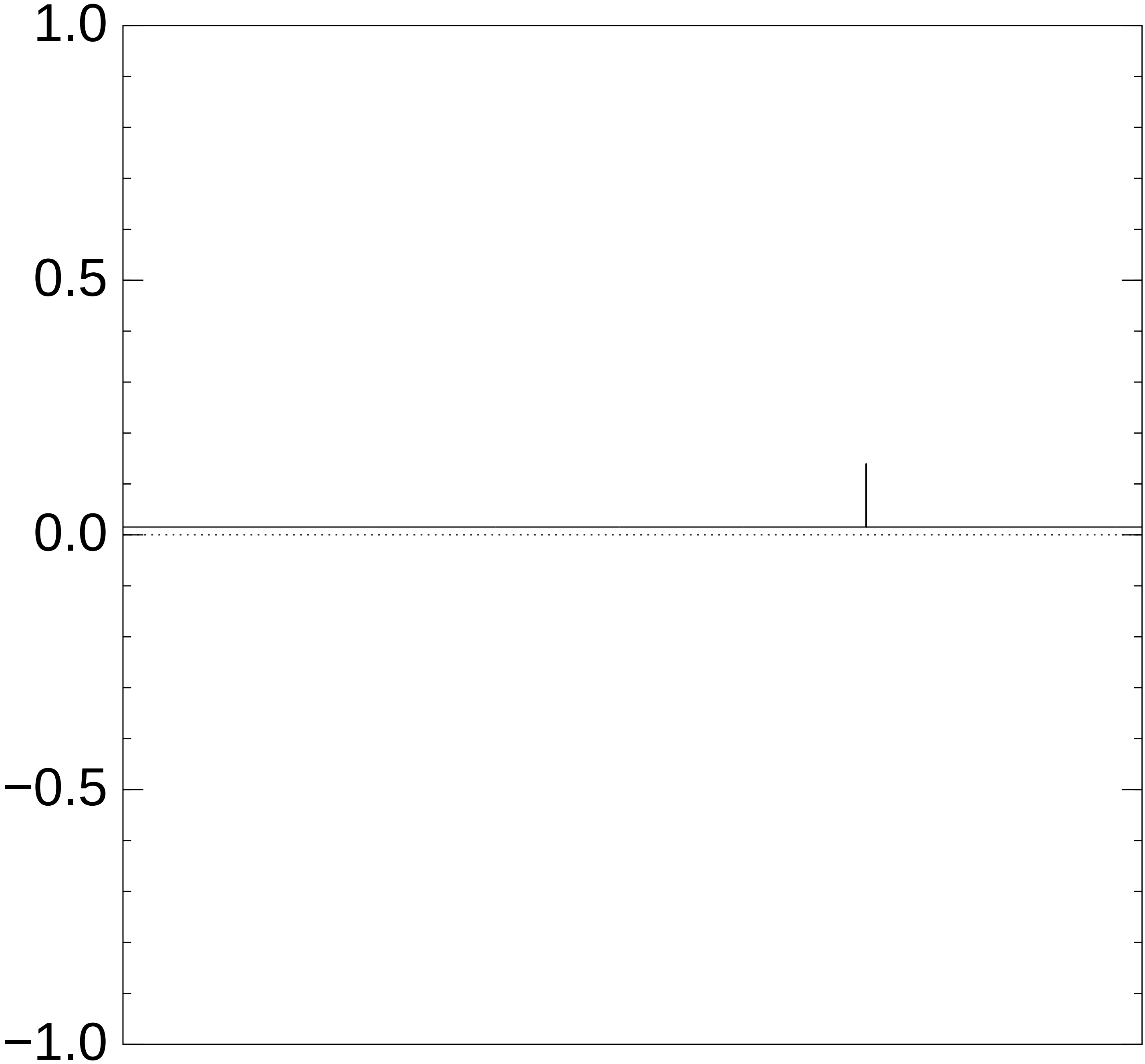
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $2 \times$ (Step 1 + Step 2):



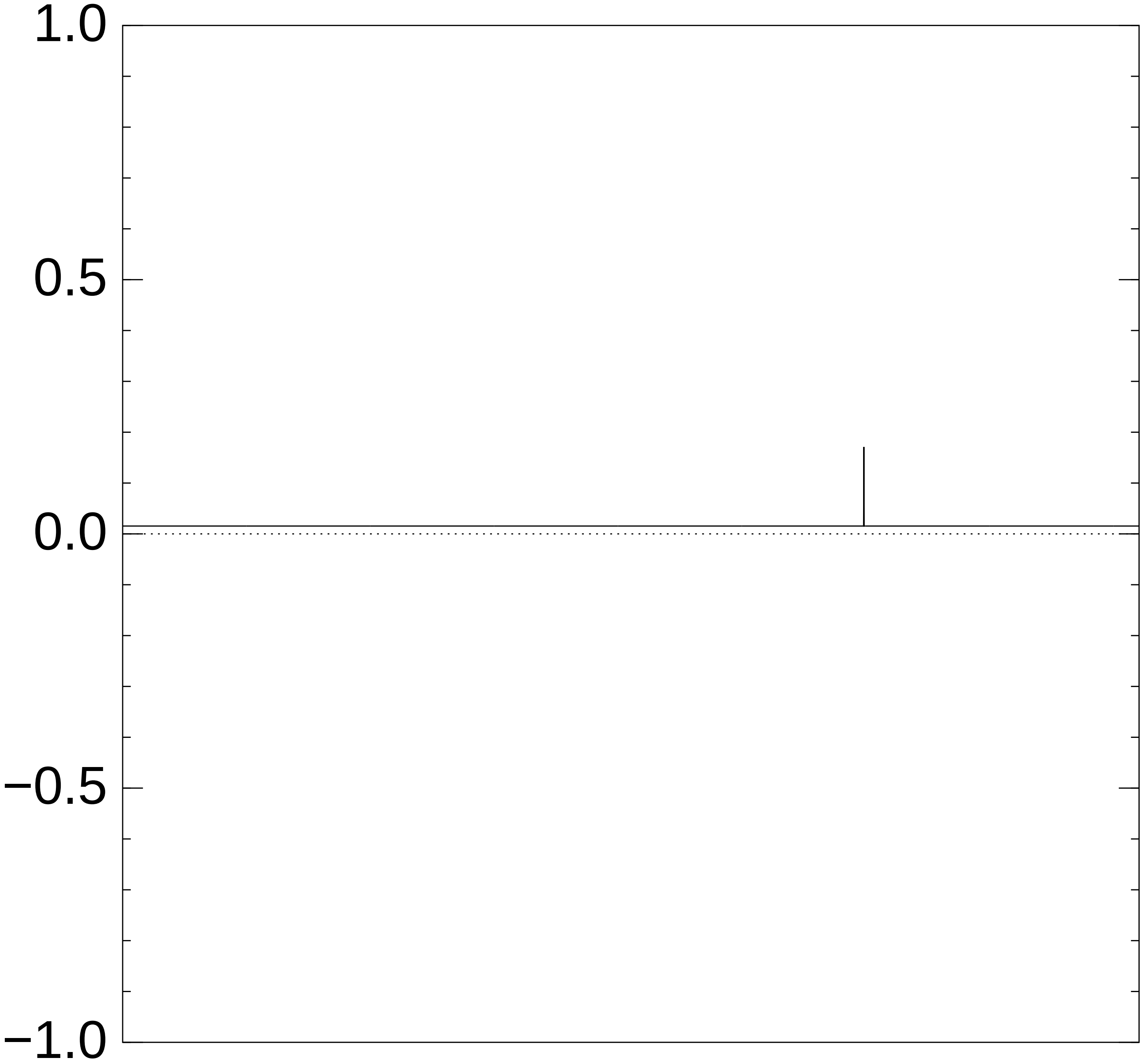
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $3 \times$ (Step 1 + Step 2):



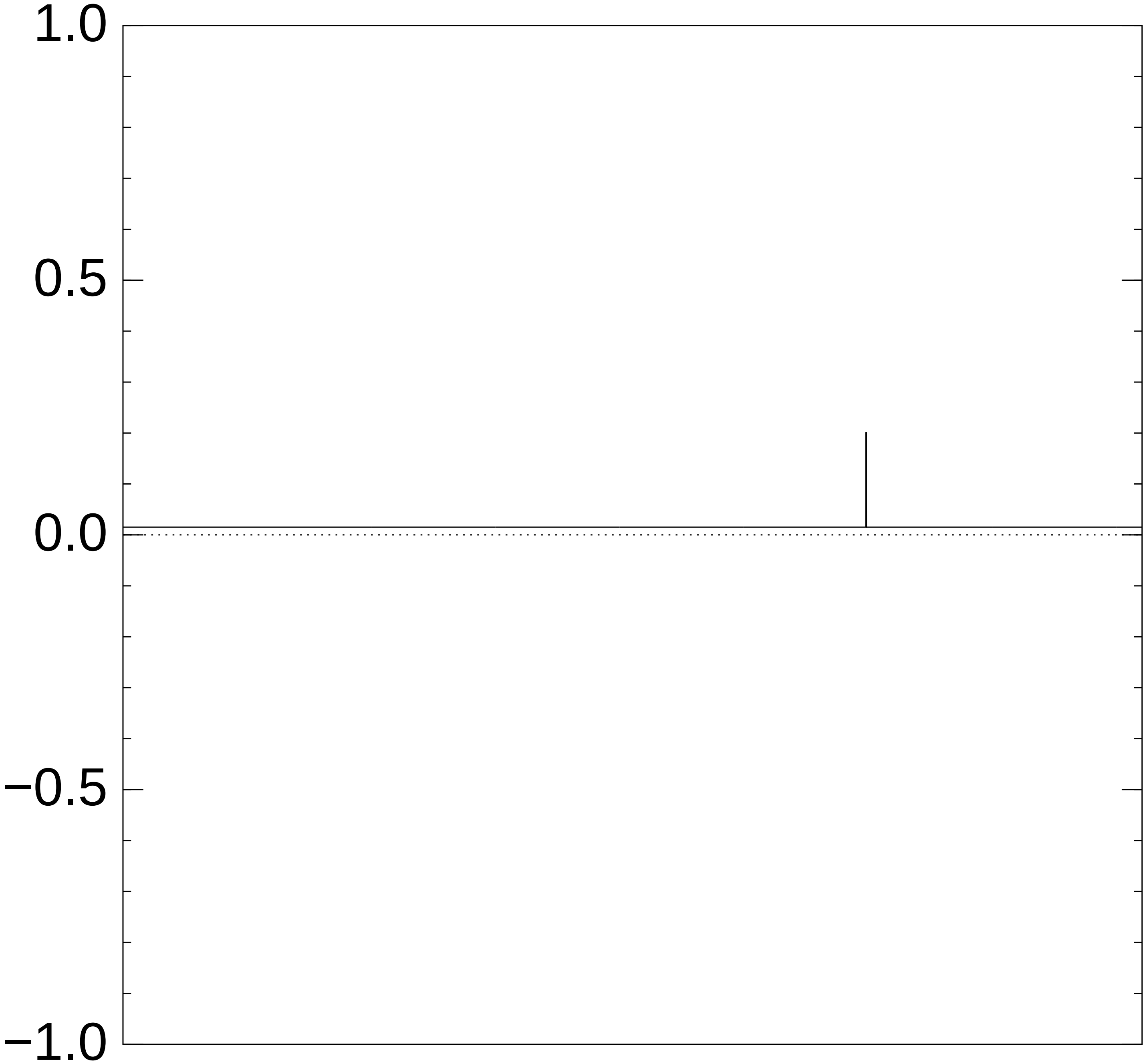
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $4 \times$ (Step $1 +$ Step 2):



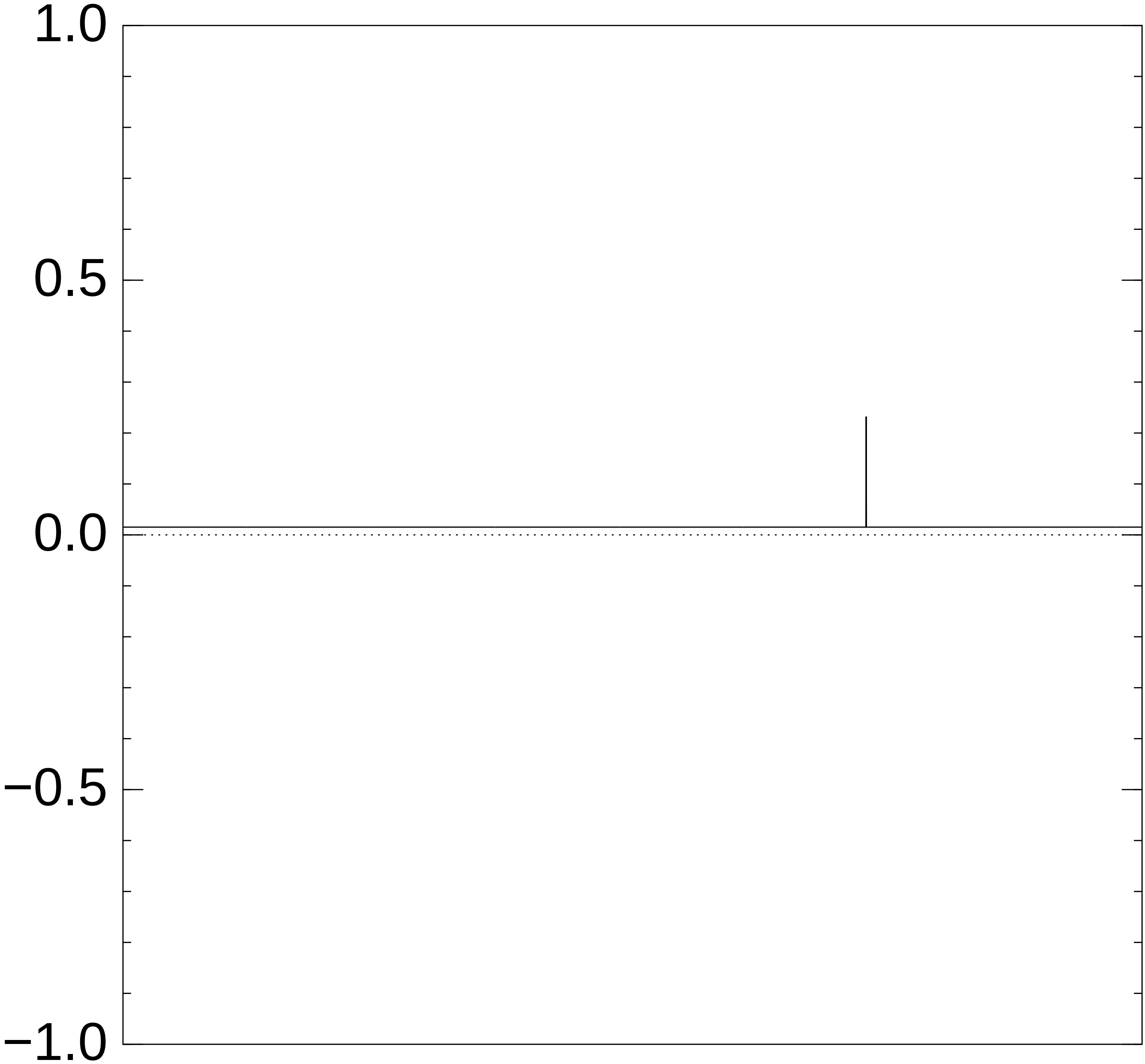
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $5 \times$ (Step 1 + Step 2):



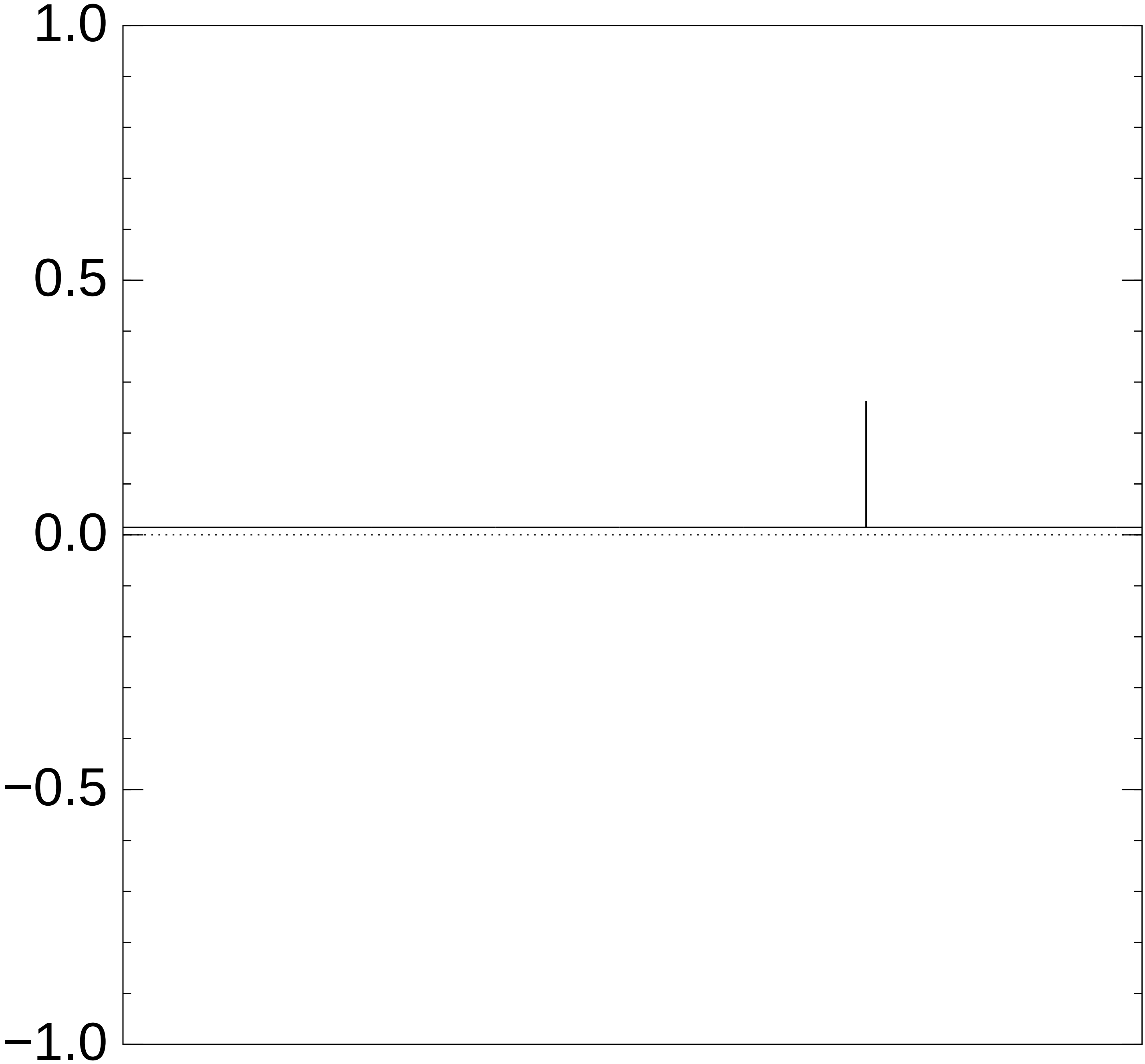
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $6 \times (\text{Step } 1 + \text{Step } 2)$:



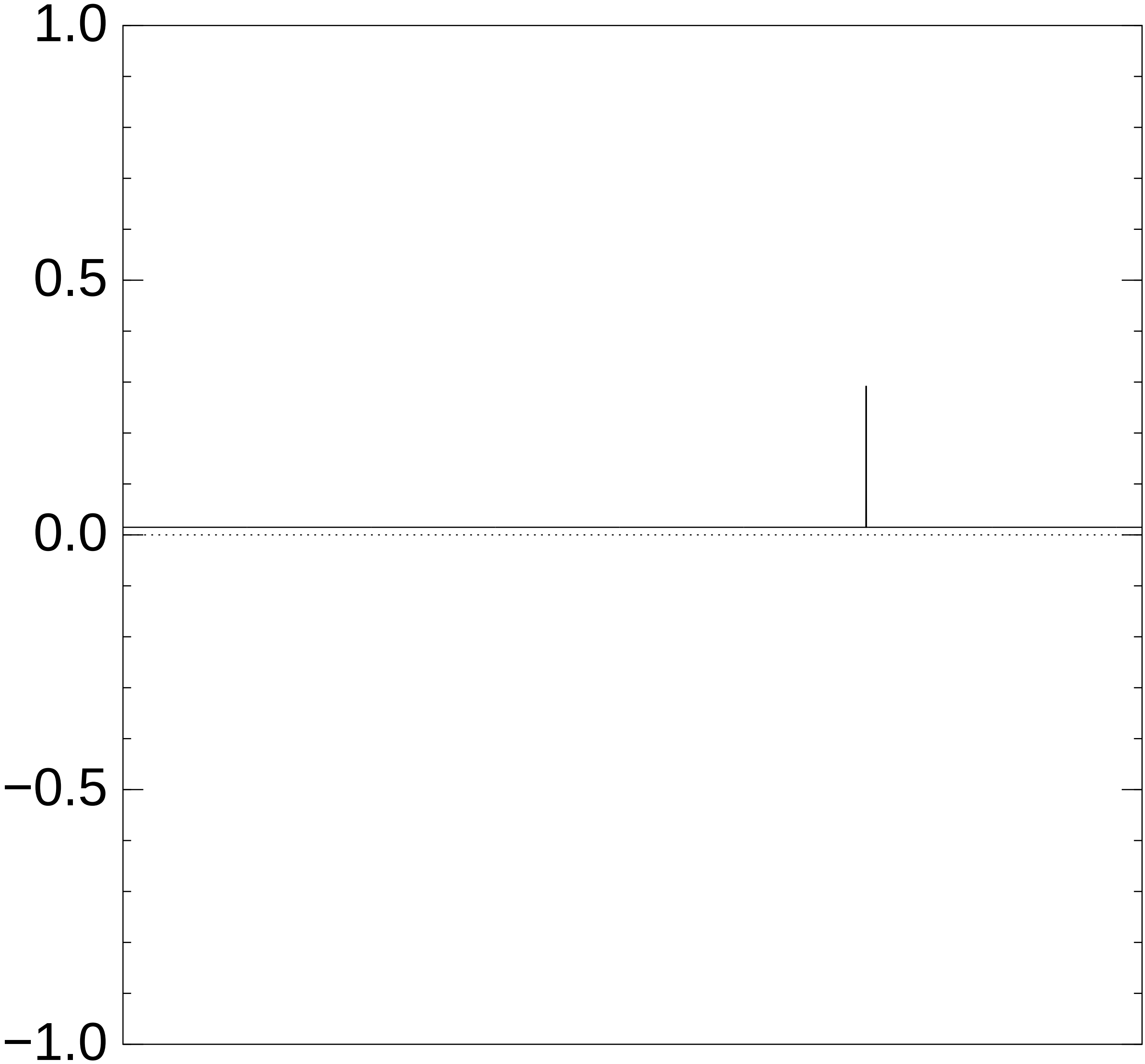
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $7 \times$ (Step 1 + Step 2):



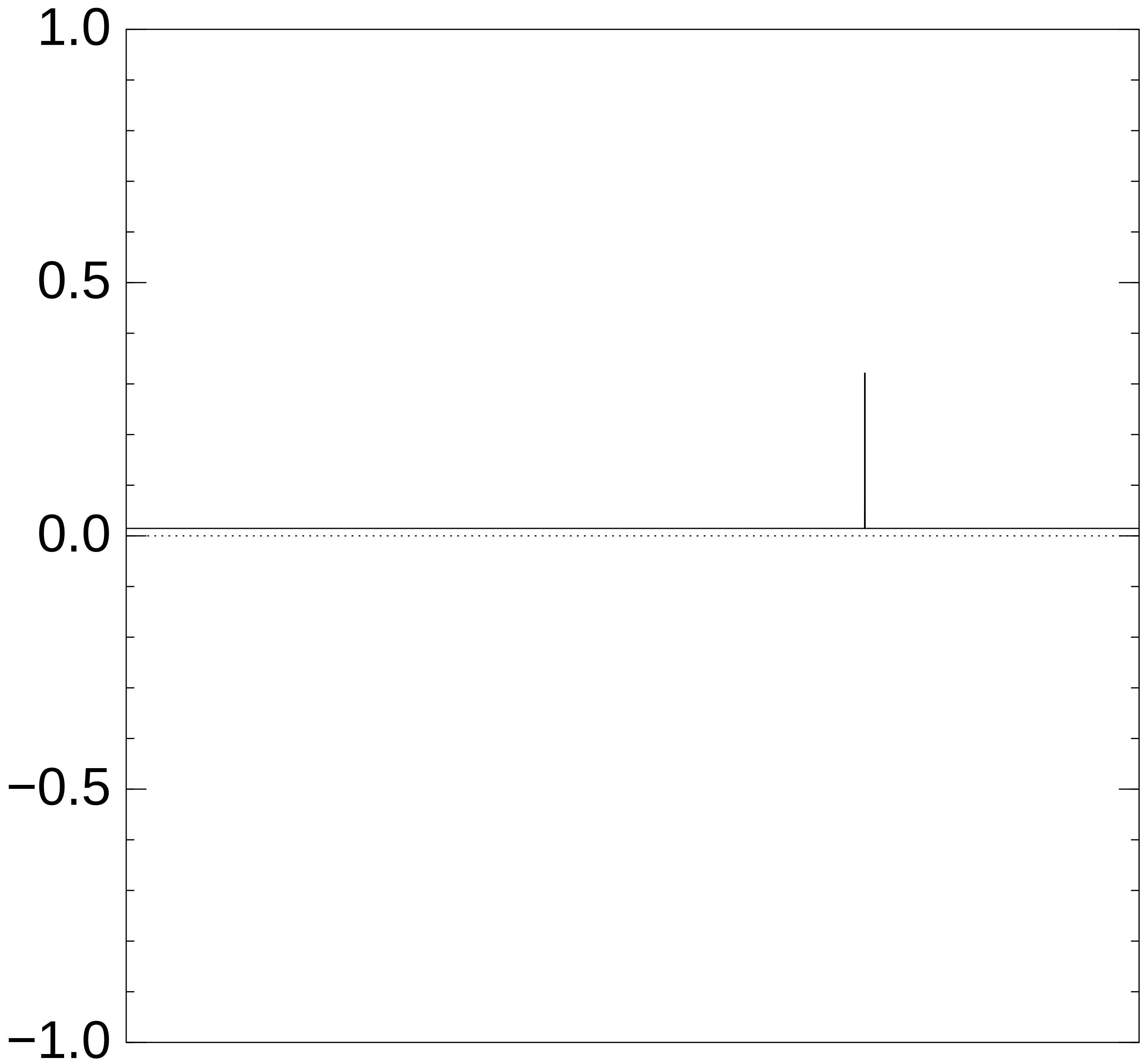
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $8 \times$ (Step 1 + Step 2):



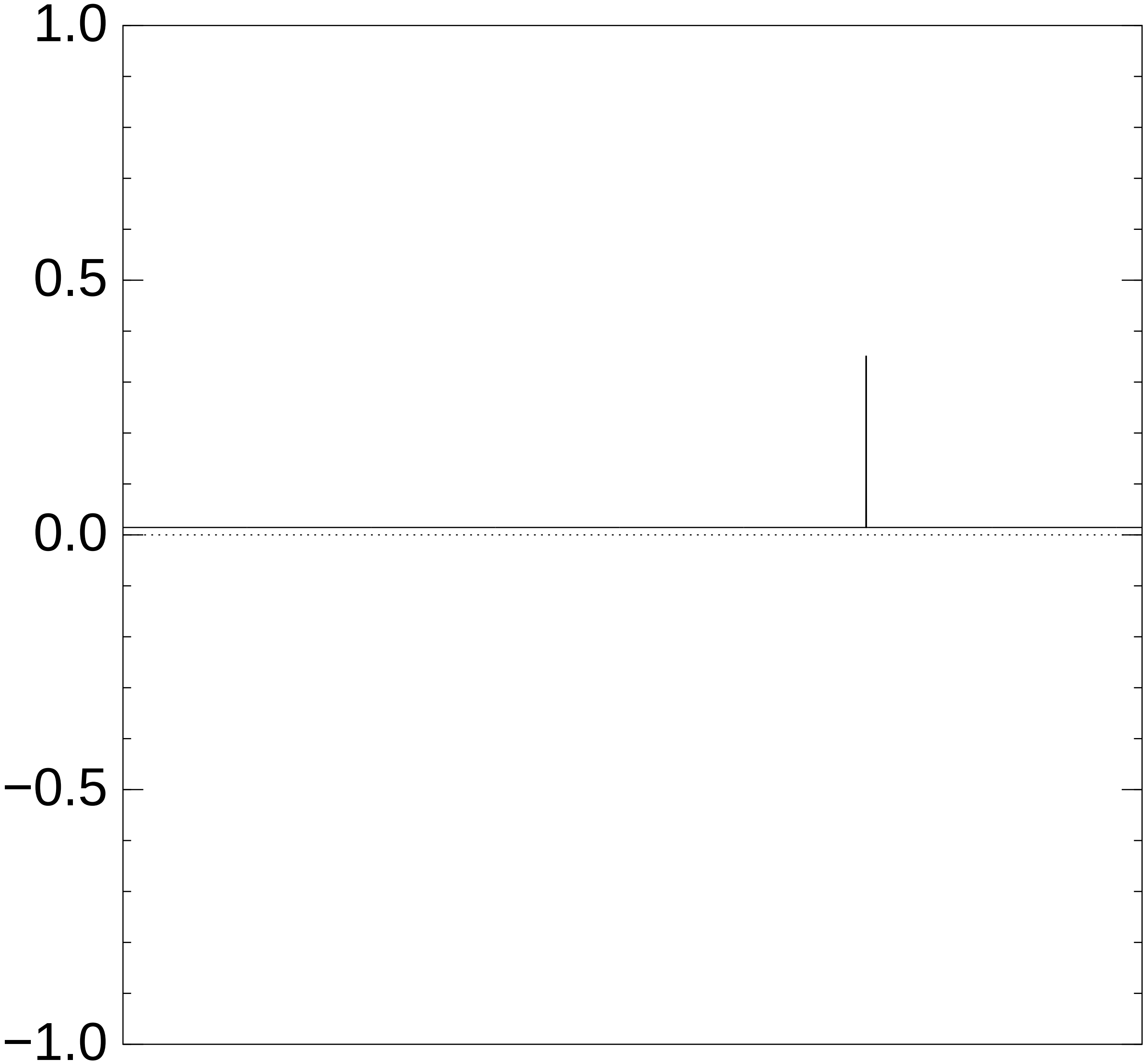
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $9 \times (\text{Step } 1 + \text{Step } 2)$:



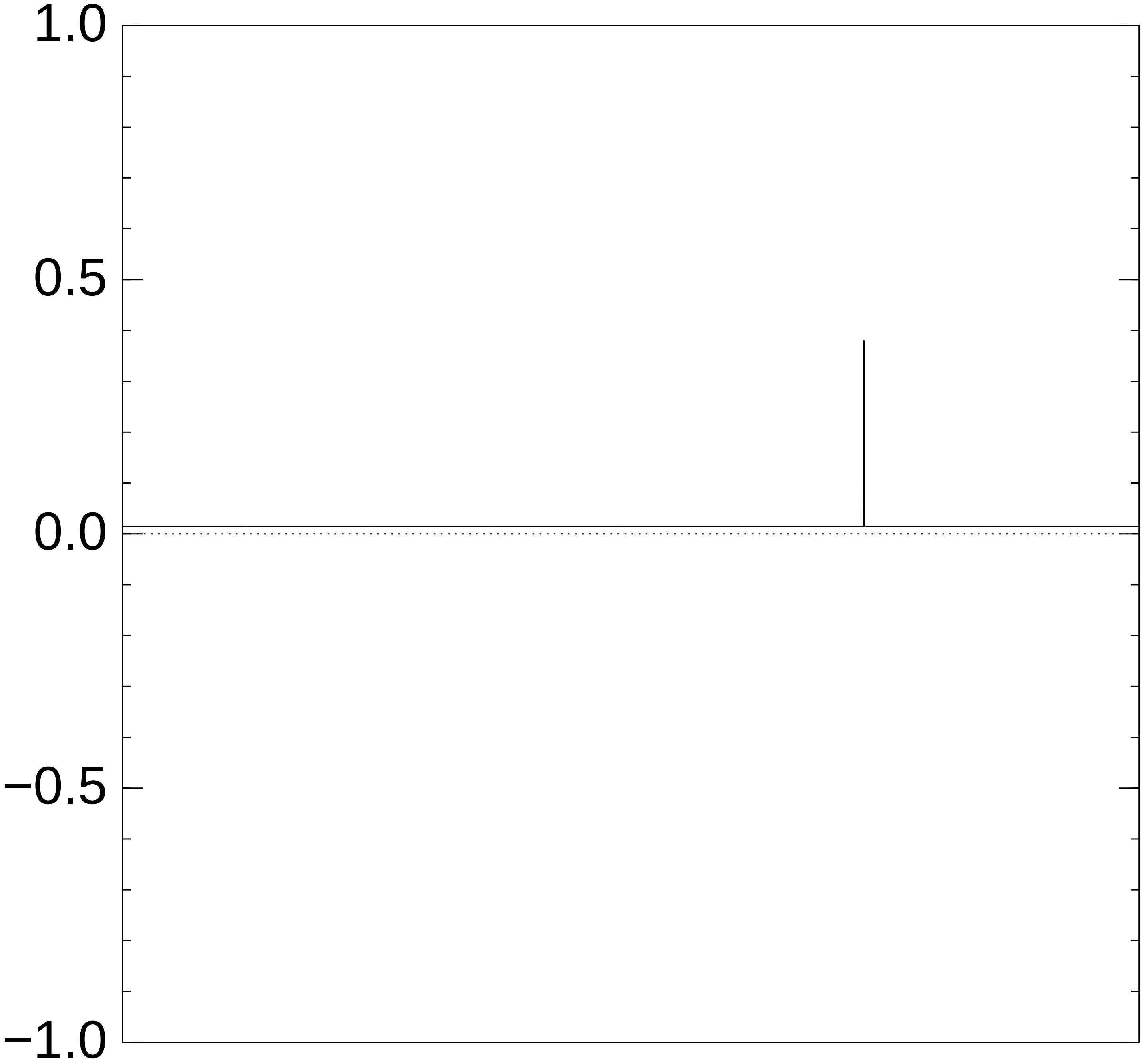
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $10 \times$ (Step $1 +$ Step 2):



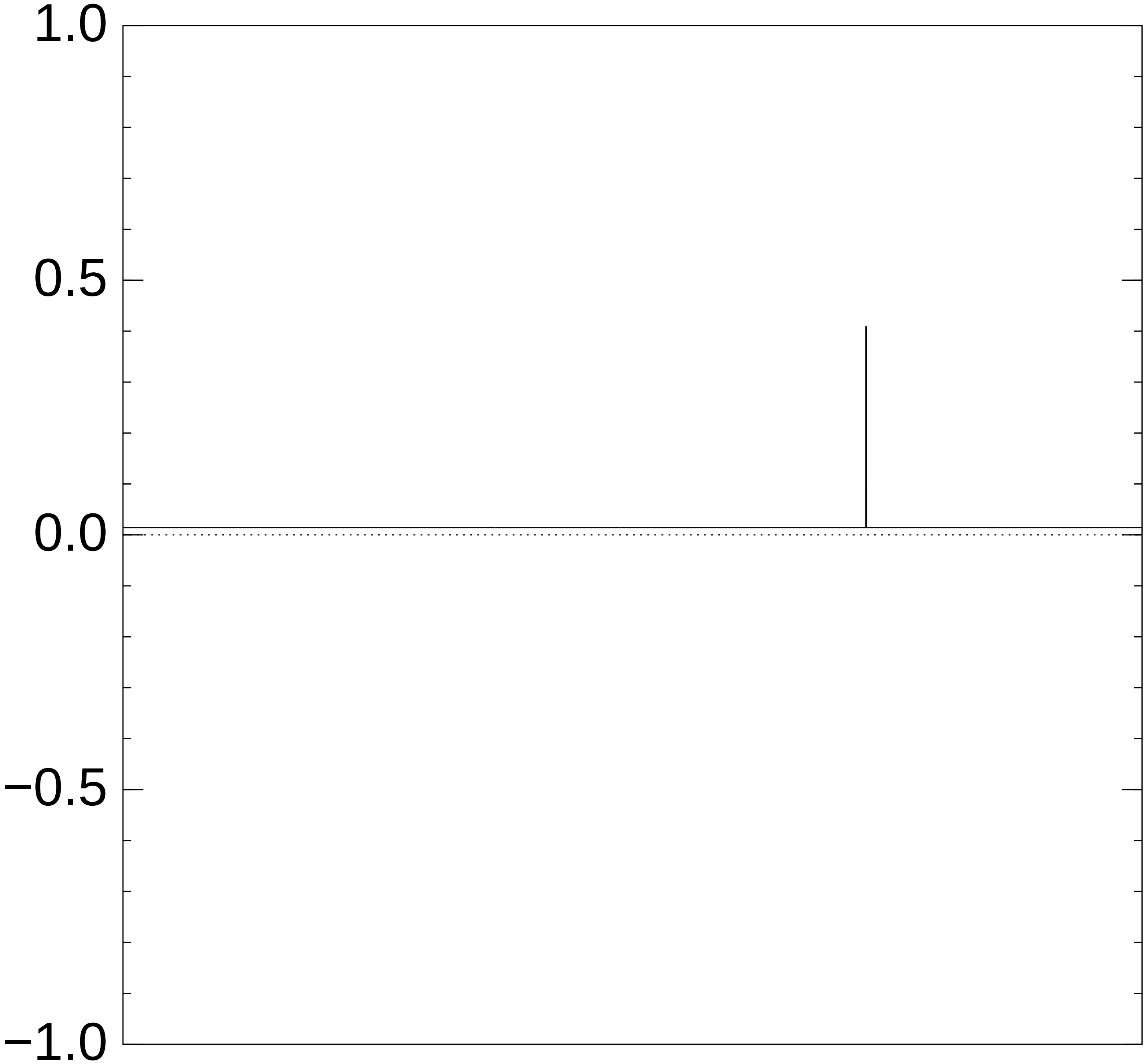
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $11 \times$ (Step 1 + Step 2):



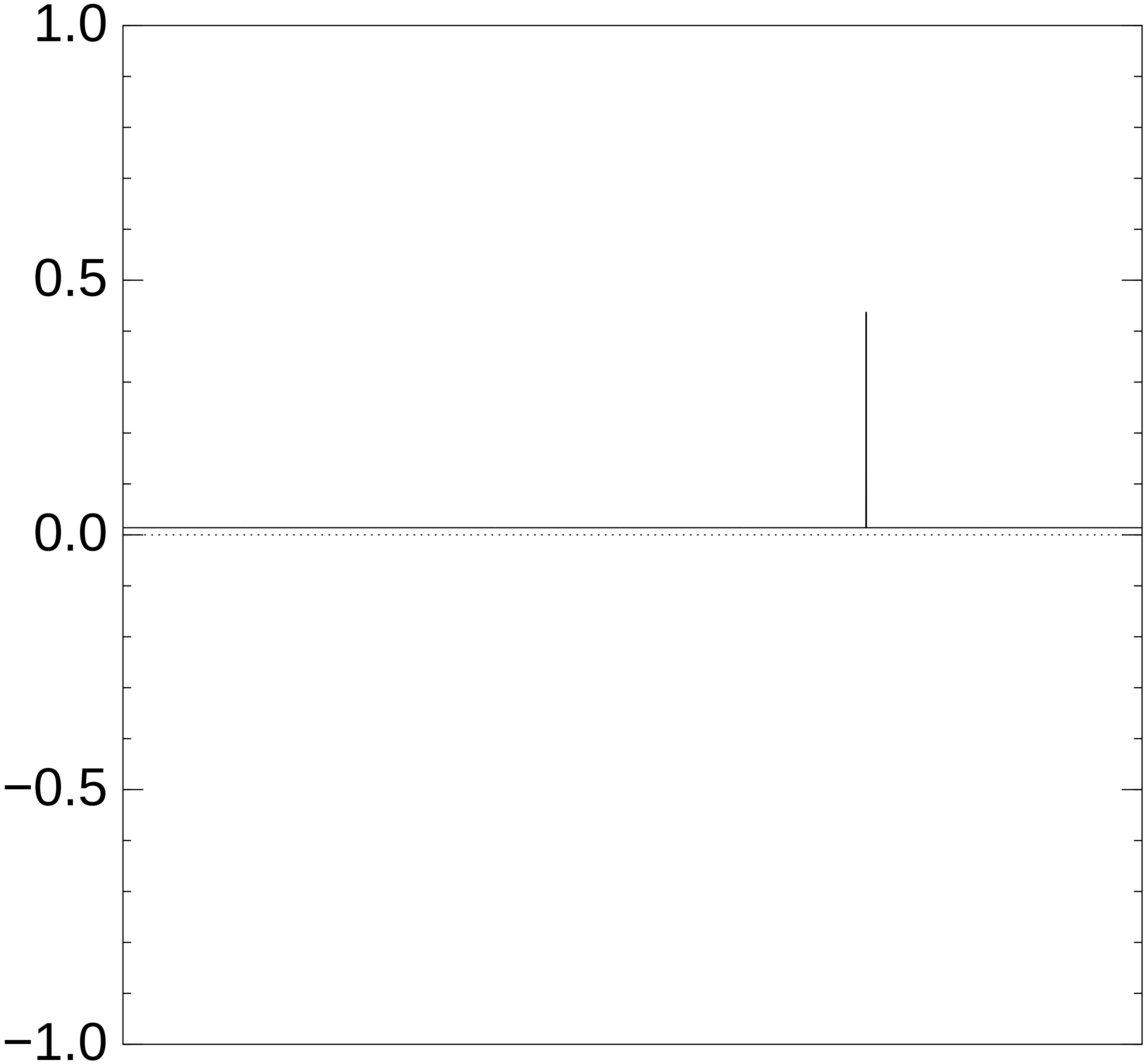
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $12 \times$ (Step 1 + Step 2):



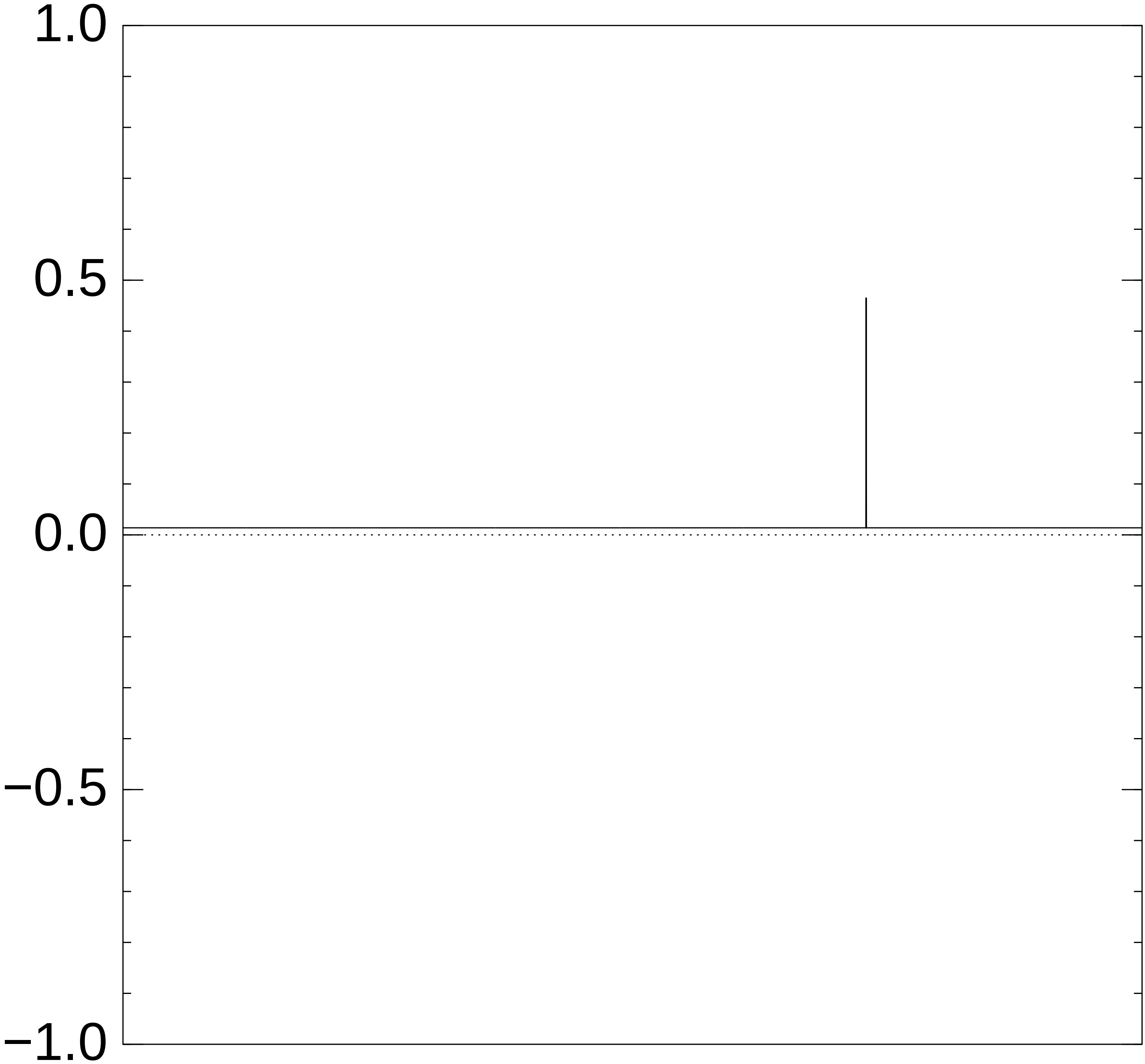
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $13 \times (\text{Step } 1 + \text{Step } 2)$:



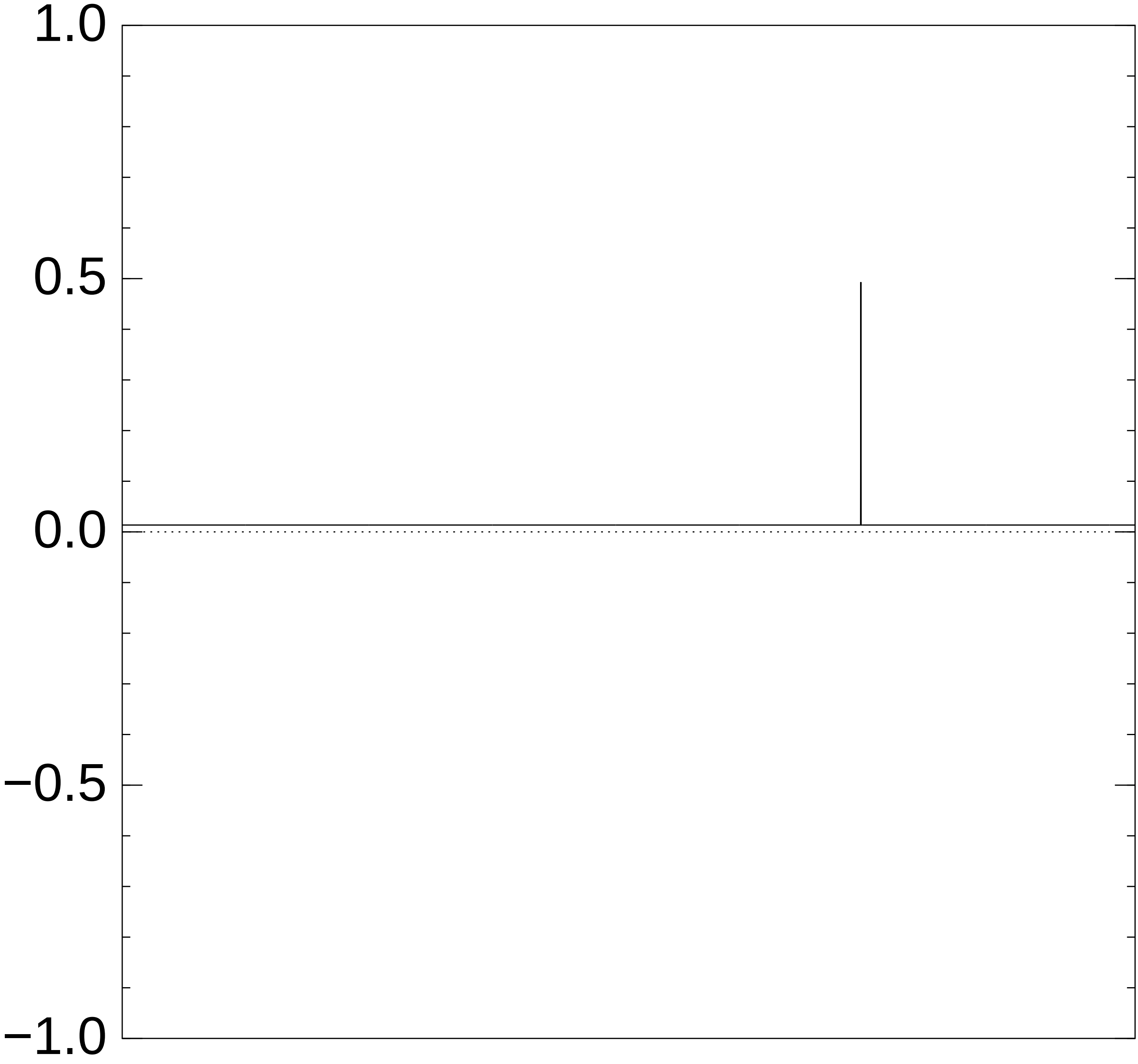
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $14 \times (\text{Step } 1 + \text{Step } 2)$:



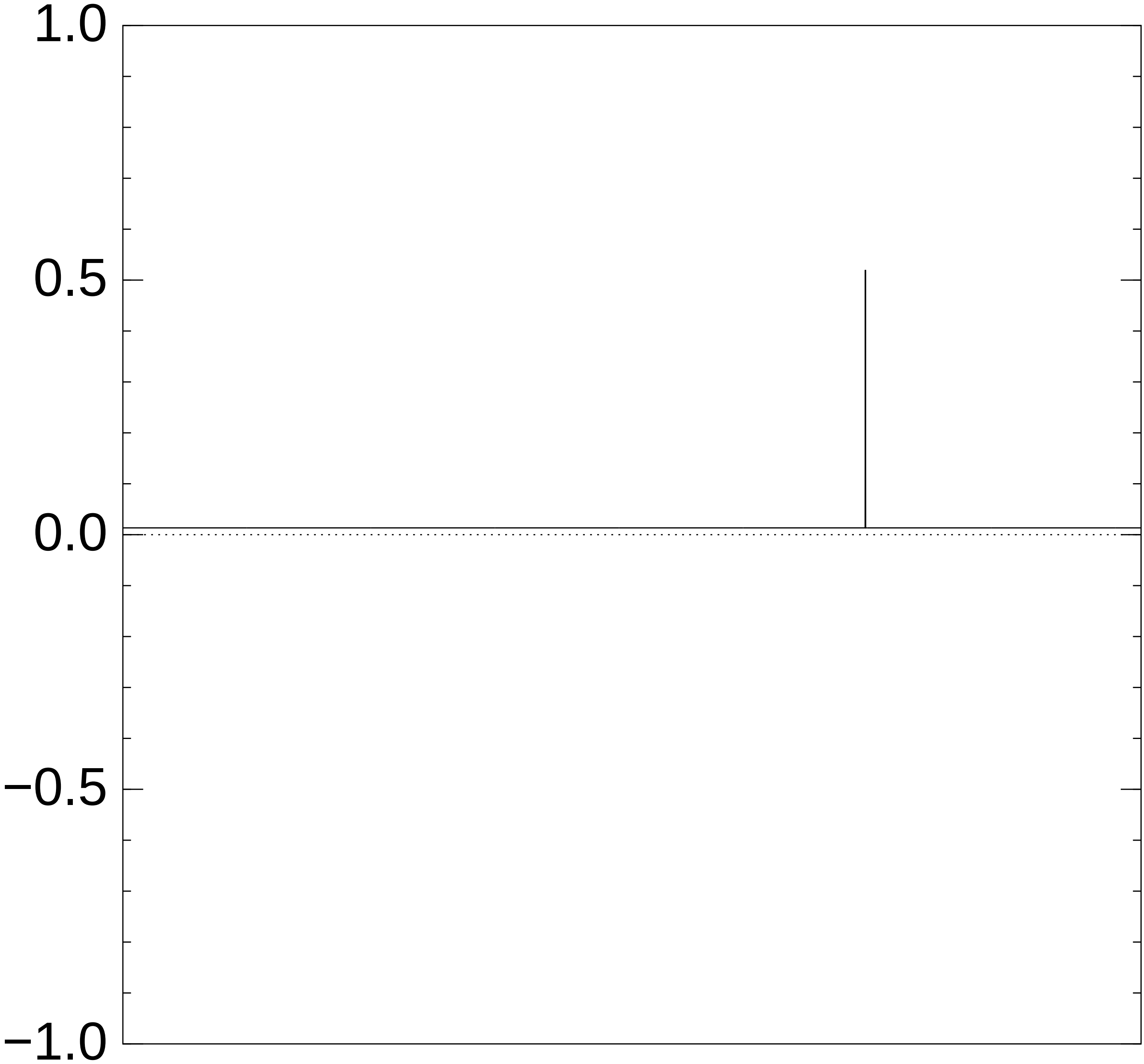
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $15 \times (\text{Step } 1 + \text{Step } 2)$:



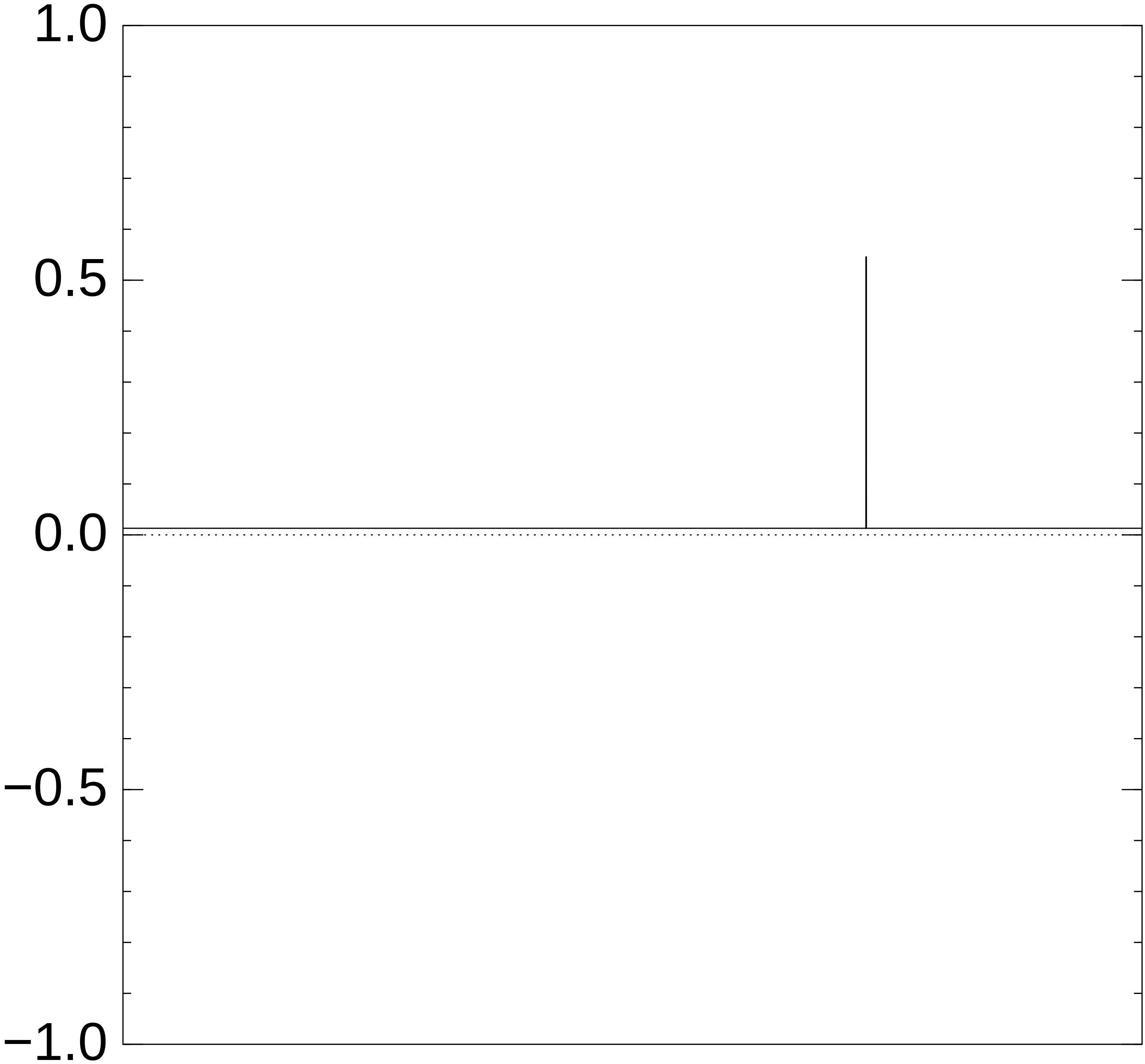
Normalized graph of $u \mapsto a_u$ for an example with $n = 12$ after $16 \times (\text{Step 1} + \text{Step 2})$:



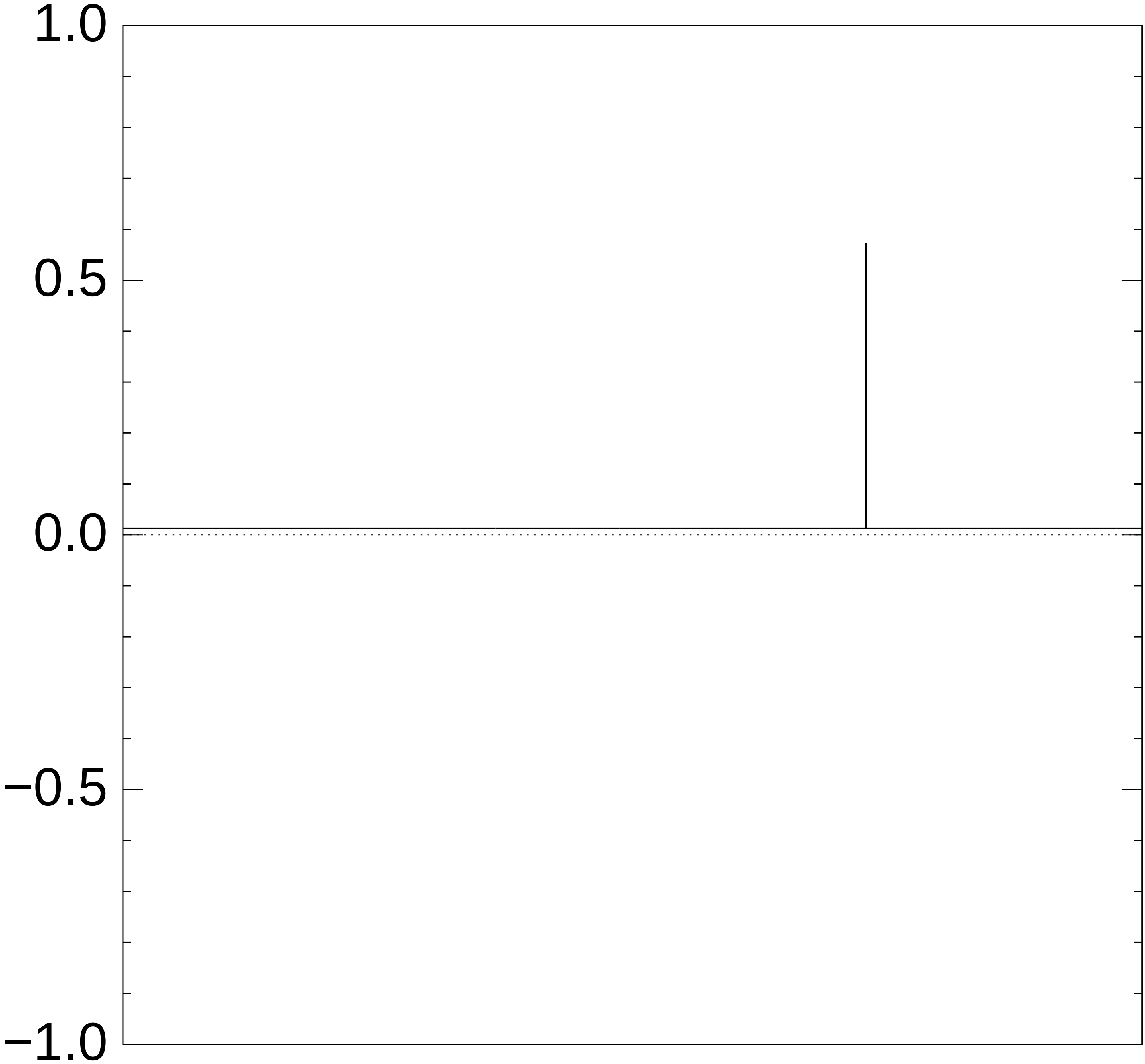
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $17 \times$ (Step $1 +$ Step 2):



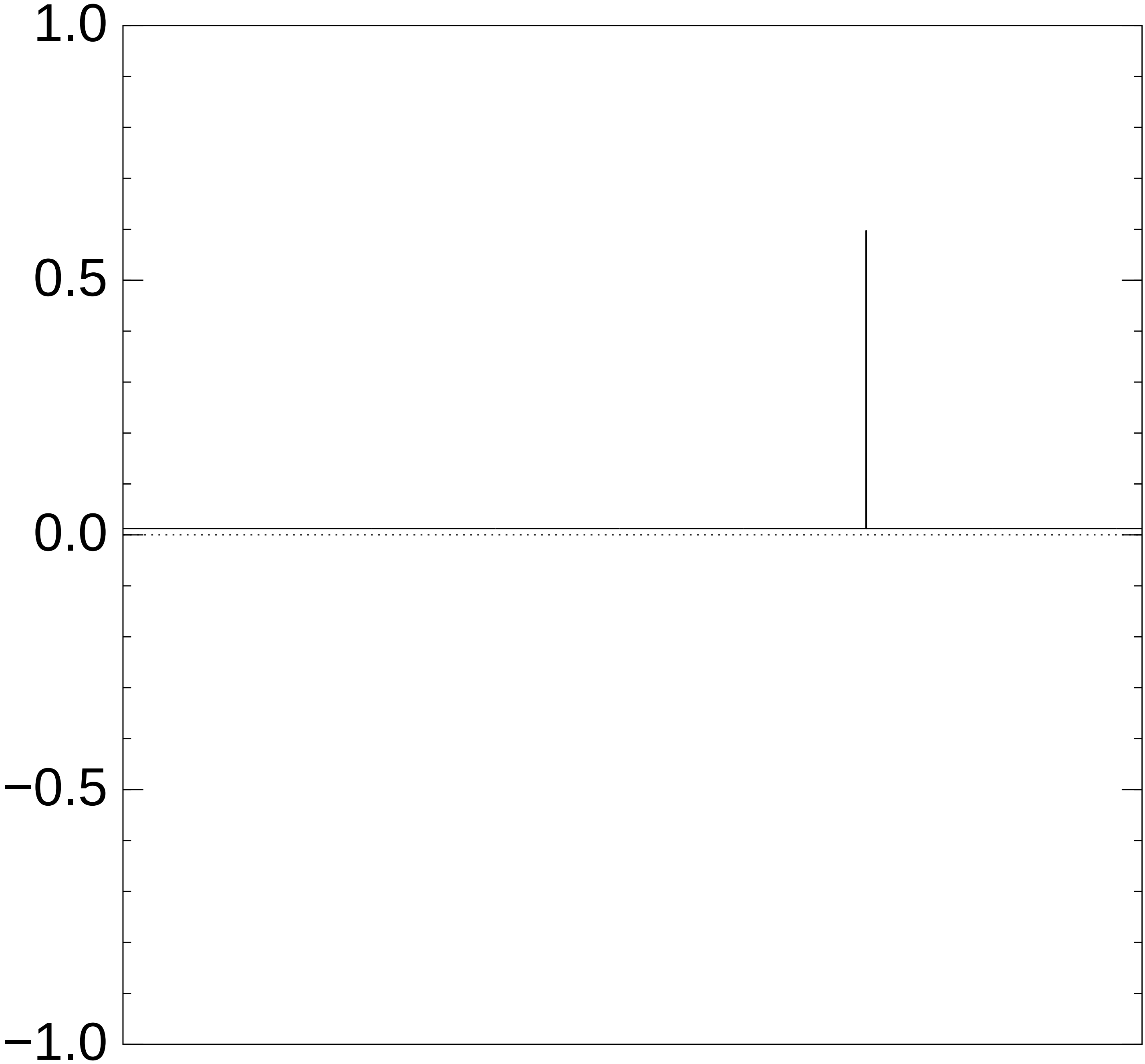
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $18 \times (\text{Step 1} + \text{Step 2})$:



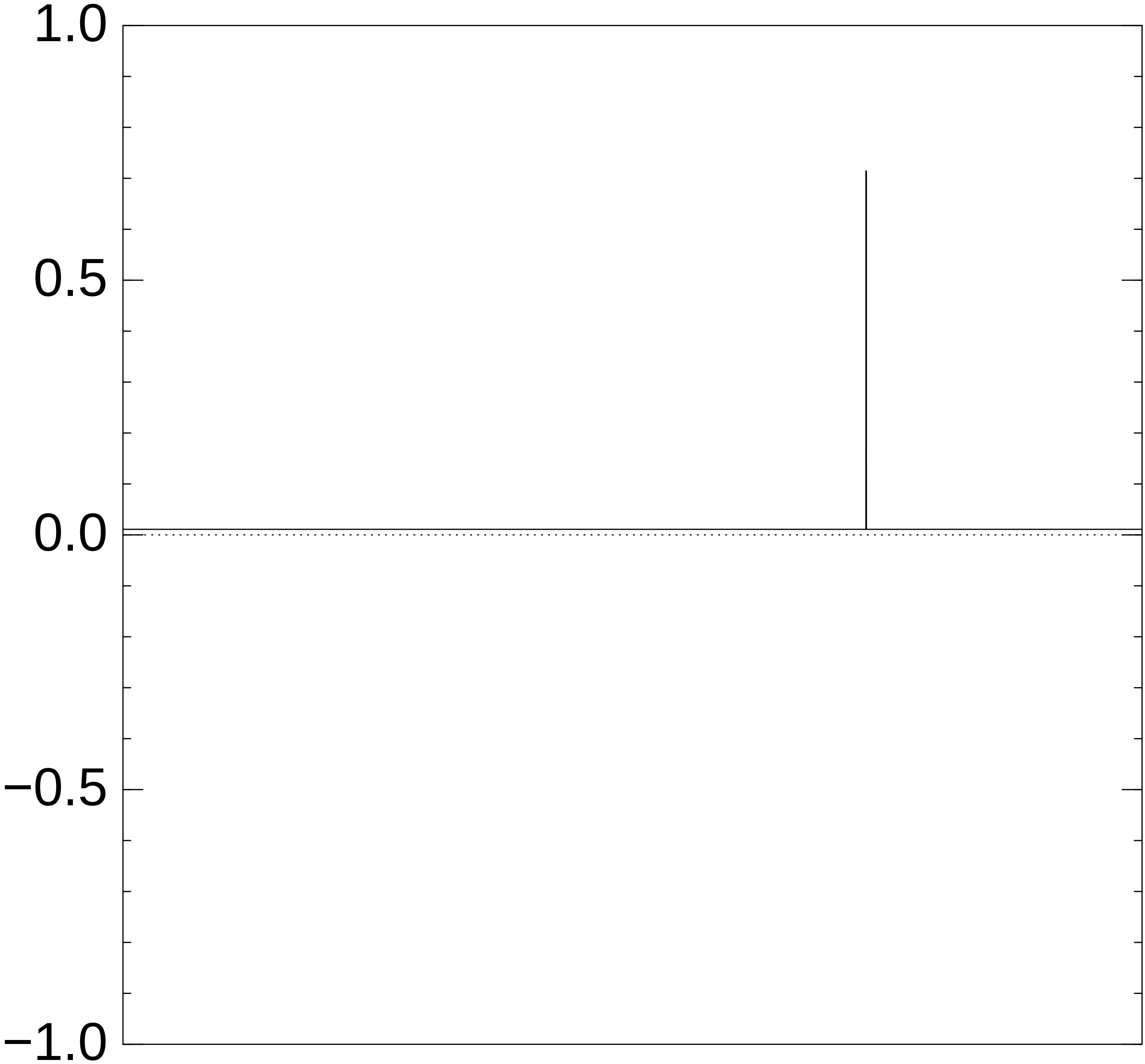
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $19 \times$ (Step $1 +$ Step 2):



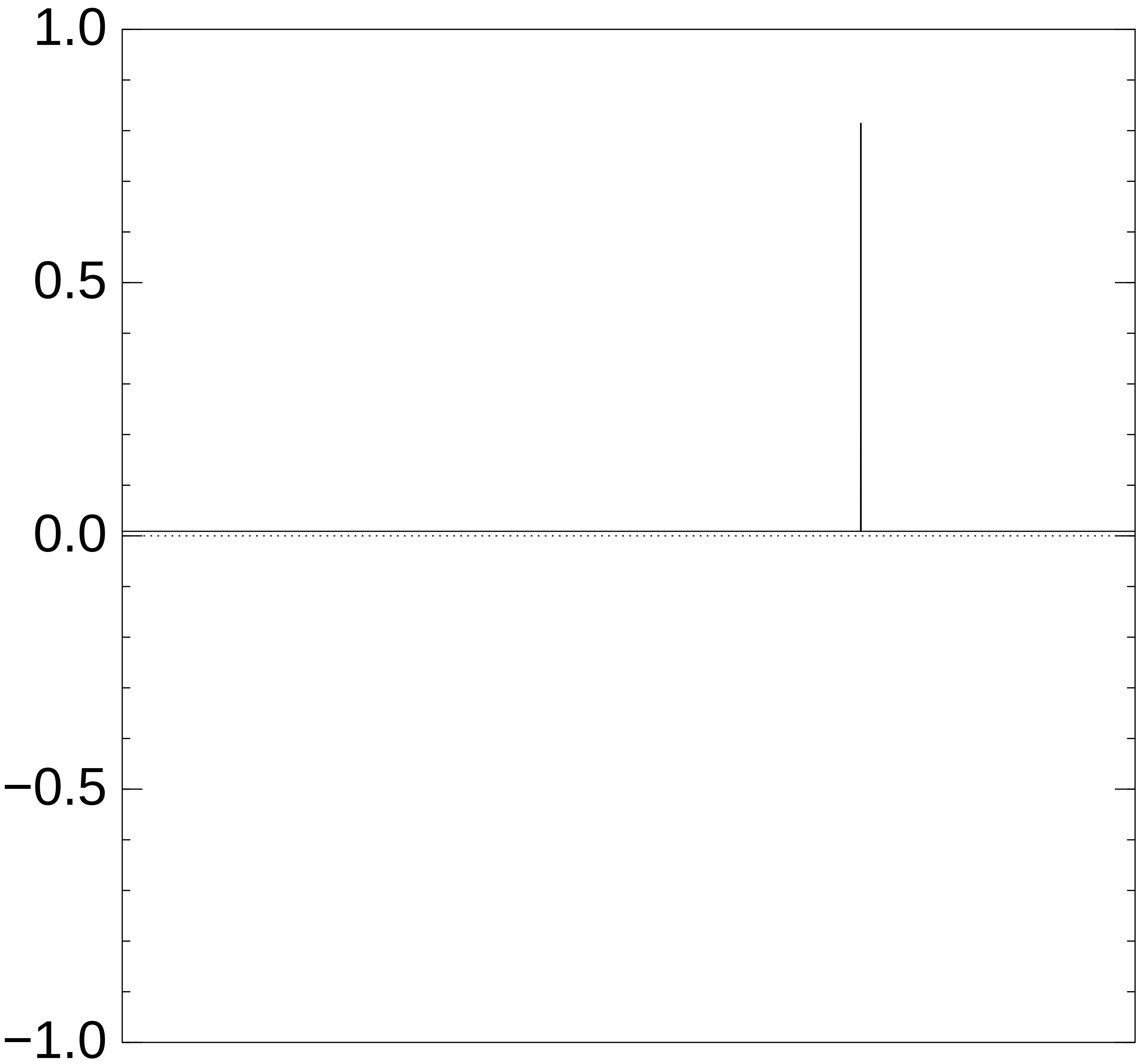
Normalized graph of $u \mapsto a_u$ for an example with $n = 12$ after $20 \times (\text{Step } 1 + \text{Step } 2)$:



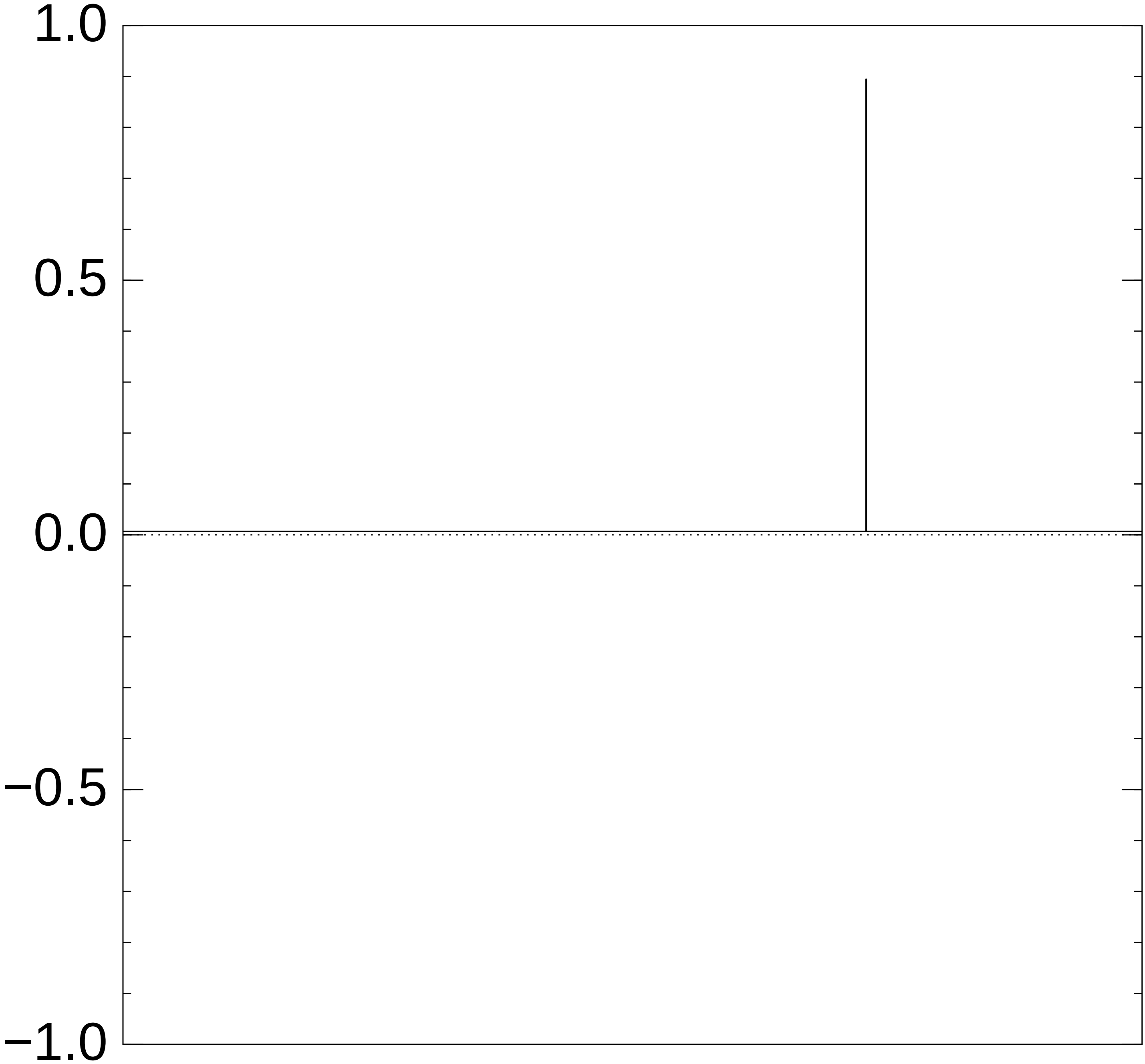
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $25 \times (\text{Step } 1 + \text{Step } 2)$:



Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $30 \times$ (Step 1 + Step 2):

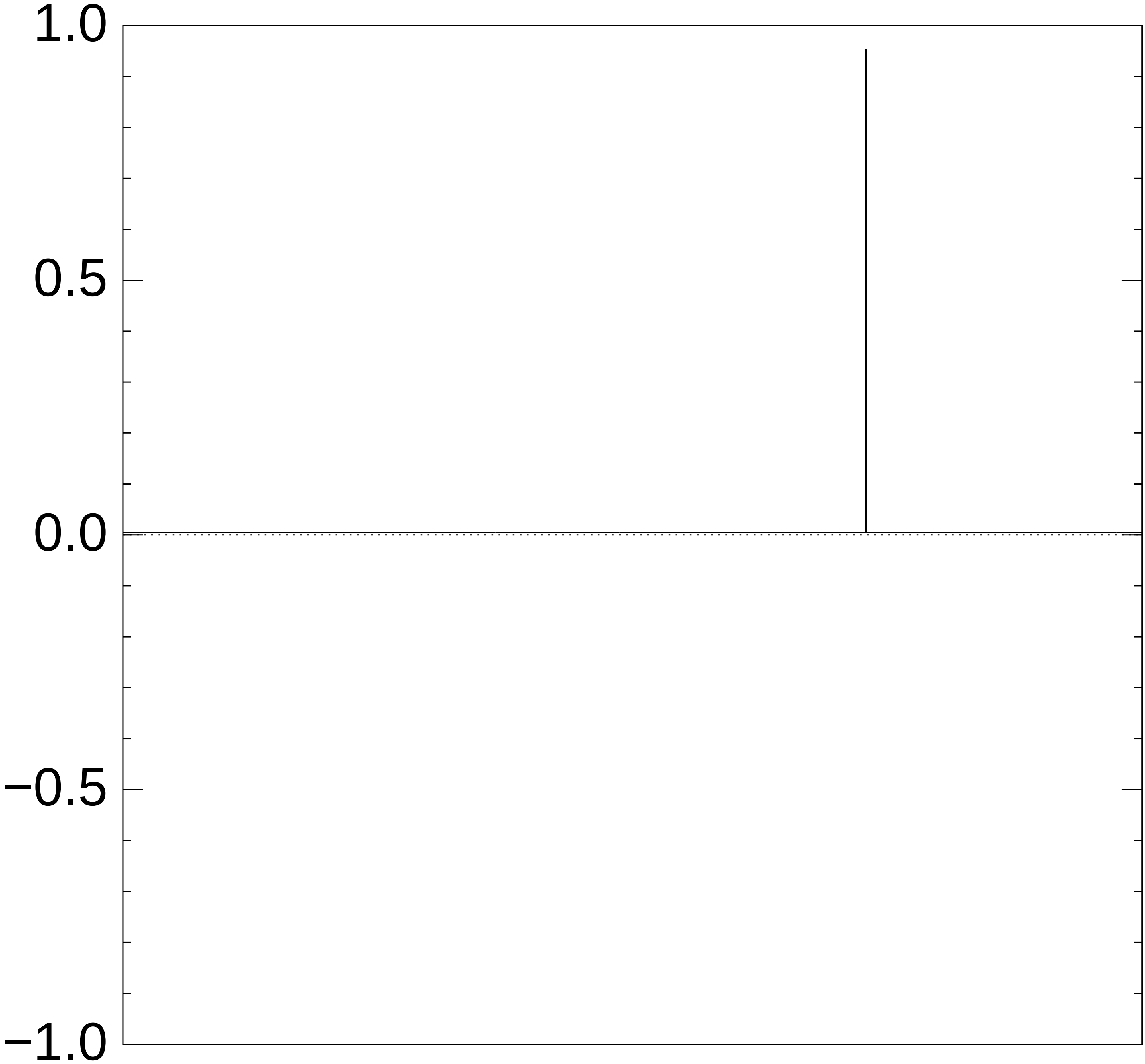


Normalized graph of $u \mapsto a_u$ for an example with $n = 12$ after $35 \times$ (Step 1 + Step 2):

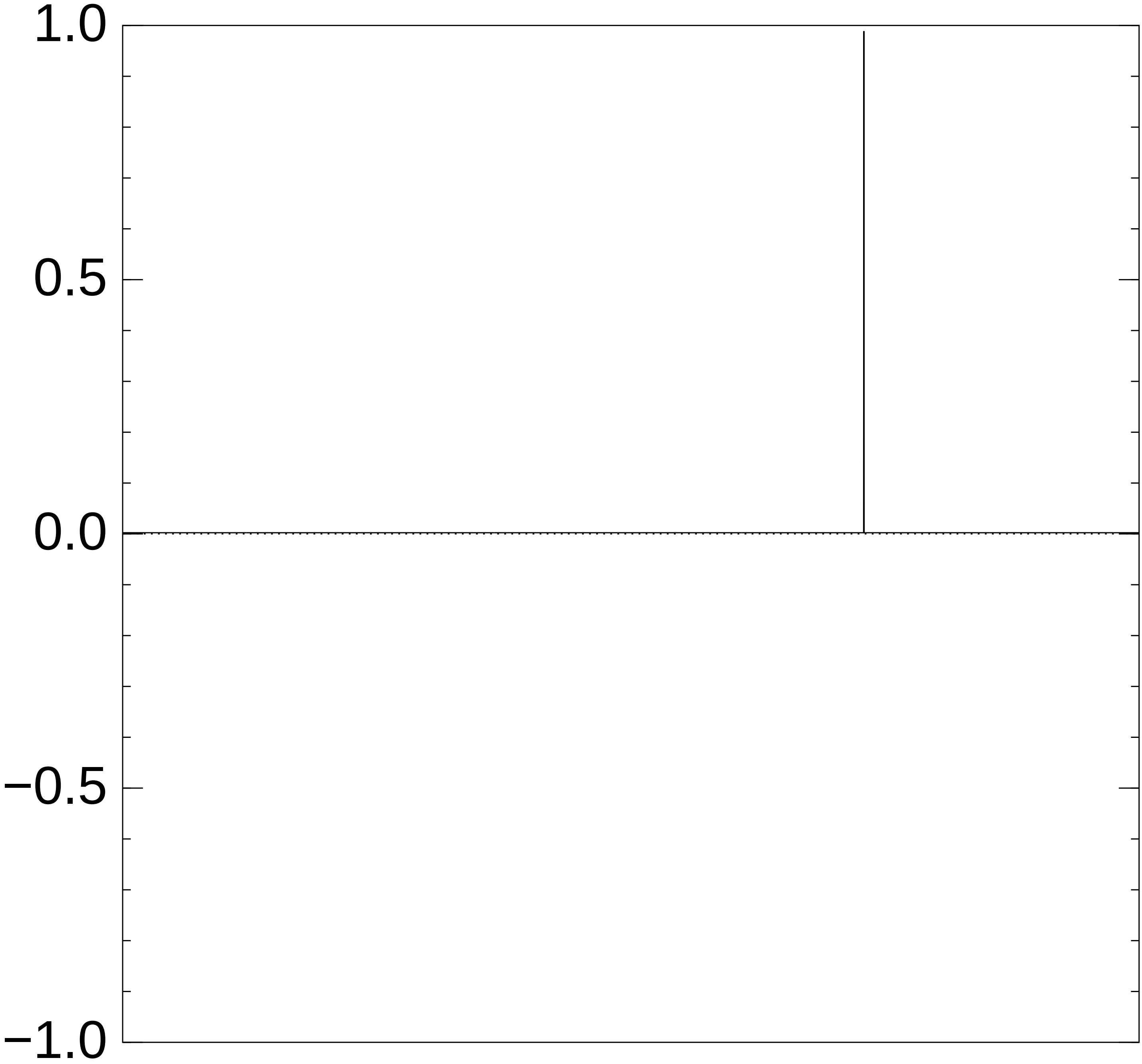


Good moment to stop, measure.

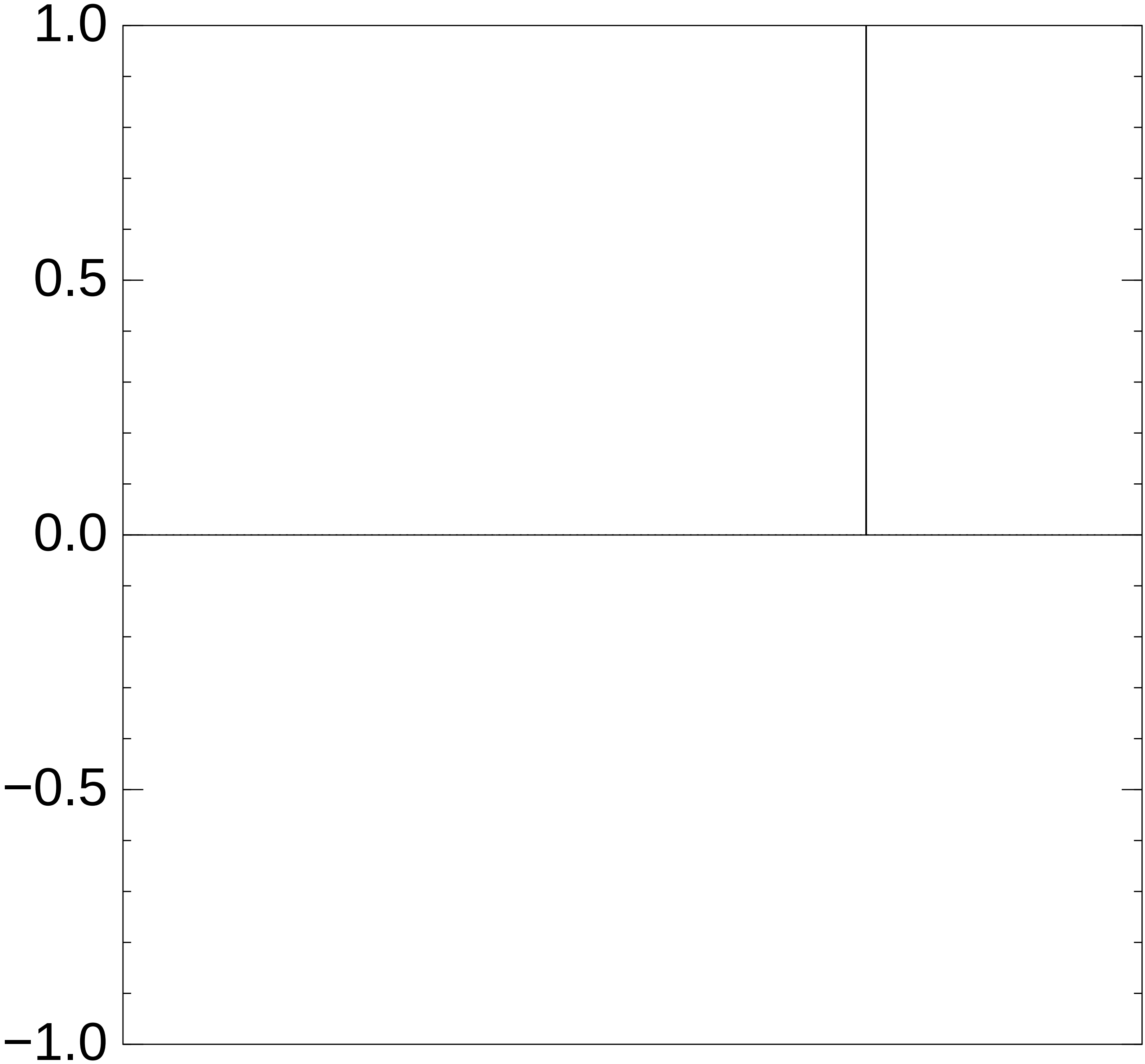
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $40 \times$ (Step $1 +$ Step 2):



Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $45 \times (\text{Step } 1 + \text{Step } 2)$:

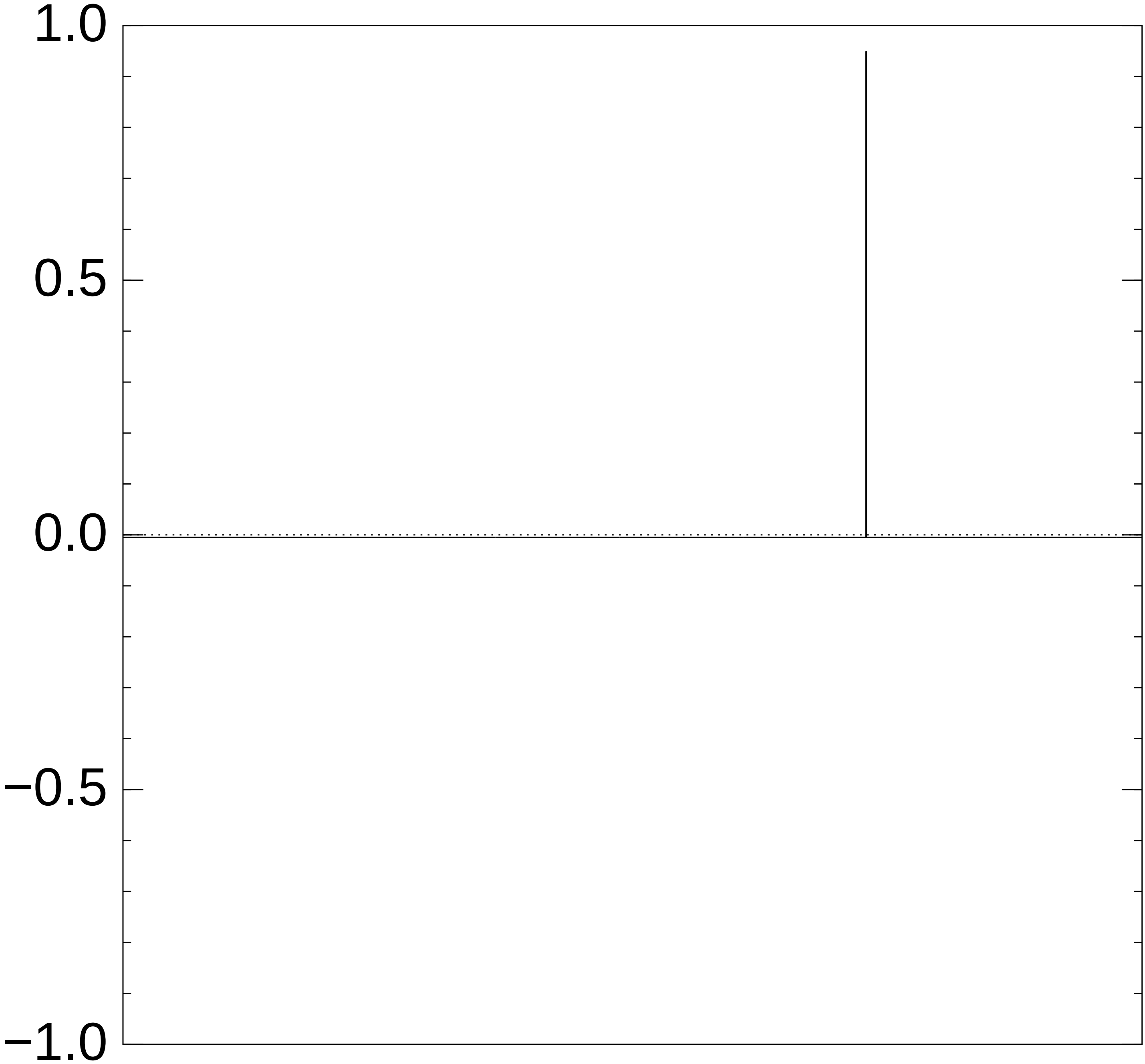


Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $50 \times$ (Step $1 +$ Step 2):

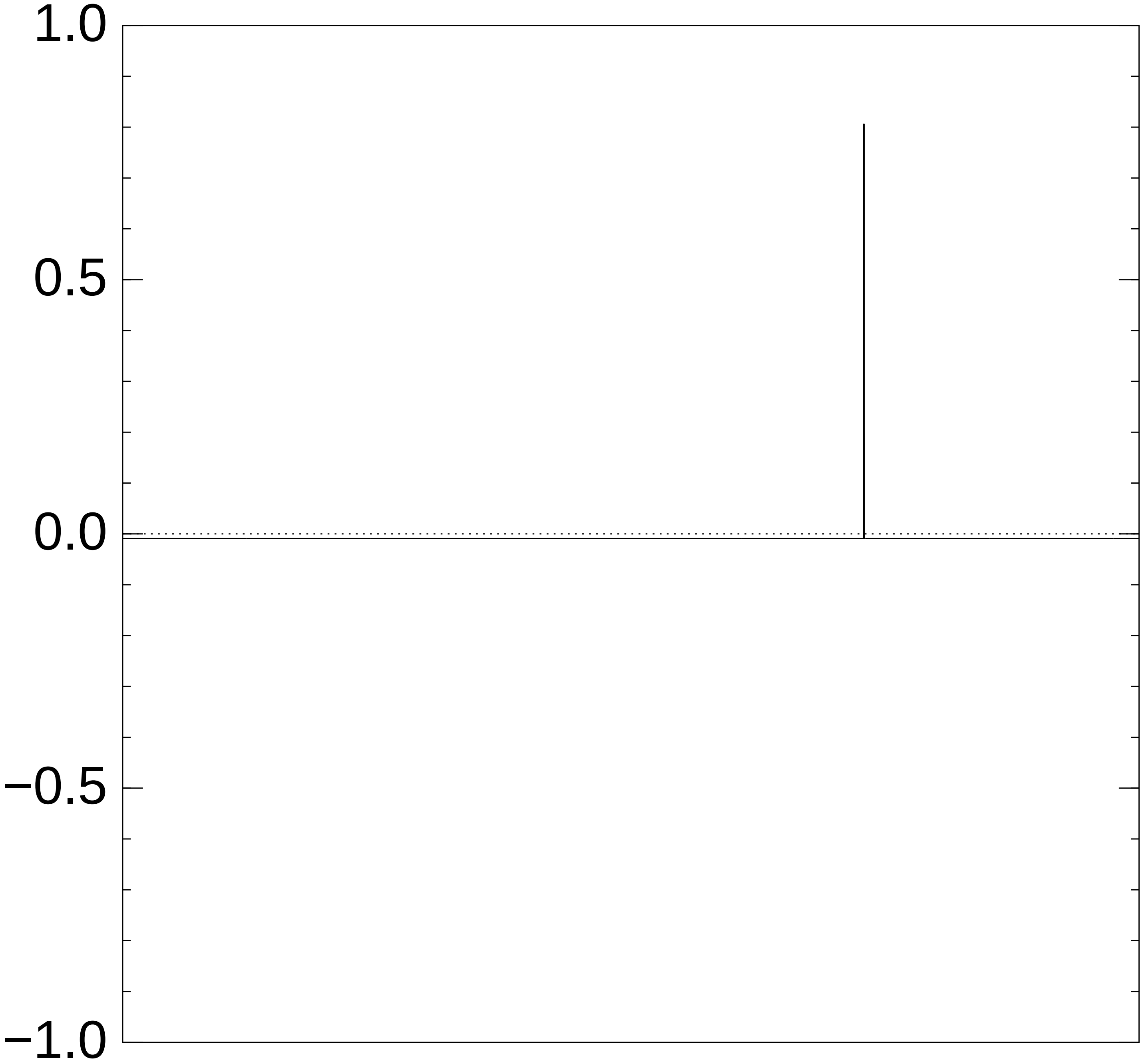


Traditional stopping point.

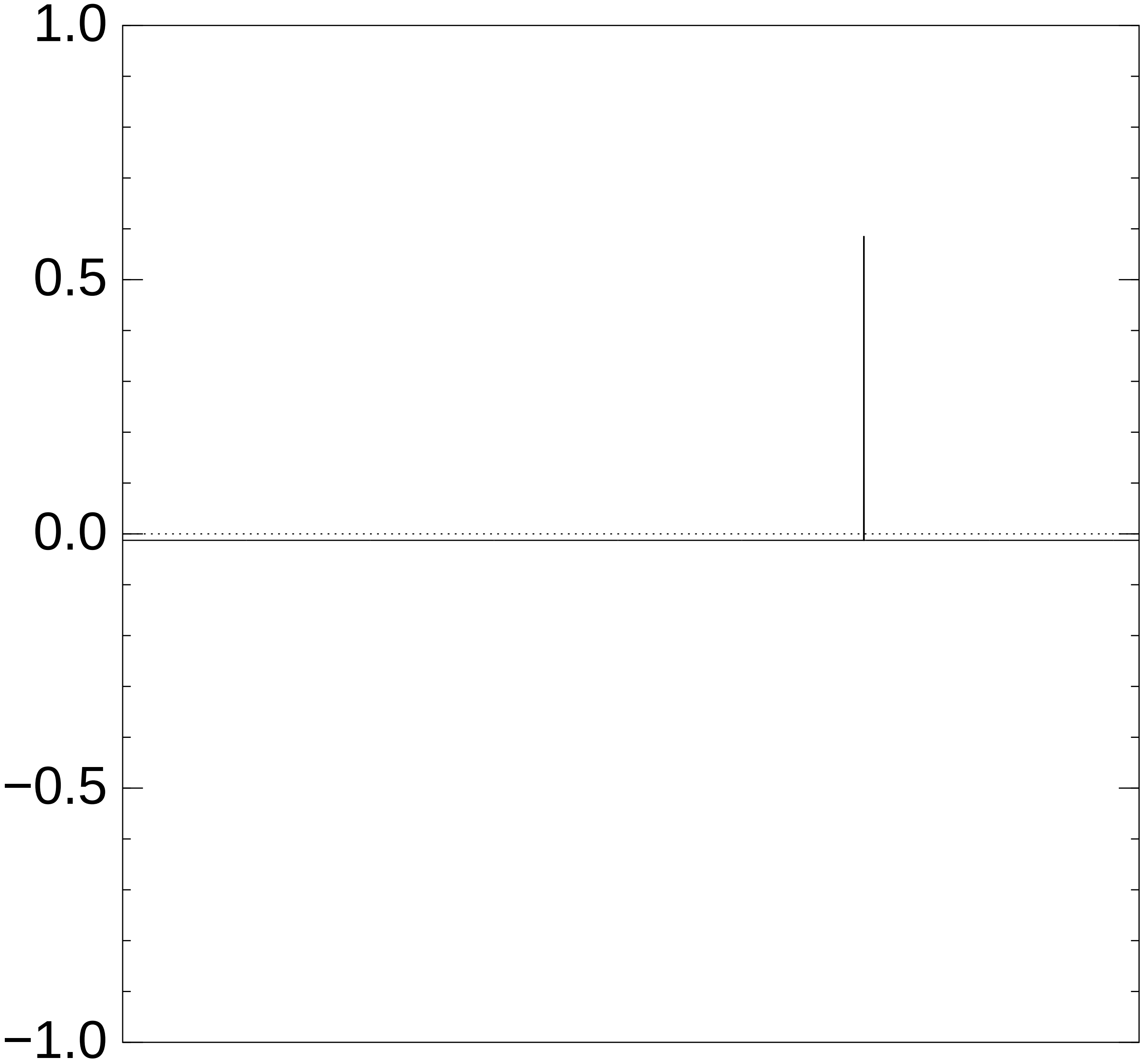
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $60 \times (\text{Step } 1 + \text{Step } 2)$:



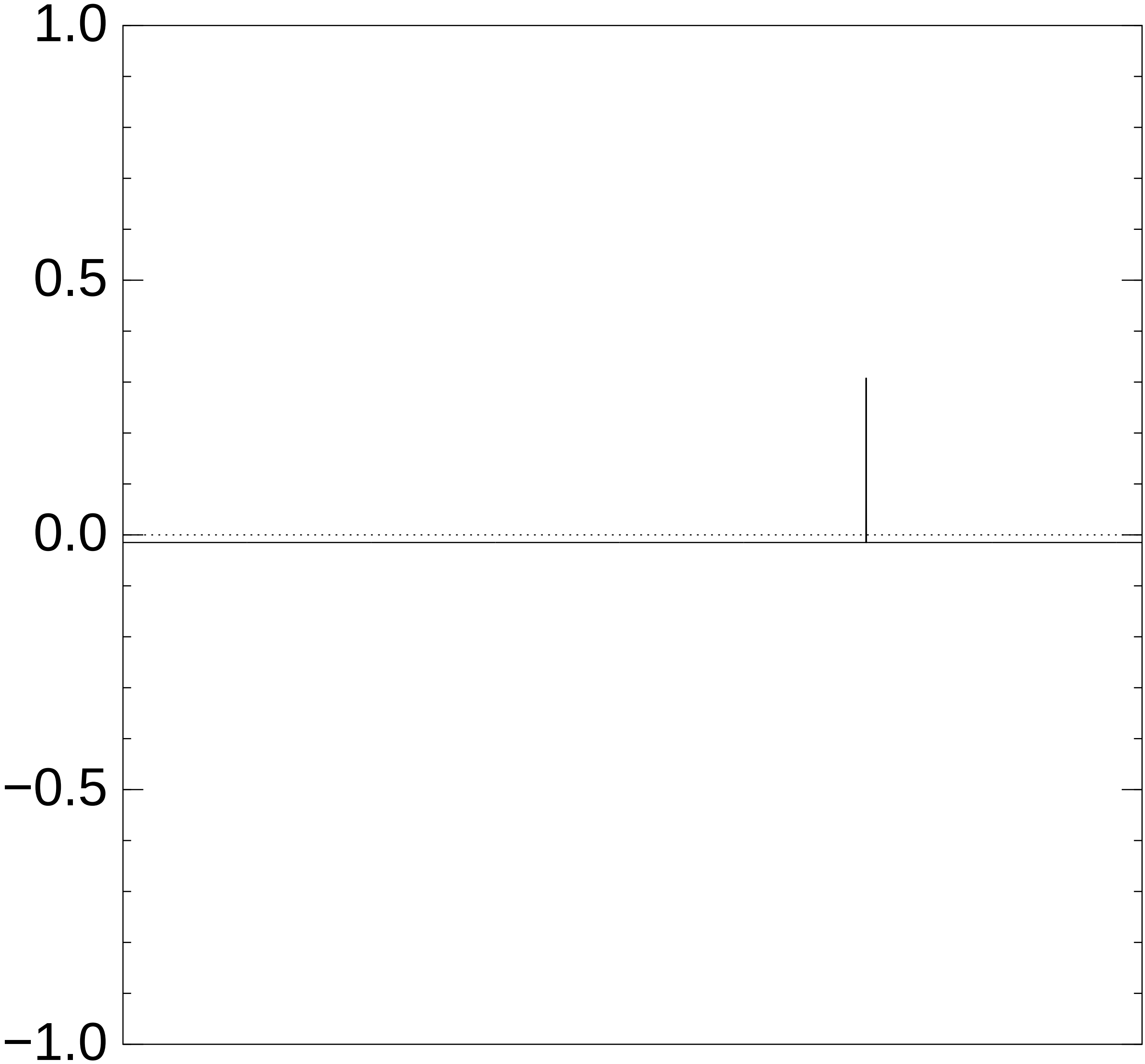
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $70 \times$ (Step $1 +$ Step 2):



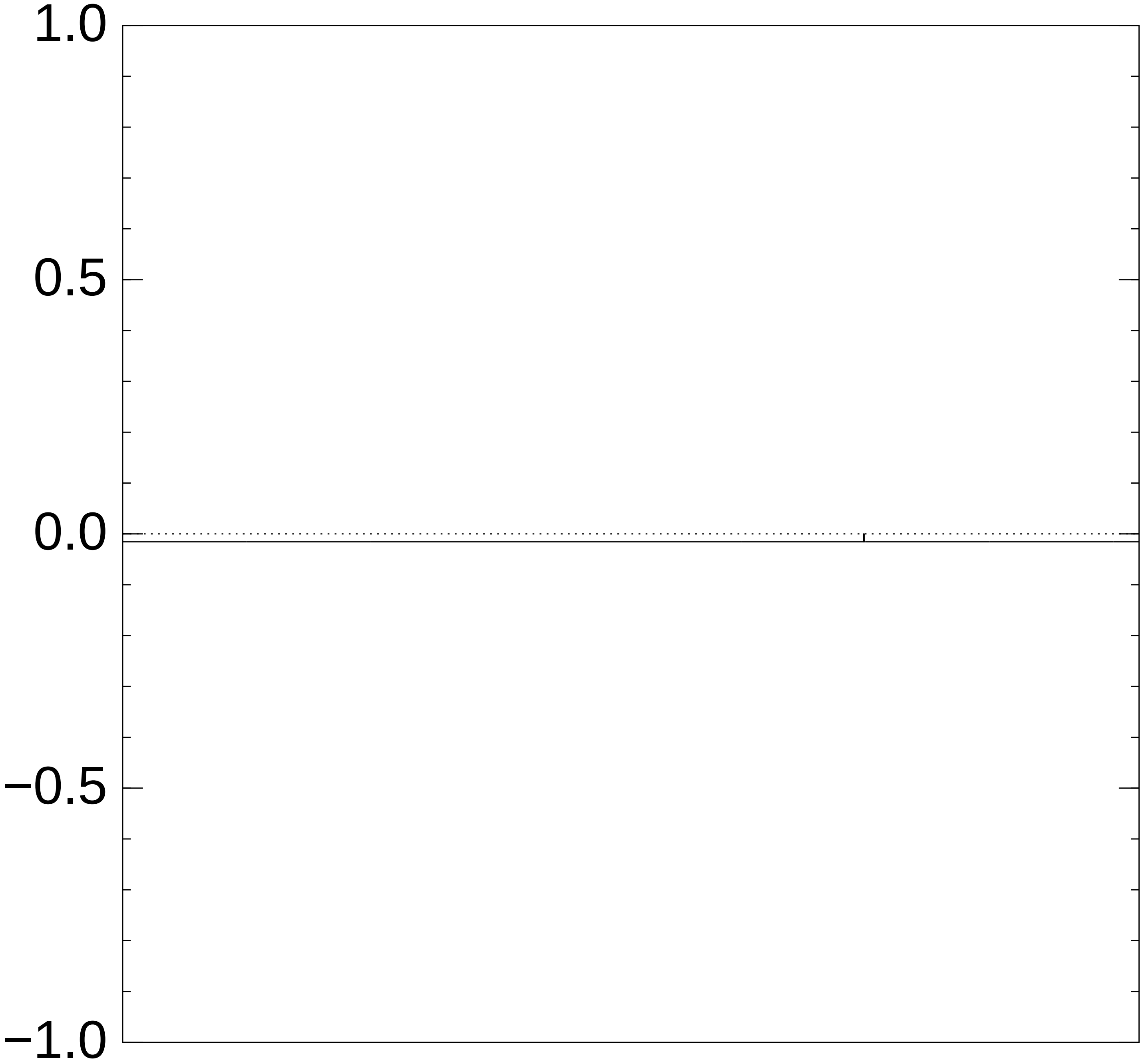
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $80 \times$ (Step $1 +$ Step 2):



Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $90 \times$ (Step $1 +$ Step 2):



Normalized graph of $u \mapsto a_u$ for an example with $n = 12$ after $100 \times (\text{Step } 1 + \text{Step } 2)$:



Very bad stopping point.

$u \mapsto a_u$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) a_u for roots u;
(2) a_u for non-roots u.

$u \mapsto a_u$ is completely described
by a vector of two numbers
(with fixed multiplicities):
(1) a_u for roots u;
(2) a_u for non-roots u.

Step 1 + Step 2
act linearly on this vector.

$u \mapsto a_u$ is completely described
by a vector of two numbers
(with fixed multiplicities):
(1) a_u for roots u;
(2) a_u for non-roots u.

Step 1 + Step 2
act linearly on this vector.

Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover's algorithm.
$\Rightarrow$ Probability is $\approx$1
after $\approx(\pi/4)2^{0.5n}$ iterations.

Many more applications

Shor generalizations:
e.g., poly-time attack breaking "cyclotomic" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices".

Grover generalizations:
e.g., fastest subset-sum attacks use "quantum walks".

Not just Shor and Grover:
e.g., subexponential-time CRS/CSIDH isogeny attack uses "Kuperberg's algorithm".