Quantum cryptanalysis

Daniel J. Bernstein

Main question in
quantum cryptanalysis:
What is the most efficient
quantum algorithm
to attack this cryptosystem?

(For comparison, main question
in non-quantum cryptanalysis:
What is the most efficient
non-quantum algorithm
to attack this cryptosystem?)

"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?

(Something to think about:
Do we really know the answer
for non-quantum computers?)

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 today.

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 today.

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 today.

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 1980 Manin (English version:
paper appendix), 1982 Feynman.

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 1980 Manin (English version:
paper appendix), 1982 Feynman.

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.

State of a non-quantum computer

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

State of a non-quantum computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

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

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

State of a non-quantum 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.: $(0, 1, 1)$.

State of a non-quantum 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 $\{0, 1\}$.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

State of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $[3, 1, 4, 1, 5, 9, 2, 6]$.

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.: $[-2, 7, -1, 8, 1, -8, -2, 8]$.

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.: $[0, 0, 0, 0, 0, 1, 0, 0]$.

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.:
$[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3]$.

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.

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
- "collapses" 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
- "collapses" 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
- "collapses" the state.

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

"Collapse": New state is 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. $\mathrm{C}_1\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 1, 4, 5, 9, 6, 2]$.

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. $\mathrm{C}_1\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 1, 4, 5, 9, 6, 2]$.

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. $\mathrm{C}_2\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 9, 5, 6, 2]$.

Controlled-NOT (CNOT) gates

e.g. $\mathrm{C}_1\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 1, 4, 5, 9, 6, 2]$.

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. $\mathrm{C}_2\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 9, 5, 6, 2]$.

e.g. $\mathrm{C}_0\mathrm{NOT}_2$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 9, 4, 6, 5, 1, 2, 1]$.

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. $\mathrm{C_2C_1NOT_0}$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 5, 9, 6, 2]$.

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. $\mathrm{C}_2\mathrm{C}_1\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 5, 9, 6, 2]$.

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

e.g. $\mathrm{C}_0\mathrm{C}_1\mathrm{NOT}_2$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 6, 5, 9, 2, 1]$.

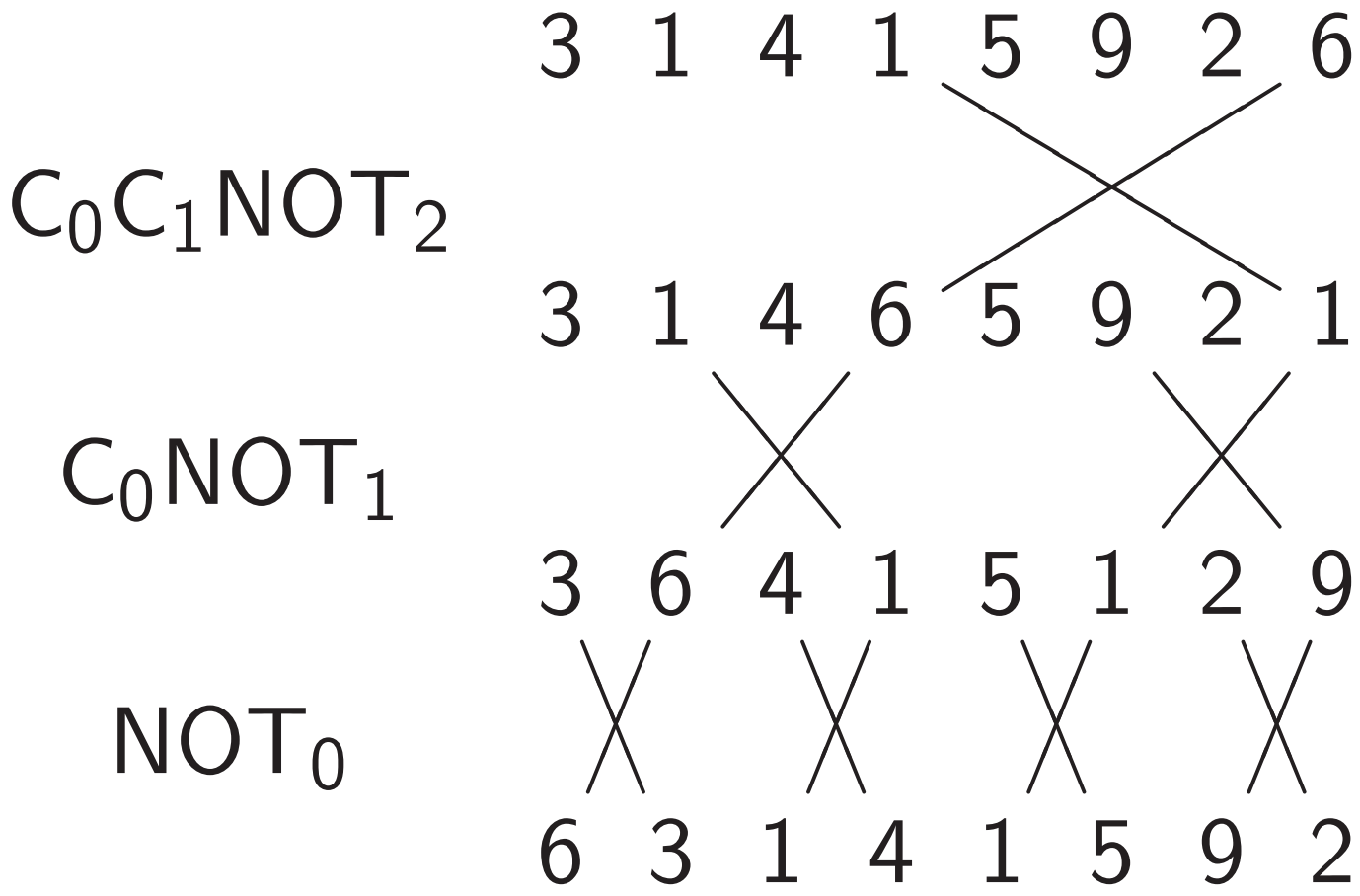
More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

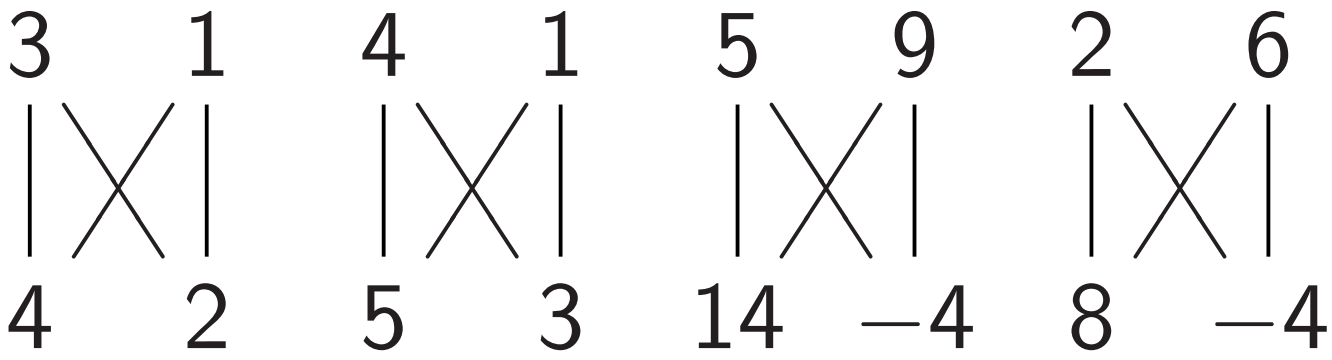
e.g. series of gates to
rotate 8 positions by distance 1:



Hadamard gates

Hadamard$_0$:

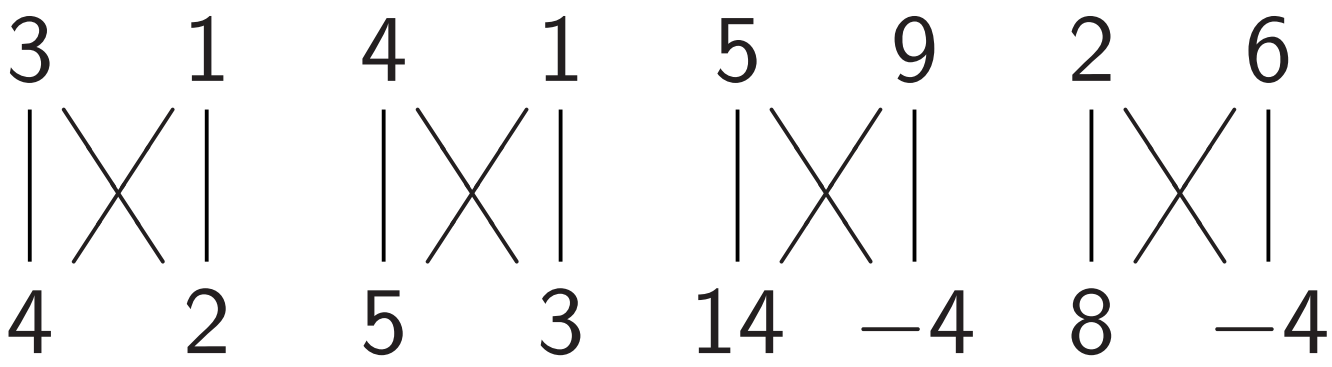
$[a, b] \mapsto [a + b, a - b]$.



Hadamard gates

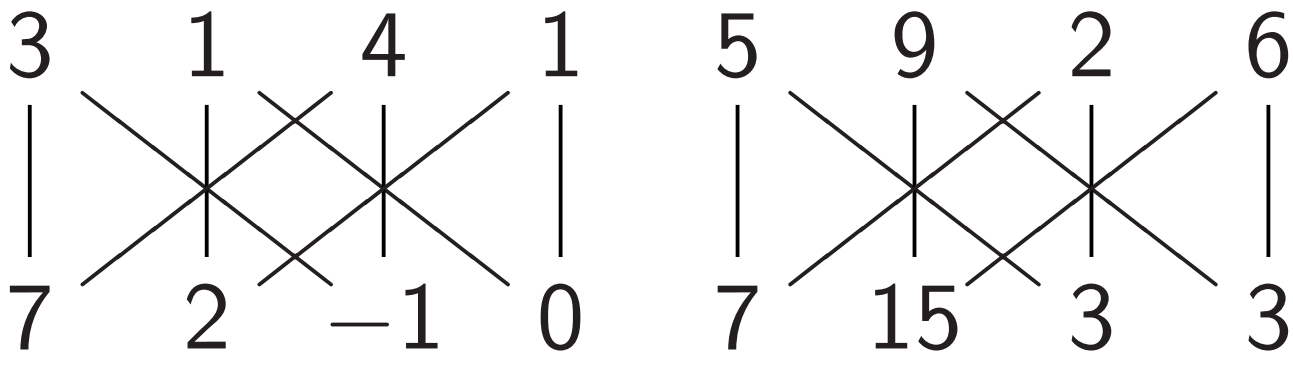
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



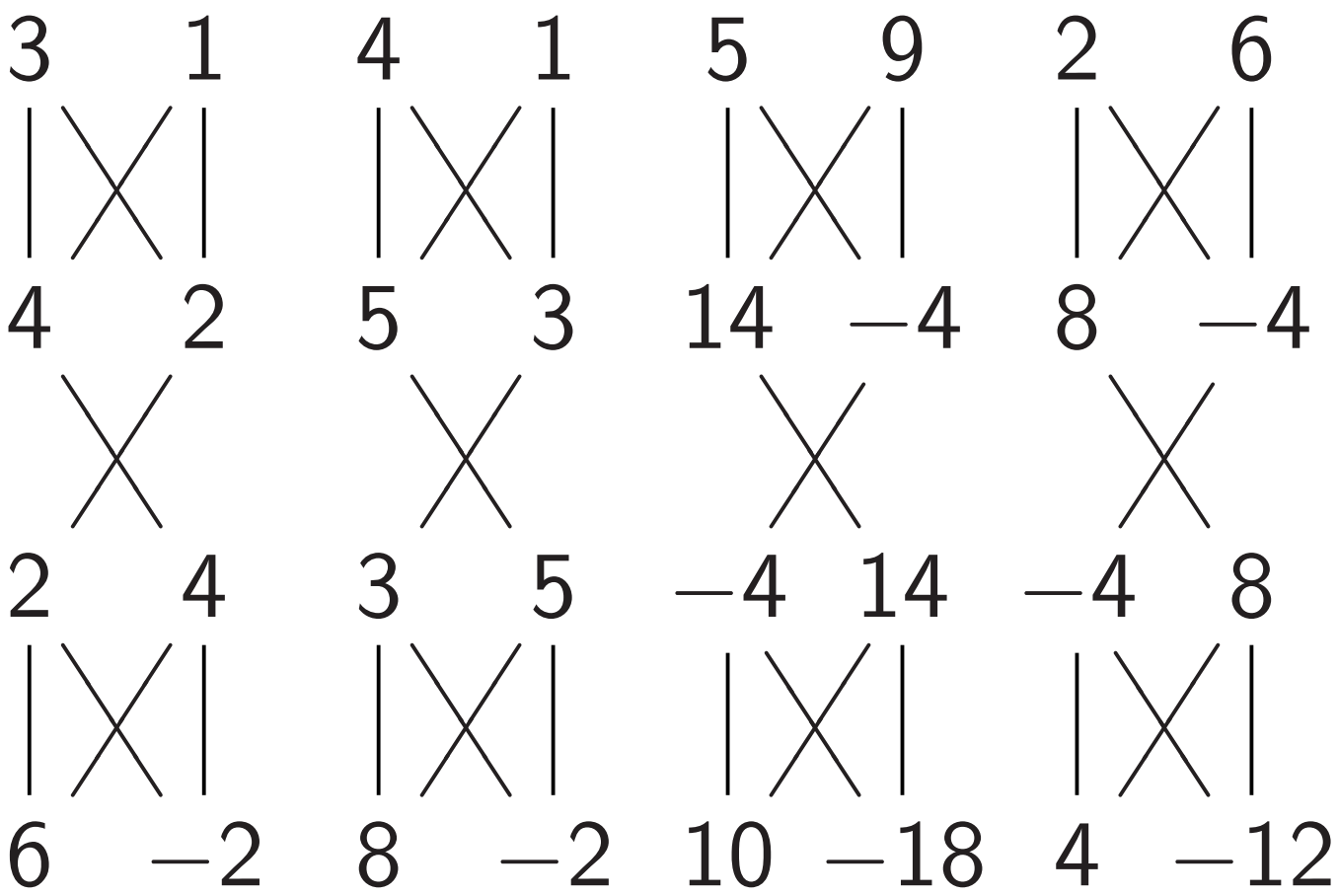
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



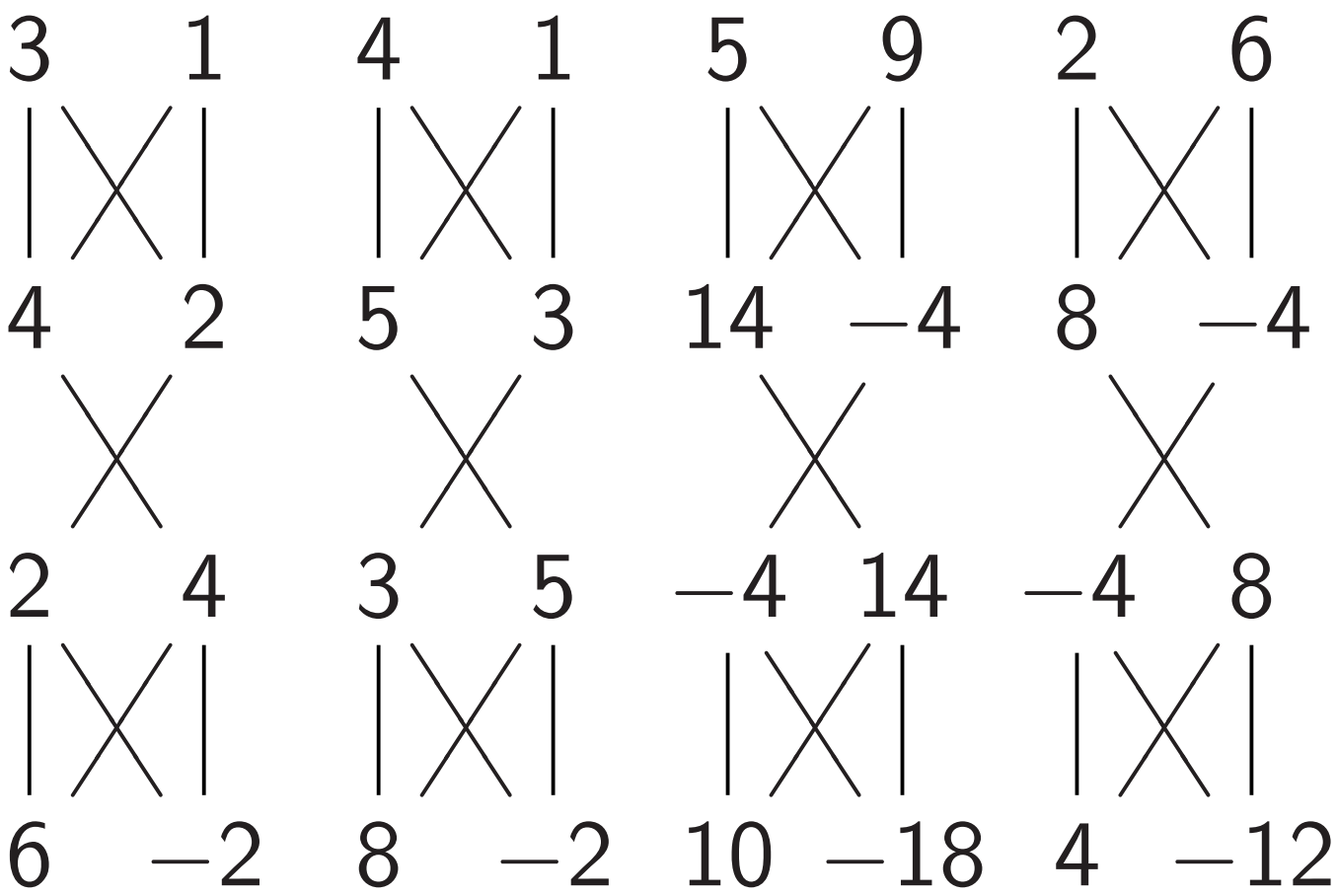
Some uses of Hadamard gates

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



Some uses of Hadamard gates

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

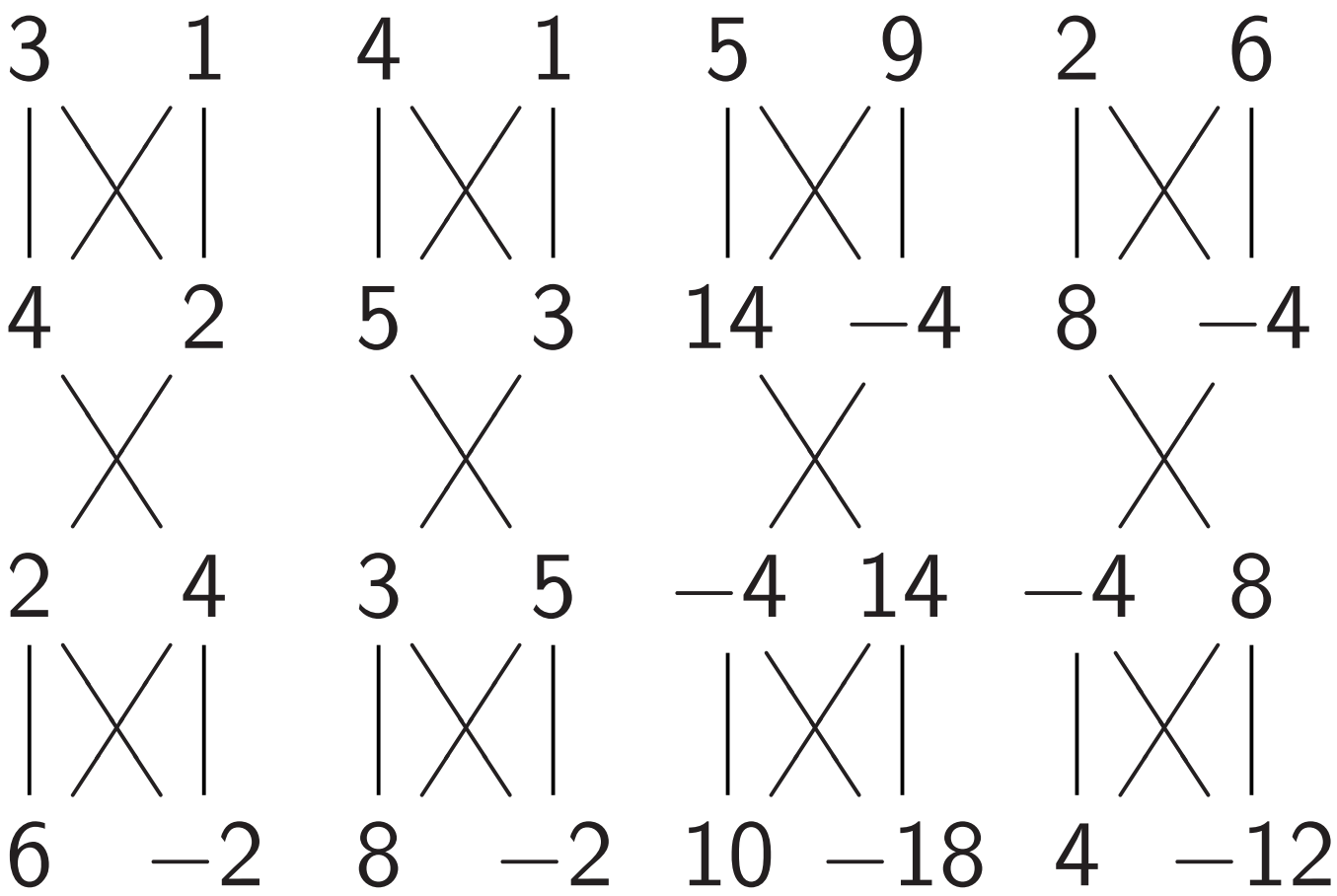


"Multiplied each amplitude by 2."

This is not physically observable.

Some uses of Hadamard gates

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



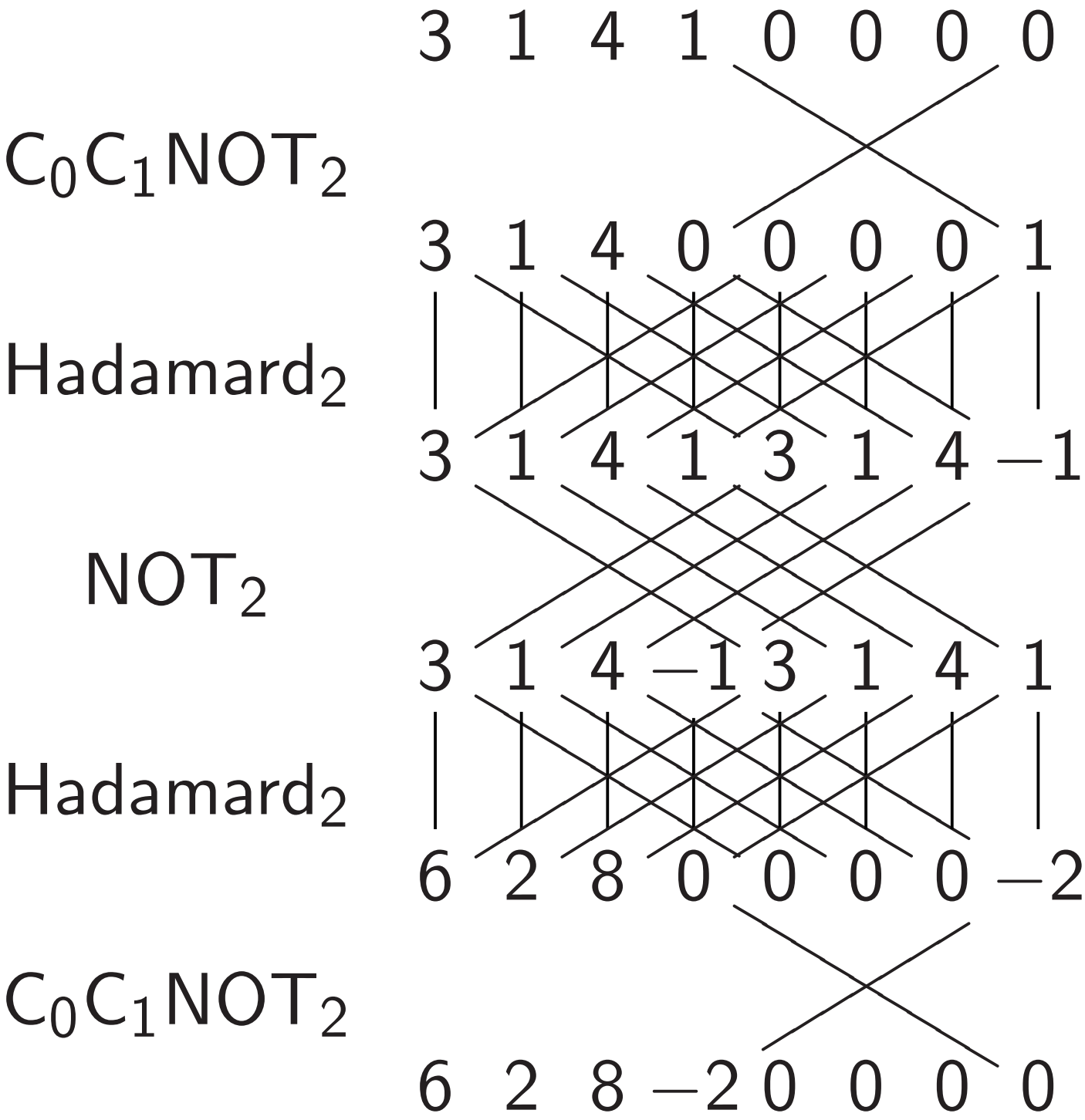
"Multiplied each amplitude by 2."
This is not physically observable.

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

Fancier example:

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

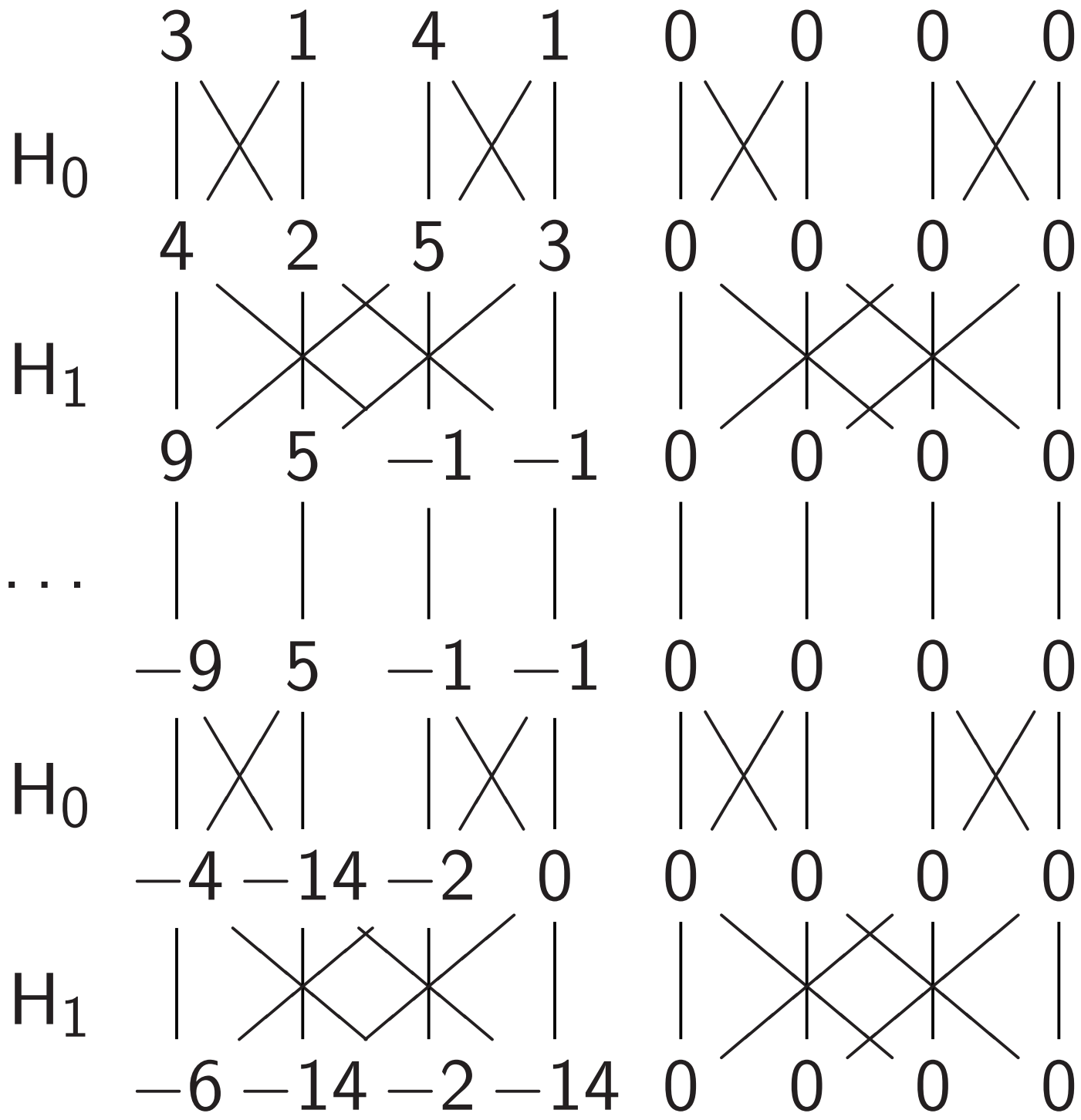
Assumes $q_2 = 0$: "ancilla" qubit.



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.

Non-quantum 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum evaluations 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.

This example is for a function f with 3-bit input and 3-bit output.

Example of Simon's algorithm

Step 2.0. 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 2.1. 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 2.2. 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. Step 3 will apply the function f (a specific function in this example), computing $f(u)$ in universe u.

Example of Simon's algorithm

Step 3a. 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 3b. More entry 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 3c. More entry 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 3d. More entry 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 3e. More entry 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 3f. More entry 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 3g. More entry 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 3h. More entry 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 3i. More entry 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 3j. Final entry 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 3j. Final entry 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 4.0. 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 4.1. Hadamard_1:

$$
\begin{array}{cccccccc}
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
1, & \overline{1}, & \overline{1}, & 1, & 1, & 1, & \overline{1}, & \overline{1}, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
1, & 1, & \overline{1}, & \overline{1}, & 1, & \overline{1}, & \overline{1}, & 1, \\
1, & \overline{1}, & 1, & \overline{1}, & 1, & 1, & 1, & 1, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
1, & 1, & 1, & 1, & 1, & \overline{1}, & 1, & \overline{1}.
\end{array}
$$

Example of Simon's algorithm

Step 4.2. Hadamard_2:

$$
\begin{array}{cccccccc}
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & \overline{2}, & 0, & 0, & \overline{2}, & 0, & 2, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & \overline{2}, & 0, & 0, & 2, & 0, & \overline{2}, \\
2, & 0, & 2, & 0, & 0, & \overline{2}, & 0, & \overline{2}, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
2, & 0, & 2, & 0, & 0, & 2, & 0, & 2.
\end{array}
$$

Example of Simon's algorithm

Step 4.2. 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 5: 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 3 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 3 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 3 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 3 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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm

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

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$
quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

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

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

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

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

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

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

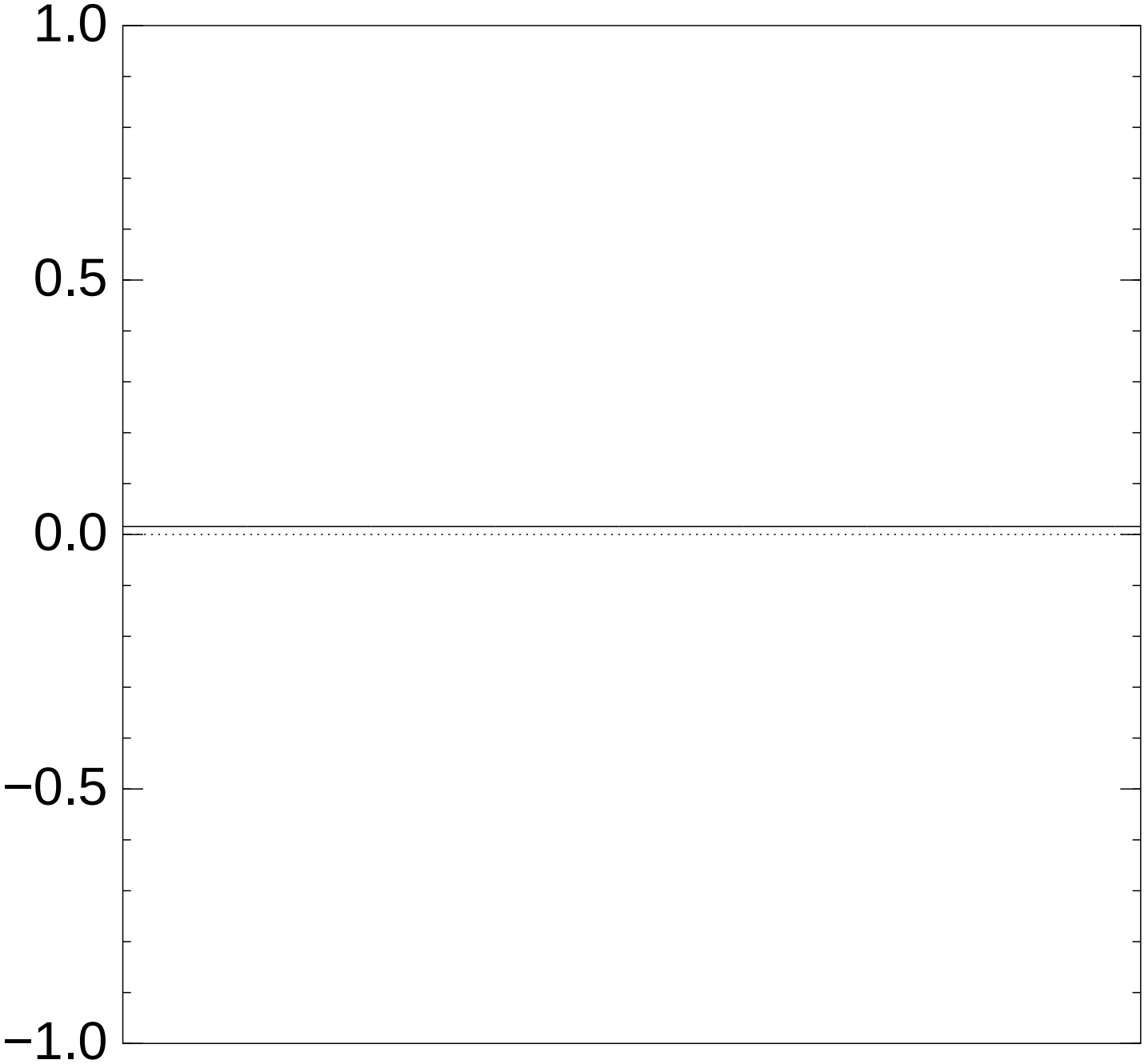
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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

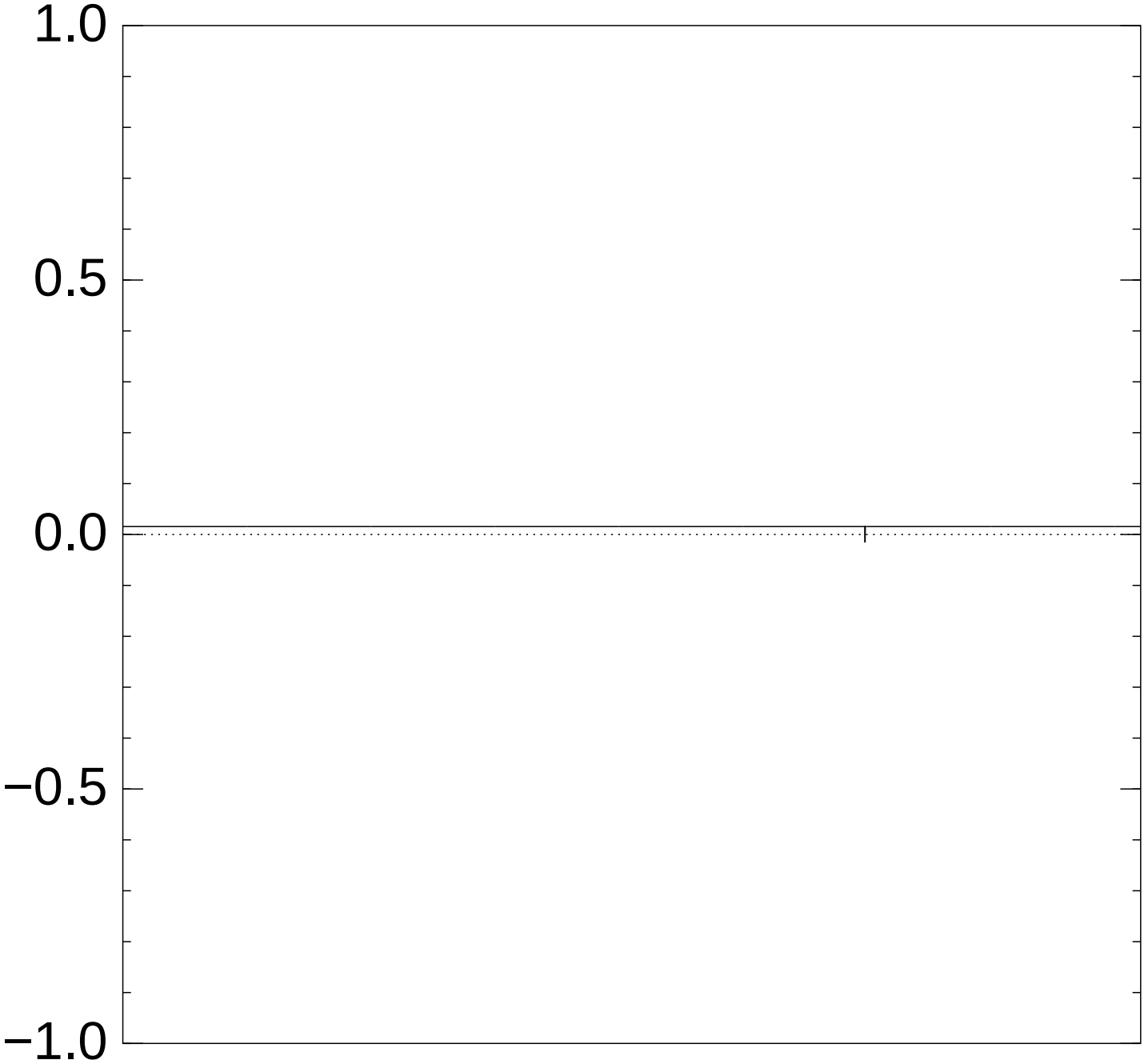
Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ times.

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

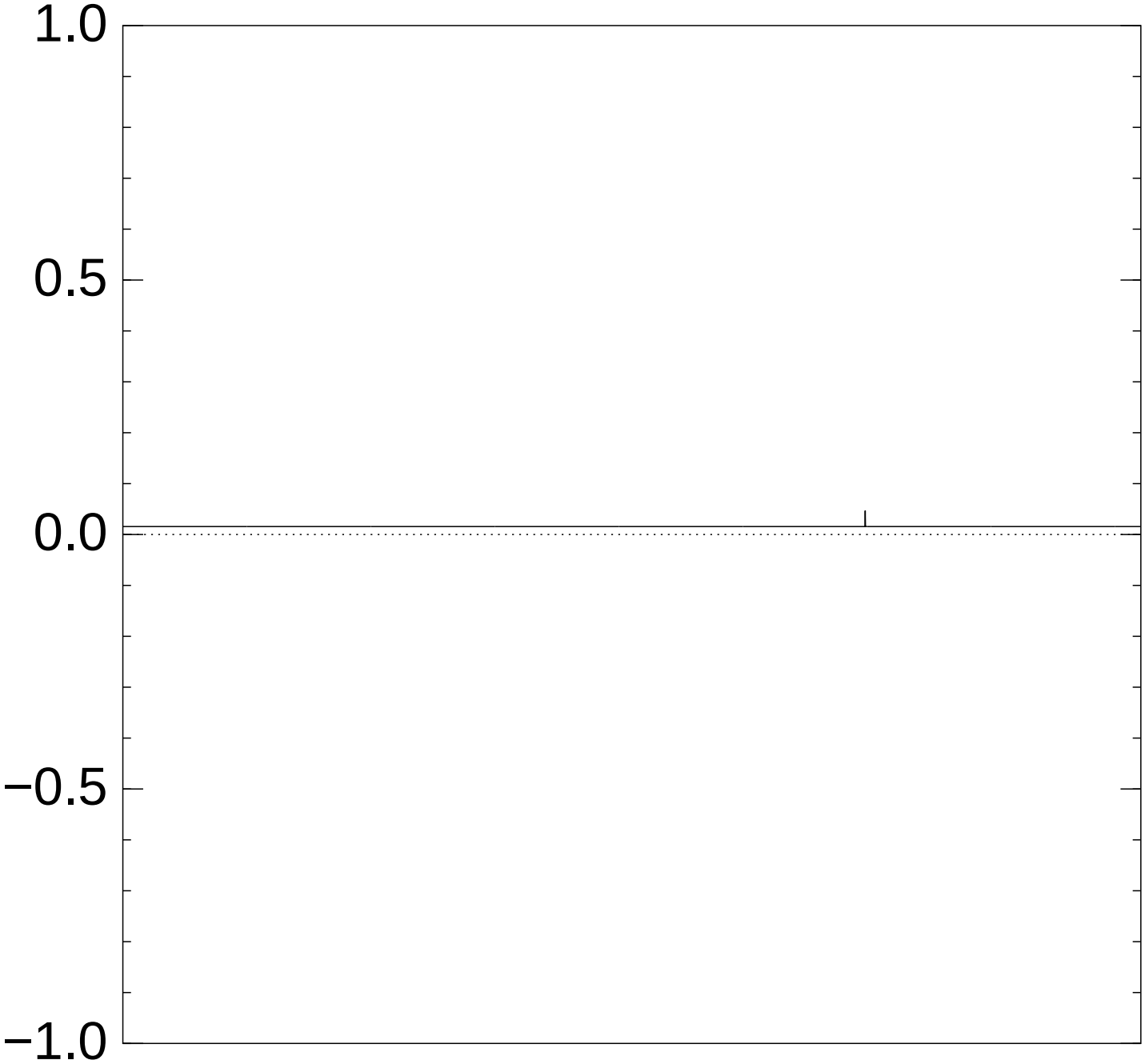
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after 0 steps:



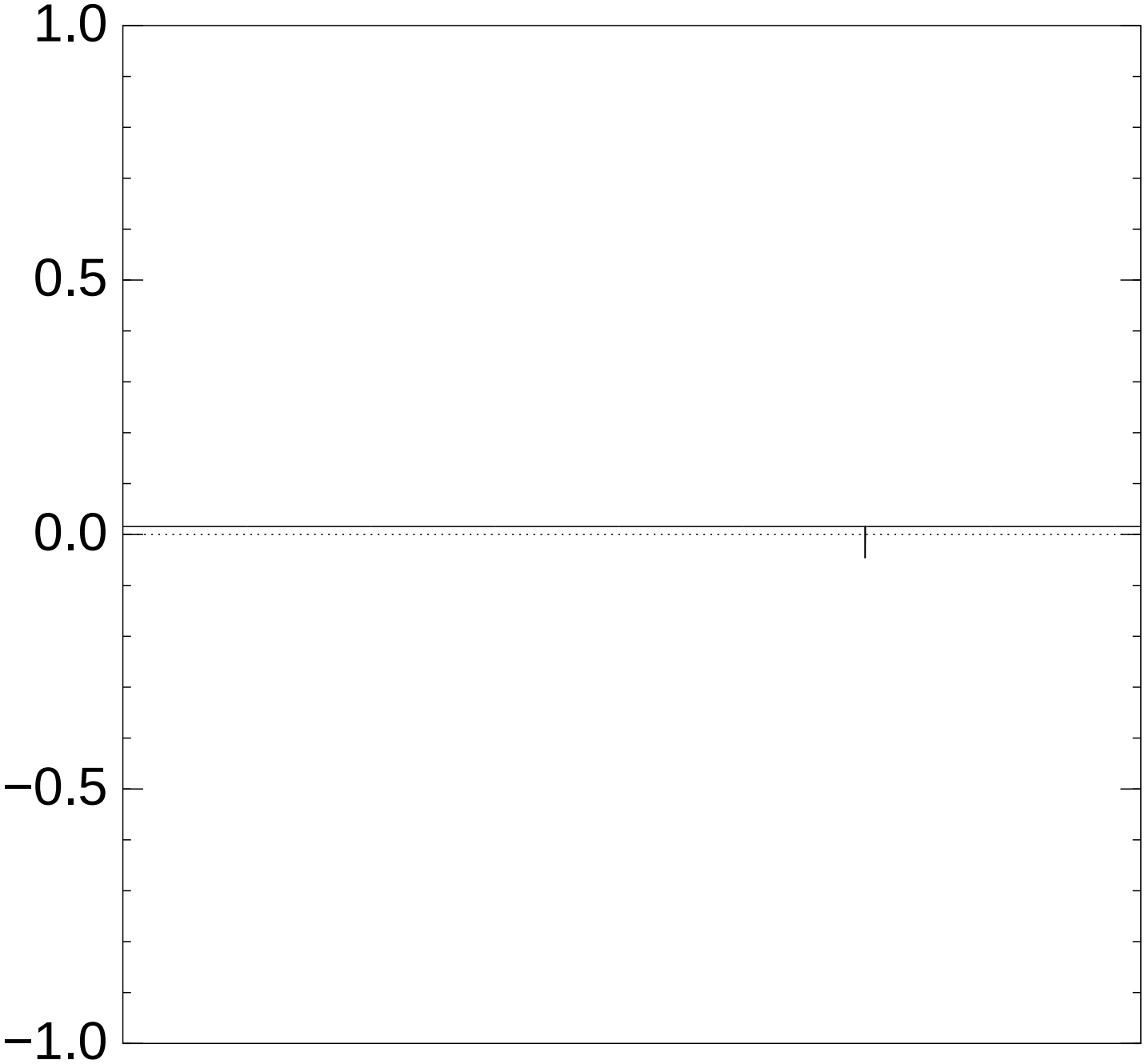
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after Step 1:



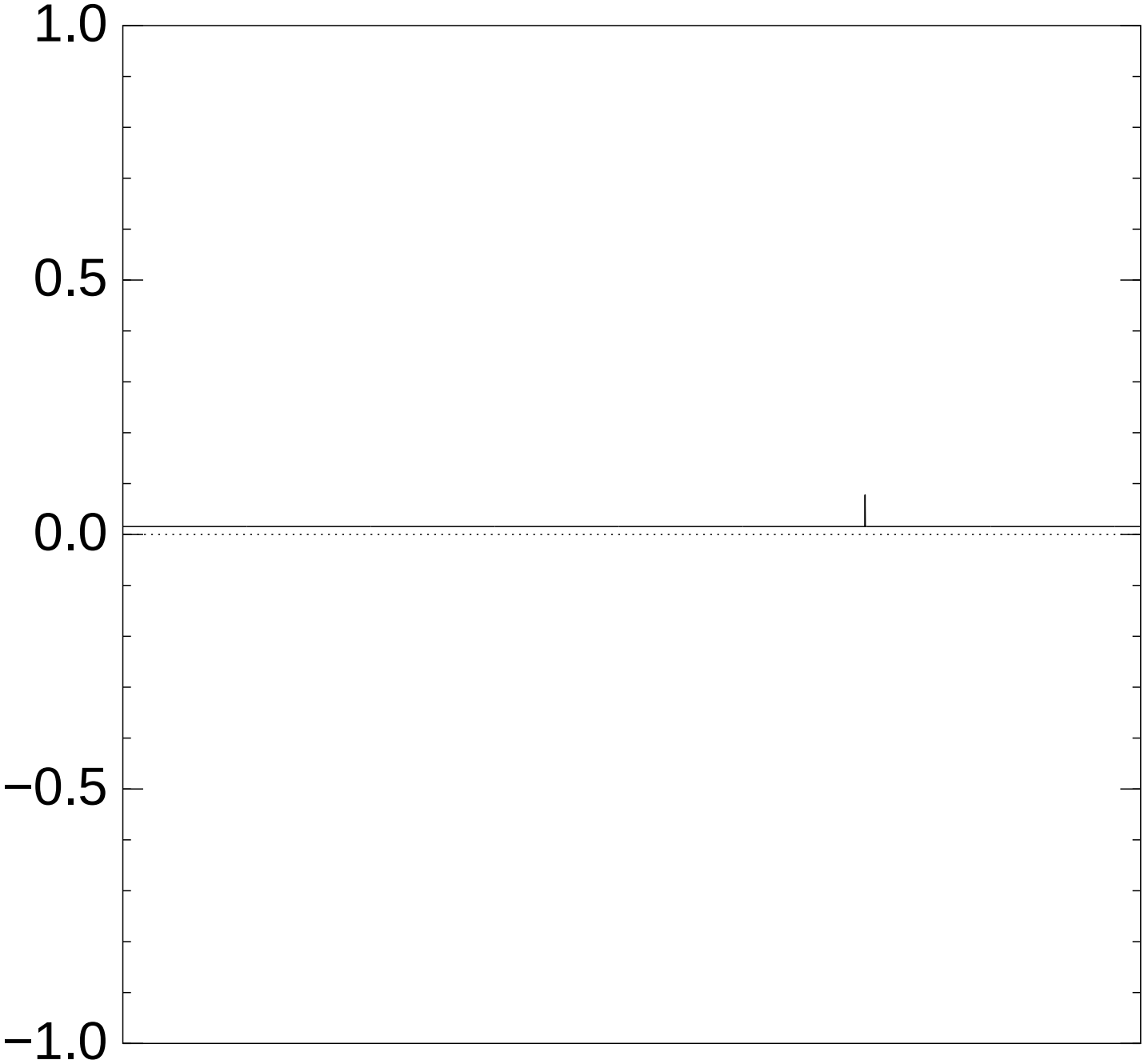
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after Step 1 + Step 2:



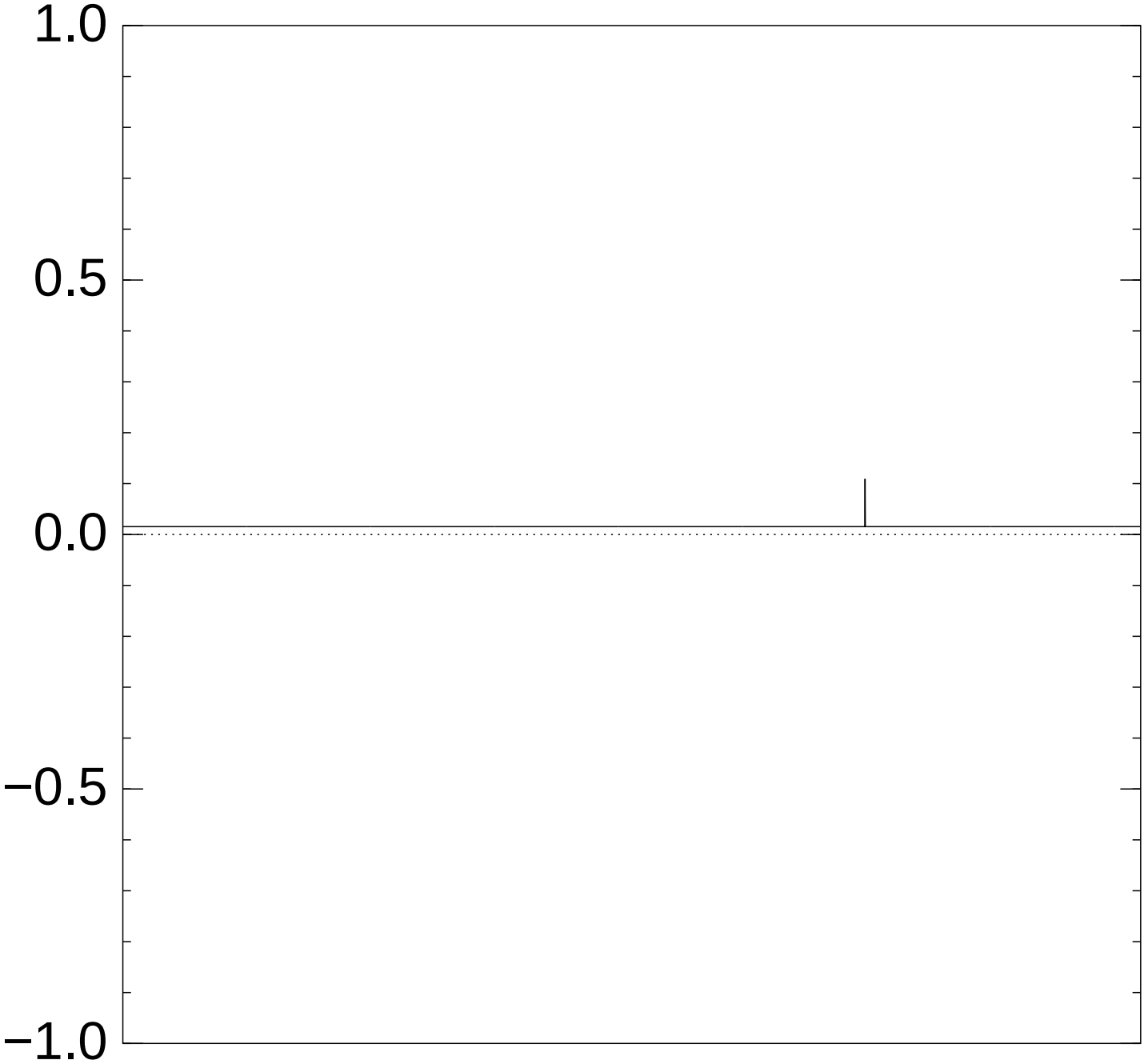
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after Step 1 + Step 2 + Step 1:



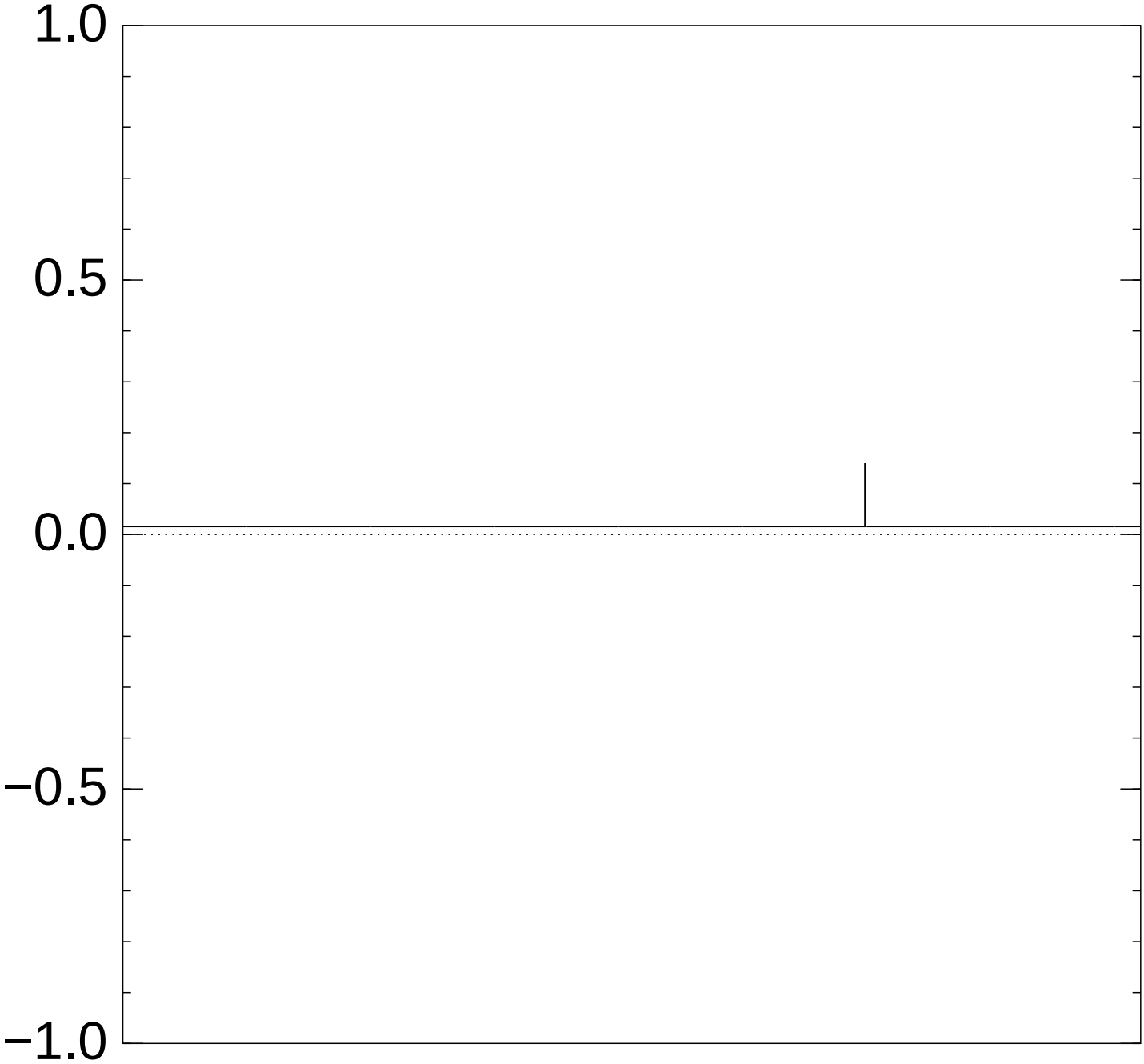
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $2 \times (\text{Step } 1 + \text{Step } 2)$:



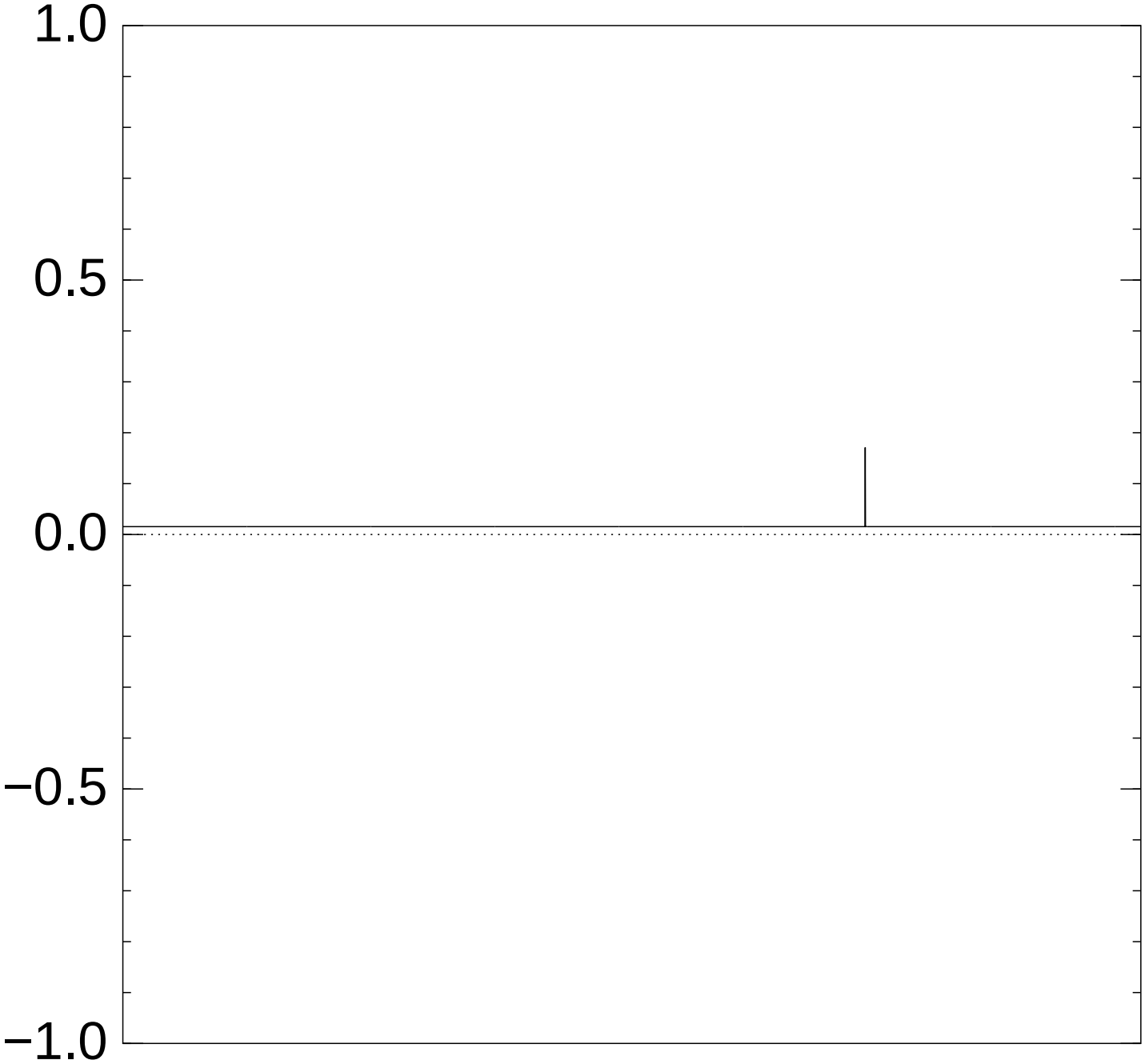
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $3 \times$ (Step 1 + Step 2):



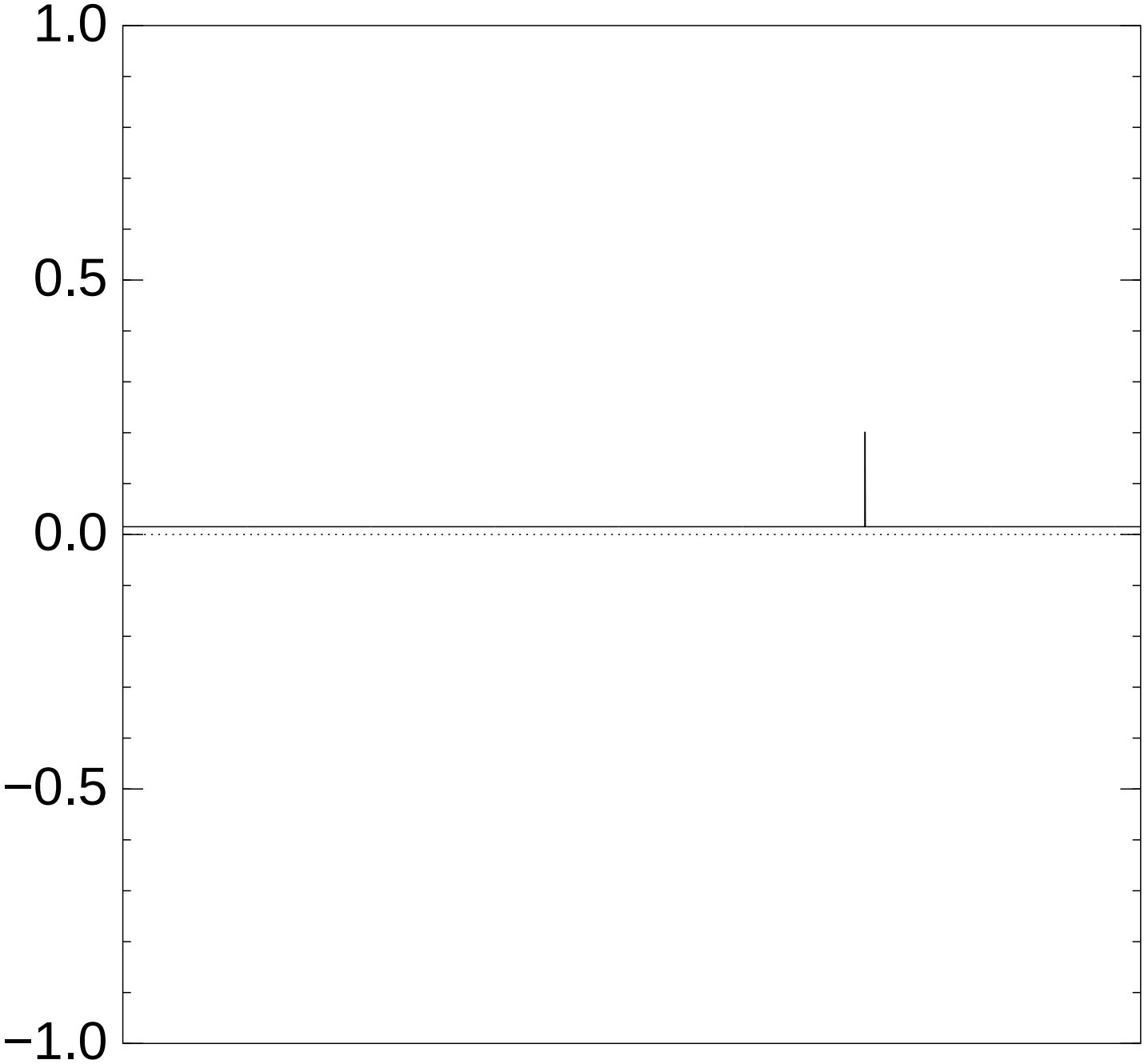
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $4 \times$ (Step 1 + Step 2):



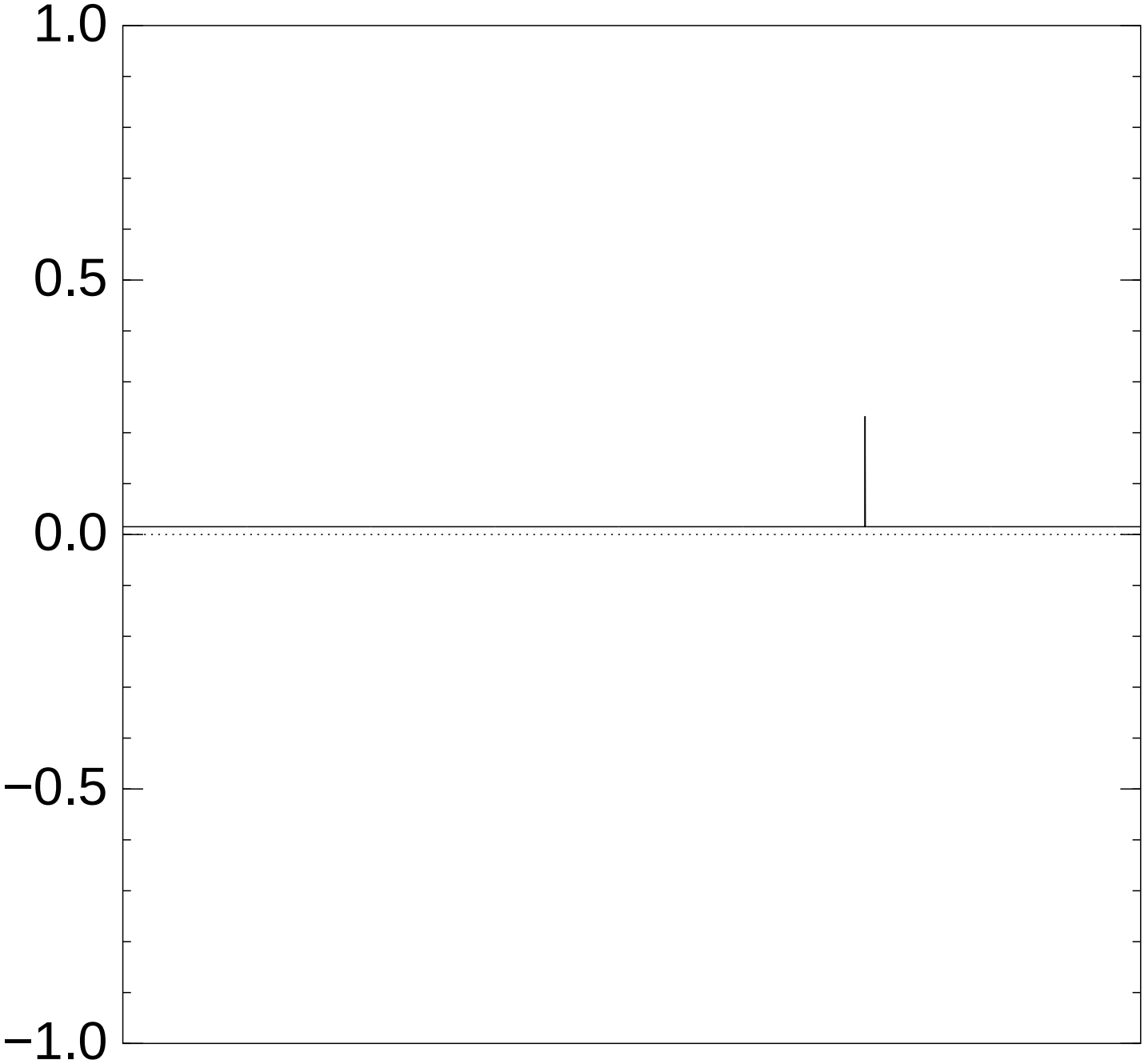
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $5 \times (\text{Step } 1 + \text{Step } 2)$:



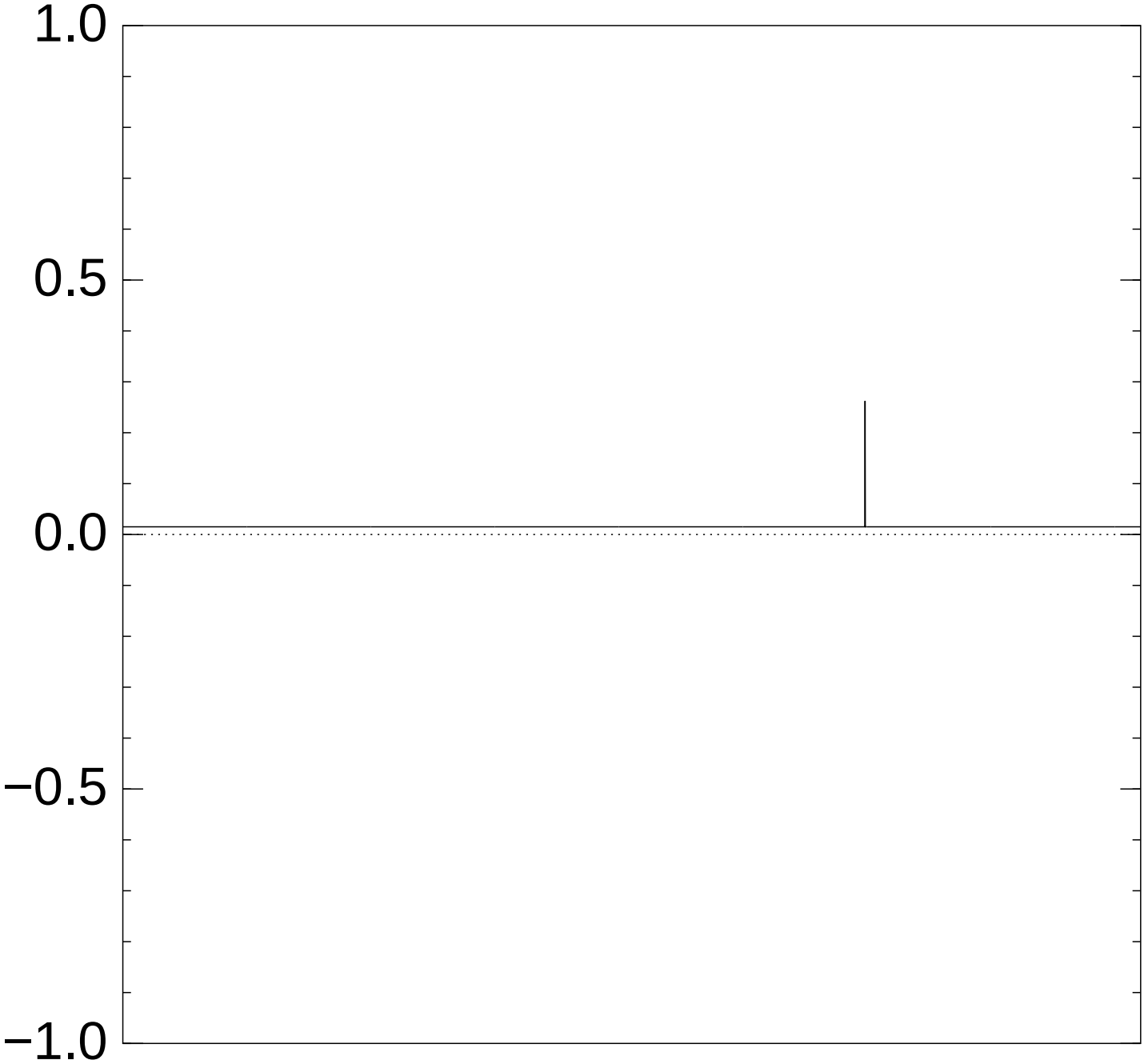
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $6 \times (\text{Step } 1 + \text{Step } 2)$:



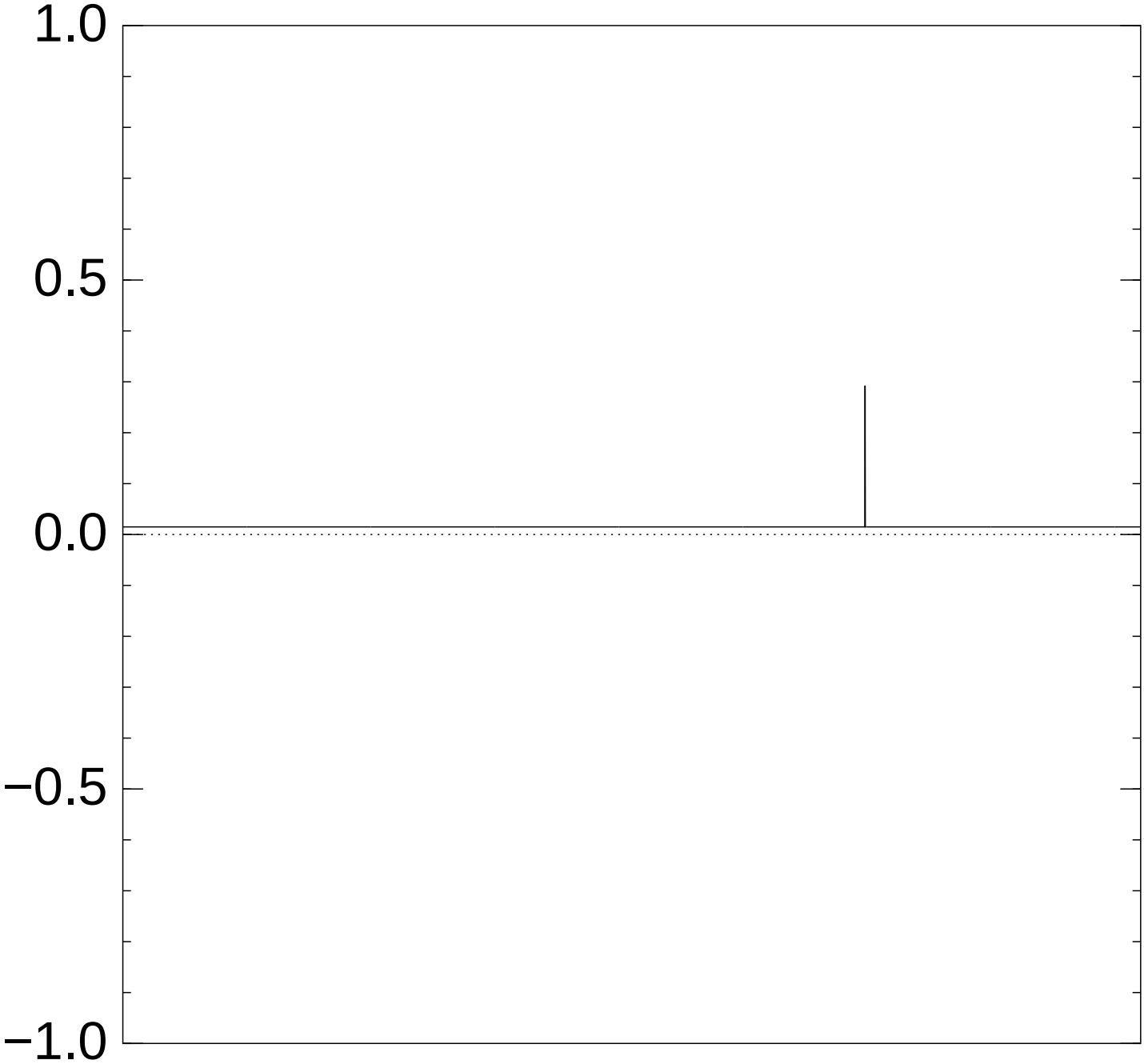
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $7 \times (\text{Step } 1 + \text{Step } 2)$:



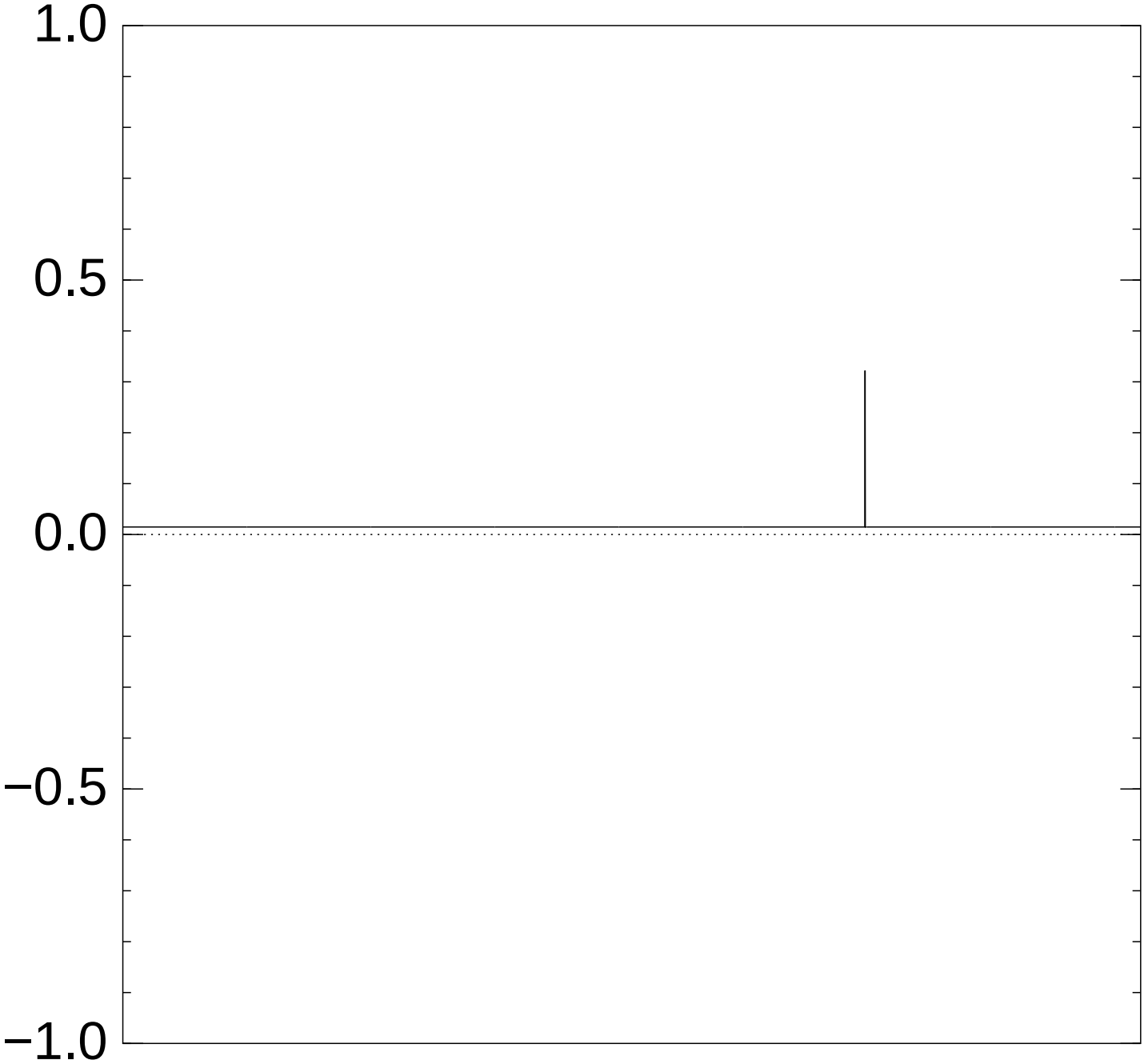
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $8 \times (\text{Step } 1 + \text{Step } 2)$:



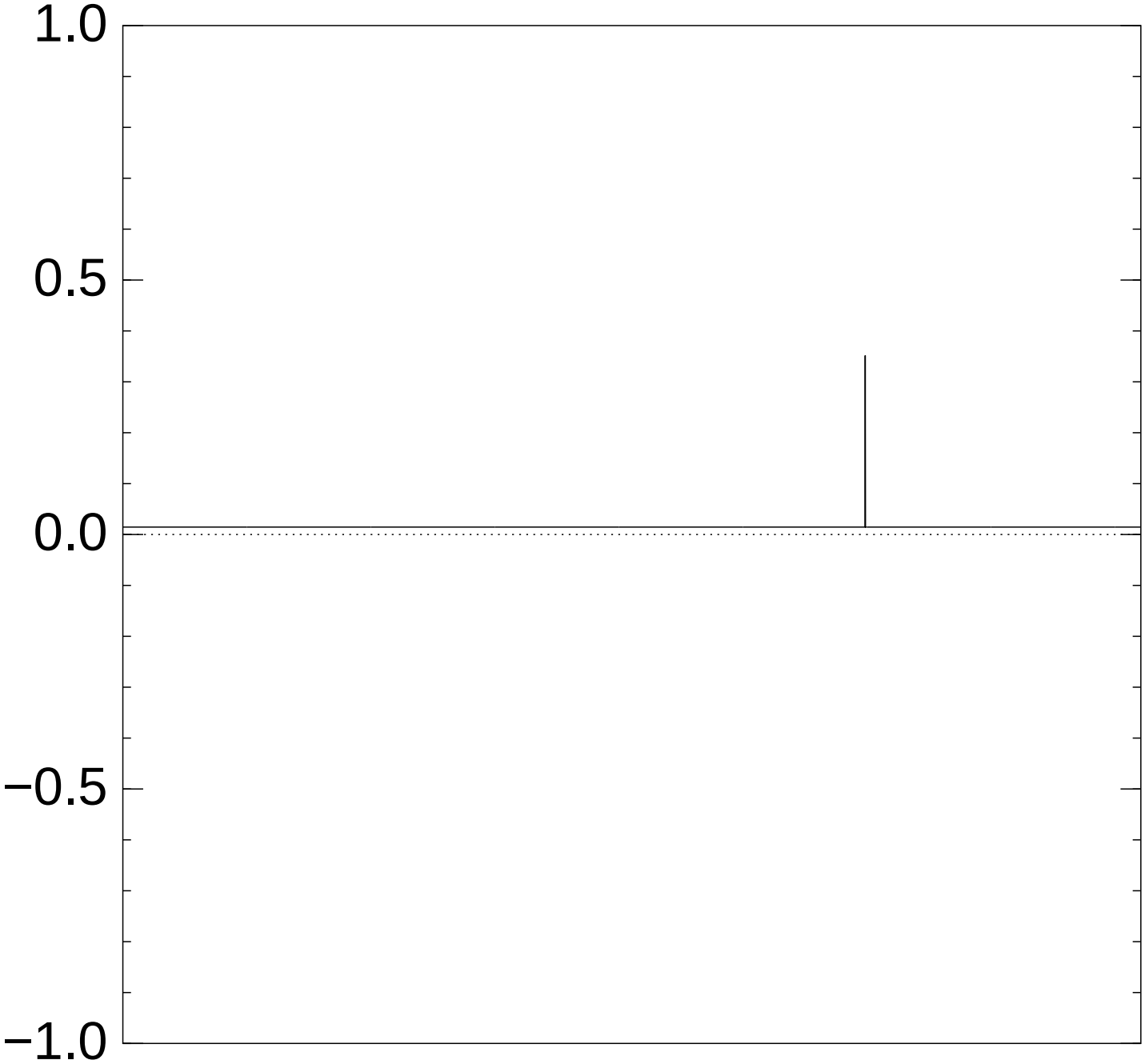
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $9 \times (\text{Step } 1 + \text{Step } 2)$:



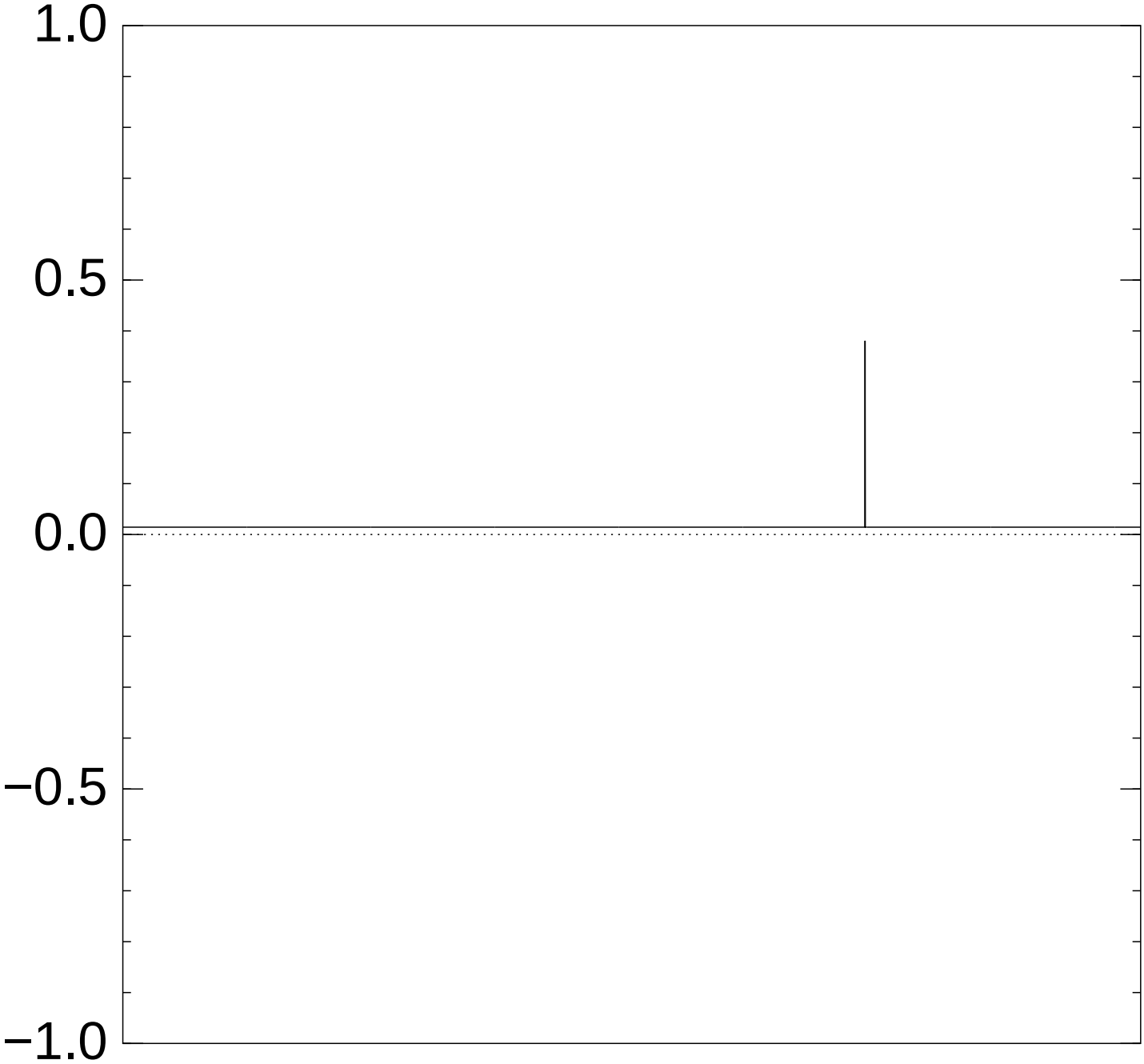
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $10 \times$ (Step 1 + Step 2):



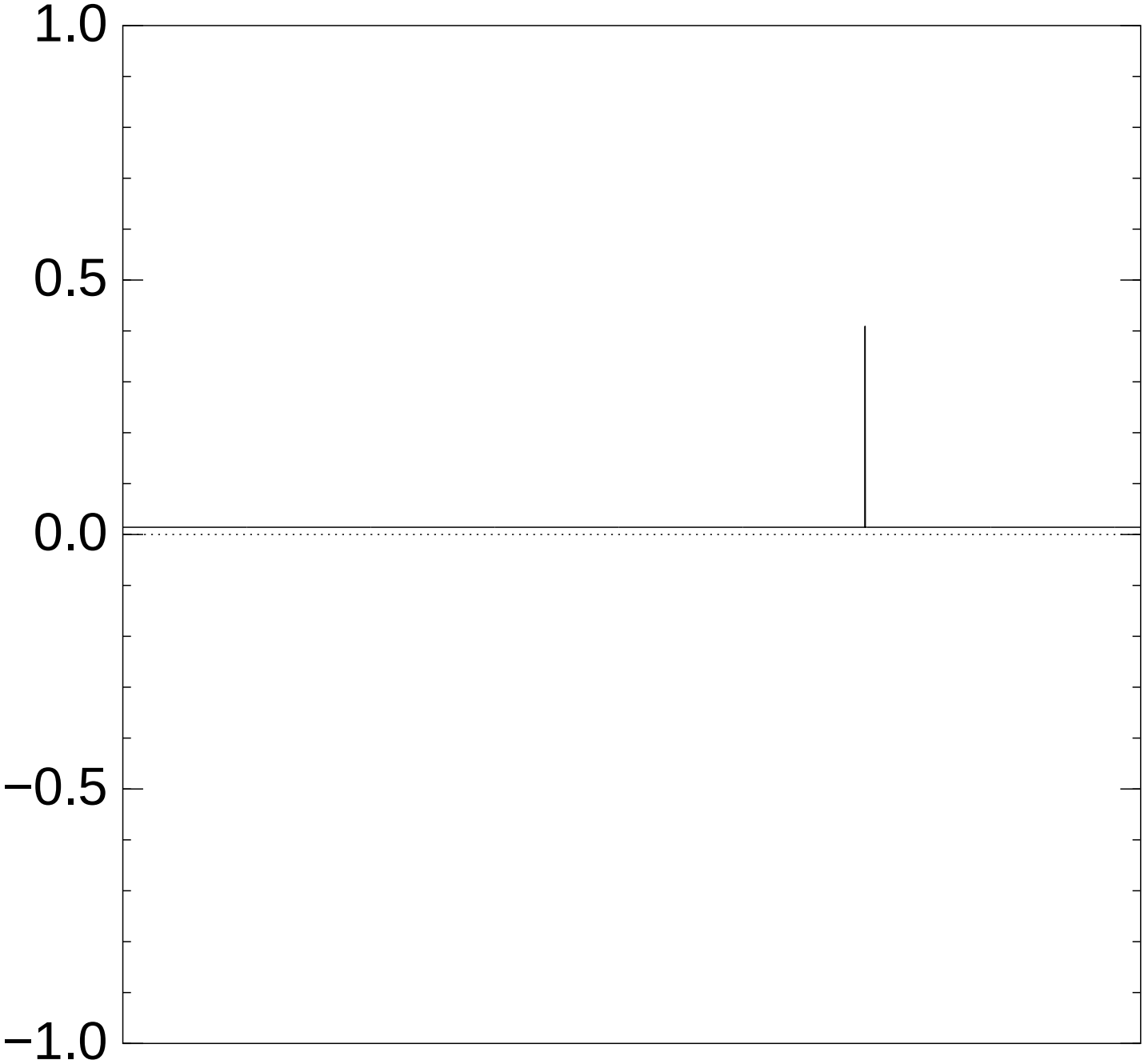
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $11 \times$ (Step 1 + Step 2):



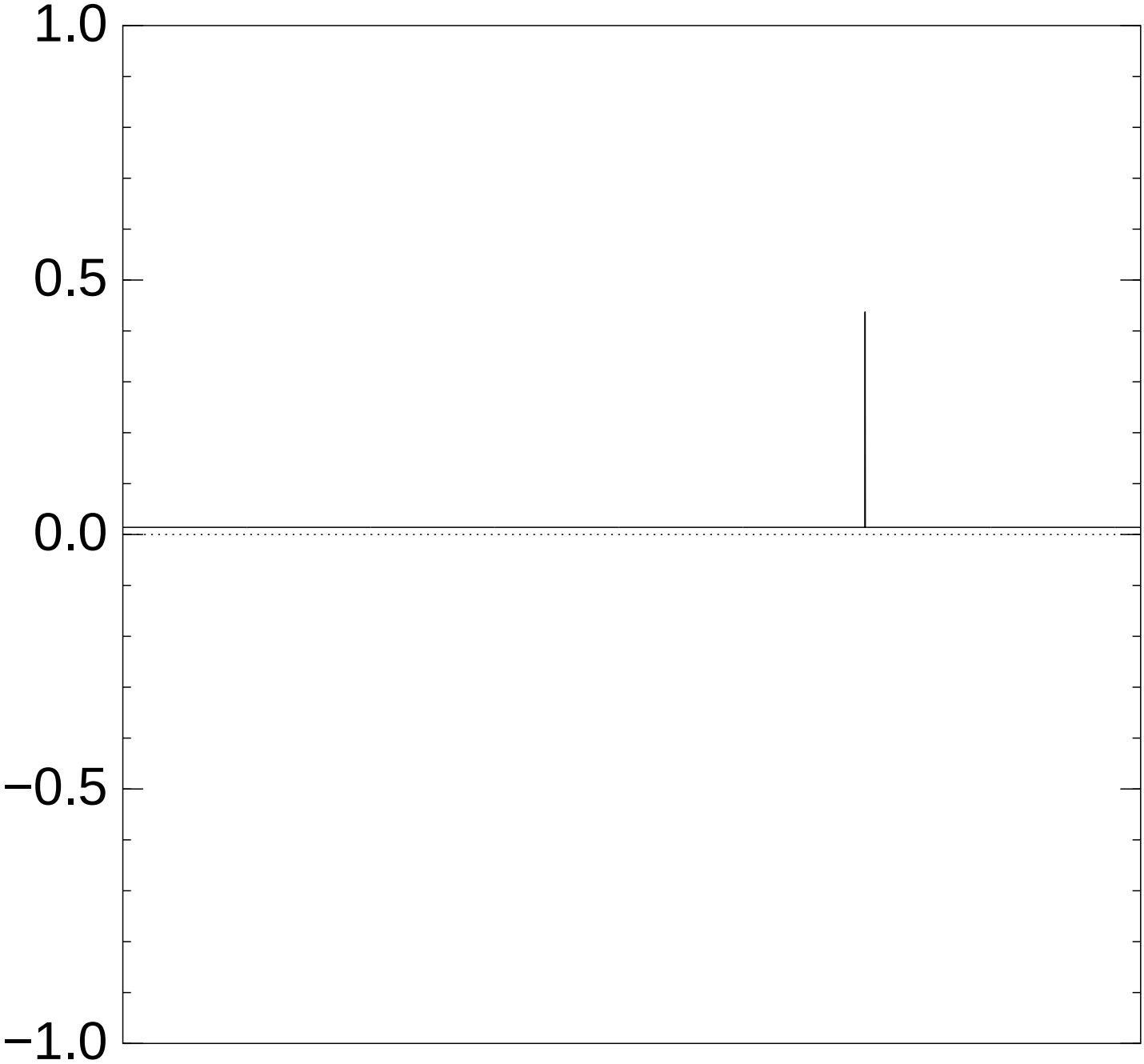
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $12 \times$ (Step 1 + Step 2):



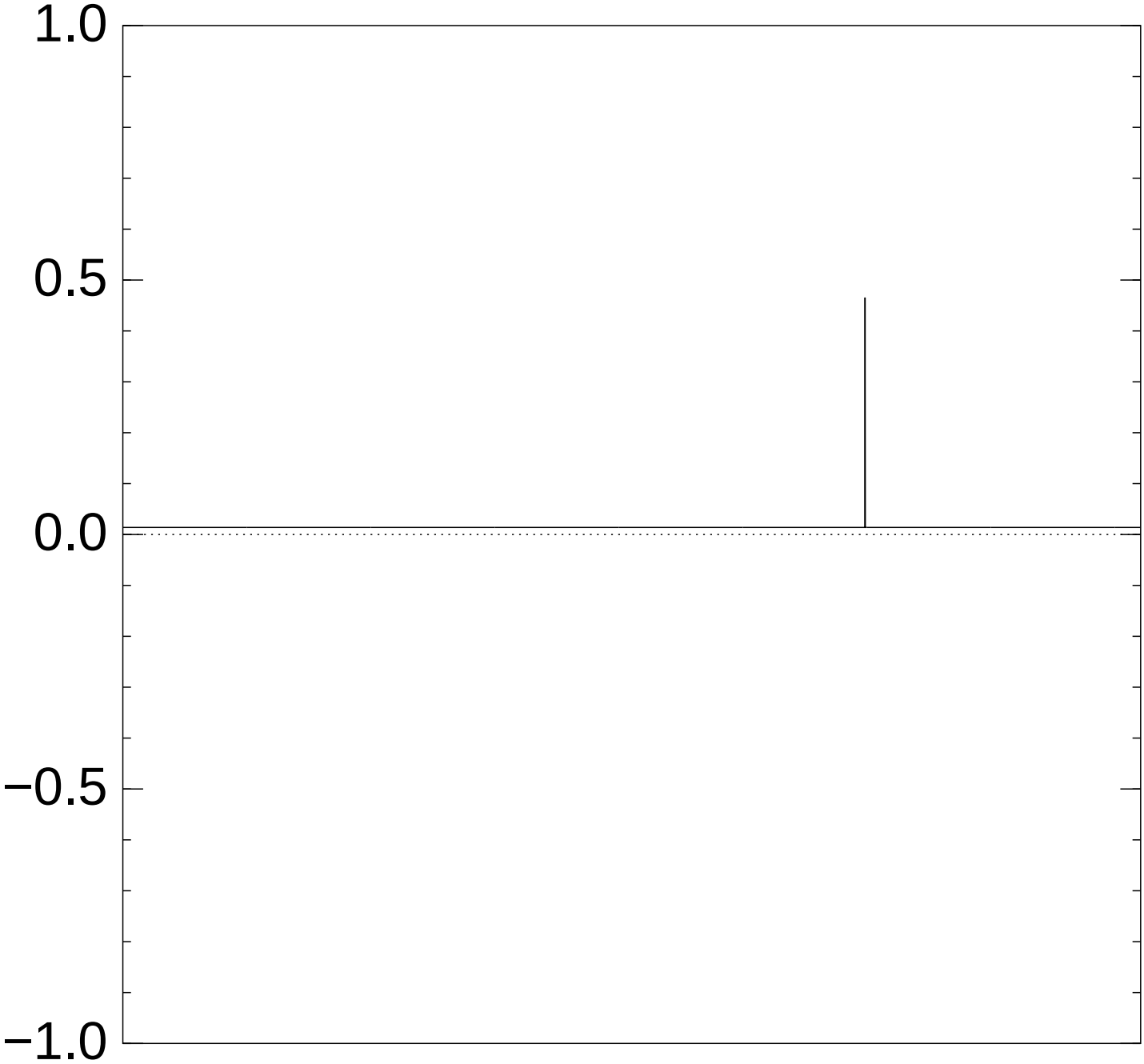
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $13 \times$ (Step 1 + Step 2):



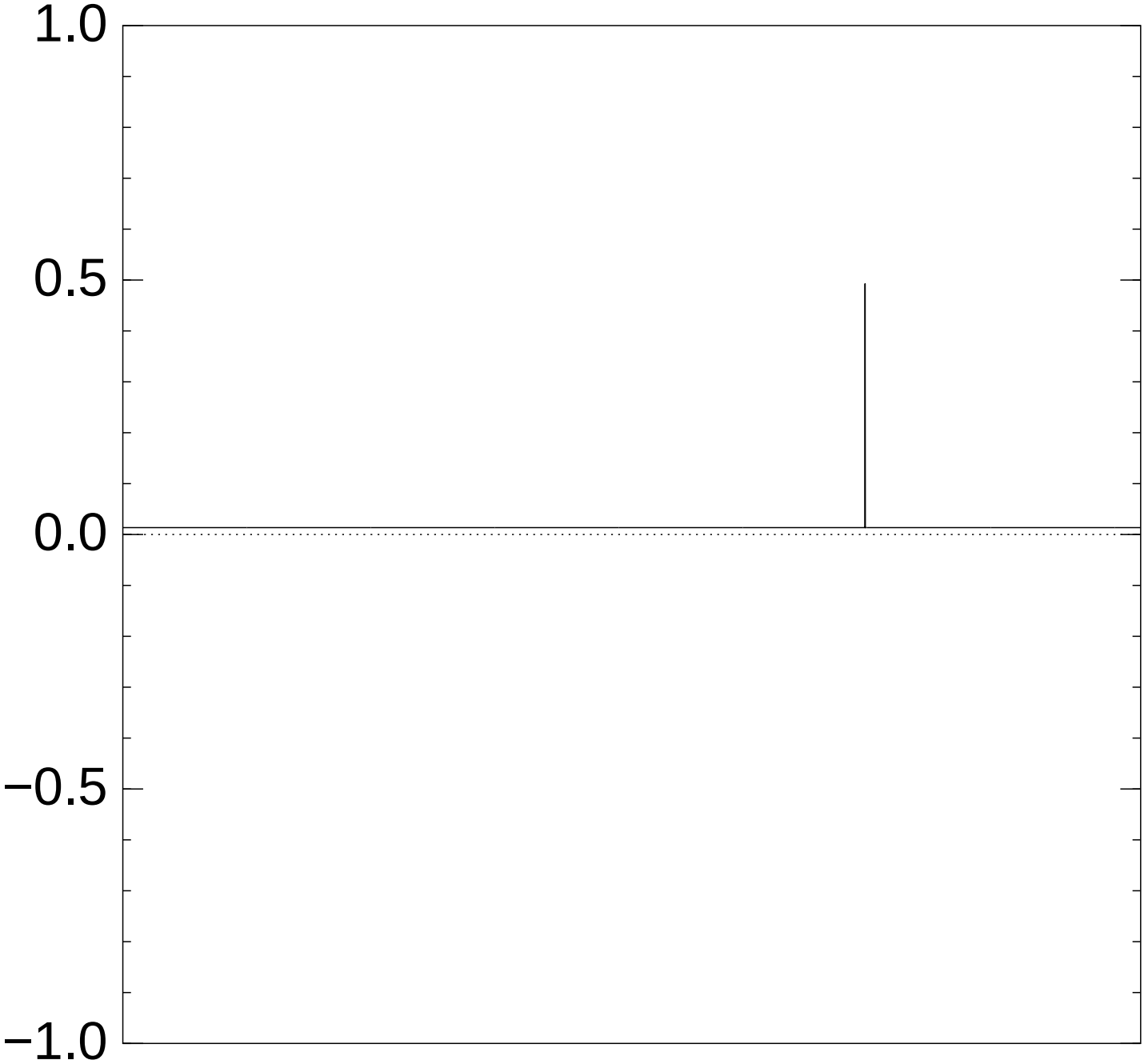
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $14 \times$ (Step 1 + Step 2):



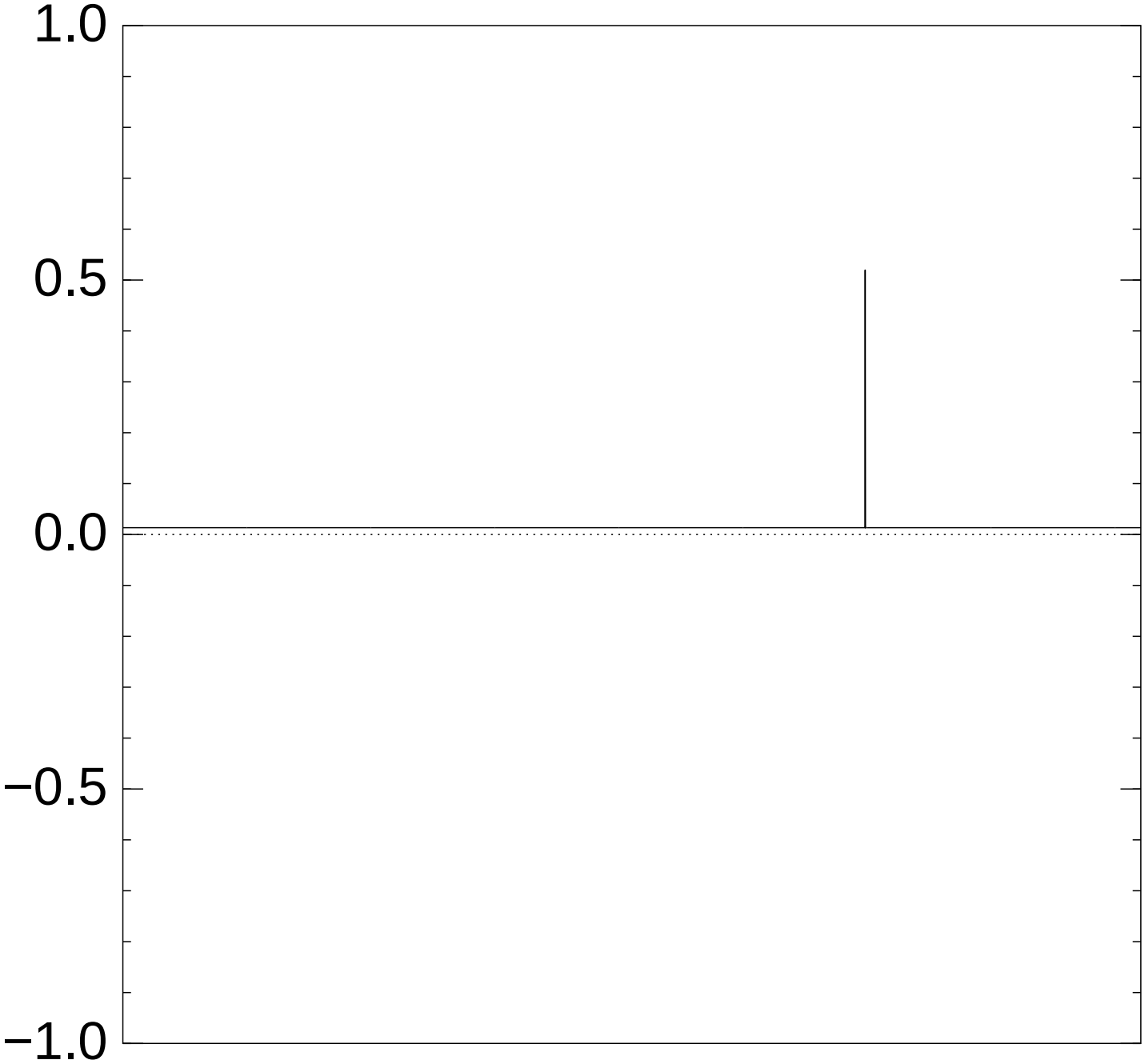
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $15 \times$ (Step 1 + Step 2):



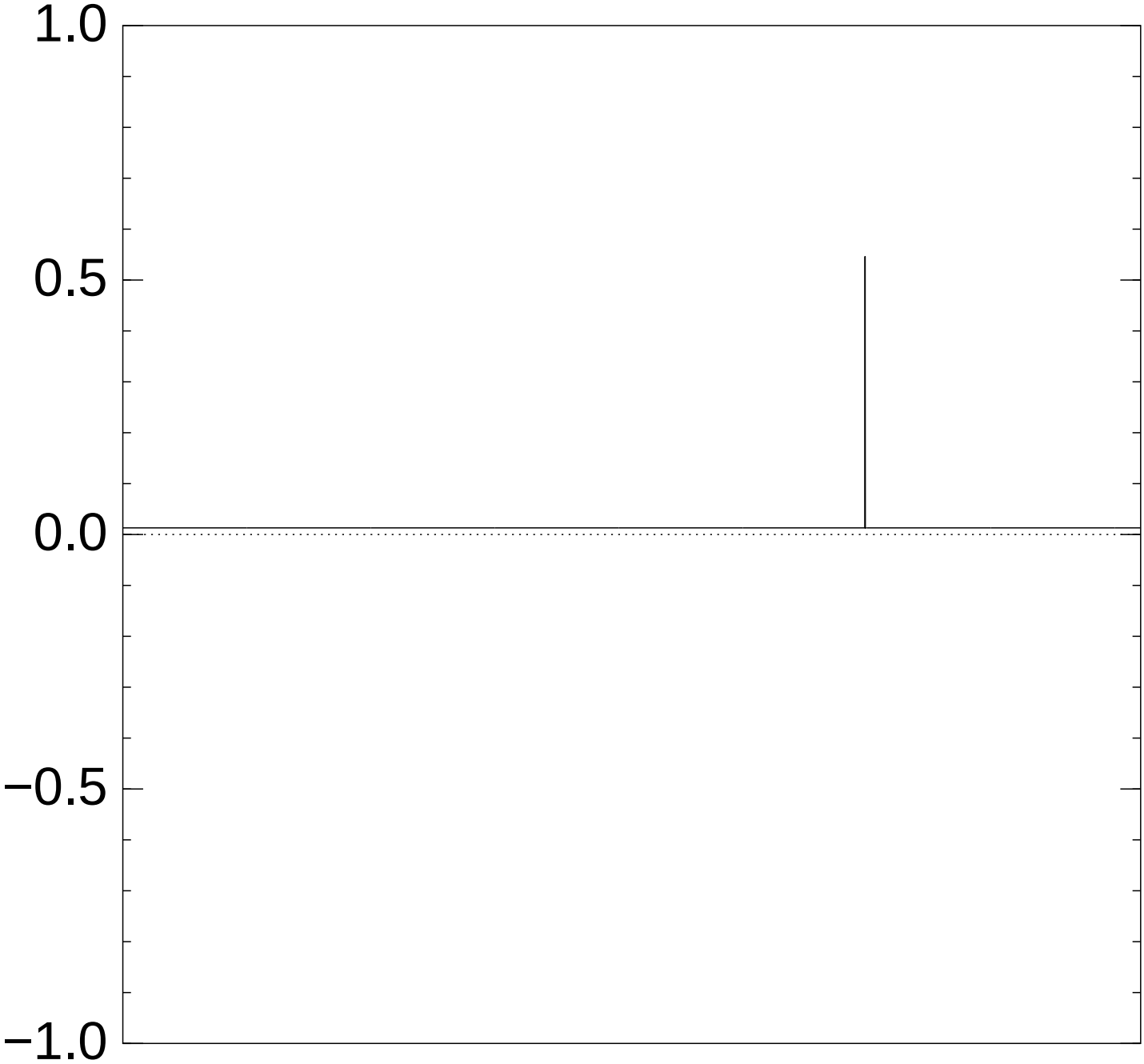
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $16 \times$ (Step 1 + Step 2):



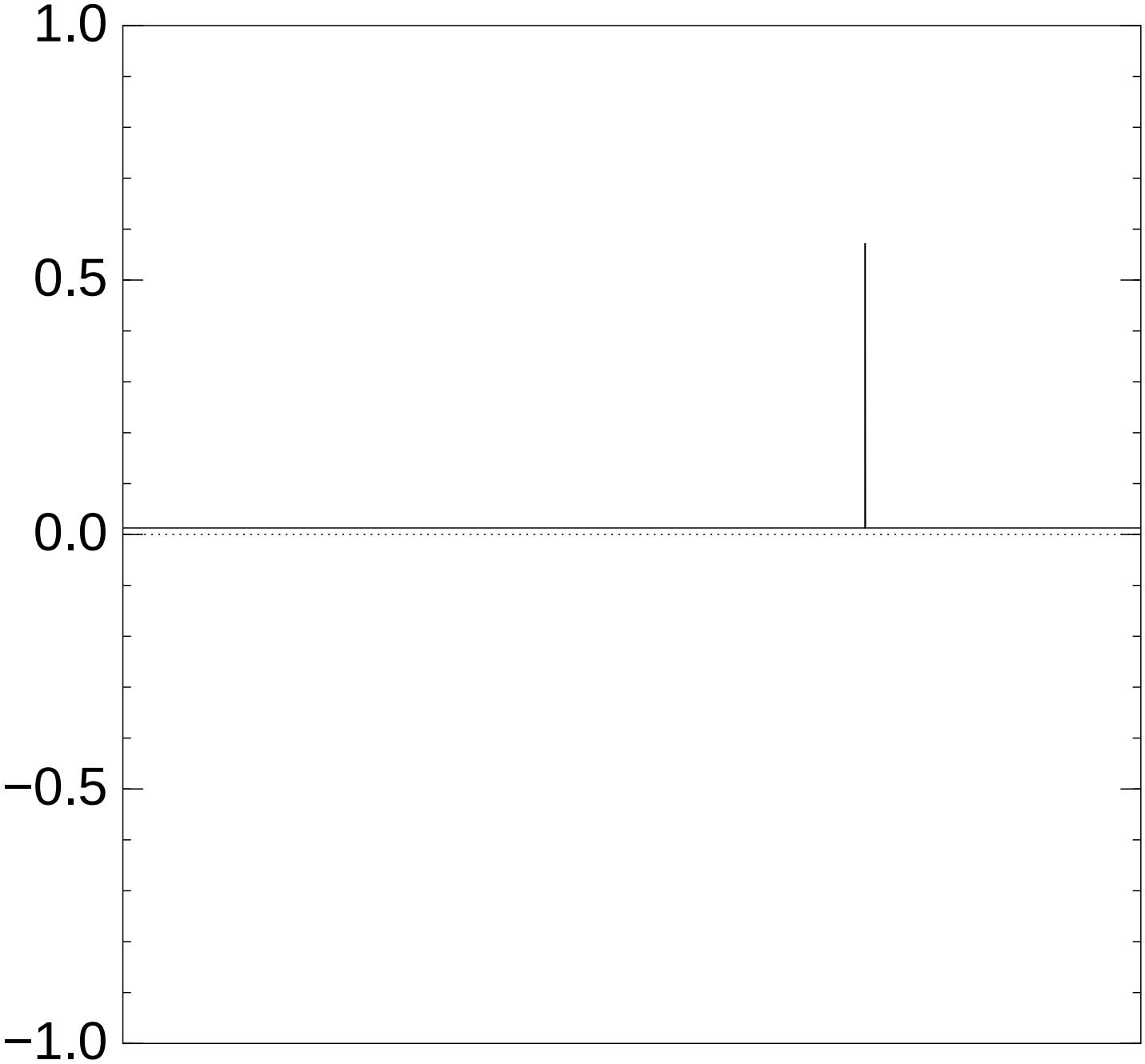
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $17 \times$ (Step 1 + Step 2):



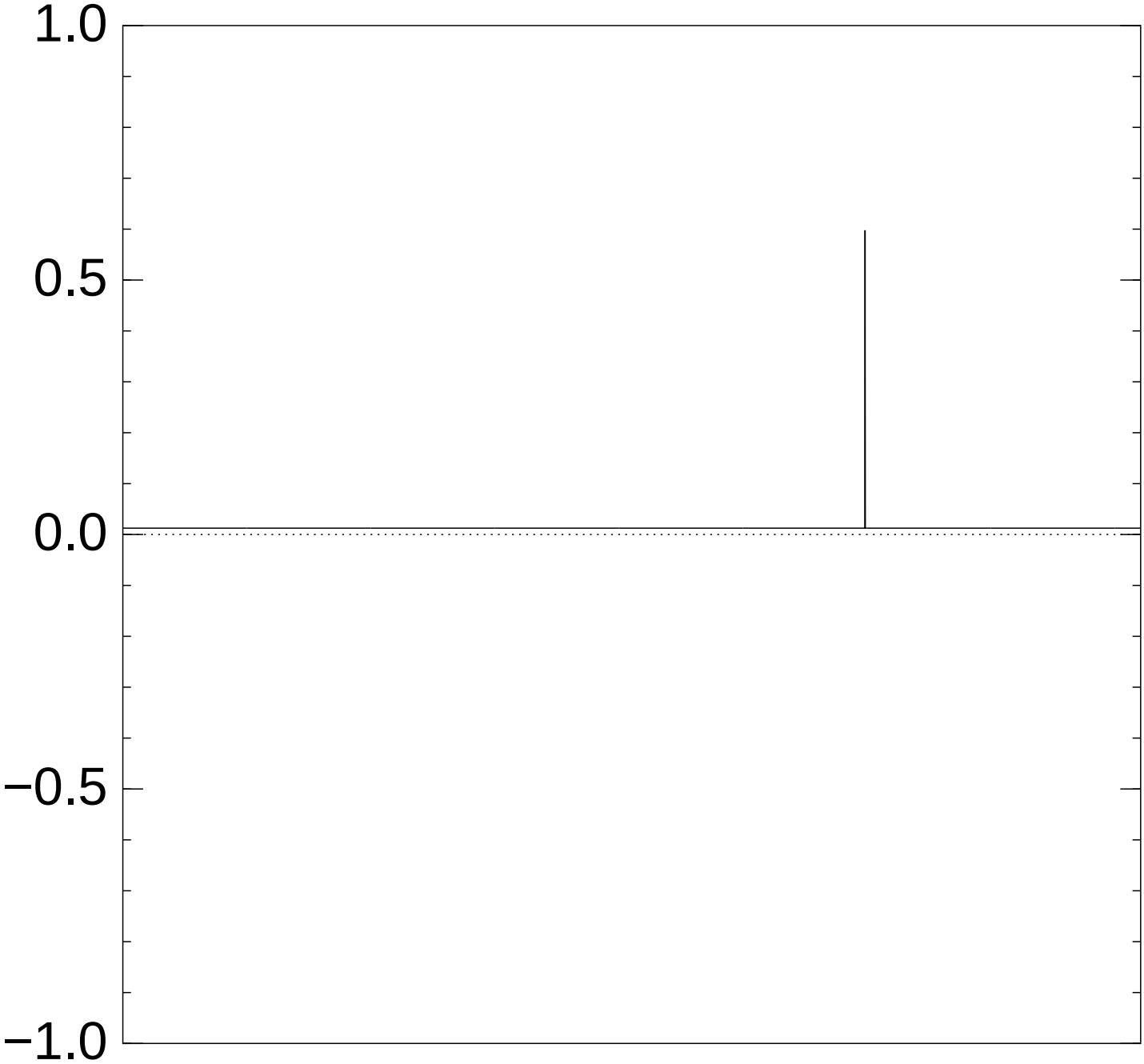
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $18 \times$ (Step 1 + Step 2):



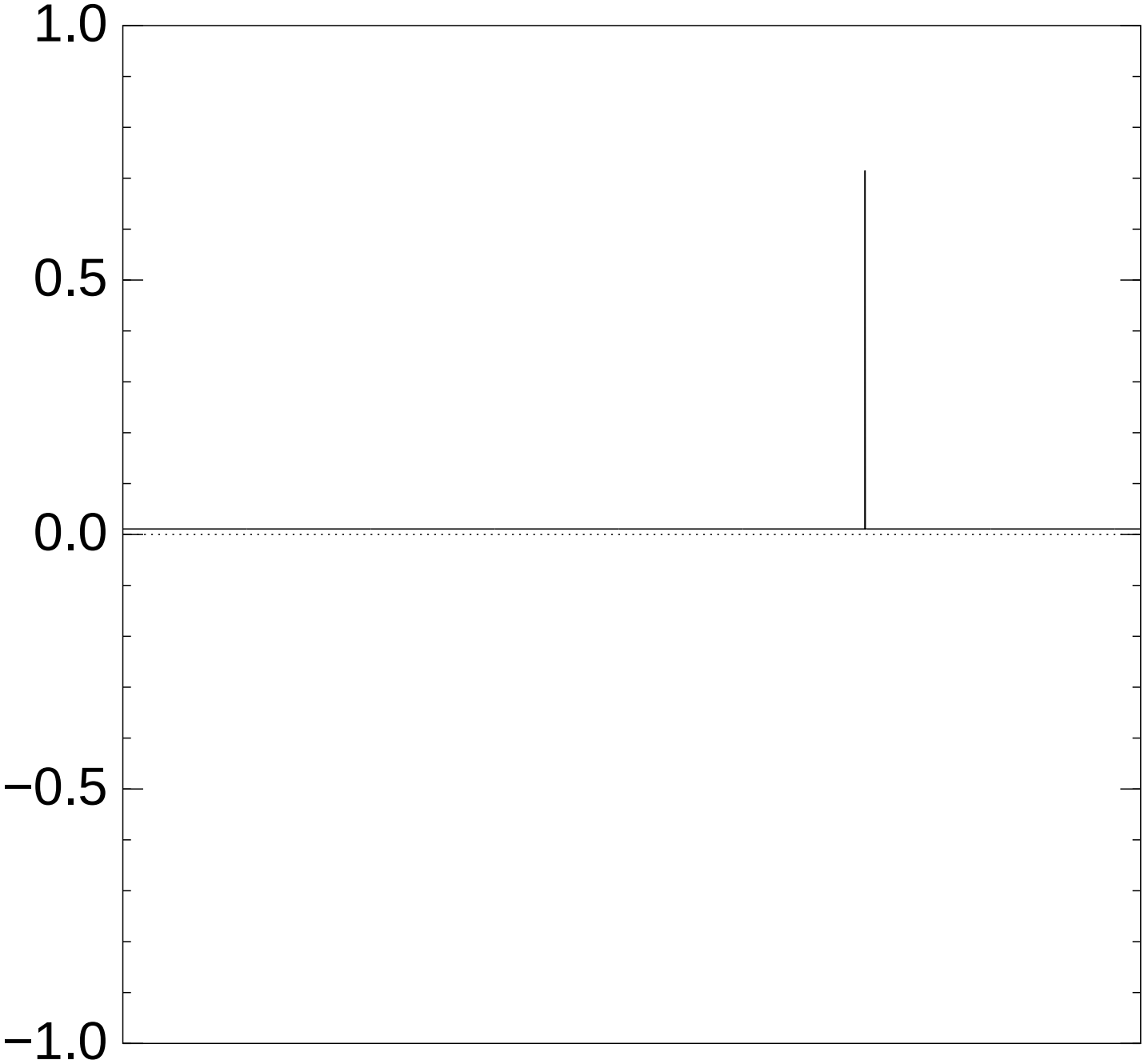
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $19 \times$ (Step 1 + Step 2):



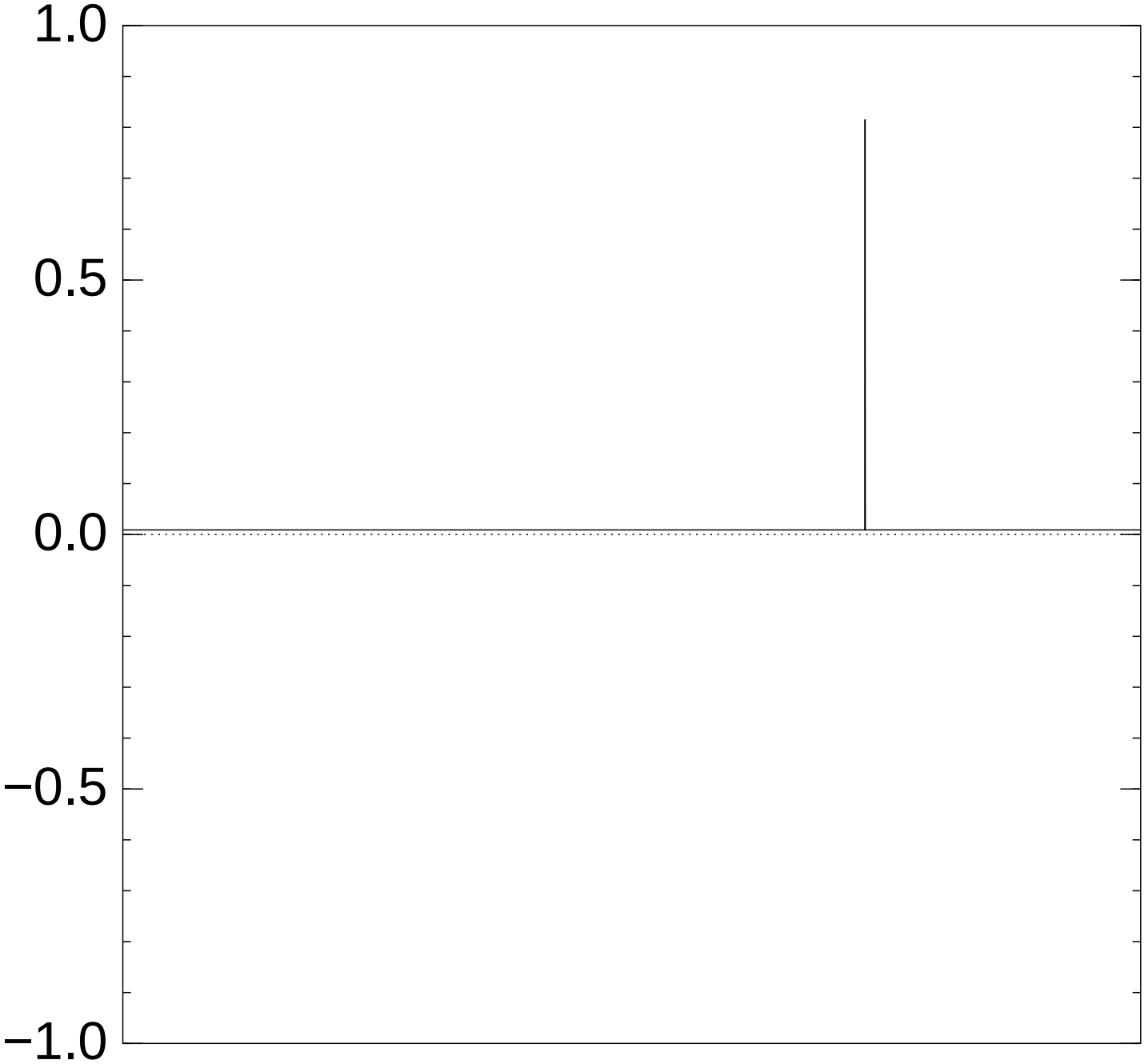
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $20 \times$ (Step 1 + Step 2):



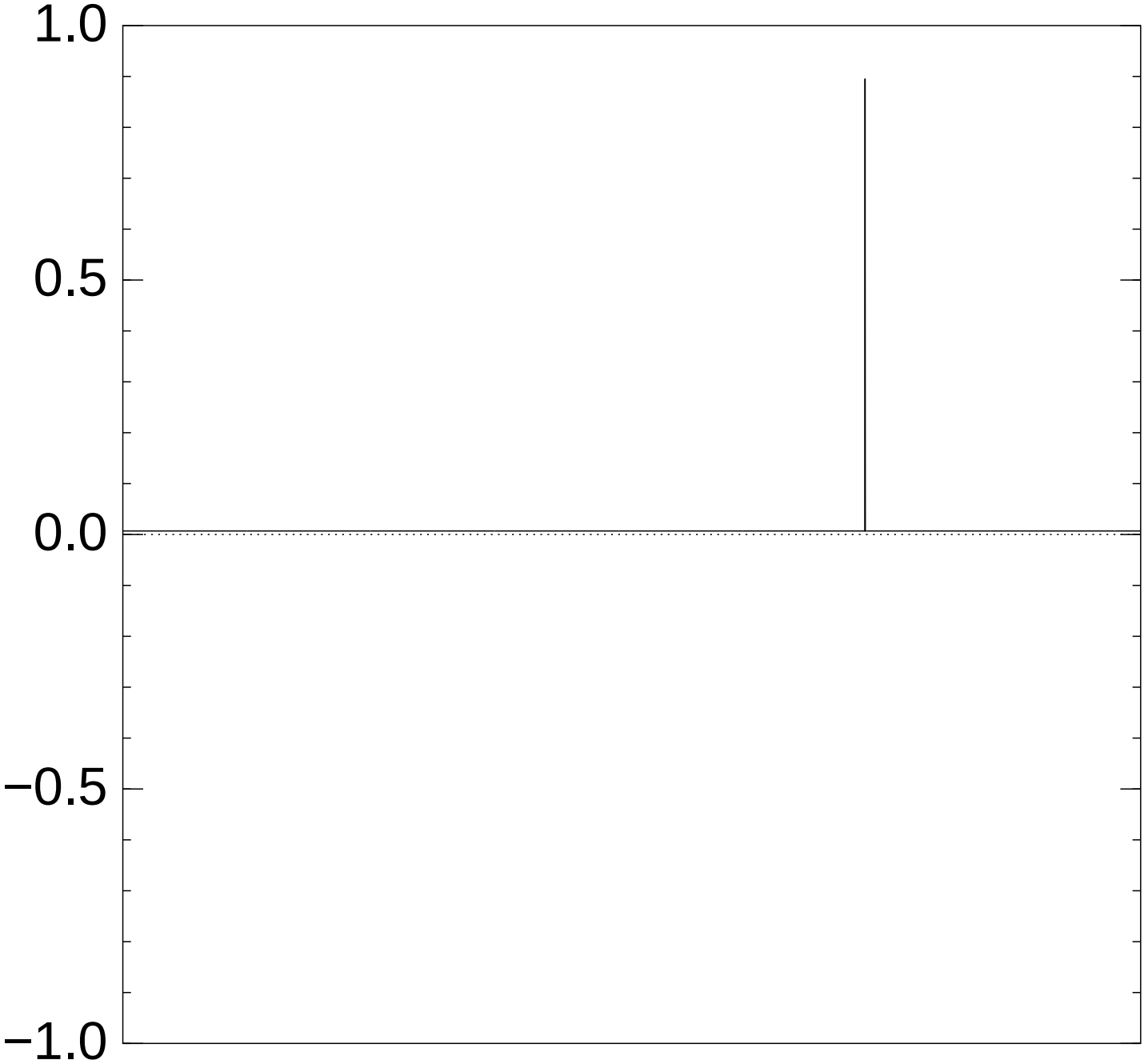
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $25 \times$ (Step 1 + Step 2):



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

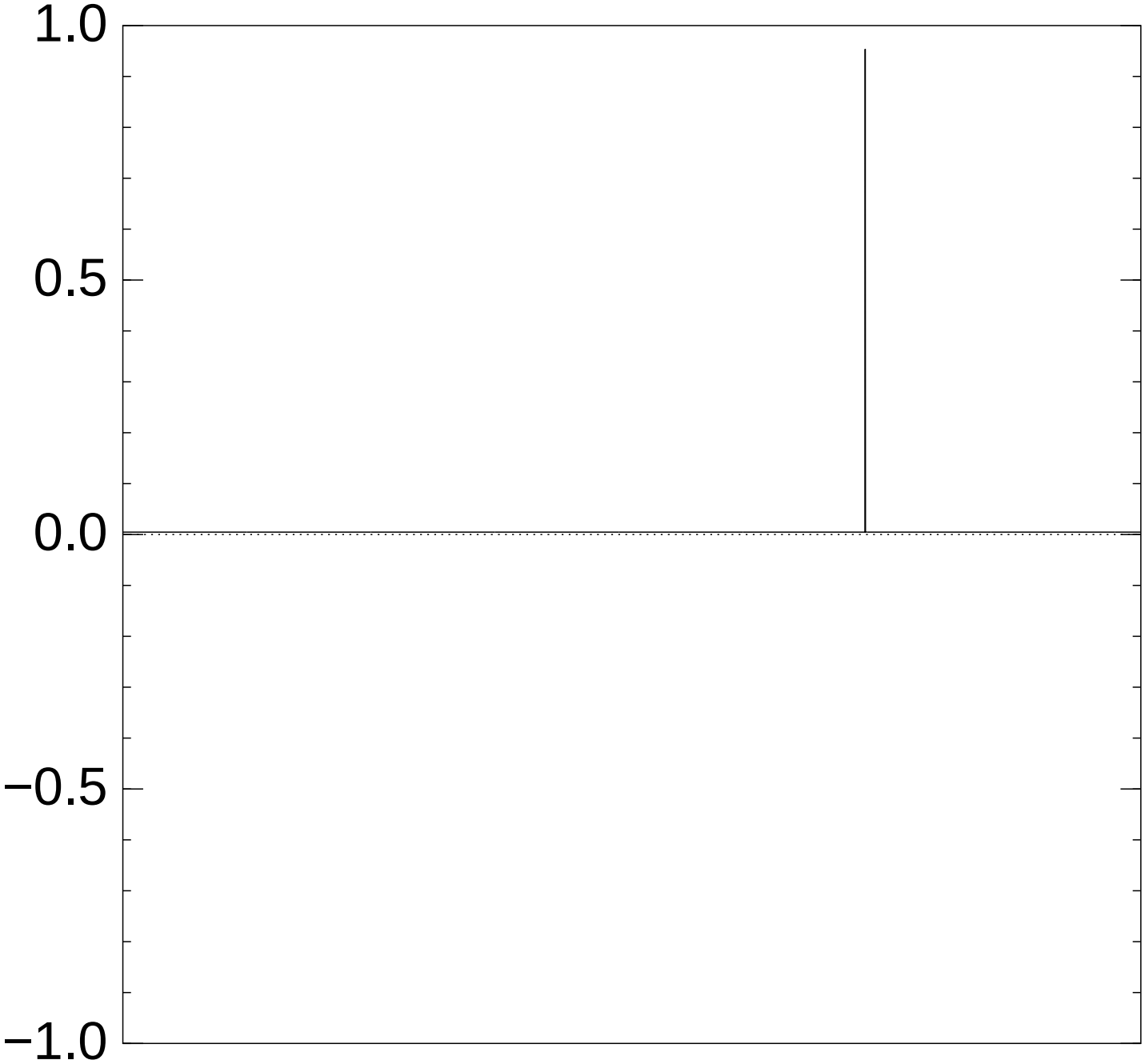


Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $35 \times$ (Step 1 + Step 2):

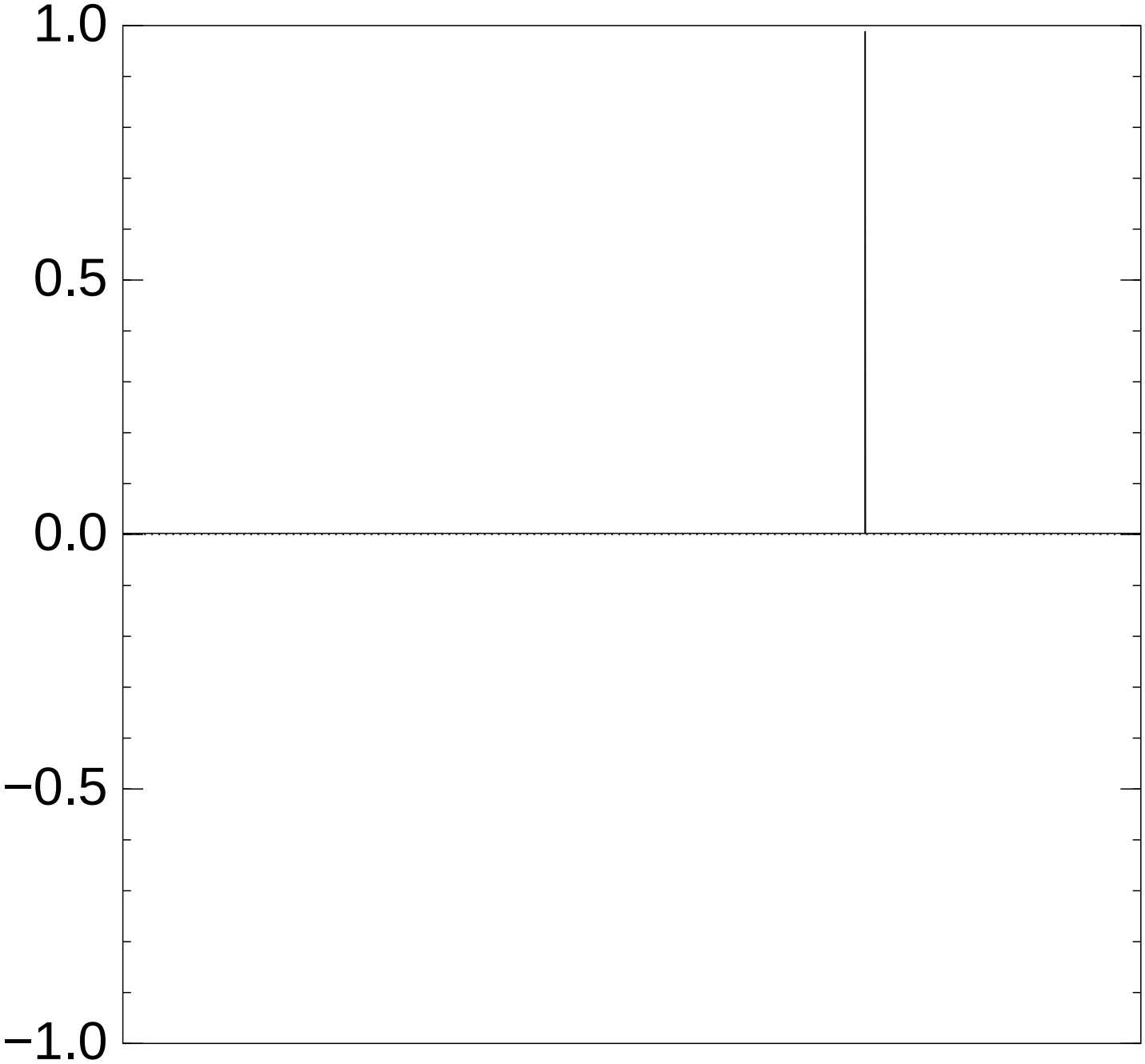


Good moment to stop, measure.

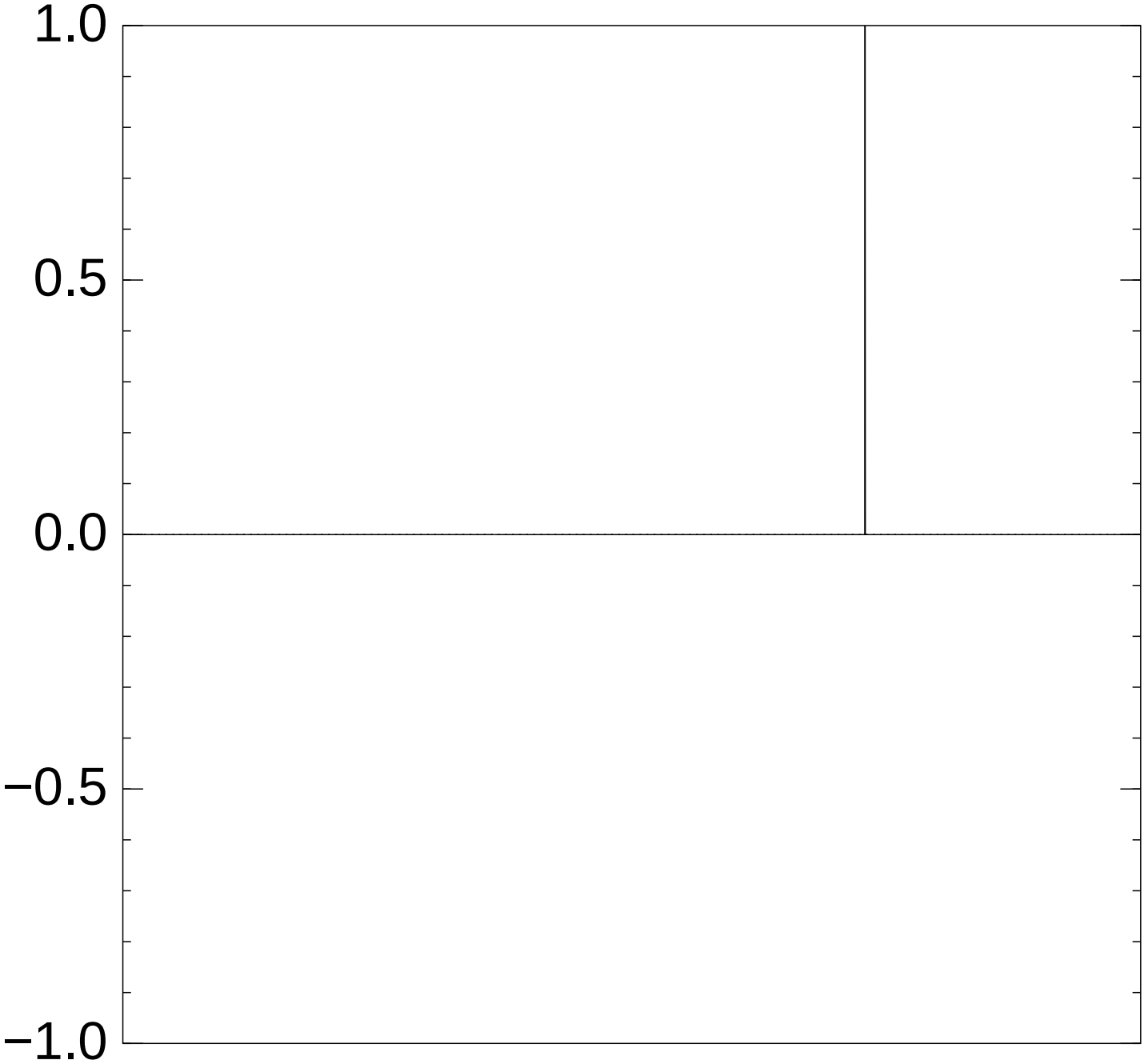
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $40 \times$ (Step 1 + Step 2):



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

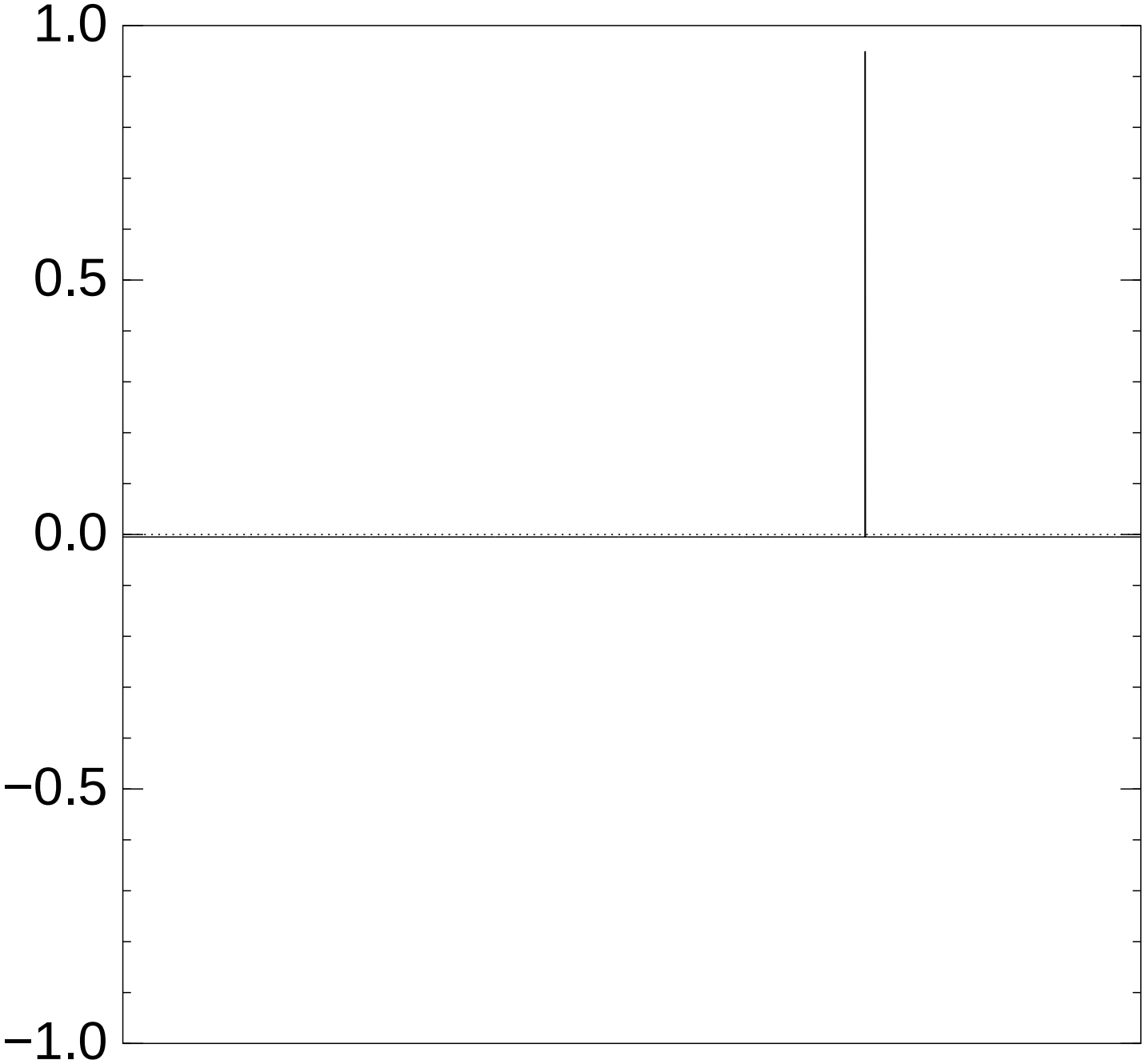


Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $50 \times$ (Step 1 + Step 2):

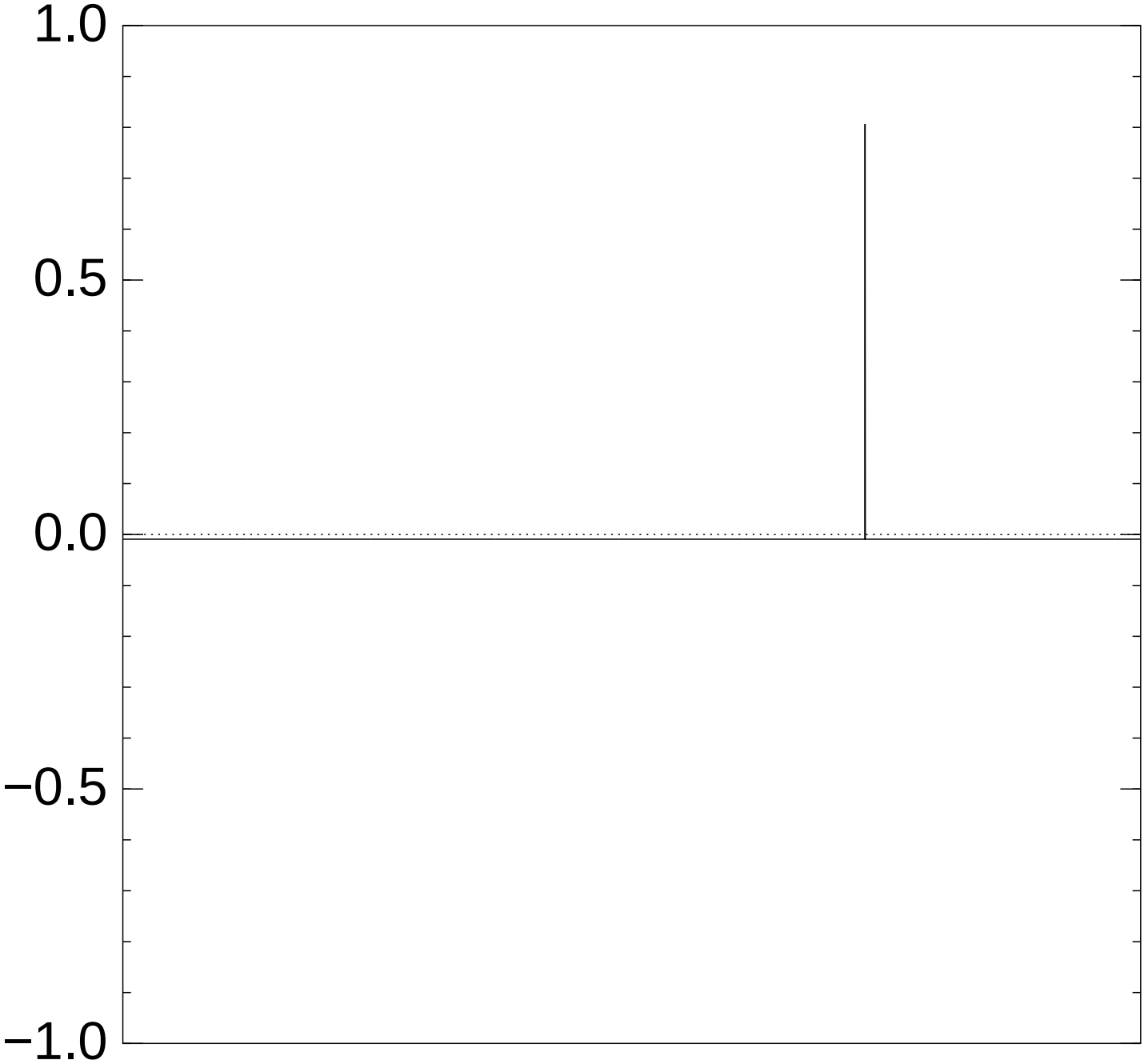


Traditional stopping point.

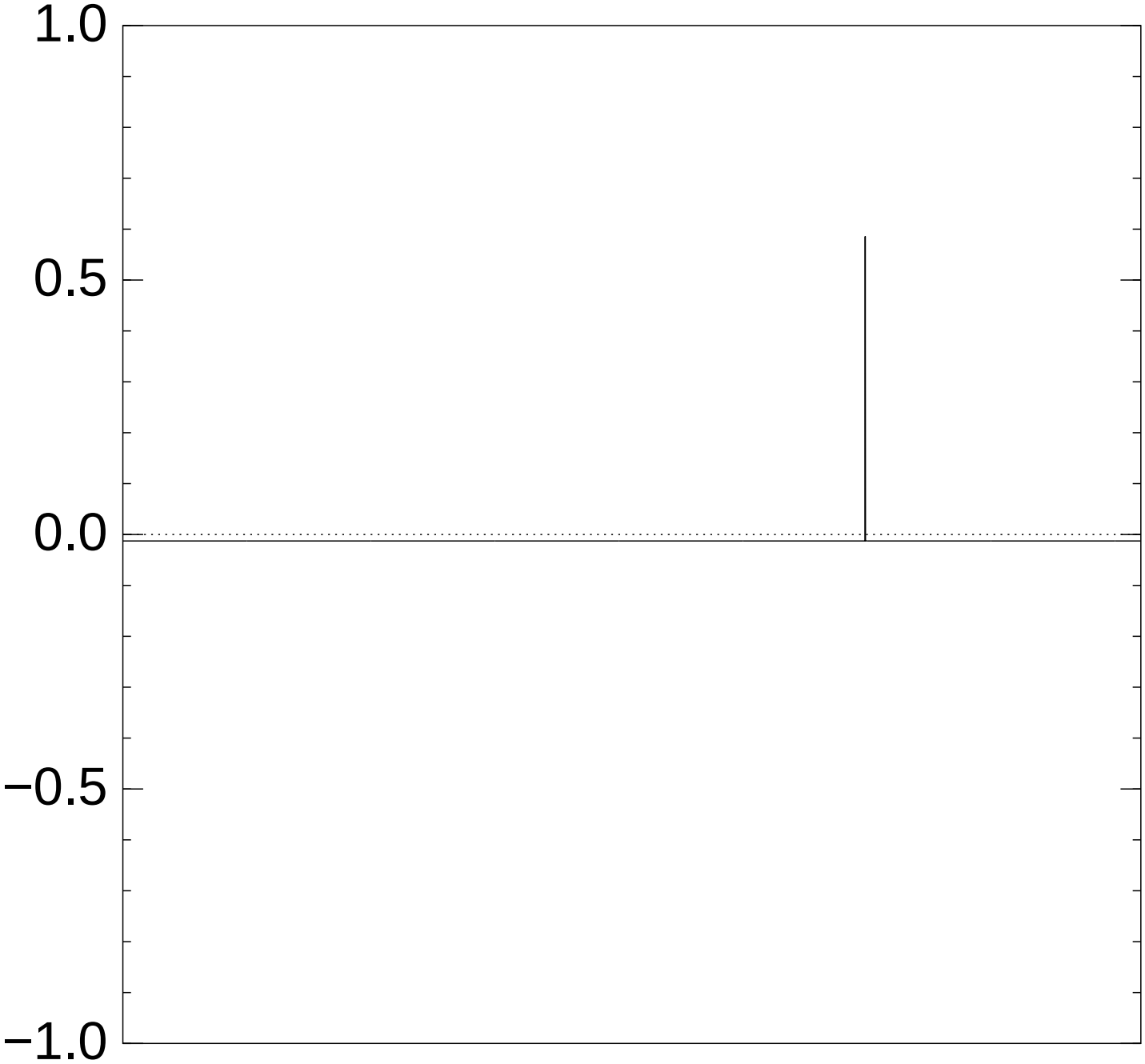
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $60 \times$ (Step 1 + Step 2):



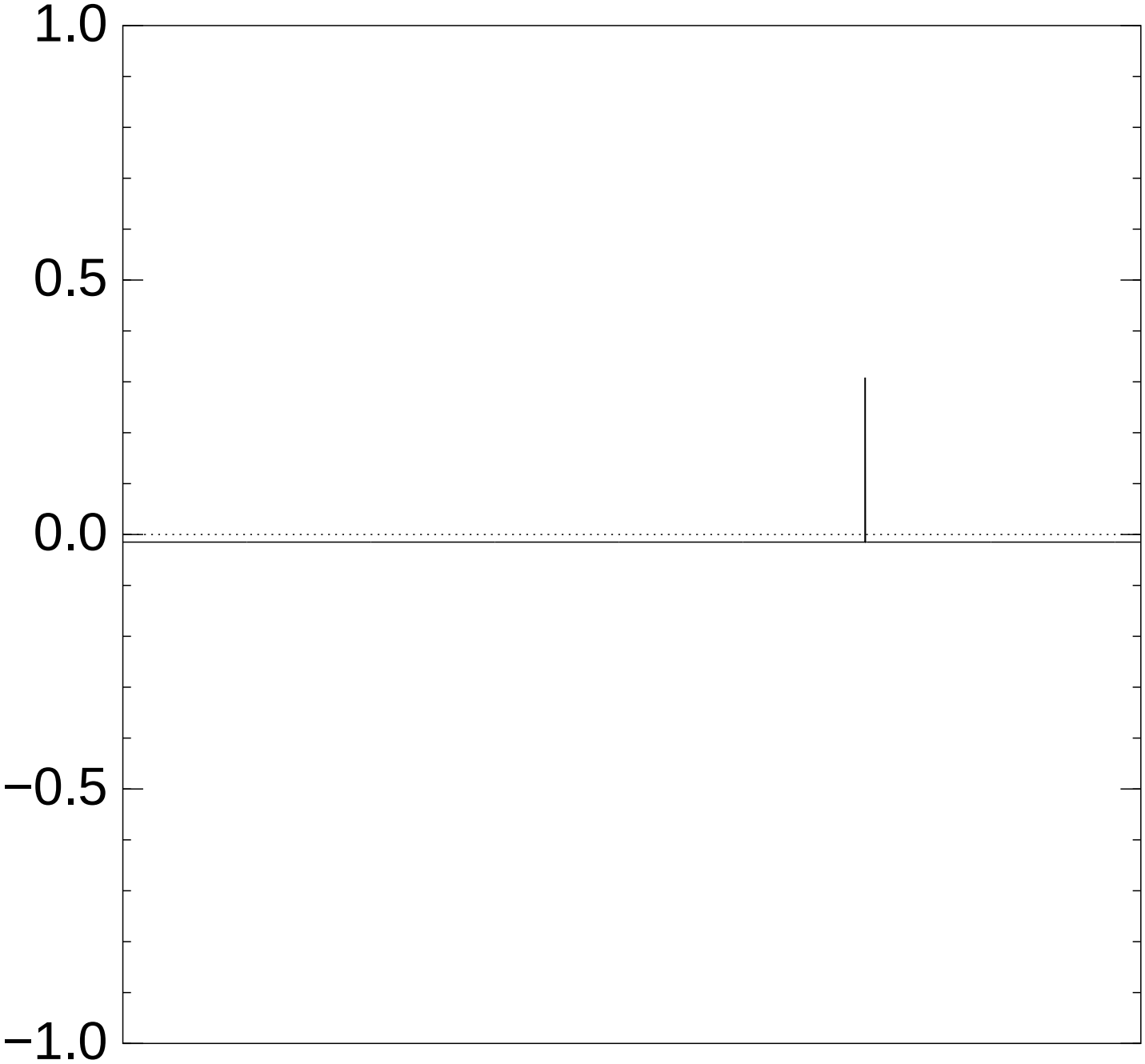
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $70 \times$ (Step 1 + Step 2):



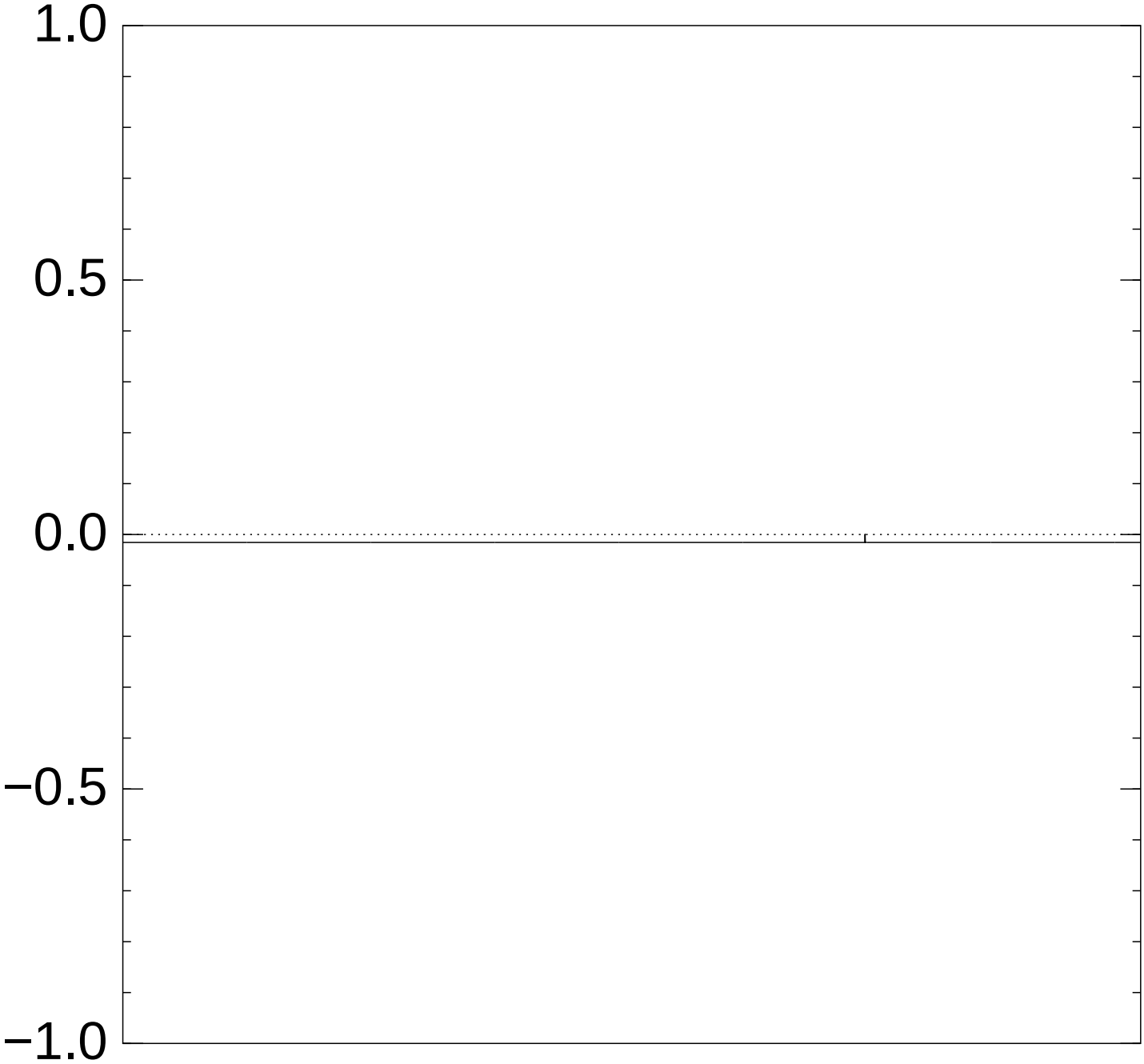
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $80 \times$ (Step 1 + Step 2):



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 $100 \times$ (Step 1 + 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 ≈ 1
after $\approx (\pi/4)2^{0.5n}$ iterations.

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor "hard" integers, etc. are outside this subset. Learn how to design quantum algorithms!

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor "hard" integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2021: Shor's algorithm and Grover's algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk: Start with one u; compute $f(u)$. Replace u by random neighbor; repeat enough times for mixing; check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

Extreme example of walk:

Completely unrestricted neighbors.

Recompute f at each step.

Mixes instantly. Same as Grover.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs,
exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm:
nearly 2^n calls to f.
Ambainis, using quantum walk:
$\approx 2^{2n/3}$ calls to f.

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$,
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$.
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S':
delete one elt, insert one elt.

Non-quantum setup cost σ;
then inner $\cdot$ outer loops $\sigma \cdot (2^n/\sigma)^2$.
Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.
Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: $\{$periods of $f\}$ is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: $\{$periods of $f\}$ is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

Shor's idea + extra work handles arbitrary finite groups with $O(n)$ evaluations of f. Massive caveat here: also need huge f-independent computation!

Kuperberg: For dihedral group, reduce the extra computation at some cost in f evaluations. Total cost is superpolynomial but subexponential: $2^{O(\sqrt{n})}$ evaluations of f + overhead.

Shor already handles some easy subgroups of the dihedral group. For hard cases, Kuperberg solves the "hidden-*shift* problem": find s in a commutative group given *two* functions f_0, f_1 satisfying $f_1(u) = f_0(u + s)$.

The impact on cryptography

2021.12: A Firefox connection
to `https://google.com`
is encrypted and authenticated
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH system,
with the key exchange signed
by an ECDSA-NIST-P-256 key,
with the signing key certified
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

However, AES-128 exposes many
protocols to multi-target attacks
that are already feasible today.
So use AES-256. (Or ChaCha20:
bigger security margin, no timing
attacks, no block-size attacks.)

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

For cryptographic risk
management, should presume
some Grover speedup depending
on quantum-computer progress,
but have to account for costs of
quantum evaluation of f.

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST received and posted 69 complete submissions.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven't been found yet.

Major directions so far in *quantum* cryptanalysis of PQC:

1. "Small" Grover applications.
Main design strategy:
Try to find attack components that can be viewed as huge searches for "good" objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

2. "Big" Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

4. Shor applications.

Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments:
see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

6. Analyzing and optimizing costs of all of these algorithms in much more detail.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

6. Analyzing and optimizing costs of all of these algorithms in much more detail.

7. Changing cryptosystems to enable attacks: e.g. "Please use your secret key on a quantum computer to decrypt the following superposition of ciphertexts."