Introduction to quantum algorithms and introduction to code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0, 1, \ldots, 2^n - 1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$:
Measurement produces 6.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$:
Measurement produces 6.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
$\circ$ complementing qubit 0
$\circ$ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" $=$
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" $=$
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a “Toffoli gate” $=$
“controlled controlled NOT gate”:
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" =
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible computation

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

Reversible computation

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.

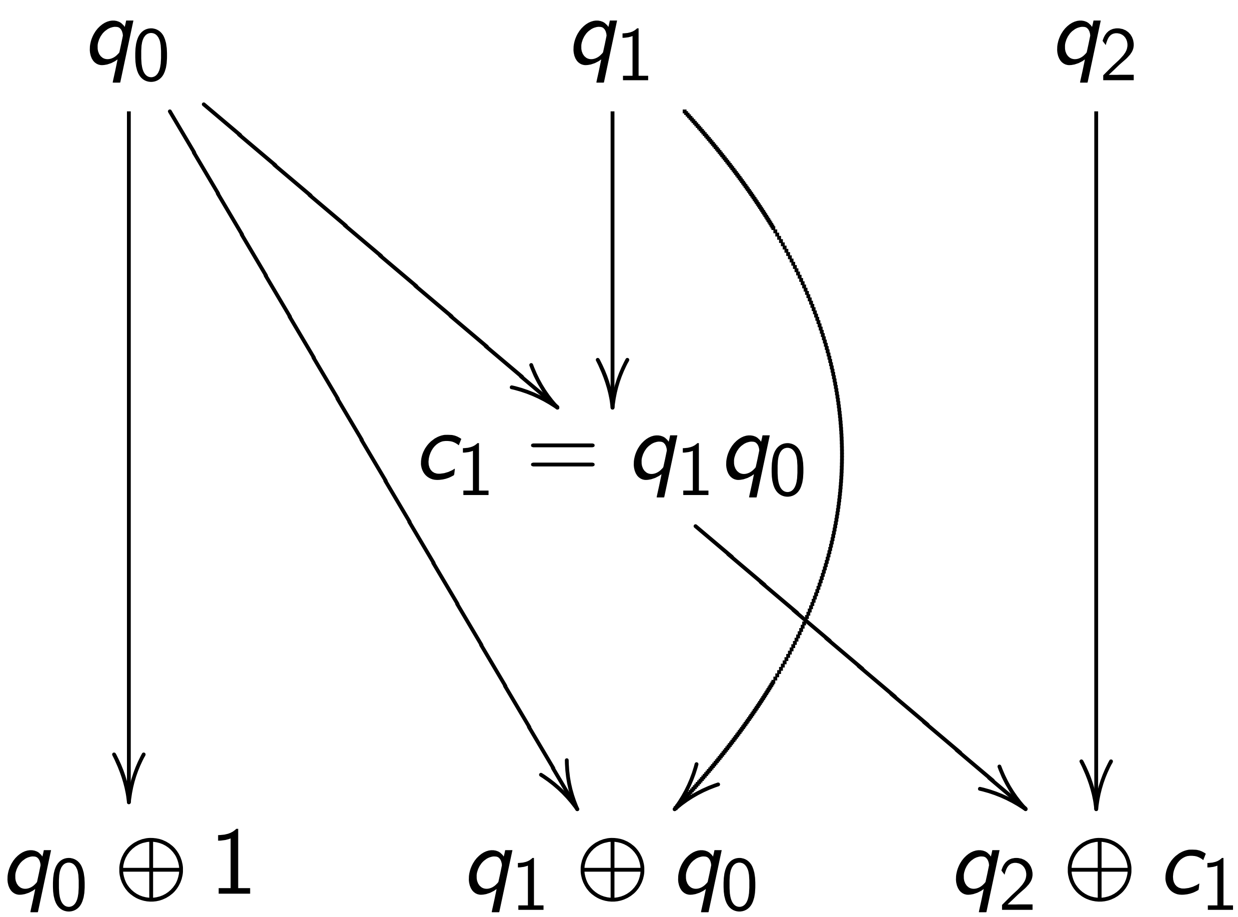
Example: Let's compute

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;

permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation example
was deceptively easy.

It didn't need many operations.

For large n, most permutations p
need many operations $\Rightarrow$ slow.
Really want *fast* circuits.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

inputs $b_1, b_2, \ldots, b_T$;

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
$(\text{inputs}) \mapsto$
$(\text{inputs}, \text{dirt}, \text{outputs})$.

Dirty reversible computation:
$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$
$(\text{inputs}, \text{dirt}, \text{outputs})$.

Clean reversible computation:
$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$
$(\text{inputs}, \text{zeros}, \text{outputs})$.

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Crude "poly-time" analyses
don't care about this,
but serious cryptanalysis
is much more precise.

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Fast quantum operations, part 2

“Hadamard”:

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Repeat n times: e.g.,
$(1,0,0,\ldots,0) \mapsto (1,1,1,\ldots,1)$.

Measuring $(1,0,0,\ldots,0)$
always produces 0.

Measuring $(1,1,1,\ldots,1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1,1,1,\ldots,1)$ to "pure state"
$(1,0,0,\ldots,0)$.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0,1\}^n$.
Can we find this period s,
given a fast circuit for f?

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0,1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits
in pure zero state:
vector $(1, 0, 0, \ldots)$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits
in pure zero state:
vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard
to move first n qubits
into uniform superposition:
$(1, 1, 1, \ldots, 1, 0, 0, \ldots)$
with 2^n entries 1, others 0.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry, output is orthogonal to s.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

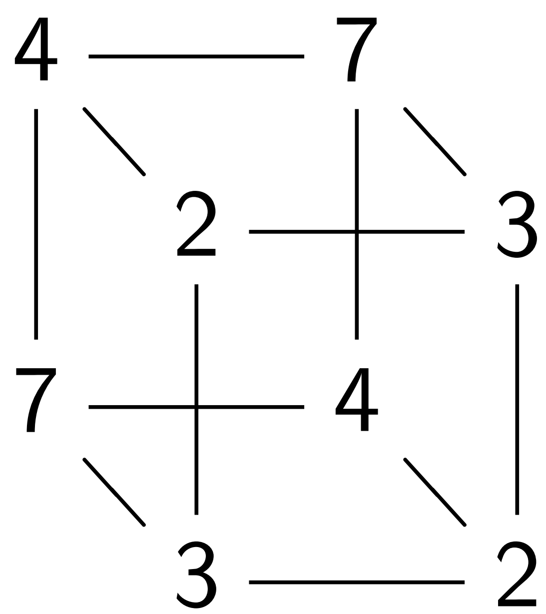
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

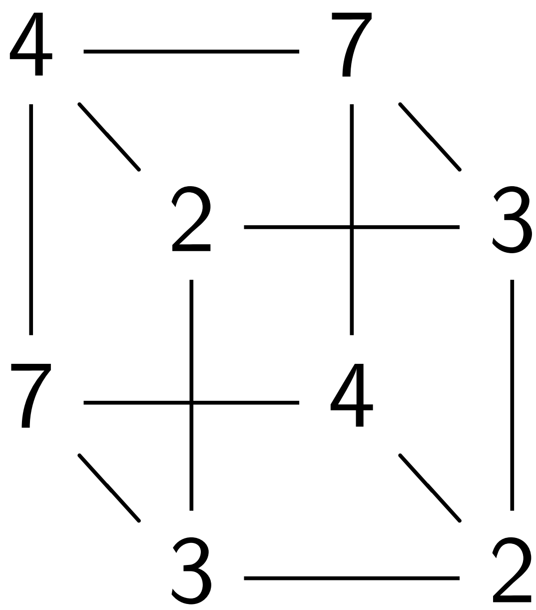
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that $f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm for f, using 6 qubits.

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.

Step 2. Hadamard on qubit 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.

Step 3. Hadamard on qubit 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.

Step 4. Hadamard on qubit 2:

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

Step 5. $(q, 0) \mapsto (q, f(q))$:

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

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

Step 6. Hadamard on qubit 0:

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

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

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

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

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

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

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

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

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

Step 7. Hadamard on qubit 1:

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

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

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

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

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

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

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

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

Step 8. Hadamard on qubit 2:

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

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

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

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

$2, 0, 2, 0, 0, 2, 0, 2,$

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

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

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

Step 8. Hadamard on qubit 2:

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

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

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

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

$2, 0, 2, 0, 0, 2, 0, 2,$

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

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

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

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

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 q.

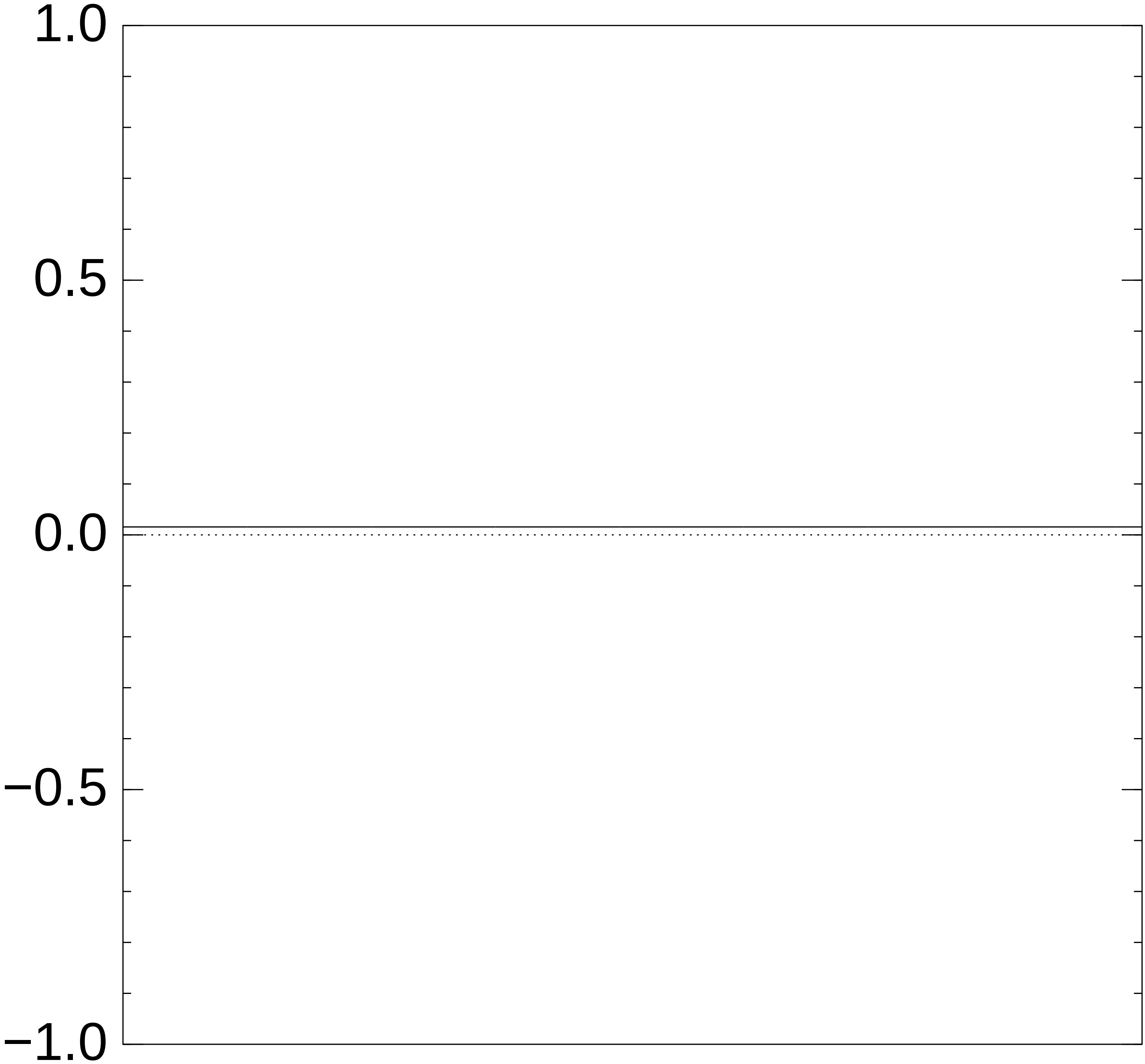
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

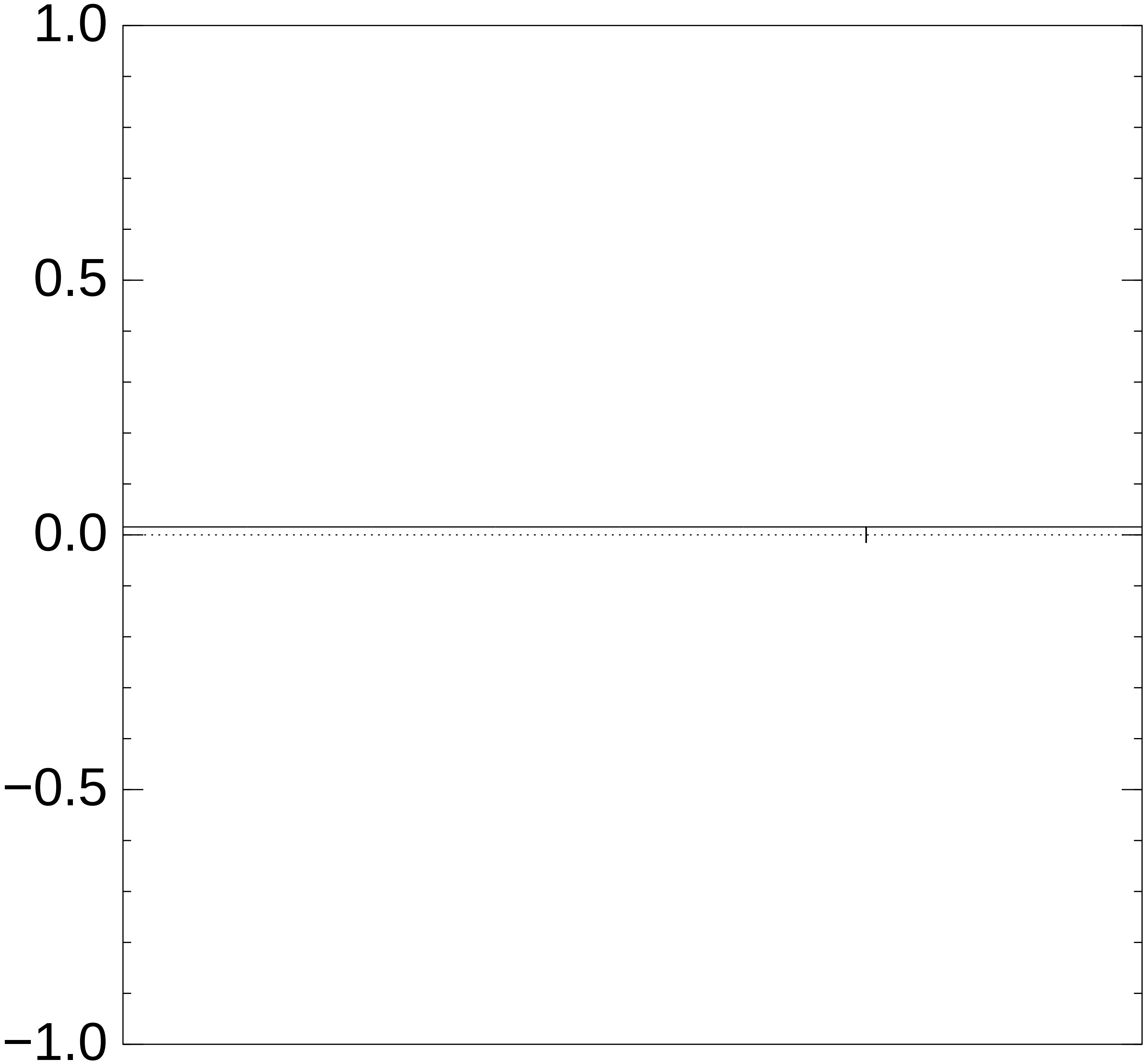
Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ times.

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

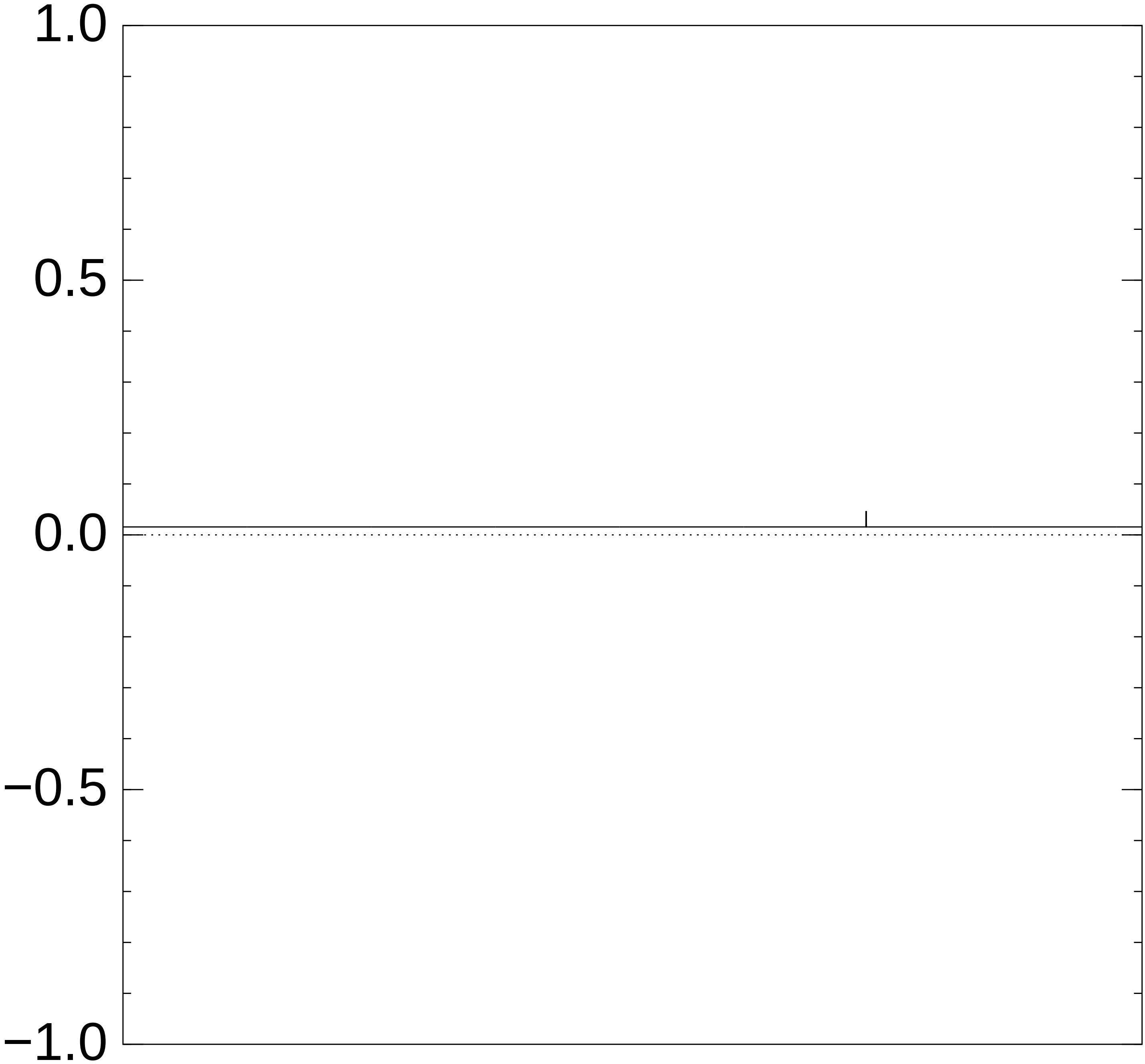
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after 0 steps:



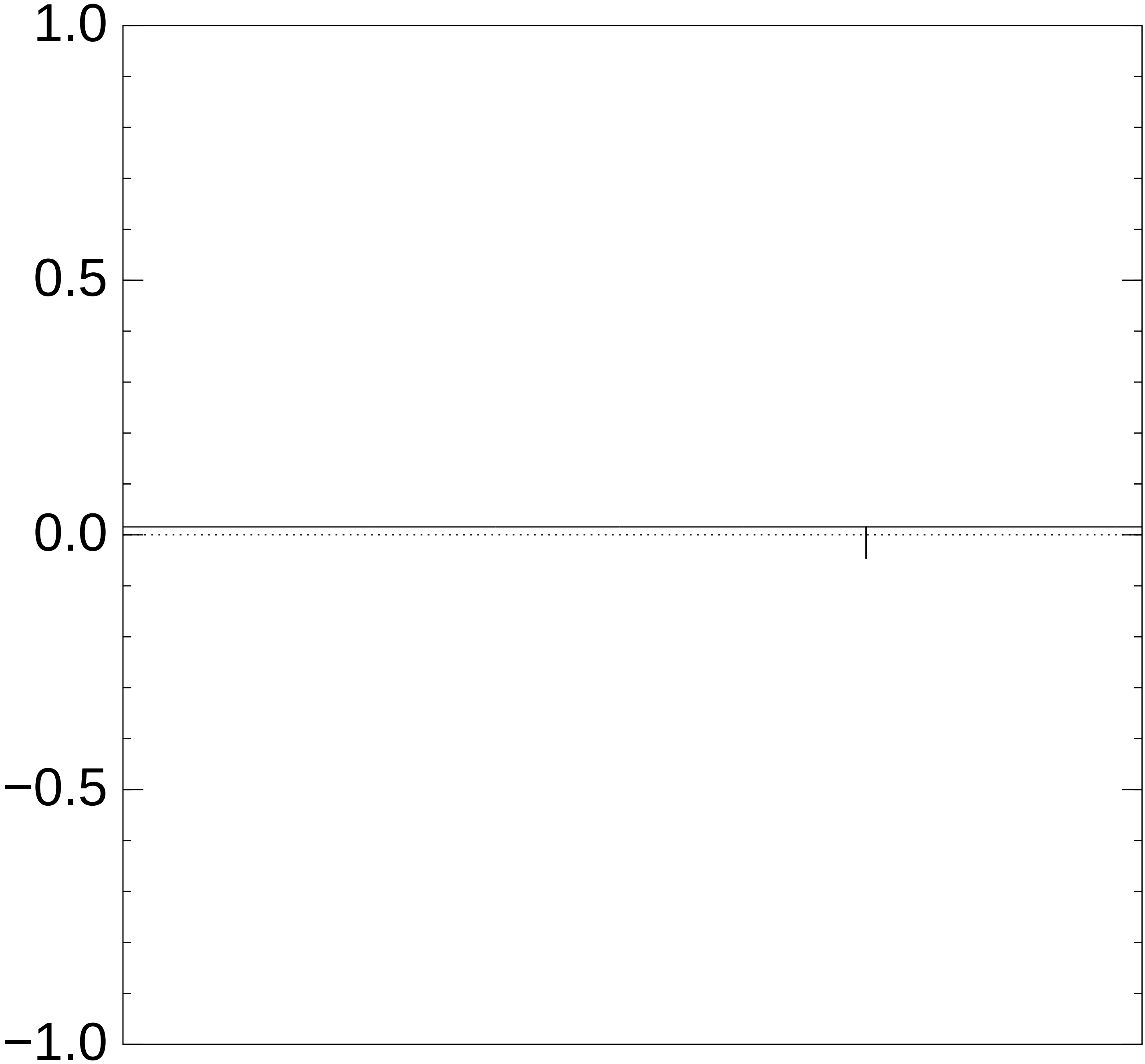
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1:



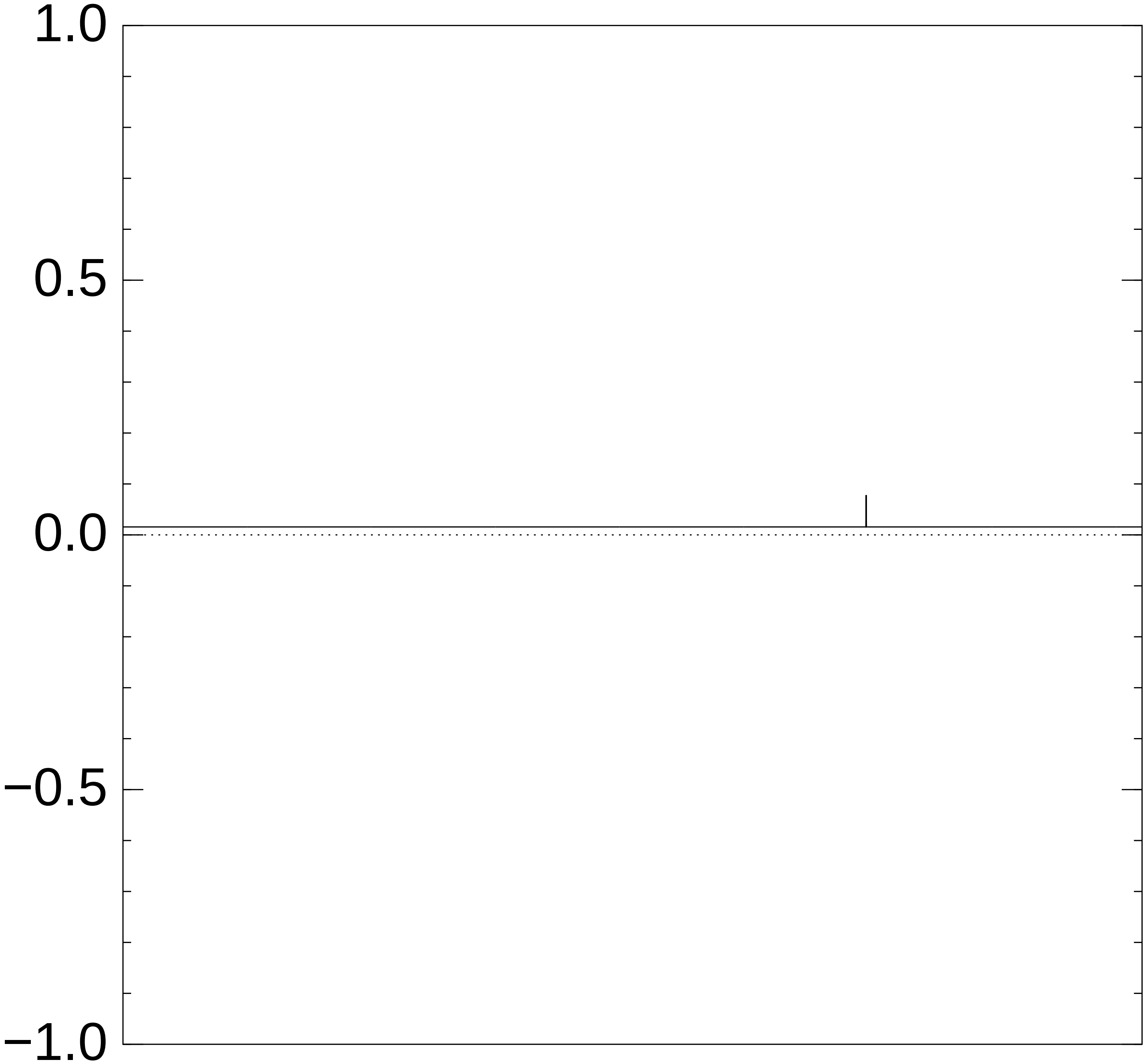
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step $1 +$ Step 2:



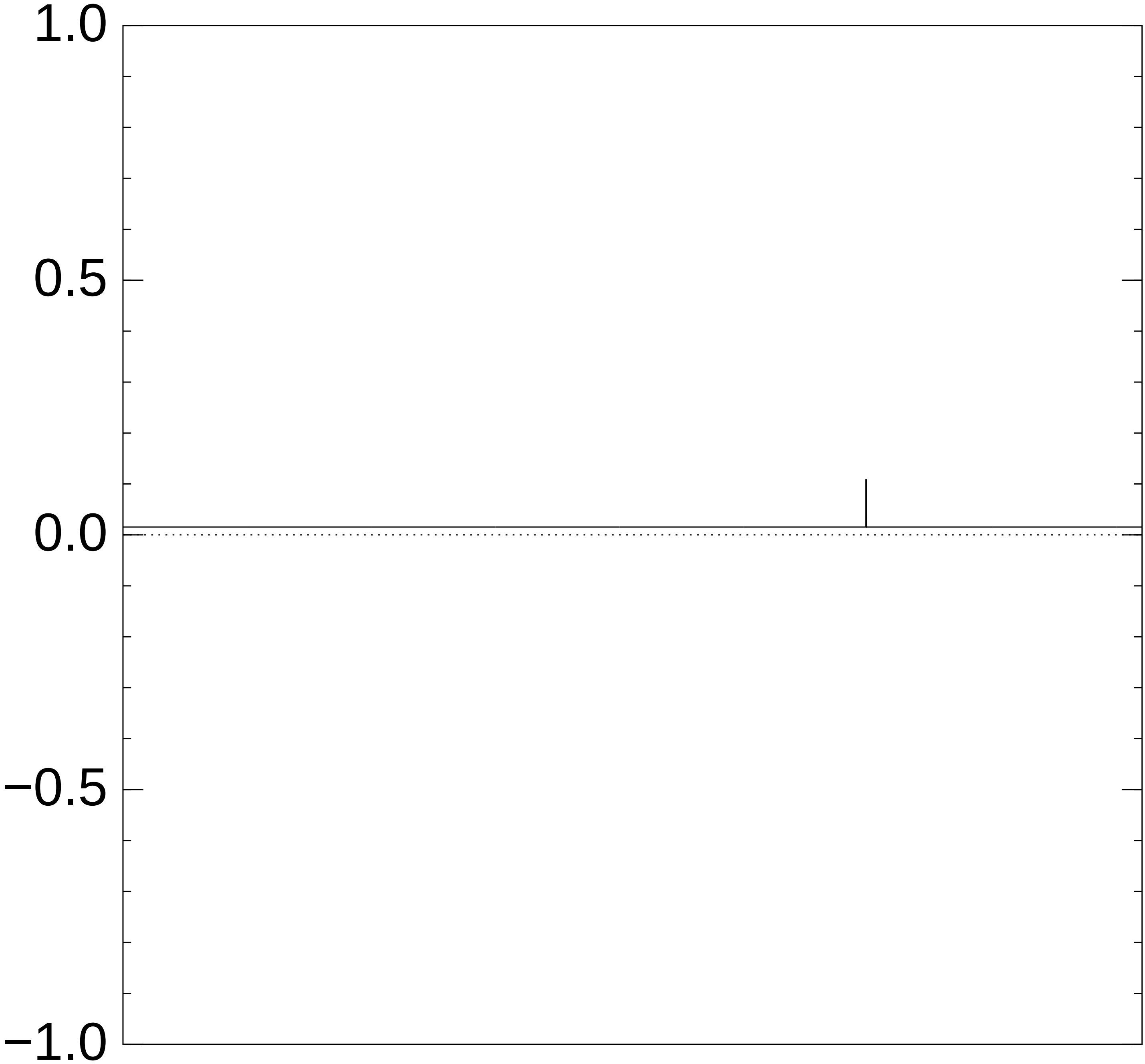
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after Step $1 +$ Step $2 +$ Step 1:



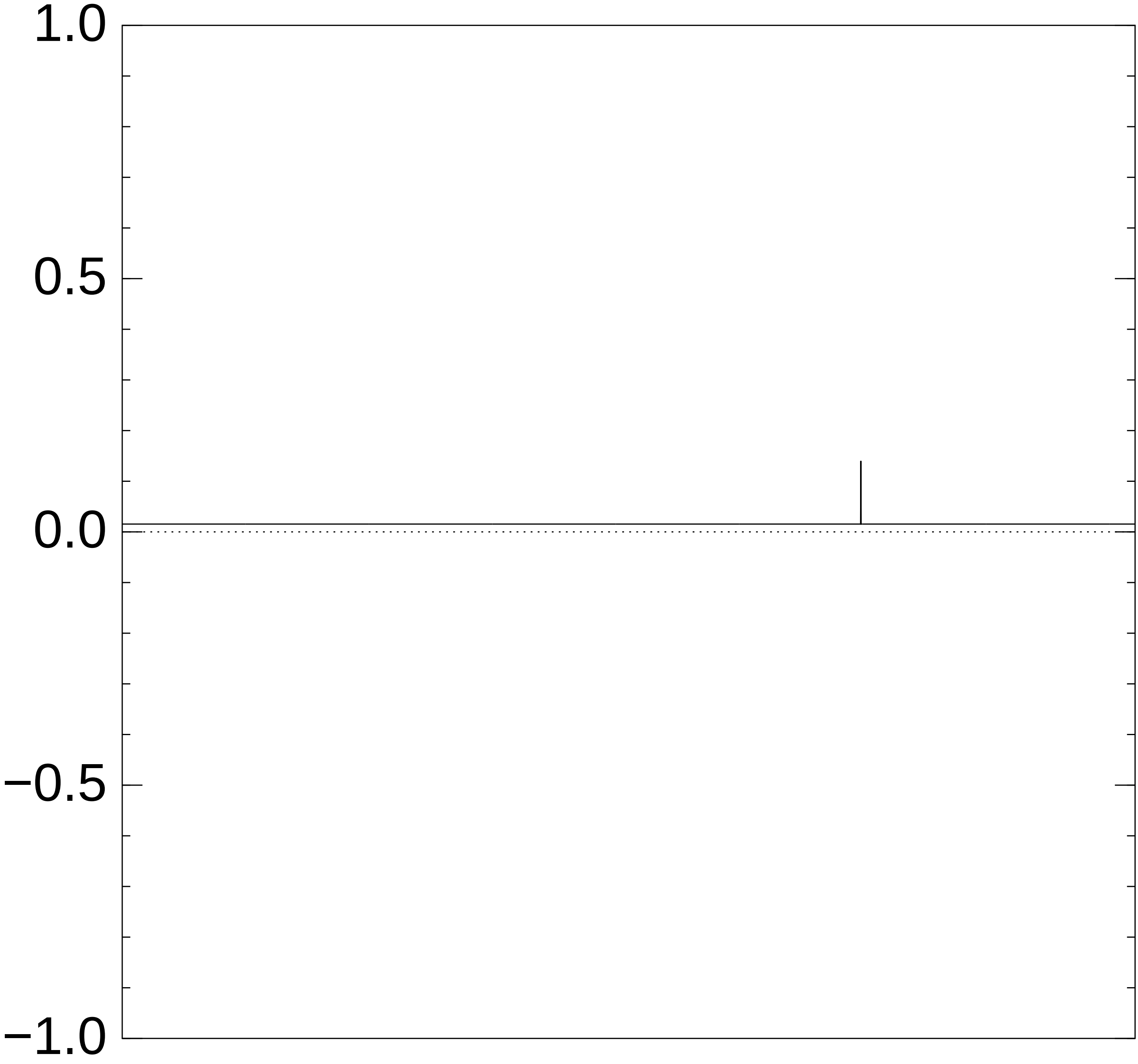
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $2 \times (\text{Step } 1 + \text{Step } 2)$:



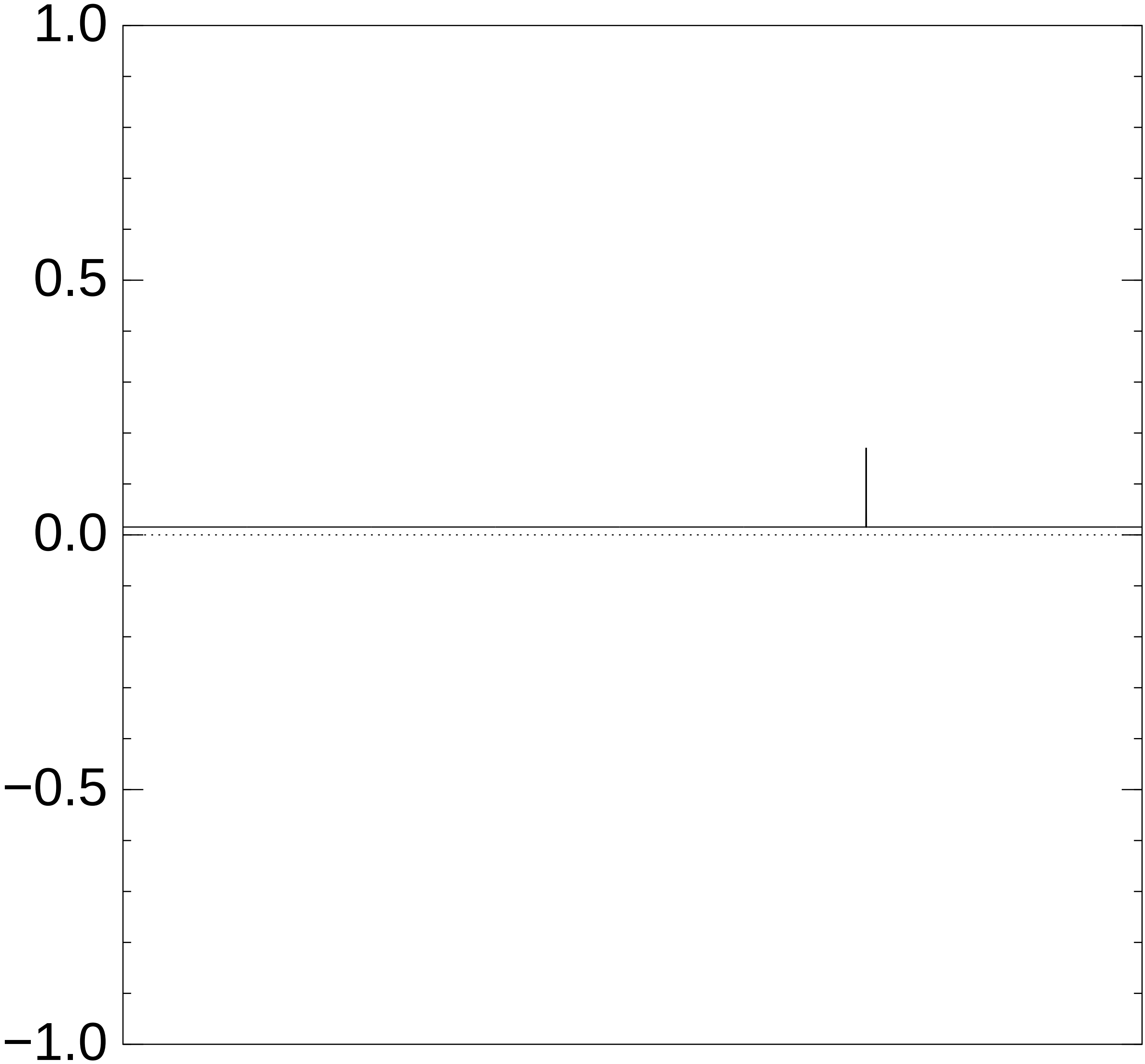
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $3 \times$ (Step $1 +$ Step 2):



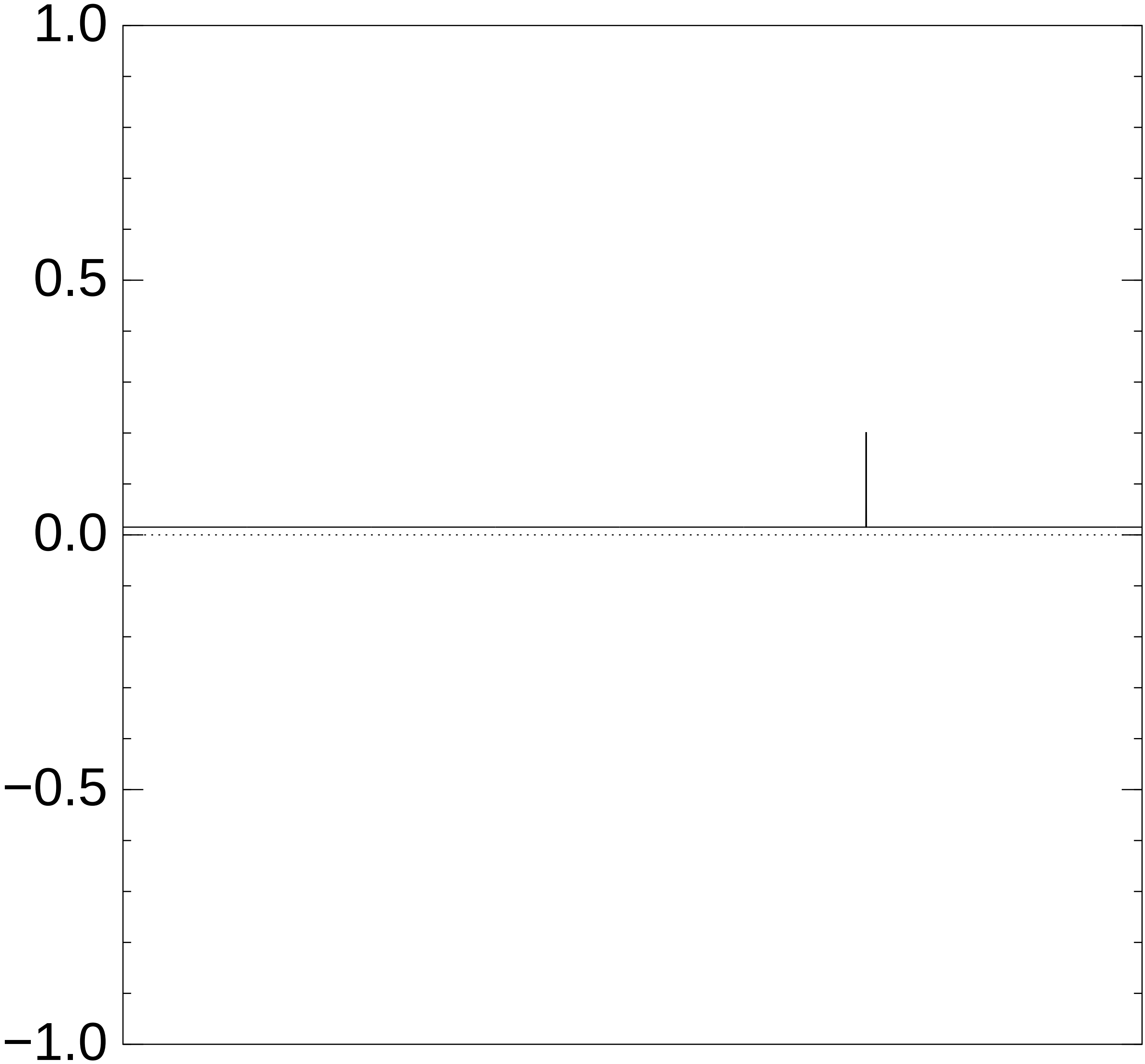
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $4 \times$ (Step 1 + Step 2):



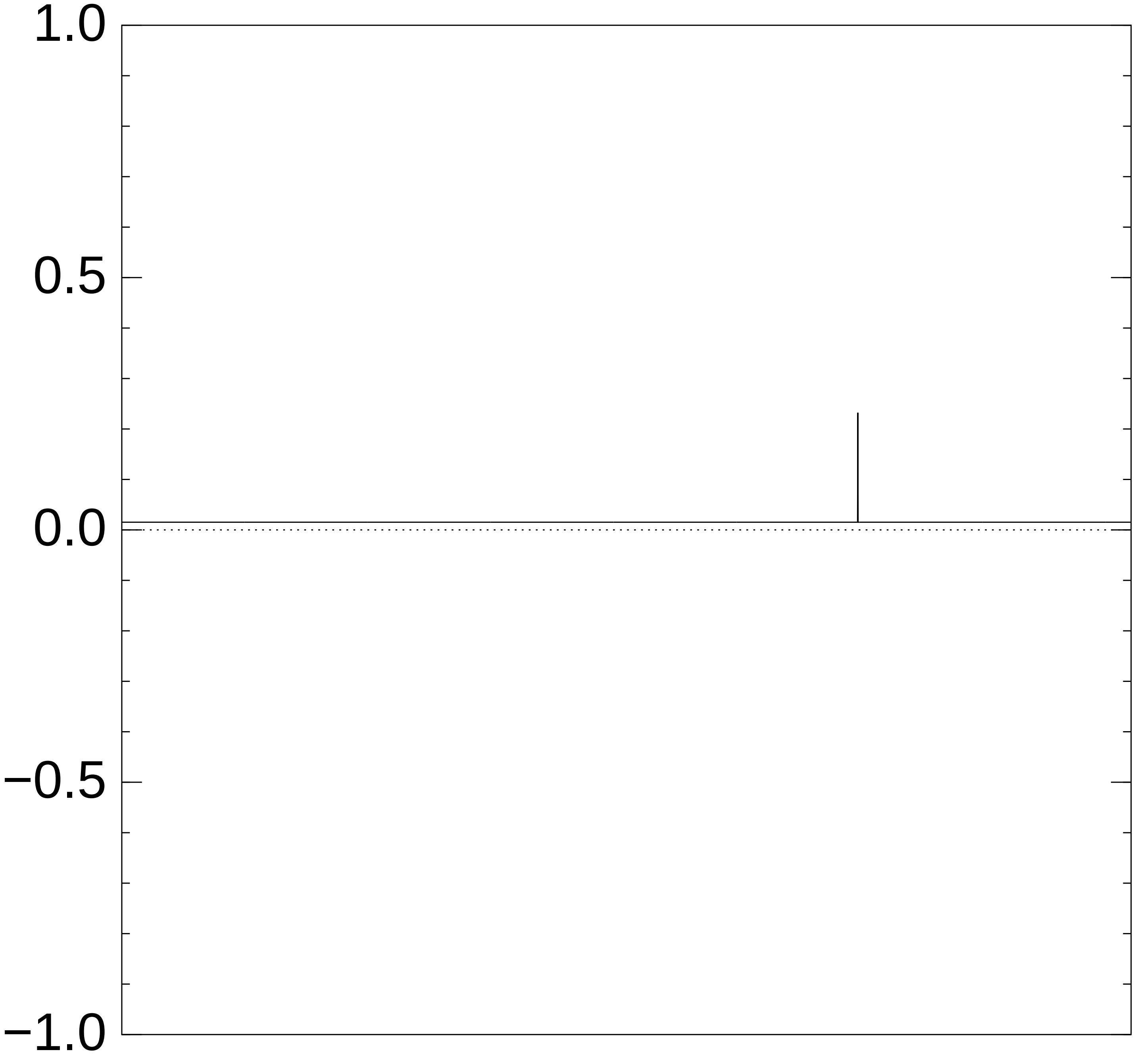
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $5 \times (\text{Step } 1 + \text{Step } 2)$:



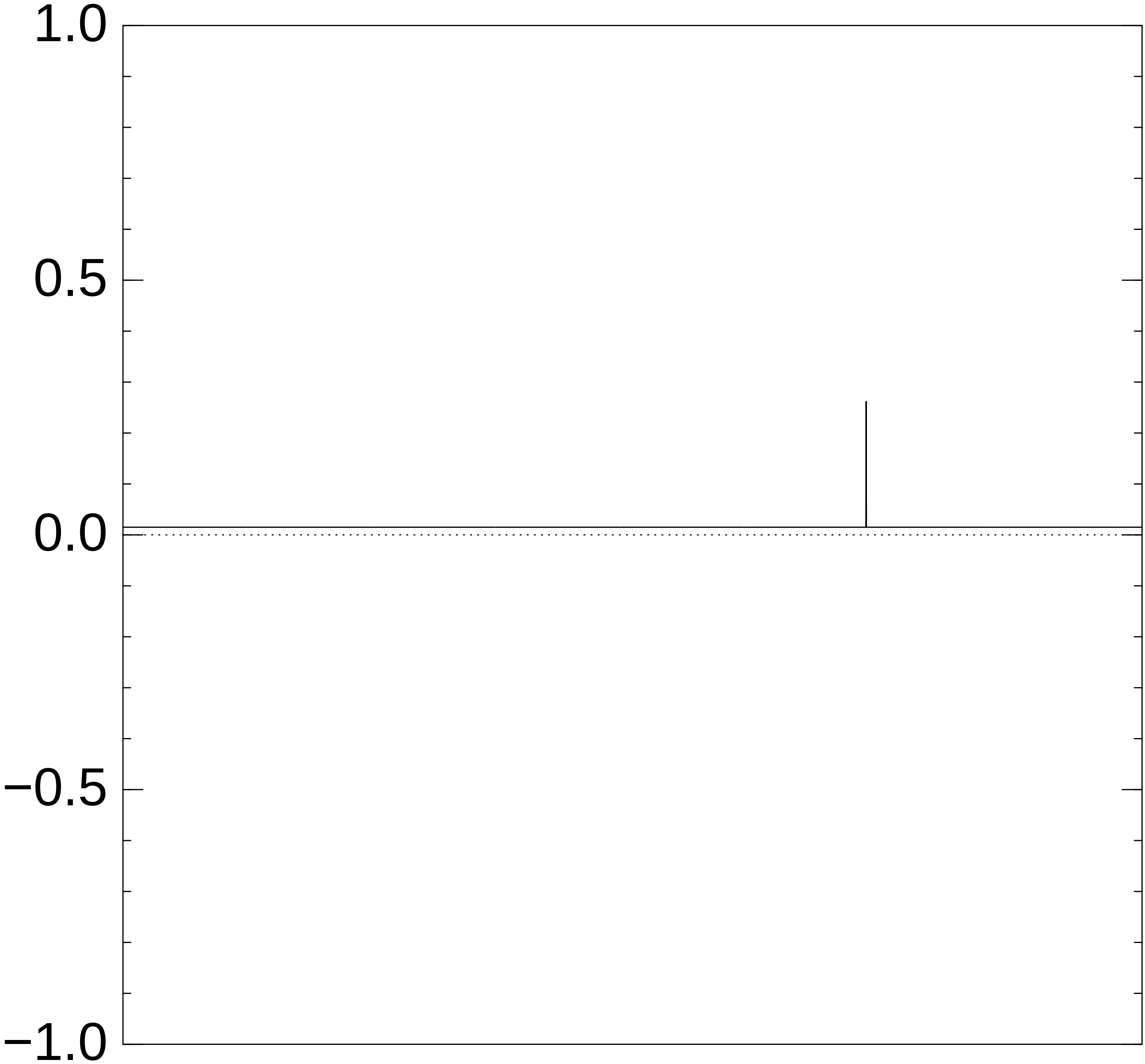
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $6 \times (\text{Step } 1 + \text{Step } 2)$:



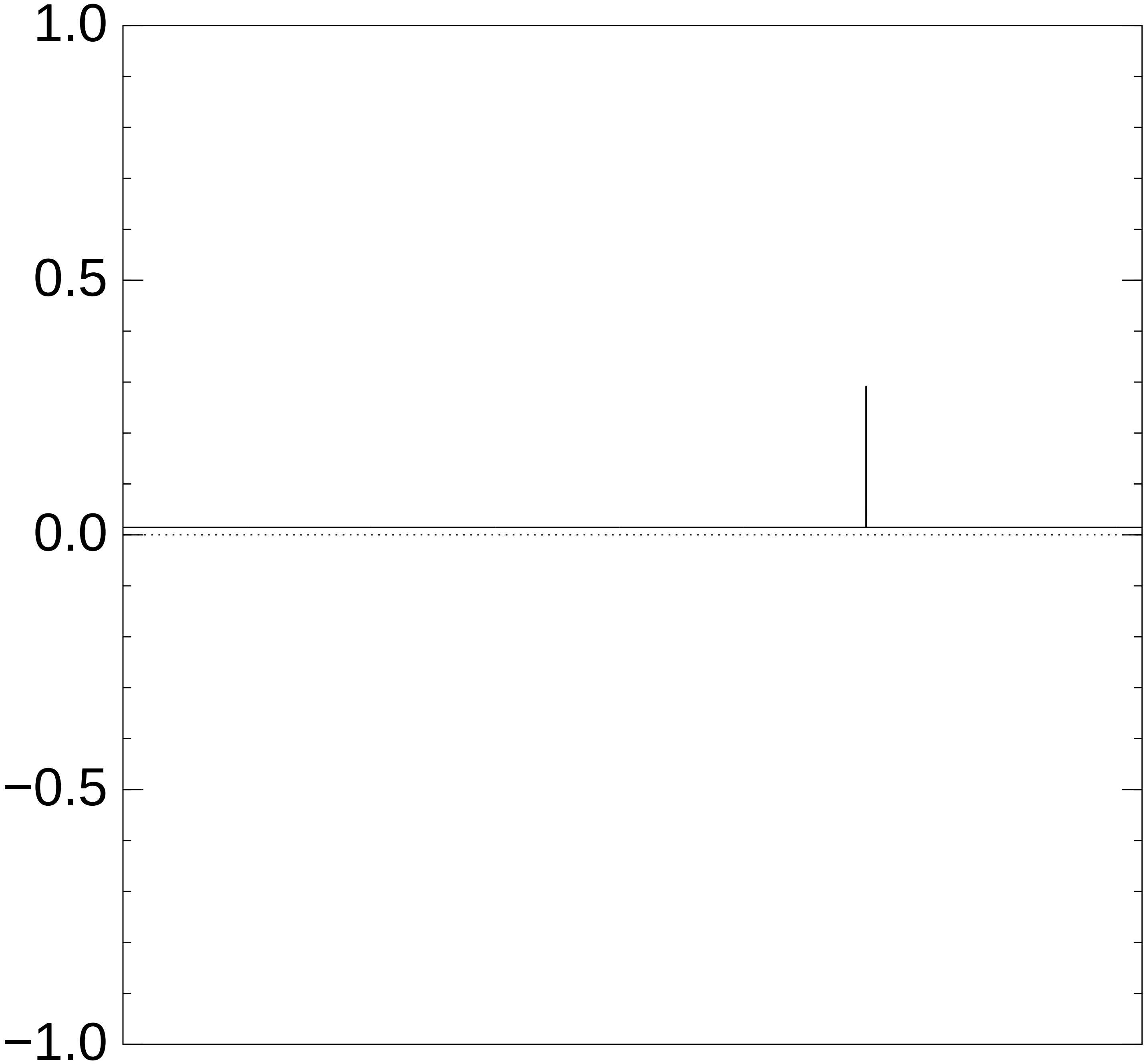
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $7 \times$ (Step $1 +$ Step 2):



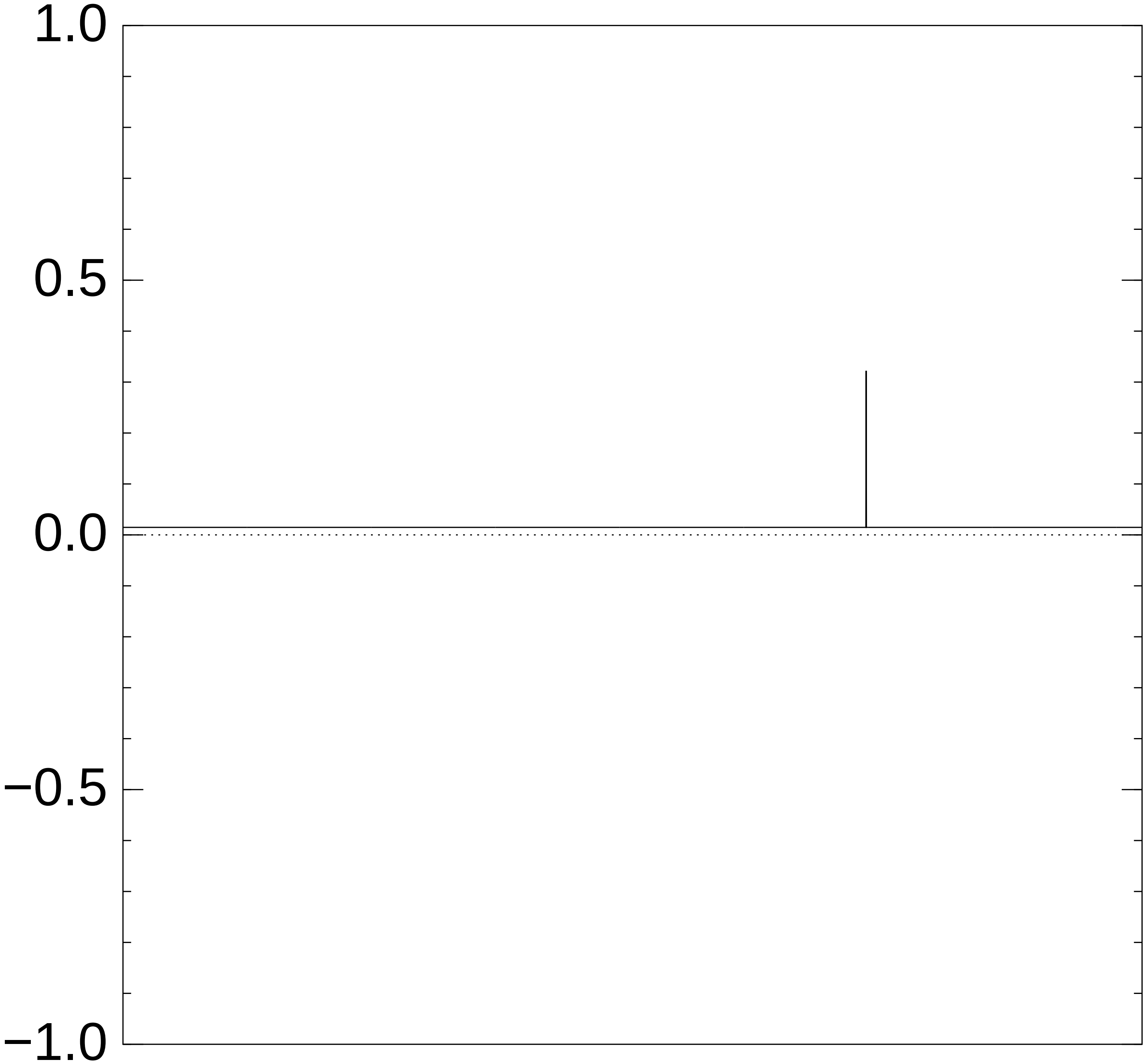
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $8 \times (\text{Step } 1 + \text{Step } 2)$:



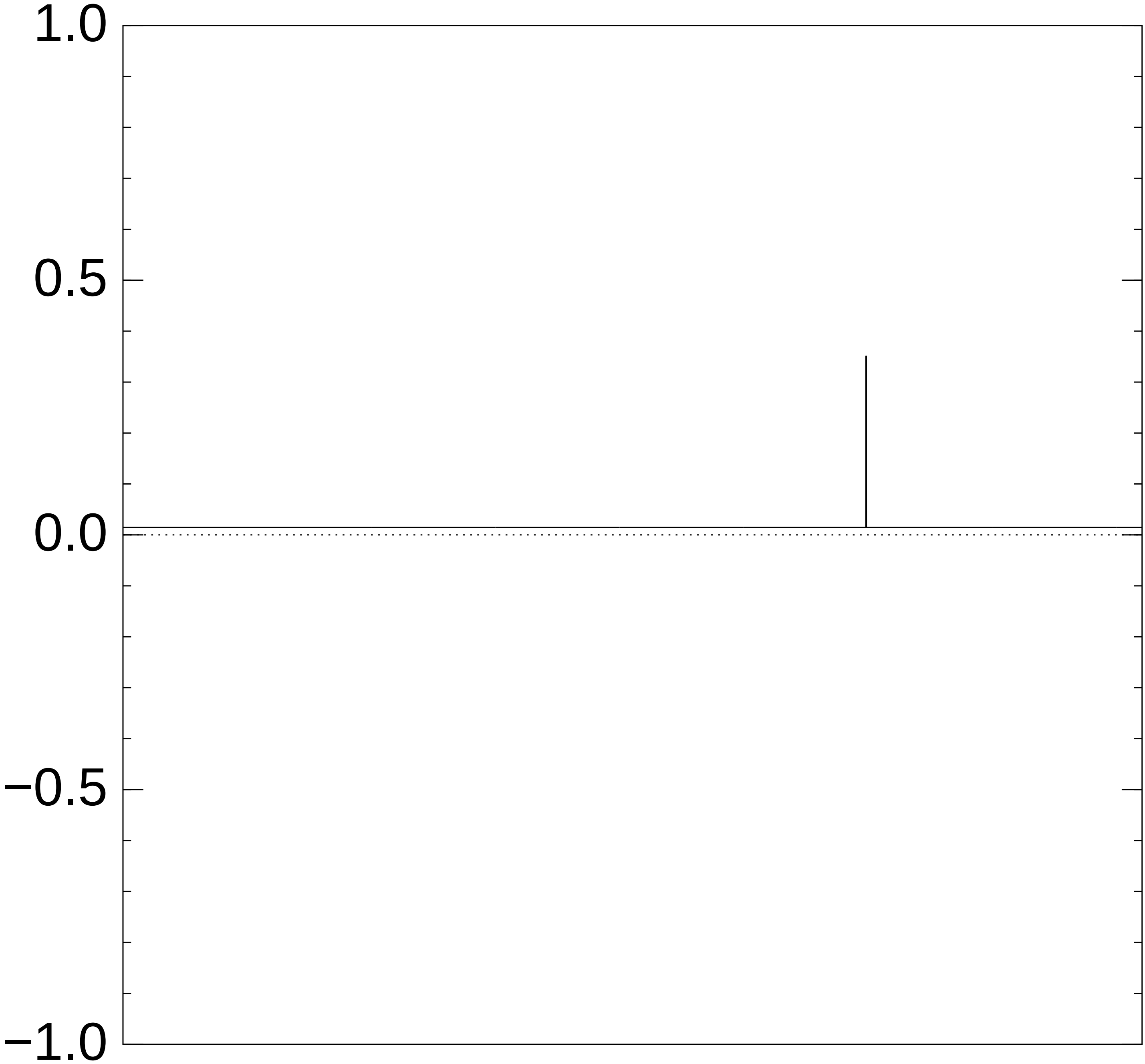
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $9 \times (\text{Step } 1 + \text{Step } 2)$:



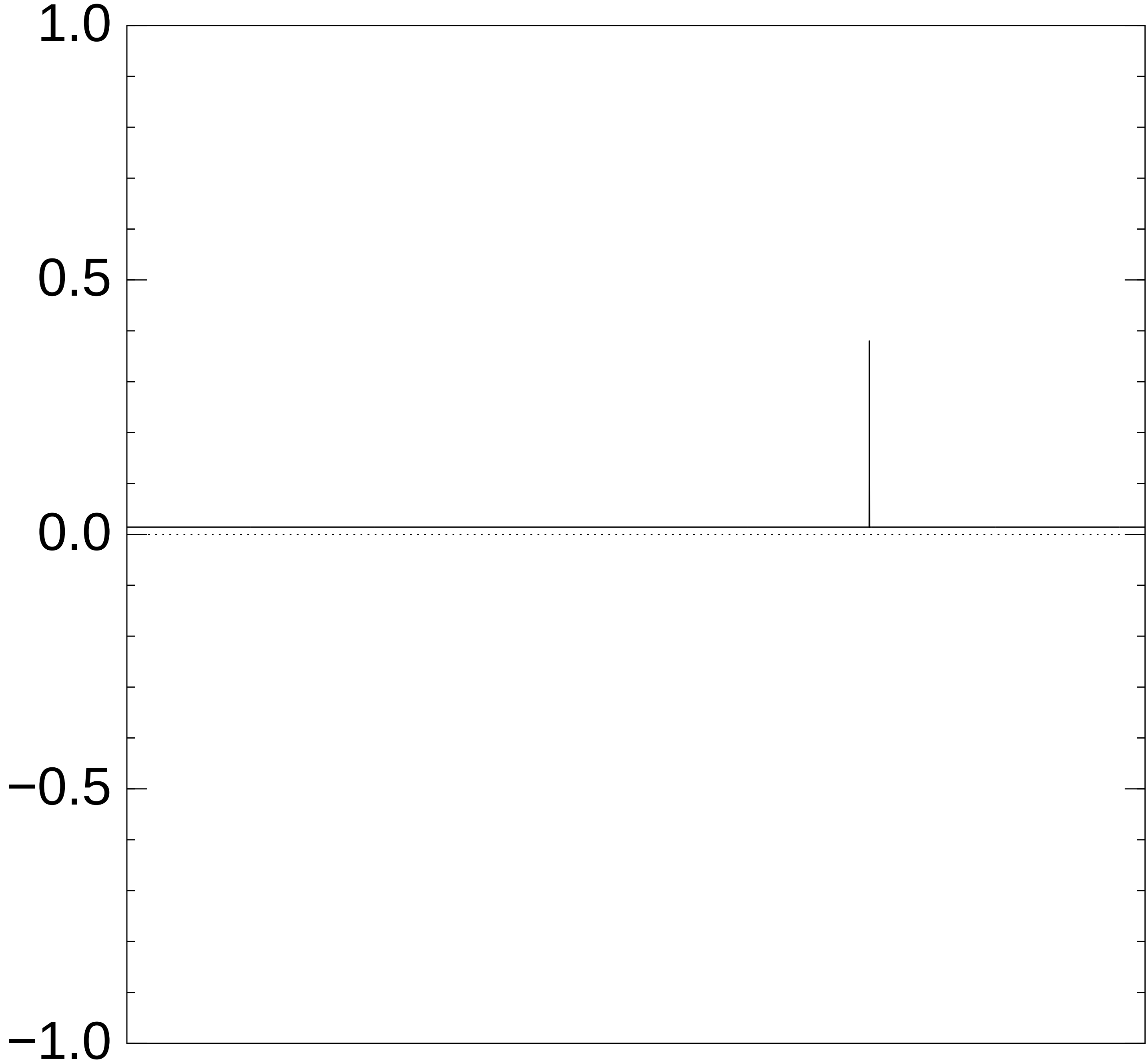
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $10 \times$ (Step $1 +$ Step 2):



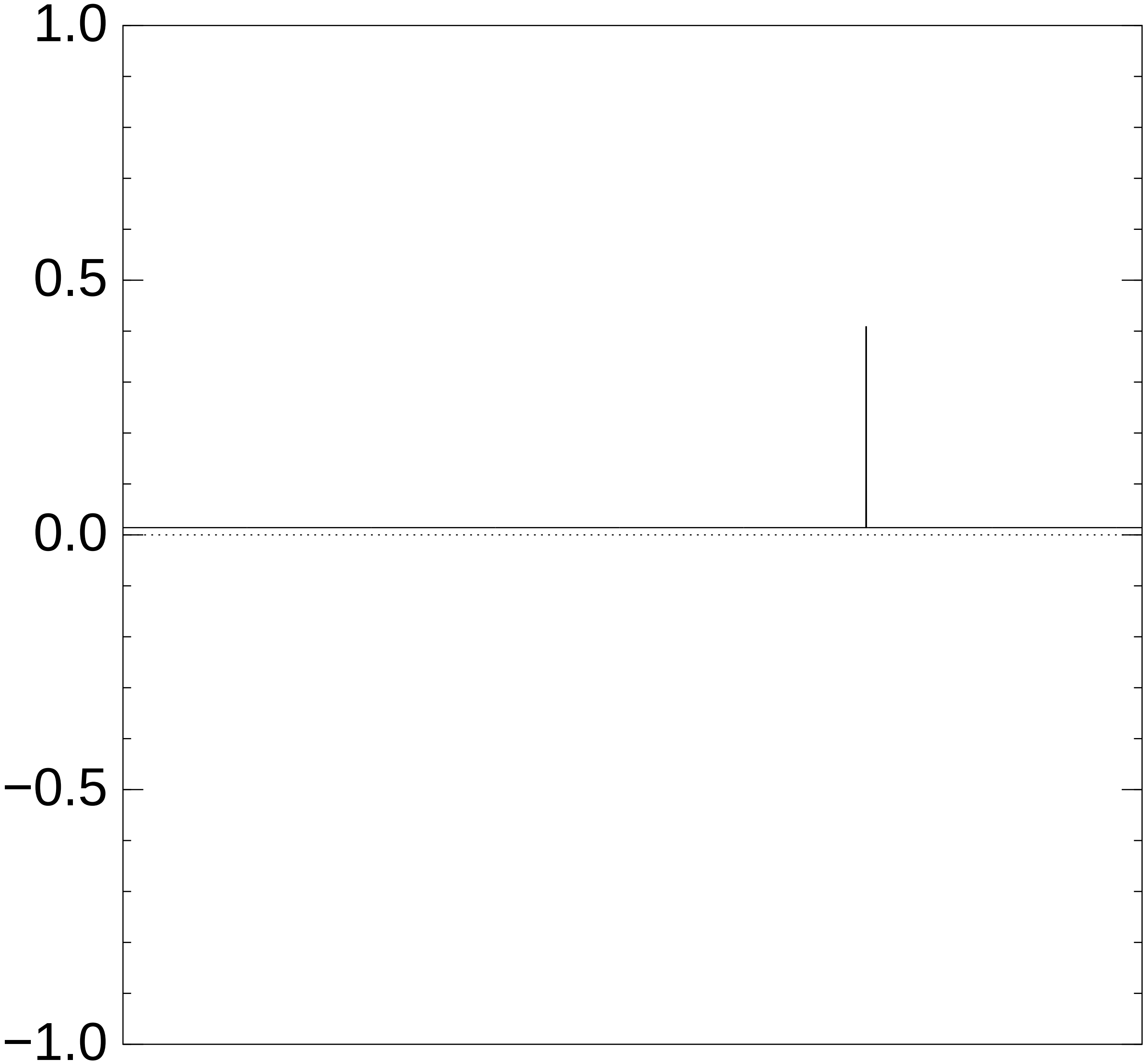
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $11 \times (\text{Step } 1 + \text{Step } 2)$:



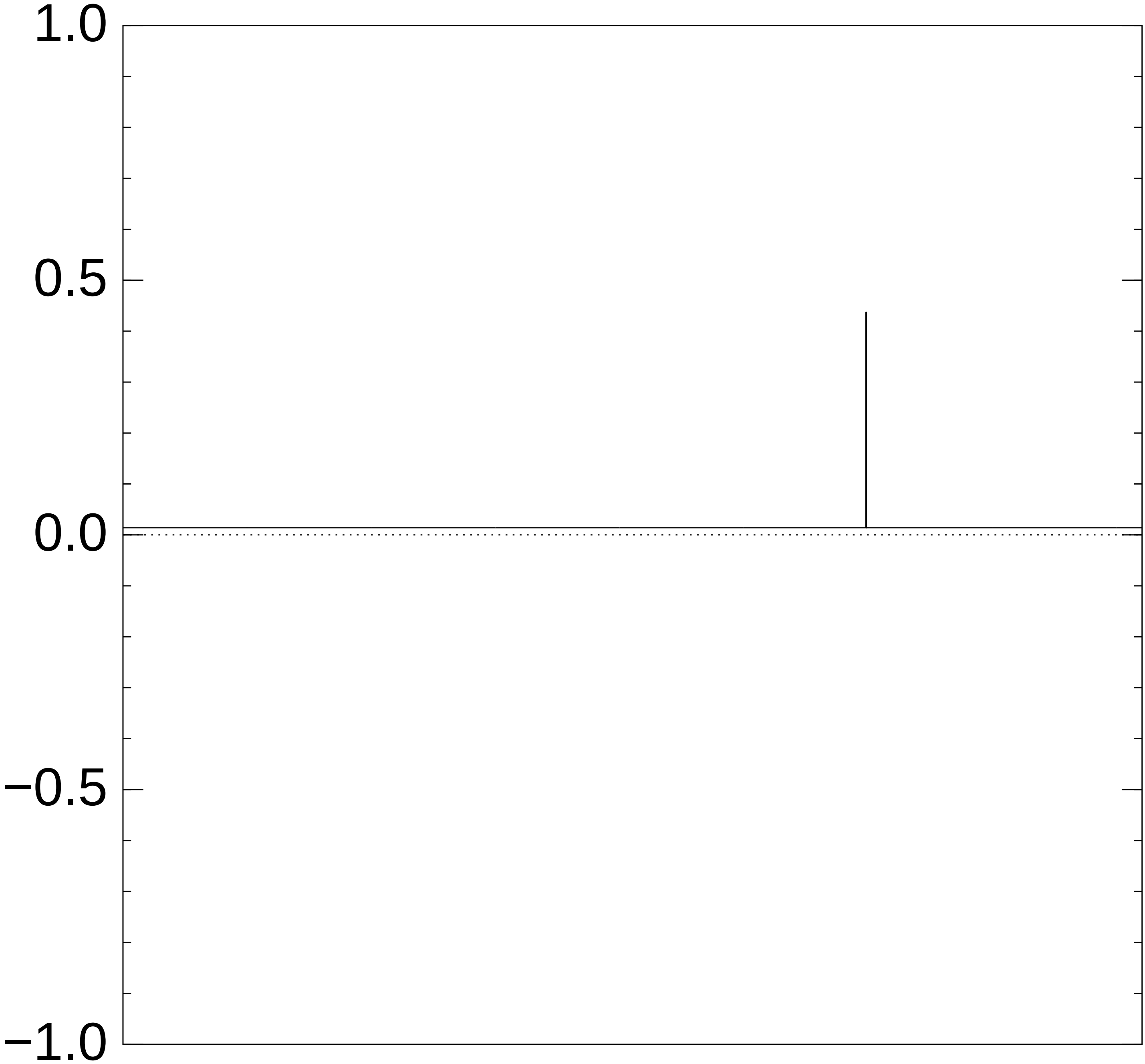
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $12 \times (\text{Step } 1 + \text{Step } 2)$:



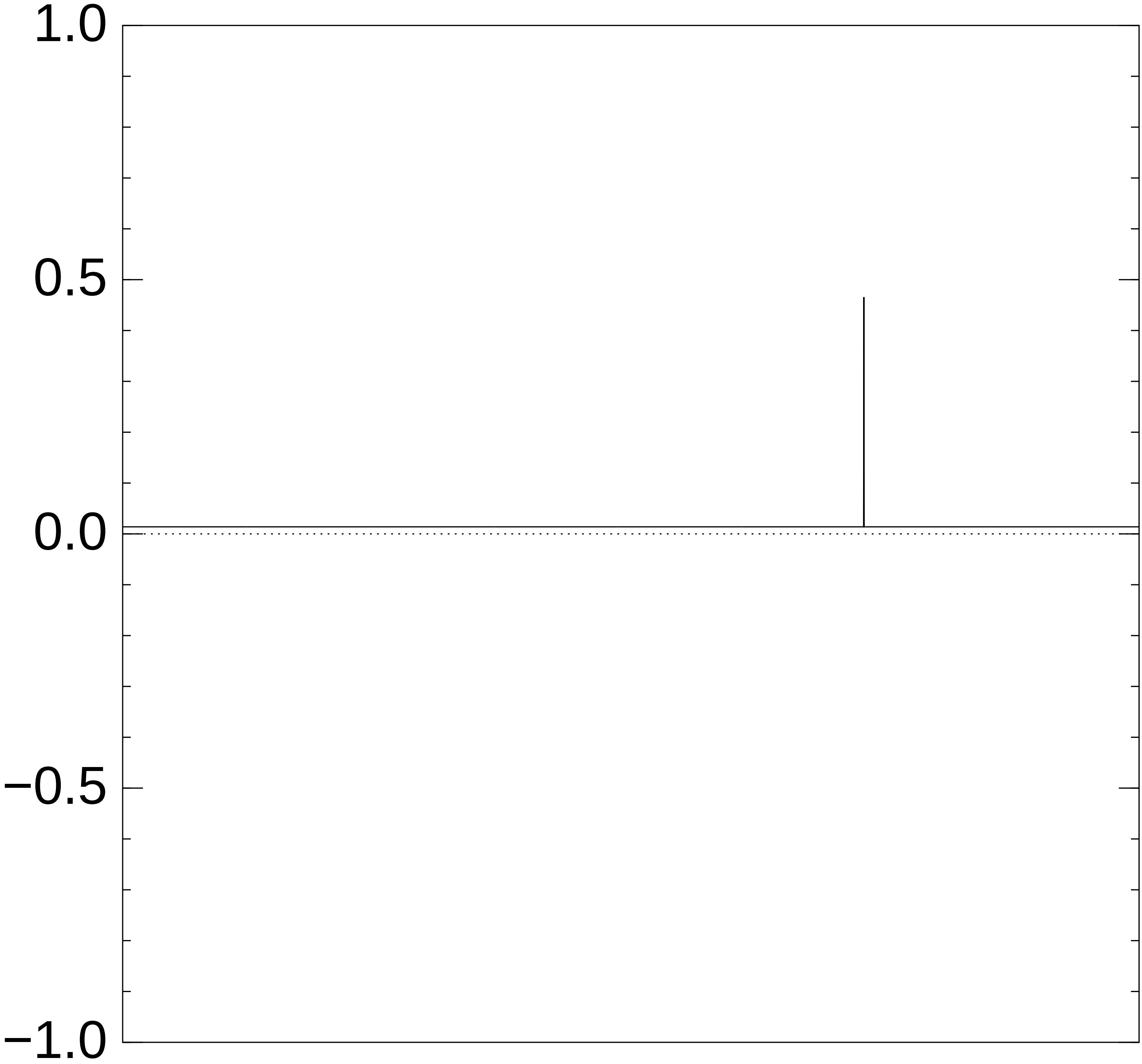
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $13 \times (\text{Step } 1 + \text{Step } 2)$:



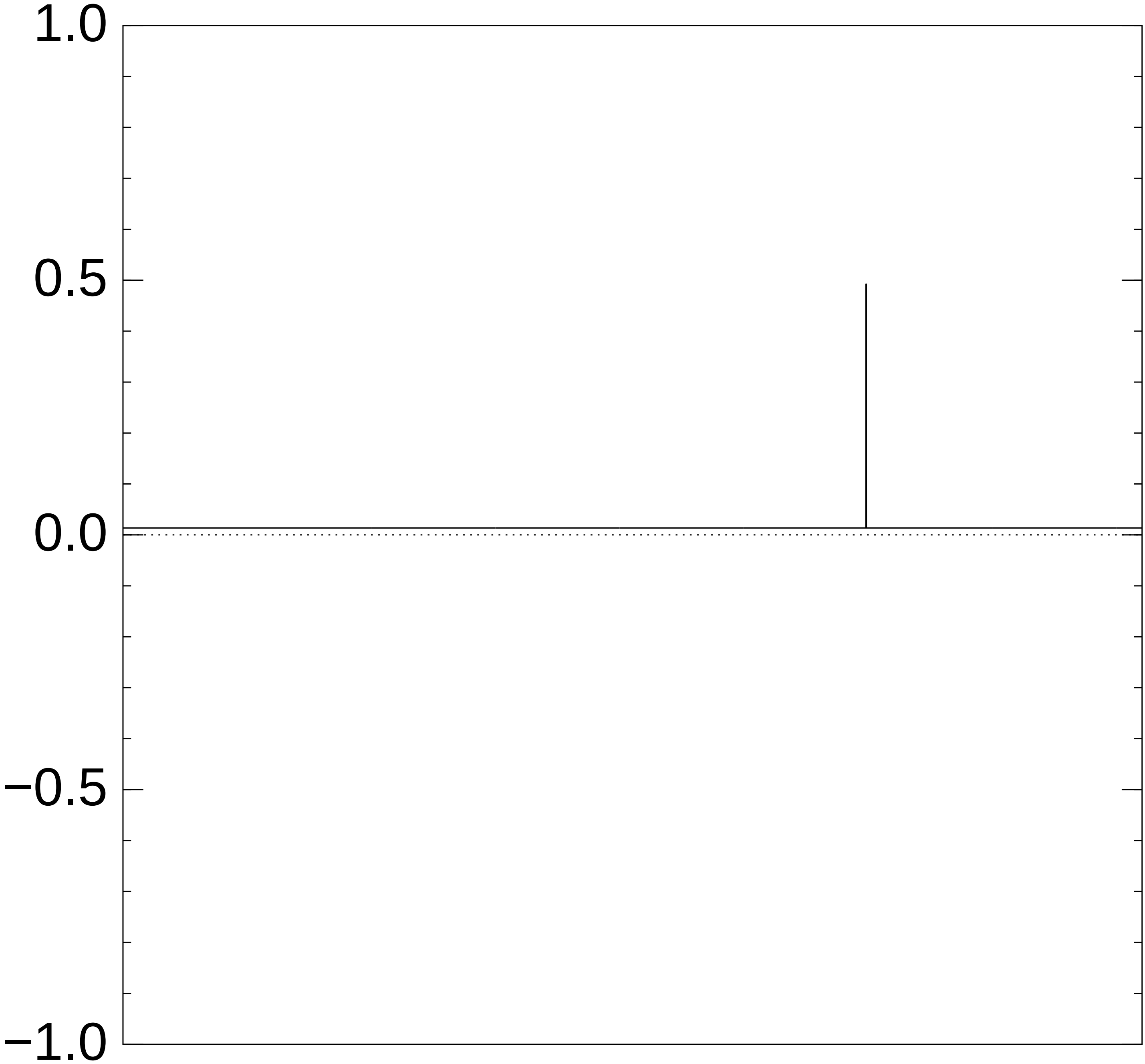
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $14 \times (\text{Step } 1 + \text{Step } 2)$:



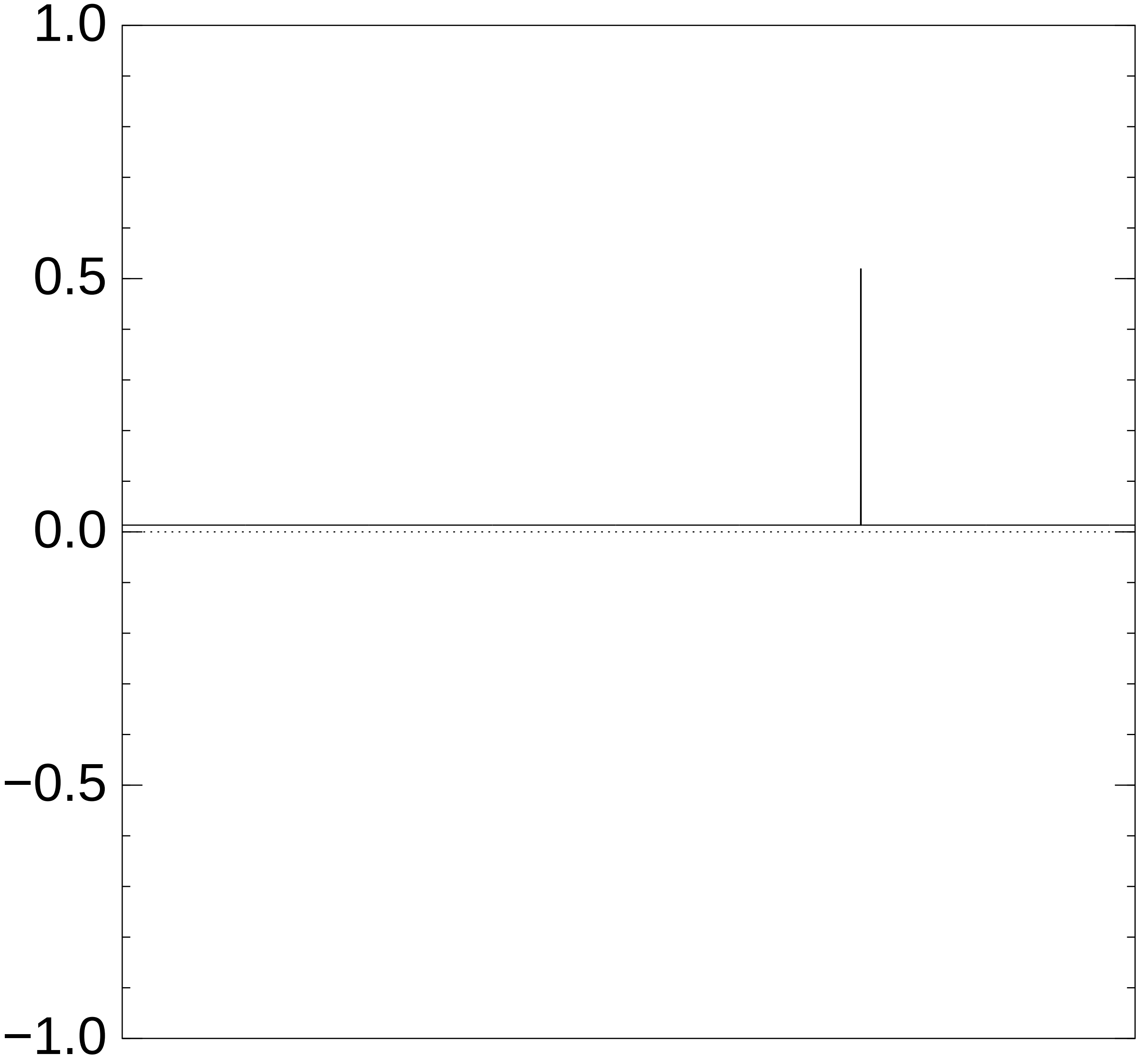
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $15 \times$ (Step $1 +$ Step 2):



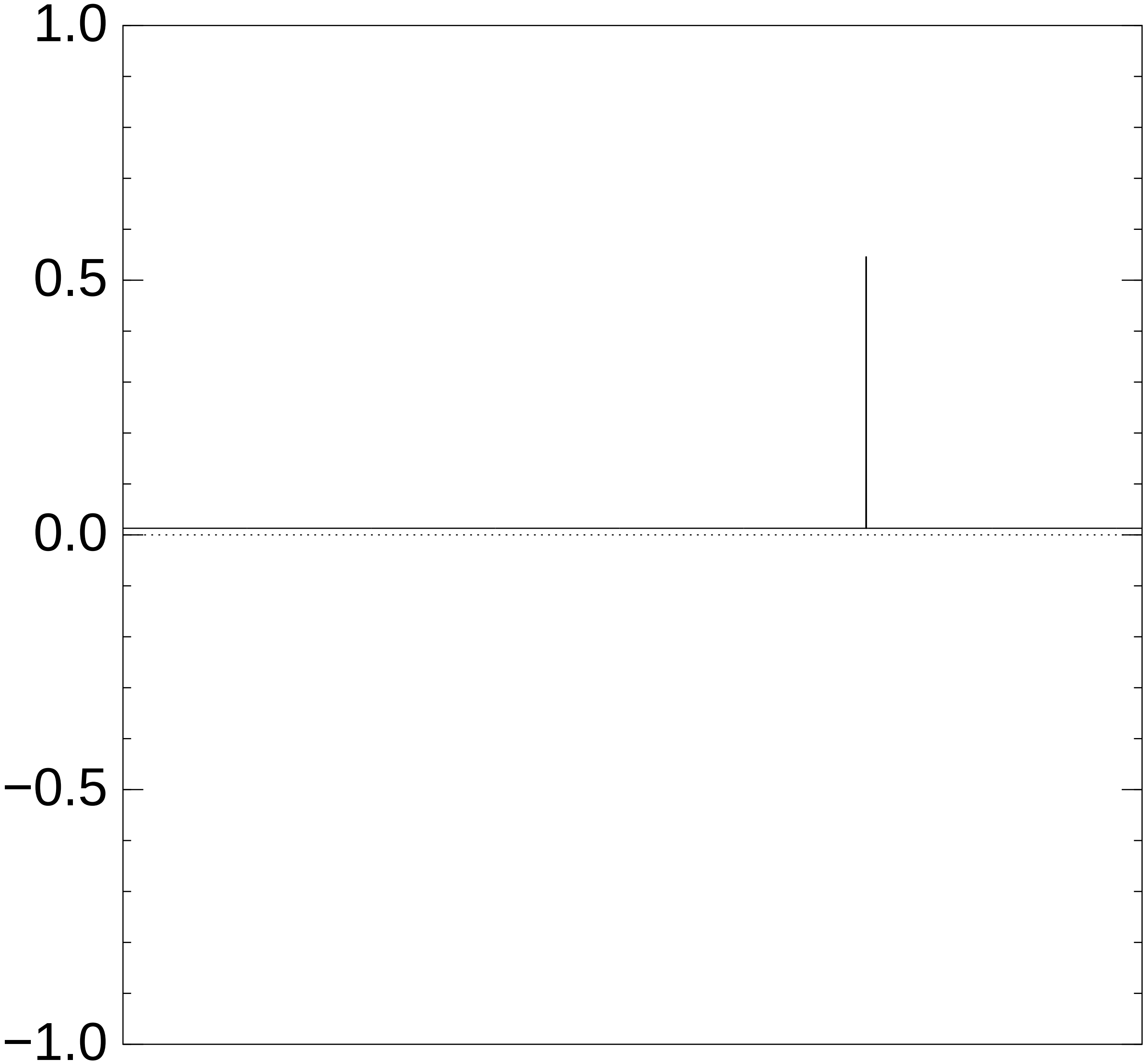
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $16 \times$ (Step $1 +$ Step 2):



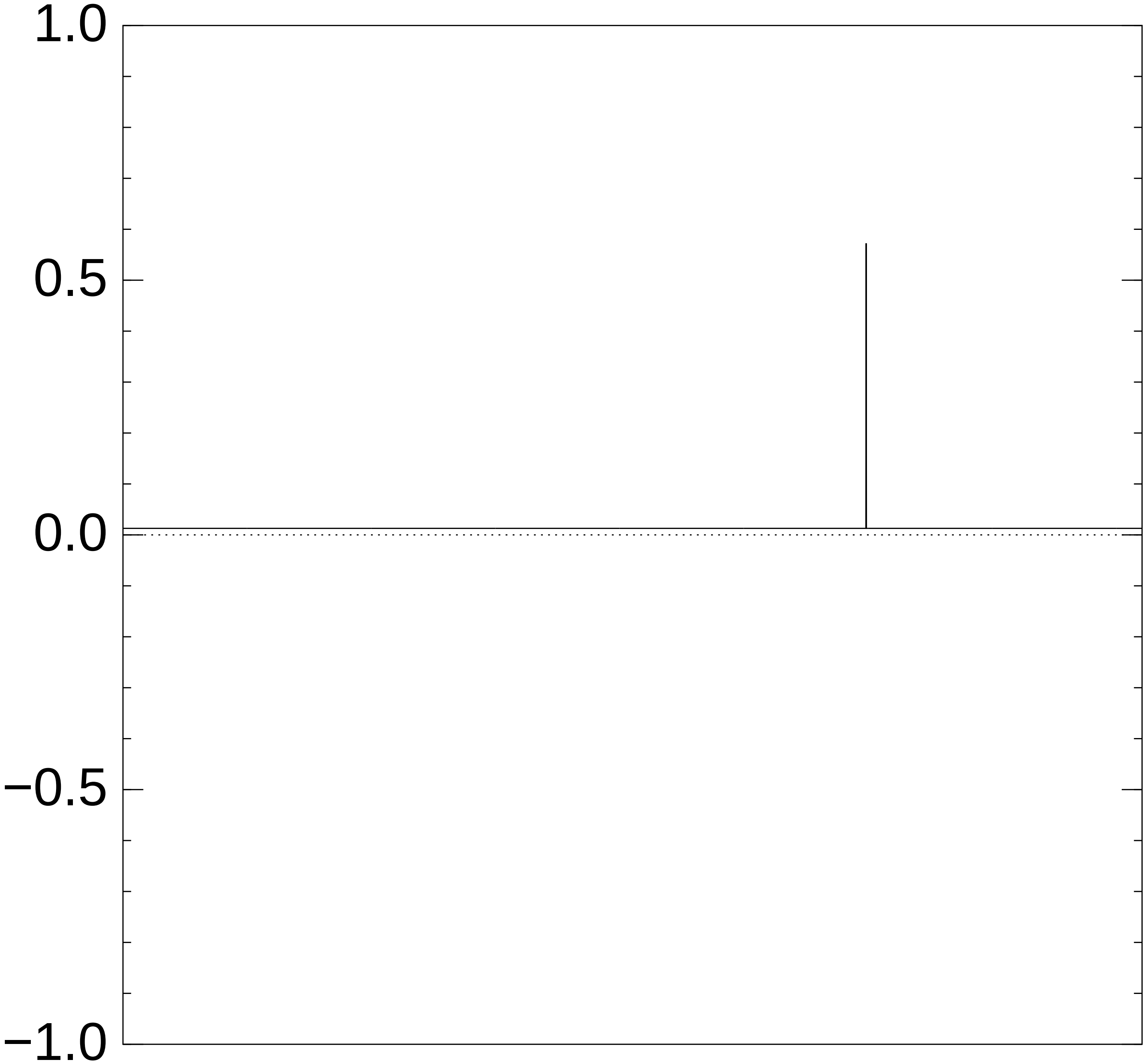
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $17 \times$ (Step $1 +$ Step 2):



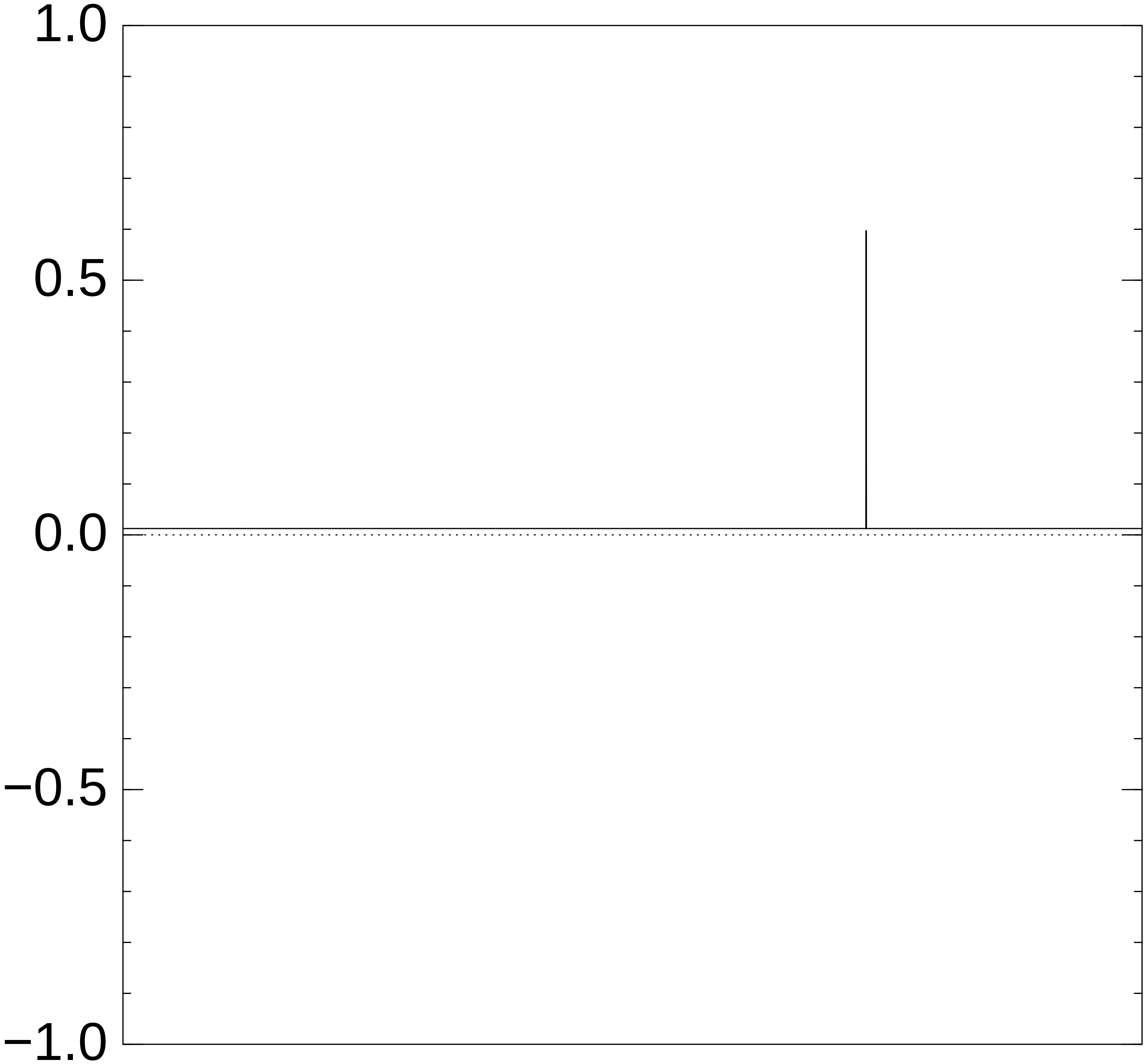
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $18 \times$ (Step $1 +$ Step 2):



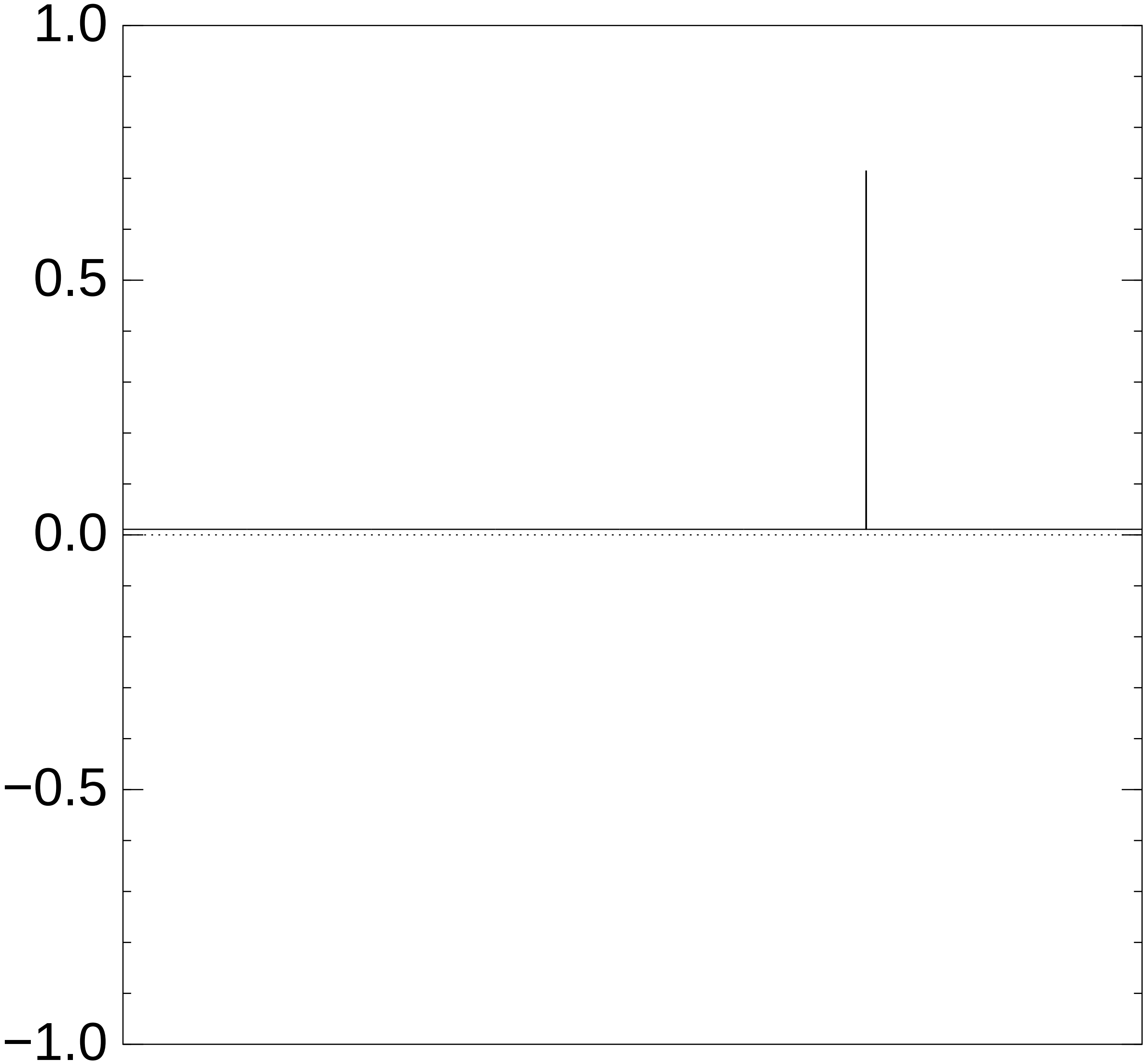
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $19 \times$ (Step $1 +$ Step 2):



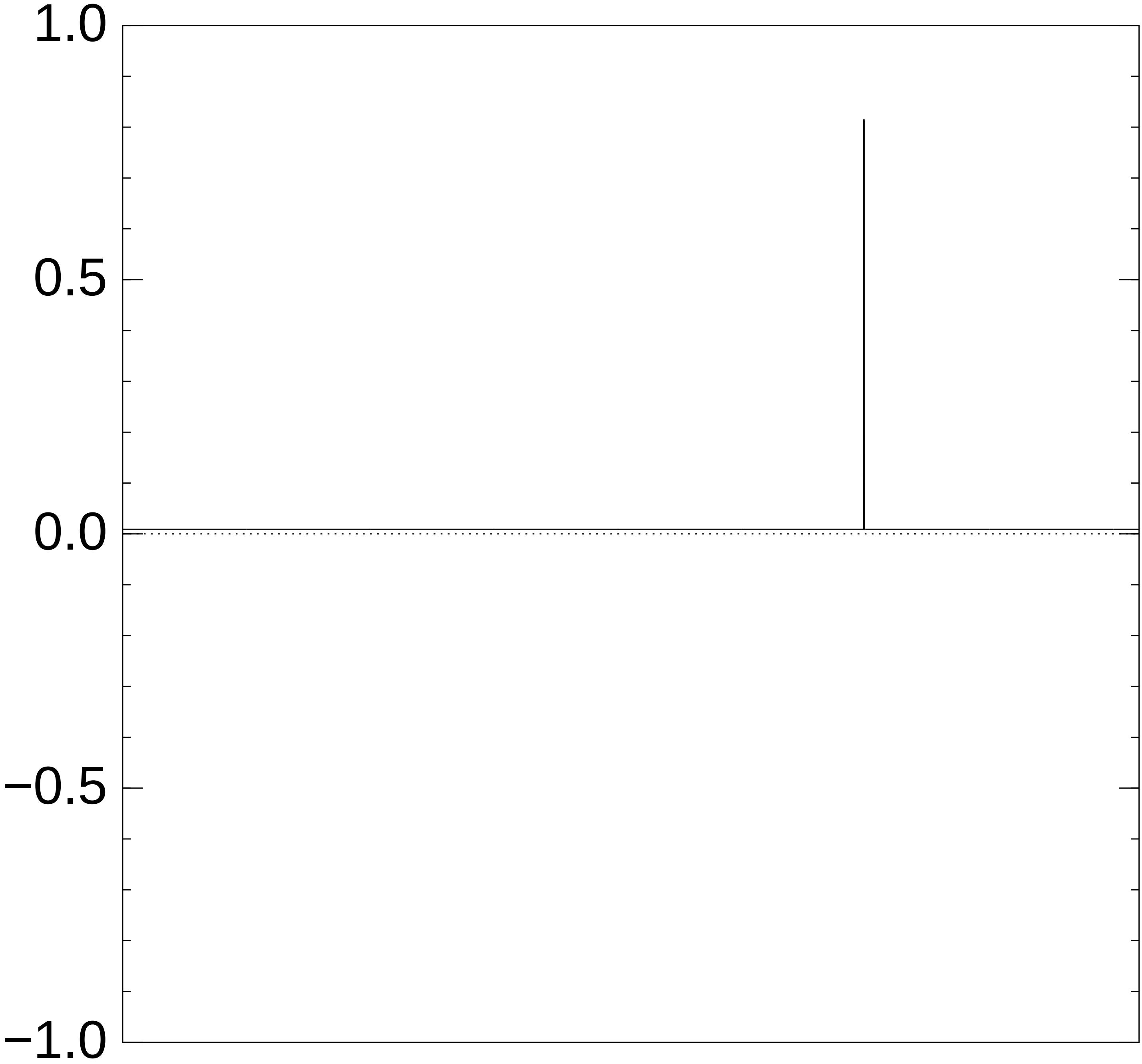
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $20 \times (\text{Step } 1 + \text{Step } 2)$:



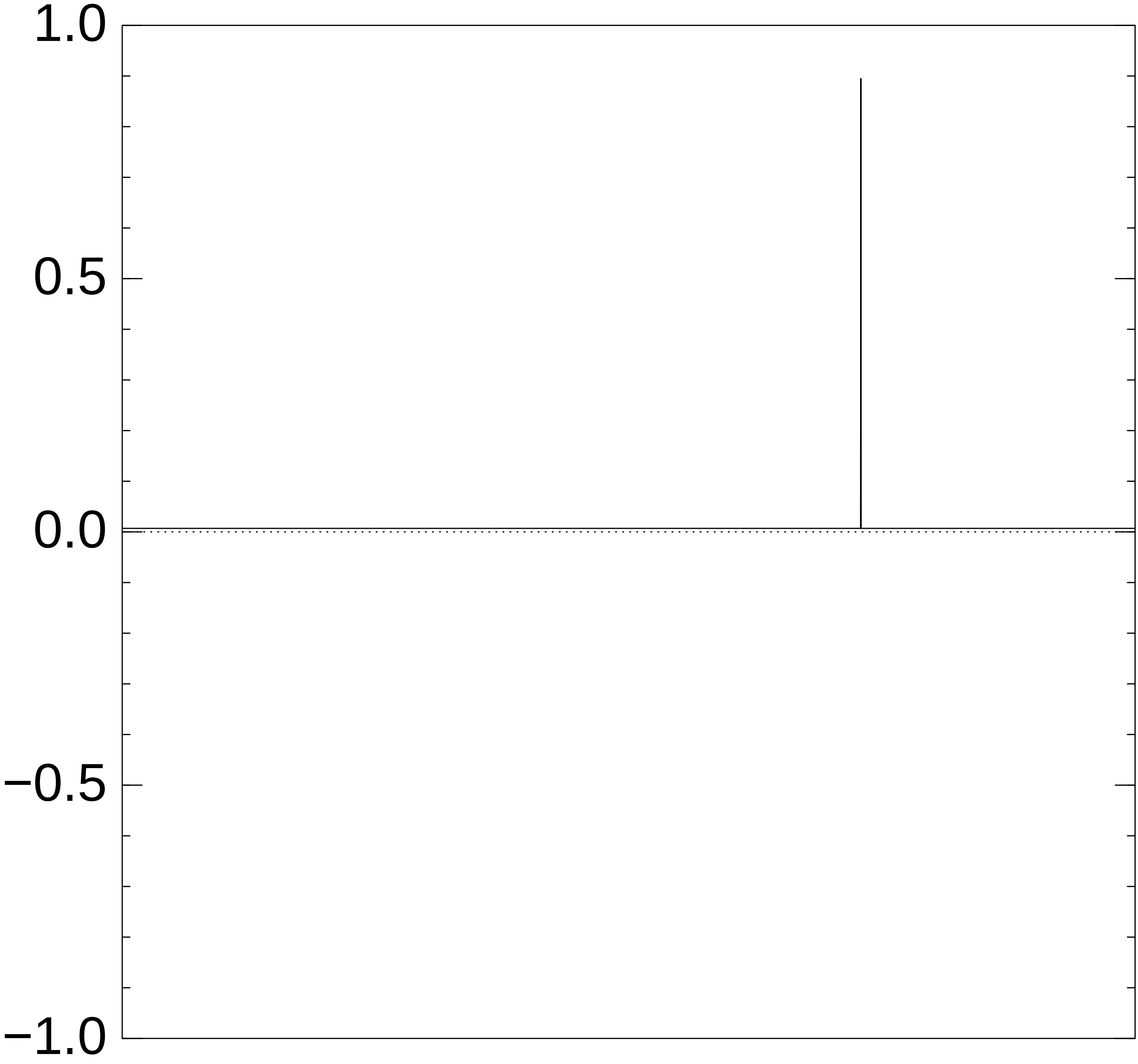
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $25 \times (\text{Step } 1 + \text{Step } 2)$:



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

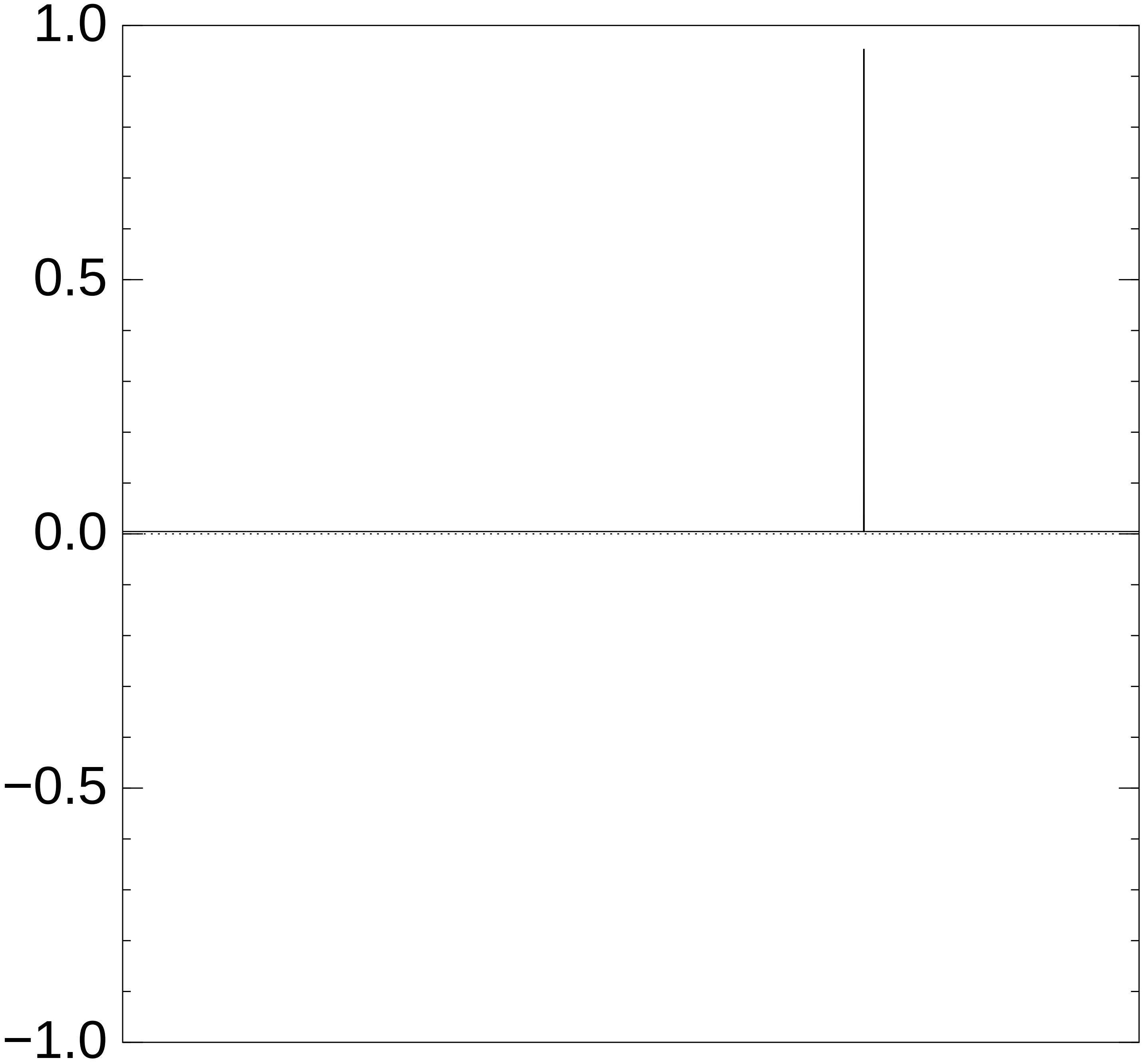


Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $35 \times$ (Step $1 +$ Step 2):

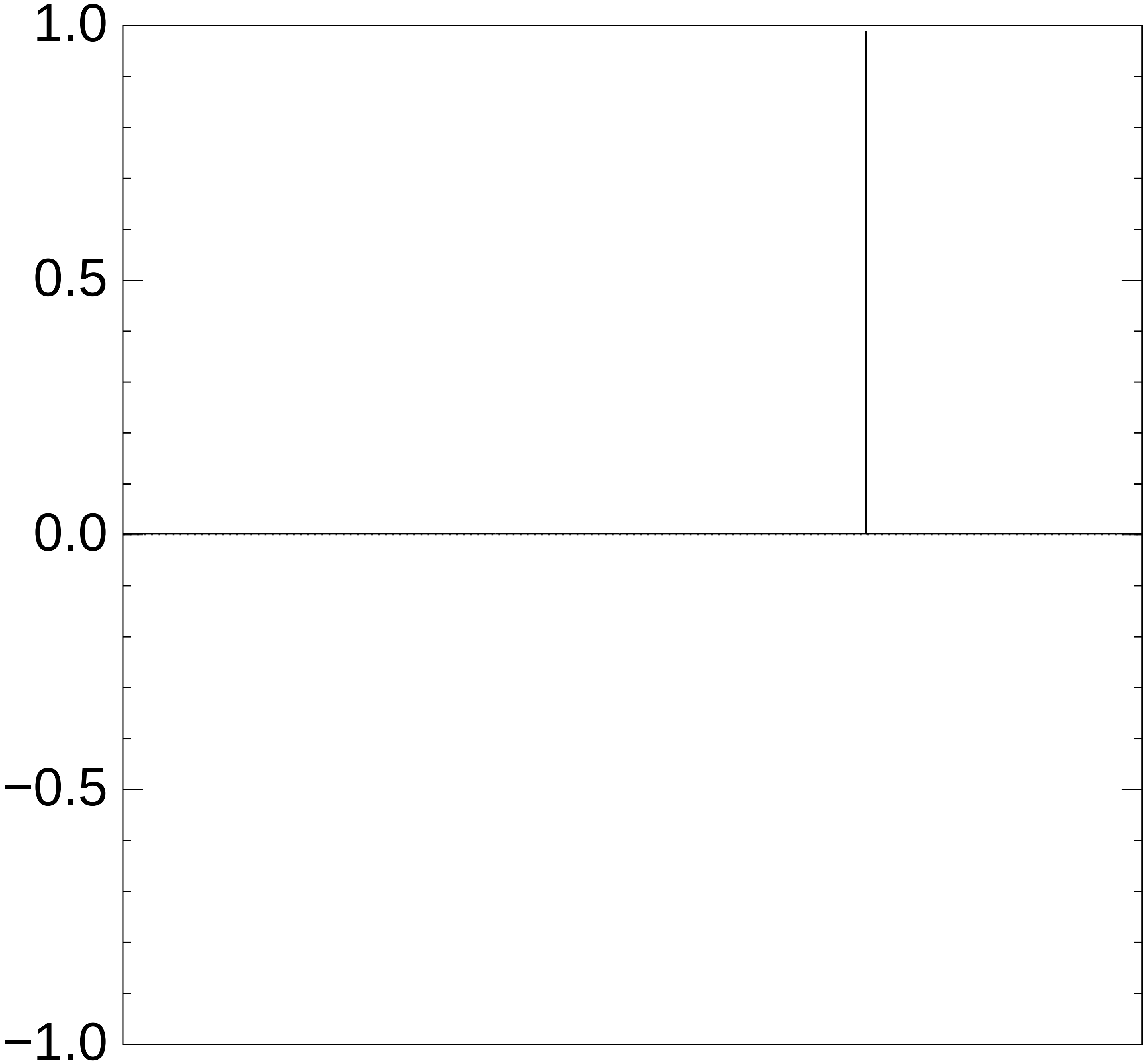


Good moment to stop, measure.

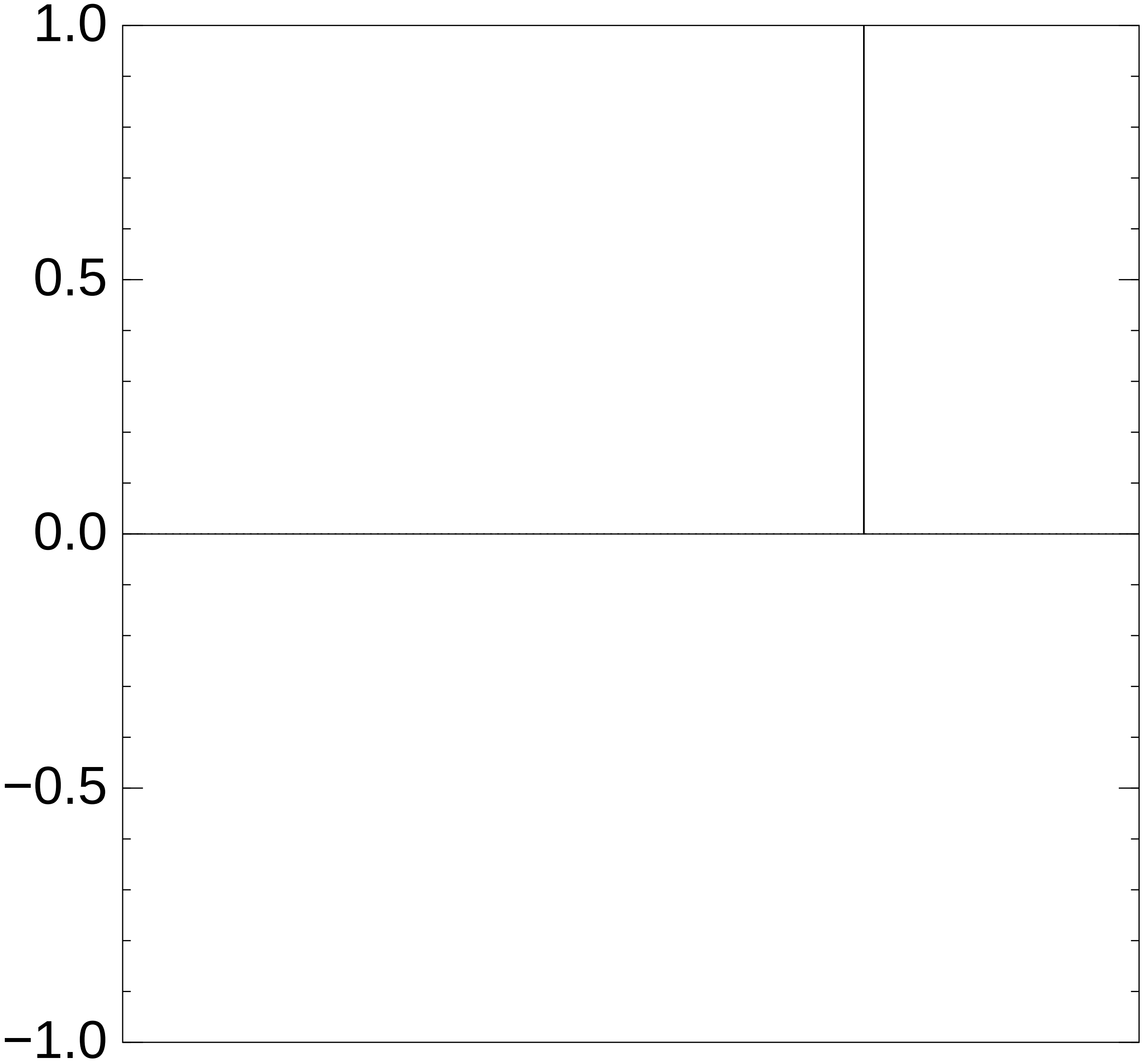
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $40 \times$ (Step $1 +$ Step 2):



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

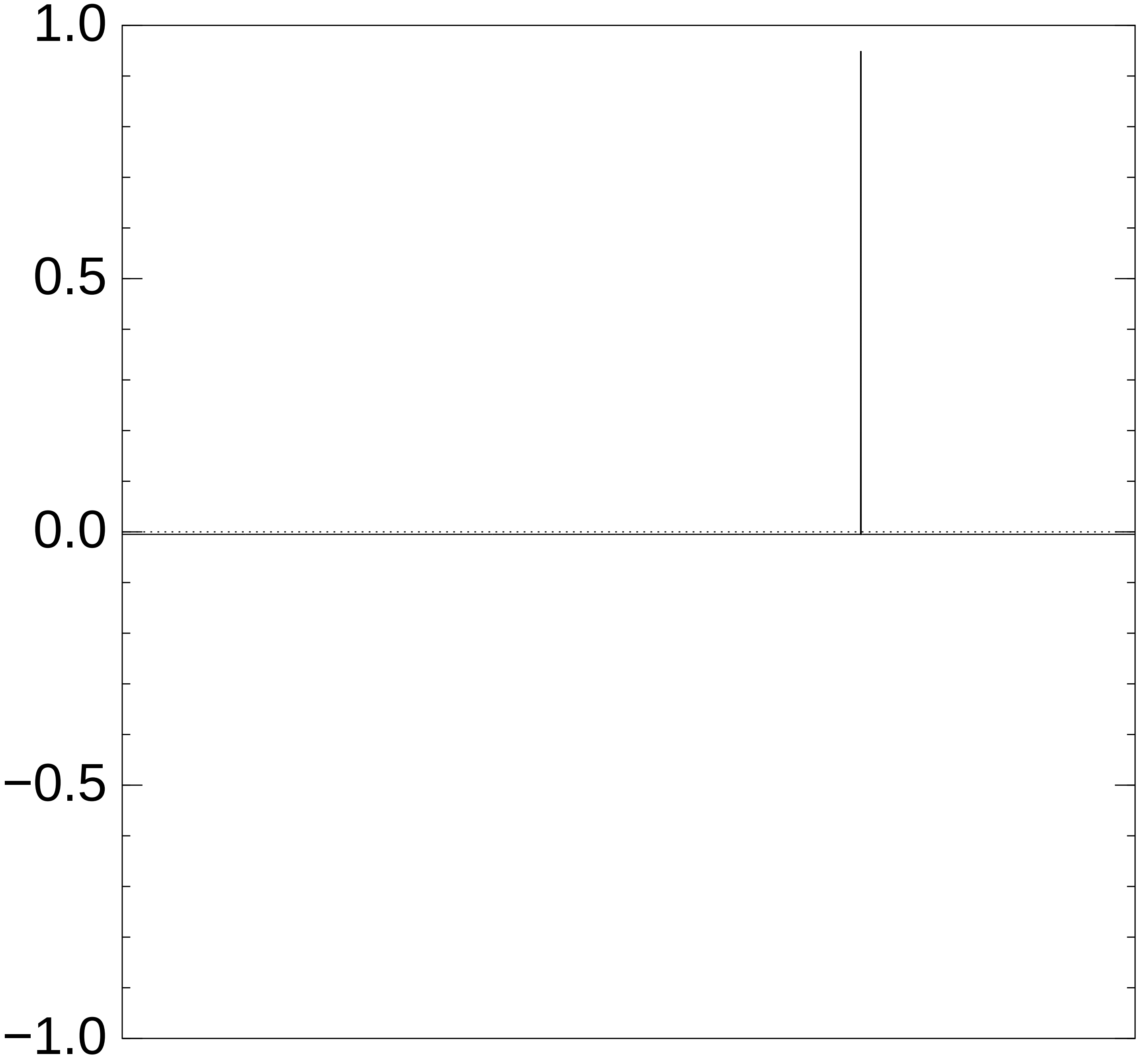


Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $50 \times$ (Step 1 + Step 2):

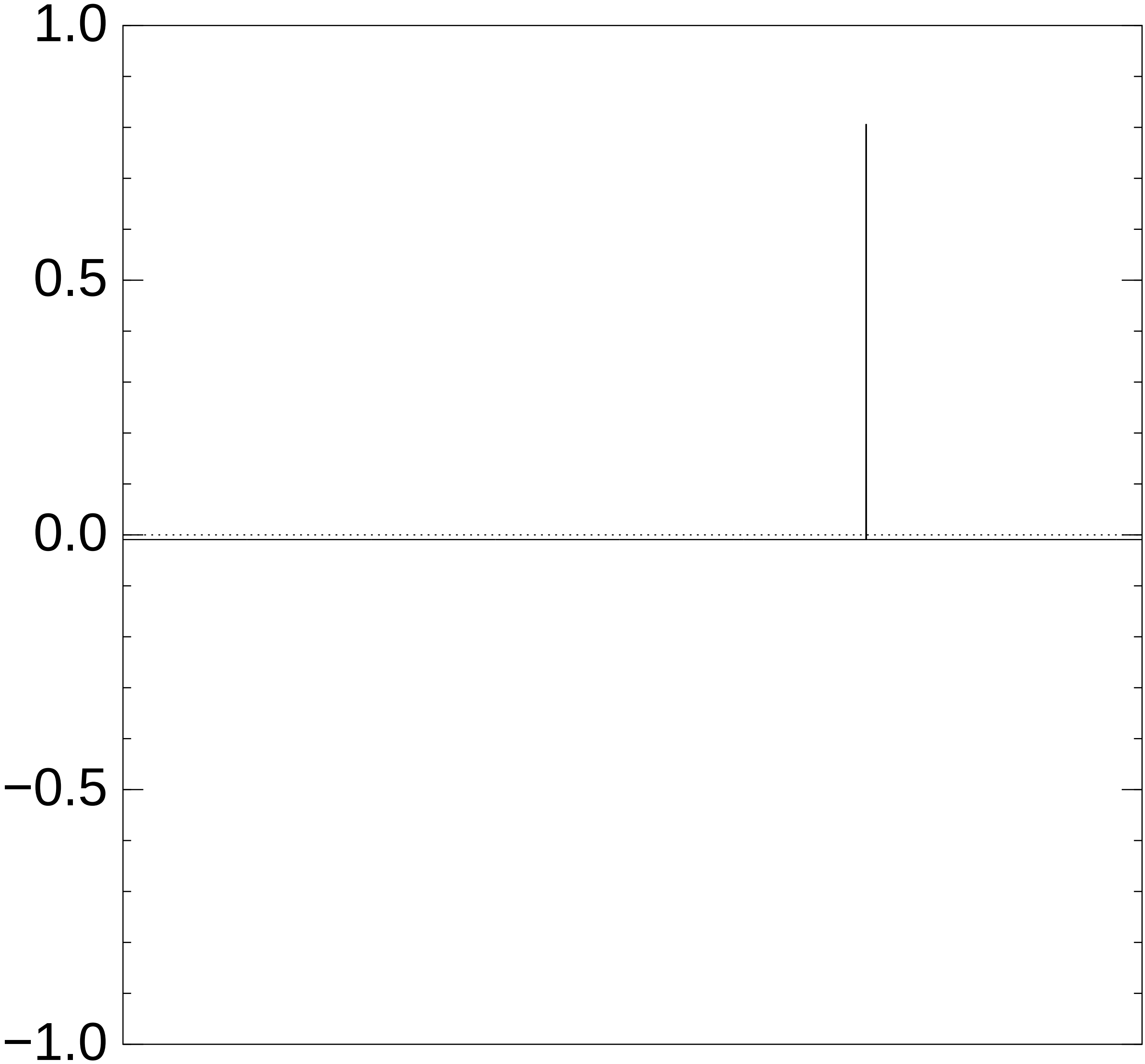


Traditional stopping point.

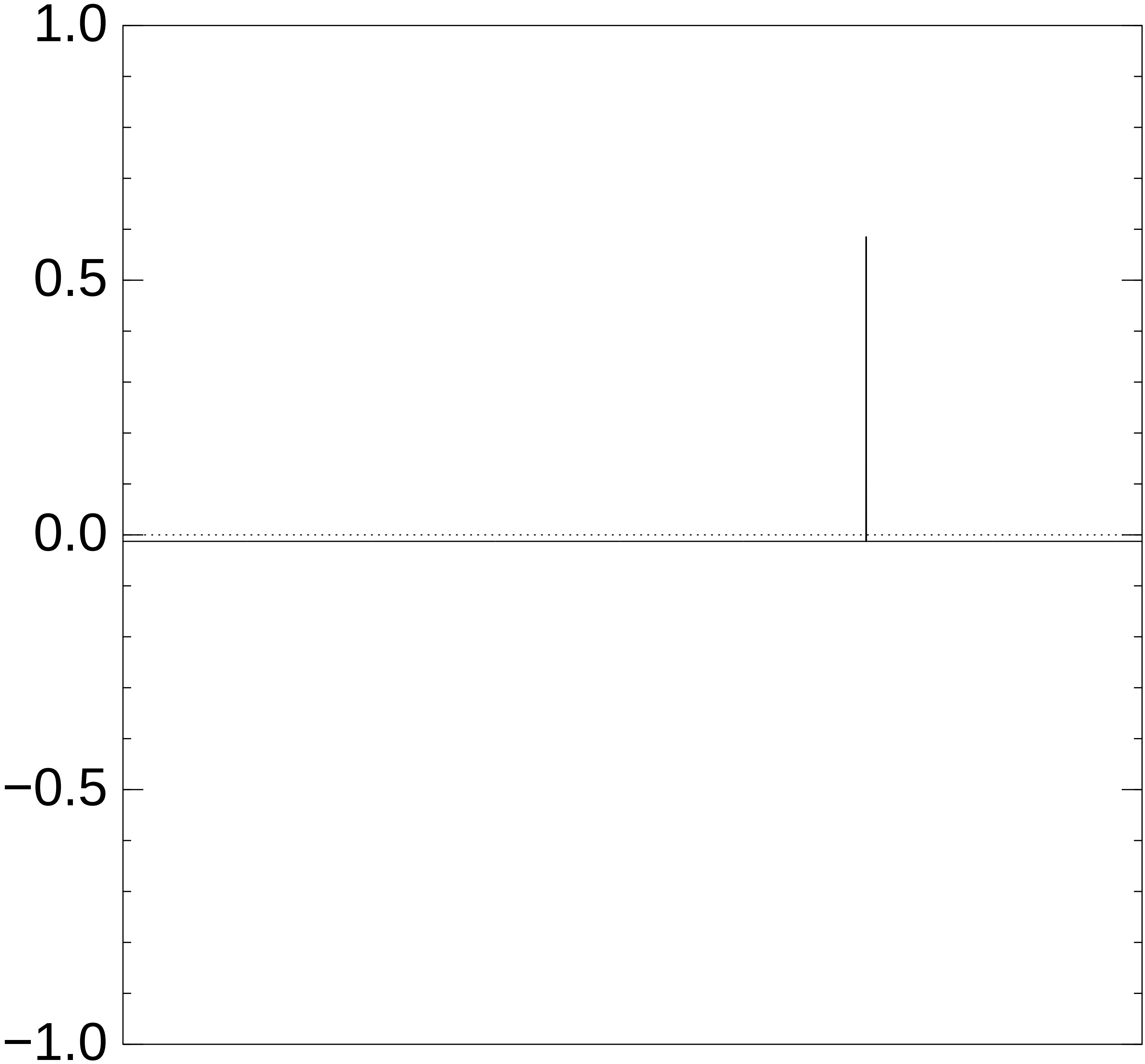
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $60 \times$ (Step $1 +$ Step 2):



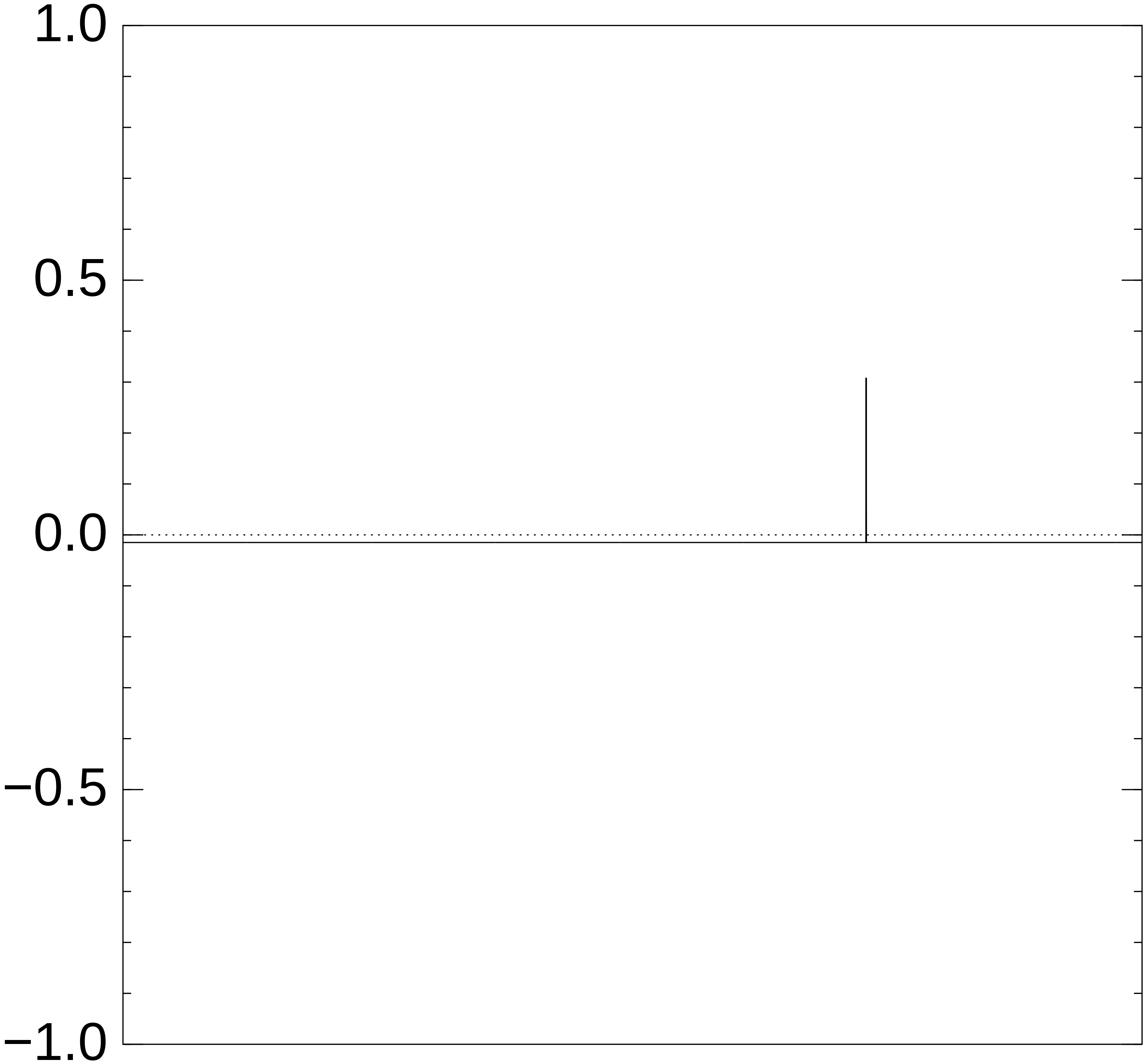
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $70 \times (\text{Step } 1 + \text{Step } 2)$:



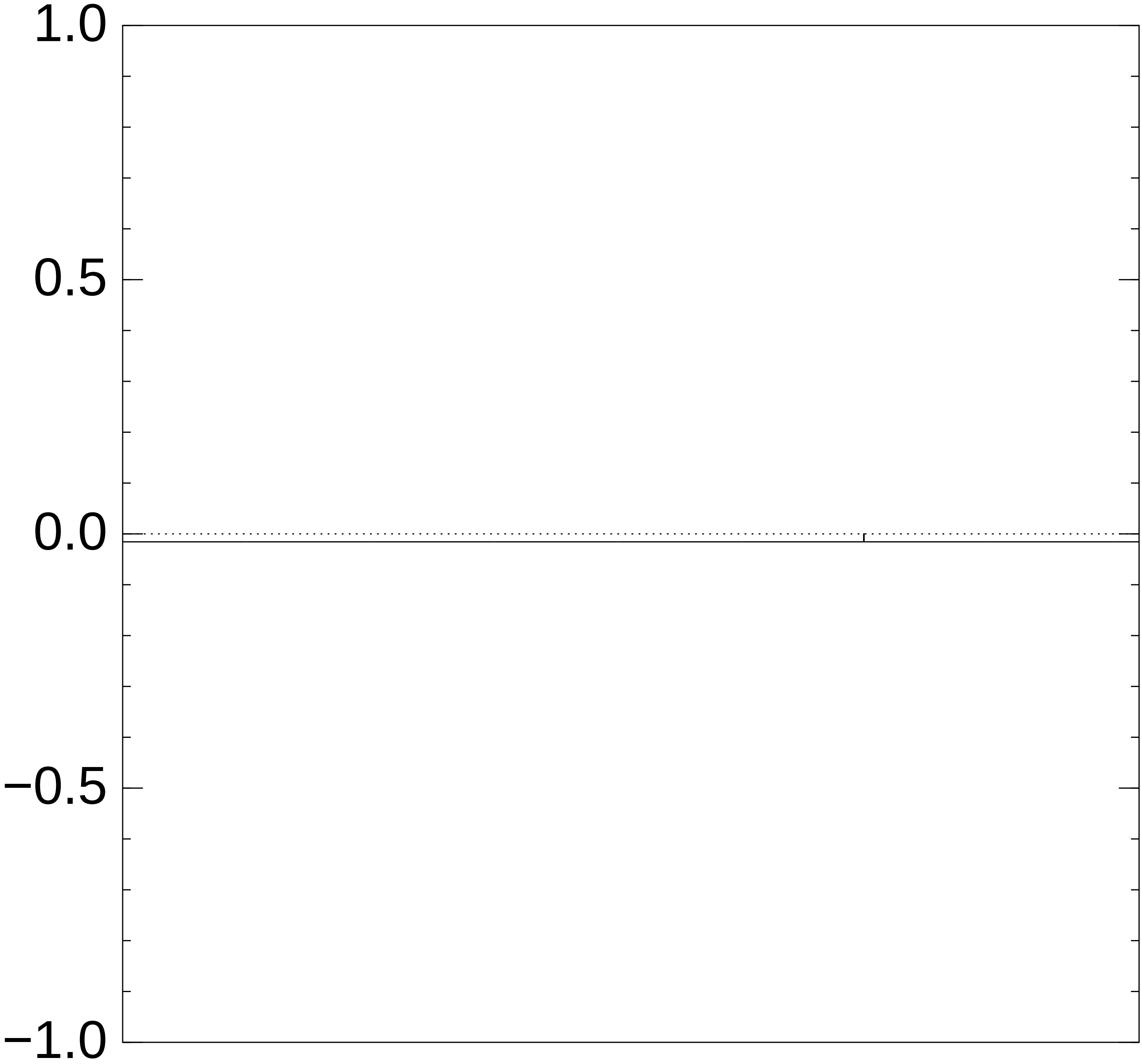
Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $80 \times (\text{Step } 1 + \text{Step } 2)$:



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



Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times$ (Step $1 +$ Step 2):



Very bad stopping point.

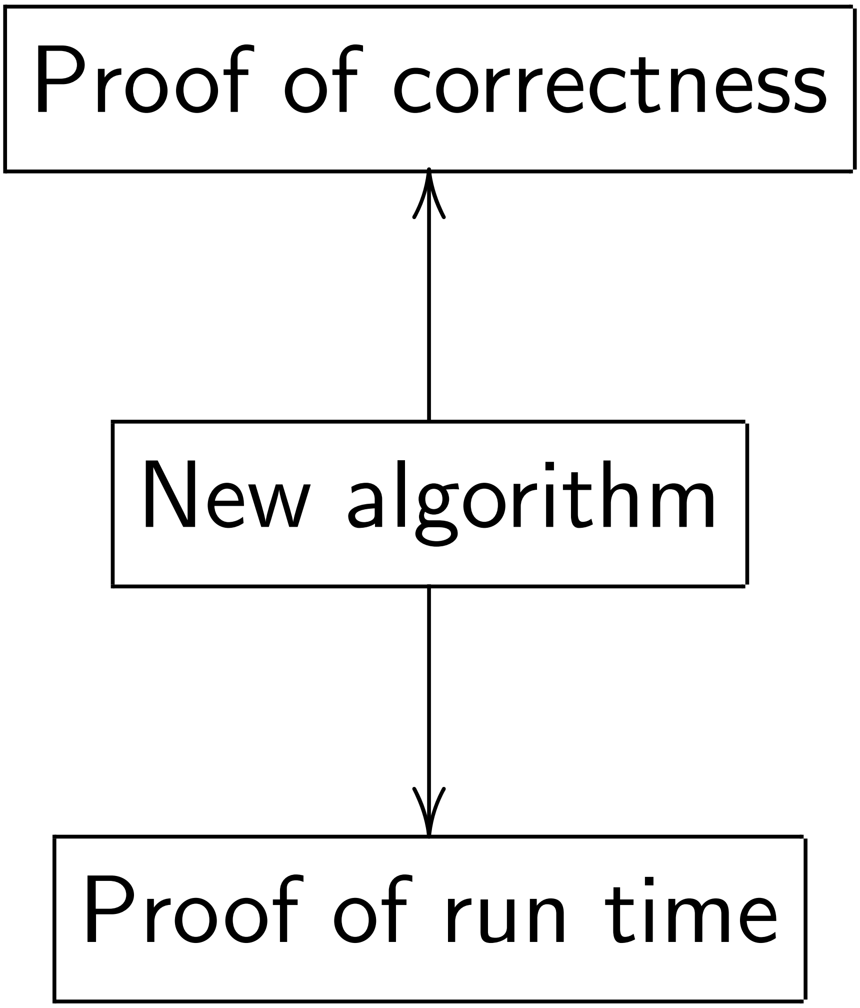
$q \mapsto a_q$ is completely described
by a vector of two numbers
(with fixed multiplicities):
(1) a_q for roots q;
(2) a_q for non-roots q.

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.

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response:
"Work harder, find proofs!"

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response:
"Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Reality: state-of-the-art
cryptanalytic algorithms
are almost never proven.

Ignorant response:
"Work harder, find proofs!"

Consensus of the experts:
proofs probably do not *exist*
for most of these algorithms.
So demanding proofs is silly.

Without proofs, how do we
analyze correctness+speed?
Answer: Real algorithm analysis
relies critically on heuristics and
computer experiments.

What about quantum algorithms?
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum attack.

What about quantum algorithms?
Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

What about quantum algorithms?
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific
simulation? Yes, sometimes.

What about quantum algorithms?

Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.** Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Fix: Switch to AES-256.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds.
Maybe 12 rounds are enough
for 2^{128} post-quantum security?
Maybe 10 rounds are enough?

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."
But some systems seem safe.

Hash-based signatures.
Example: 1979 Merkle hash-tree public-key signature system.

Code-based cryptography.
Example: 1978 McEliece hidden-Goppa-code public-key encryption system.

Lattice-based cryptography.
Example: 1998 "NTRU".

Multivariate-quadratic-equations cryptography.
Example:
1996 Patarin "HFE^{v-}"
public-key signature system.

Daniel J. Bernstein
Johannes Buchmann
Erik Dahmen
Editors

Post-Quantum Cryptography

Springer

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random"
500×1024 matrix K over $\mathbf{F}_2$.
Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption:
1024-bit strings of weight 50.
$\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear algebra,
easily works backwards
from Ke to *some* $v \in \mathbf{F}_2^{1024}$
such that $Kv = Ke$.

Attacker, by linear algebra,
easily works backwards
from Ke to *some* $v \in \mathbf{F}_2^{1024}$
such that $Kv = Ke$.

i.e. Attacker finds *some*
element $v \in e + \operatorname{Ker} K$.
Note that $\#\operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v:
to find element of $\operatorname{Ker} K$
at distance only 50 from v.
Presumably unique, revealing e.

Attacker, by linear algebra,
easily works backwards
from Ke to *some* $v \in \mathbf{F}_2^{1024}$
such that $Kv = Ke$.

i.e. Attacker finds *some*
element $v \in e + \operatorname{Ker} K$.
Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v:
to find element of $\operatorname{Ker} K$
at distance only 50 from v.
Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 subset
$S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance
that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$
is invertible.

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

Information-set decoding

Choose random size-500 subset
$S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance
that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$
is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
Apply inverse map to Ke,
revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again.
$\approx 2^{80}$ bit operations in total.

Information-set decoding

Choose random size-500 subset
$S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance
that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$
is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
Apply inverse map to Ke,
revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again.
$\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

Analyzing and optimizing attacks:
1962 Prange. 1981 Omura.
1988 Lee–Brickell. 1988 Leon.
1989 Krouk. 1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg. 1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.
1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters:
more speedups; $\approx 2^{60}$ cycles;
attack **actually carried out**.

2009 Bernstein–Lange–
Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–
Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

Modern McEliece

Easily rescue system by using a larger public key: “random” $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

After extensive research, 2015 attack cost: $2^{(1+o(1))w}$.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

After extensive research, 2015 attack cost: $2^{(1+o(1))w}$.

Post-quantum: $2^{(0.5+o(1))w}$. e.g. $\approx 2^{26}$ Grover iterations to search 2^{53} choices of S.