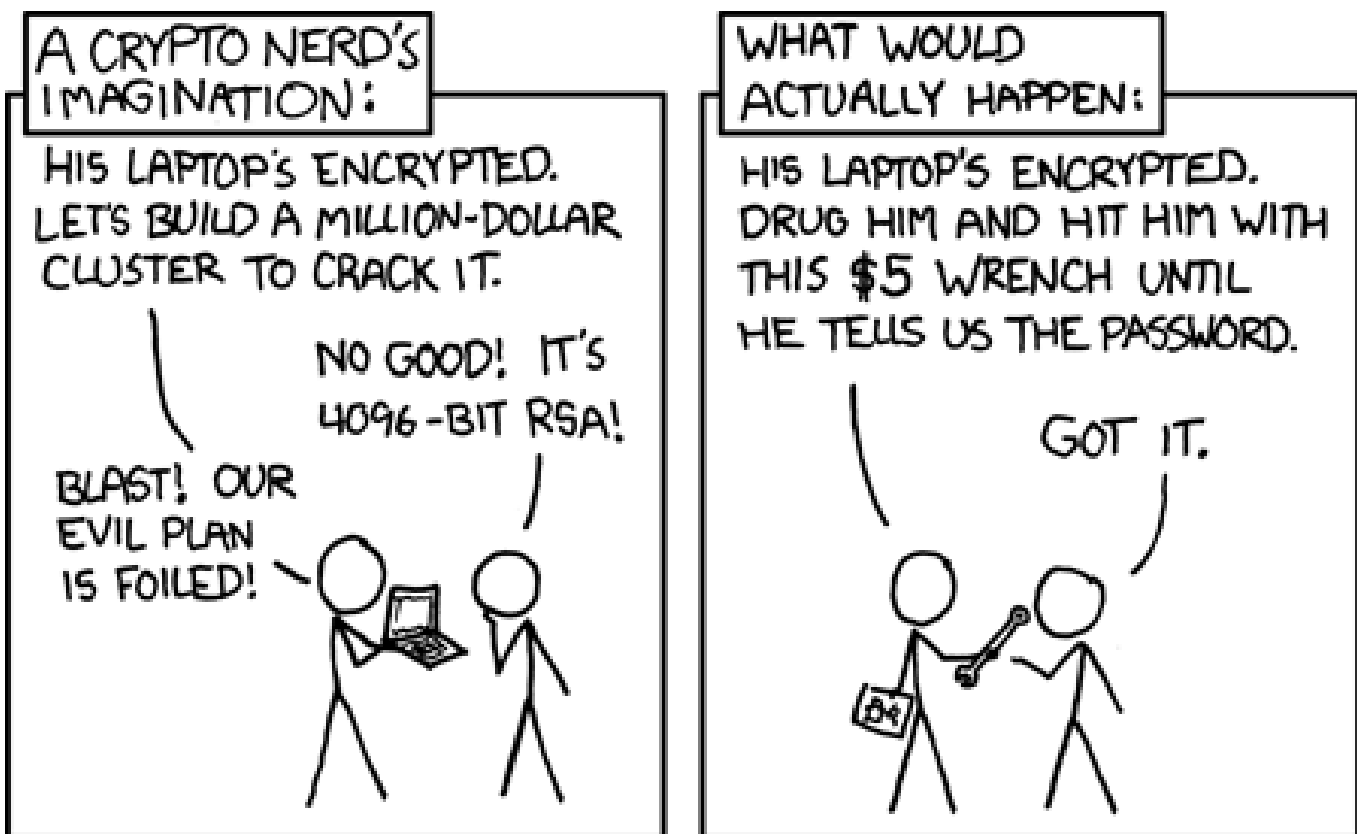


Failures of secret-key cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven



<http://xkcd.com/538/>

2011 Grigg–Gutmann (and again
2012 Gutmann): In the past 15
years “no one ever lost money to
an attack on a properly designed
cryptosystem (meaning one that
didn’t use homebrew crypto
or toy keys) in the Internet or
commercial worlds” .

2011 Grigg–Gutmann (and again 2012 Gutmann): In the past 15 years “no one ever lost money to an attack on a properly designed cryptosystem (meaning one that didn’t use homebrew crypto or toy keys) in the Internet or commercial worlds” .

2002 Shamir: “Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis.”

2013.03 Bernstein:

“Do these people mean that
it’s actually infeasible
to break real-world crypto?”

2013.03 Bernstein:

“Do these people mean that it’s actually infeasible to break real-world crypto?”

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

2013.03 Bernstein:

“Do these people mean that it’s actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

Or are they simply wrong: real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security?”

2013.03 Bernstein:

“Do these people mean that it’s actually infeasible to break real-world crypto?

Or do they mean that breaks are feasible but still not worthwhile for the attackers?

Or are they simply wrong: real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security?”

Let’s look at some examples.

Windows code signatures

Flame broke into computers, spied on audio, keystrokes, etc.

2012.06.03 Microsoft:

“We recently became aware of a complex piece of targeted malware known as ‘Flame’ and immediately began examining the issue. . . . We have discovered through our analysis that some components of the malware have been signed by certificates that allow software to appear as if it was produced by Microsoft.”

2012.06.07 Stevens: “A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant.”

2012.06.07 Stevens: “A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant.”

CrySyS: Flame file wavesup3.drv appeared in logs in 2007; Flame “may have been active for as long as five to eight years” .

2012.06.07 Stevens: “A chosen-prefix collision attack against MD5 has been used for Flame. More interestingly . . . not our published chosen-prefix collision attack was used, but an entirely new and unknown variant.”

CrySyS: Flame file wavesup3.drv appeared in logs in 2007; Flame “may have been active for as long as five to eight years” .

Was MD5 “homebrew crypto”?

No. Standardized, widely used.

Worthwhile to attack? Yes.

Compare to 2011 Grigg–Gutmann:

“Cryptosystem failure is orders of magnitude below any other risk.”

Compare to 2011 Grigg–Gutmann:
“Cryptosystem failure is orders of magnitude below any other risk.”



http://en.wikipedia.org/wiki/2003_Mission_Accomplished_speech

WEP

WEP introduced in 1997
in 802.11 wireless standard.

2001 Borisov–Goldberg–Wagner:
24-bit “nonce” frequently repeats,
leaking plaintext xor and
allowing very easy forgeries.

2001 Arbaugh–Shankar–Wan:
this also breaks user auth.

2001 Fluhrer–Mantin–Shamir:
WEP builds RC4 key (k, n)
from secret k , “nonce” n ;
RC4 outputs leak bytes of k .

Implementations, optimizations
of k -recovery attack: 2001
Stubblefield–Ioannidis–Rubin,
2004 KoreK, 2004 Devine, 2005
d'Otreppe, 2006 Klein, 2007
Tews–Weinmann–Pyshkin, 2010
Sepehrdad–Vaudenay–Vuagnoux,
2013 S–Sušil–V–V, . . .

Implementations, optimizations
of k -recovery attack: 2001
Stubblefield–Ioannidis–Rubin,
2004 KoreK, 2004 Devine, 2005
d'Otreppe, 2006 Klein, 2007
Tews–Weinmann–Pyshkin, 2010
Sepehrdad–Vaudenay–Vuagnoux,
2013 S–Sušil–V–V, . . .

“These are academic papers!

Nobody was actually attacked.”

Implementations, optimizations
of k -recovery attack: 2001
Stubblefield–Ioannidis–Rubin,
2004 KoreK, 2004 Devine, 2005
d'Otreppe, 2006 Klein, 2007
Tews–Weinmann–Pyshkin, 2010
Sepehrdad–Vaudenay–Vuagnoux,
2013 S–Sušil–V–V, . . .

“These are academic papers!

Nobody was actually attacked.”

Fact: WEP **blamed** for 2007 theft
of 45 million credit-card numbers
from T. J. Maxx. Subsequent
lawsuit **settled** for \$409000000.

Keeloq

Wikipedia: “KeeLoq is or was used in many remote keyless entry systems by such companies as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, Volkswagen Group, Clifford, Shurlok, Jaguar, etc.”

2007 Indesteege–Keller–
Biham–Dunkelman–Preneel

“How to steal cars” :

recover 64-bit KeeLoq key
using 2^{16} known plaintexts,
only $2^{44.5}$ encryptions.

2008 Eisenbarth–Kasper–Moradi–
Paar–Salmasizadeh–Shalmani

recovered system's *master* key,
allowing practically instantaneous
cloning of KeeLoq keys.

2008 Eisenbarth–Kasper–Moradi–
Paar–Salmasizadeh–Shalmani

recovered system's *master* key,
allowing practically instantaneous
cloning of KeeLoq keys.

1. Setup phase of this attack
watches power consumption
of Keeloq device. Is this
“bypassing” the cryptography?

2008 Eisenbarth–Kasper–Moradi– Paar–Salmasizadeh–Shalmani

recovered system's *master* key,
allowing practically instantaneous
cloning of KeeLoq keys.

1. Setup phase of this attack
watches power consumption
of Keeloq device. Is this
“bypassing” the cryptography?
2. If all the “*X* is weak” press
comes from academics, is it safe
to conclude that real attackers
aren't breaking *X*? How often do
real attackers issue press releases?

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from [VMWare](#), [Dell](#), etc.: switch from “AES-128” to “SALSA20-256” for the “best user experience”. Apparently AES slows down network graphics.

VMWare View

VMWare View is a remote desktop protocol supported by many low-cost terminals.

Recommendation from [VMWare](#), [Dell](#), etc.: switch from “AES-128” to “SALSA20-256” for the “best user experience”. Apparently AES slows down network graphics.

Closer look at documentation: “AES-128” and “SALSA20-256” are actually “AES-128-GCM” and “Salsa20-256-Round12”.

AES-128-GCM includes AES *and* message authentication.

No indication that VMWare's "Salsa20-256-Round12" includes any message authentication.

Can attacker forge packets?

One *can* easily combine Salsa20 with message authentication, but *does* VMWare do this?

Salsa20 has speed and security advantages over AES, but both Salsa20 and AES are *unauthenticated* ciphers.

User needs *authenticated* cipher.

SSL/TLS/HTTPS

Standard AES-CBC encryption
of a packet (p_0, p_1, p_2) :

send random v ,

$$c_0 = \text{AES}_k(p_0 \oplus v),$$

$$c_1 = \text{AES}_k(p_1 \oplus c_0),$$

$$c_2 = \text{AES}_k(p_2 \oplus c_1).$$

SSL/TLS/HTTPS

Standard AES-CBC encryption
of a packet (p_0, p_1, p_2) :

send random v ,

$$c_0 = \text{AES}_k(p_0 \oplus v),$$

$$c_1 = \text{AES}_k(p_1 \oplus c_0),$$

$$c_2 = \text{AES}_k(p_2 \oplus c_1).$$

AES-CBC encryption in SSL:

retrieve last block c_{-1}

from previous ciphertext; send

$$c_0 = \text{AES}_k(p_0 \oplus c_{-1}),$$

$$c_1 = \text{AES}_k(p_1 \oplus c_0),$$

$$c_2 = \text{AES}_k(p_2 \oplus c_1).$$

SSL lets attacker choose p_0
as function of c_{-1} ! Very bad.

2002 Möller:

To check a guess g for (e.g.) p_{-3} ,
choose $p_0 = c_{-1} \oplus g \oplus c_{-4}$,
compare c_0 to c_{-3} .

2006 Bard:

malicious code in browser should
be able to carry out this attack,
especially if high-entropy data
is split across blocks.

2011 Duong–Rizzo “BEAST”:

fast attack fully implemented,
including controlled variable split.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

“Authenticate-then-encrypt” :
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

Countermeasure in browsers:
send a content-free packet
just before sending real packet.

Attacker can also try to attack
CBC by forging *ciphertexts*,
but each SSL packet
includes an authenticator.

“Authenticate-then-encrypt”:
SSL appends an authenticator,
pads reversibly to full block,
encrypts with CBC.

2001 Krawczyk:

This is provably secure.

2001 Vaudenay:

This is completely broken if attacker can distinguish padding failure from MAC failure.

2001 Vaudenay:

This is completely broken if attacker can distinguish padding failure from MAC failure.

2003 Canvel:

Obtain such a padding oracle by observing SSL server timing.

2001 Vaudenay:

This is completely broken if attacker can distinguish padding failure from MAC failure.

2003 Canvel:

Obtain such a padding oracle by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2001 Vaudenay:

This is completely broken if attacker can distinguish padding failure from MAC failure.

2003 Canvel:

Obtain such a padding oracle by observing SSL server timing.

Response in OpenSSL etc.:
always compute MAC
even if padding fails.

2013.02 AlFardan–Paterson

“Lucky 13”: watch timing more closely; attack still works.

“Cryptographic algorithm agility” :

“Cryptographic algorithm agility” :

(1) the pretense that bad crypto is okay if there’s a backup plan

“Cryptographic algorithm agility” :

(1) the pretense that bad crypto

is okay if there’s a backup plan +

(2) the pretense that there

is in fact a backup plan.

“Cryptographic algorithm agility” :

(1) the pretense that bad crypto is okay if there’s a backup plan +

(2) the pretense that there is in fact a backup plan.

SSL has a crypto switch

that in theory allows

switching to AES-GCM.

But most SSL software

doesn’t support AES-GCM.

“Cryptographic algorithm agility” :
(1) the pretense that bad crypto is okay if there’s a backup plan +
(2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM.

But most SSL software doesn’t support AES-GCM.

The software does support one non-CBC option:

“Cryptographic algorithm agility” :
(1) the pretense that bad crypto is okay if there’s a backup plan +
(2) the pretense that there is in fact a backup plan.

SSL has a crypto switch that in theory allows switching to AES-GCM.

But most SSL software doesn’t support AES-GCM.

The software does support one non-CBC option: RC4.

2013.01: widely recommended, used for 50% of SSL traffic.

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: “The new attacks do not apply to RC4-based SSL.

... [protocol] designers [using RC4] should not be concerned.”

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: “The new attacks do not apply to RC4-based SSL.

... [protocol] designers [using RC4] should not be concerned.”

Problem: many nasty biases in RC4 output bytes z_1, z_2, \dots

Not as scary as WEP: SSL uses a hash to avoid related RC4 keys.

2001 Rivest: “The new attacks do not apply to RC4-based SSL.

... [protocol] designers [using RC4] should not be concerned.”

Problem: many nasty biases in RC4 output bytes z_1, z_2, \dots

2013 AlFardan–Bernstein–

Paterson–Poettering–Schuldt,

“On the security of RC4 in TLS”:

Force target cookie into many

RC4 sessions. Use RC4 biases

to find cookie from ciphertexts.

The single-byte biases:

2001 Mantin–Shamir:

$z_2 \rightarrow 0$.

The single-byte biases:

2001 Mantin–Shamir:

$$z_2 \rightarrow 0.$$

2002 Mironov:

$$z_1 \not\rightarrow 0, z_1 \not\rightarrow 1, z_1 \rightarrow 2, \text{ etc.}$$

The single-byte biases:

2001 Mantin–Shamir:

$$z_2 \rightarrow 0.$$

2002 Mironov:

$$z_1 \not\rightarrow 0, z_1 \not\rightarrow 1, z_1 \rightarrow 2, \text{ etc.}$$

2011 Maitra–Paul–Sen Gupta:

$$z_3 \rightarrow 0, z_4 \rightarrow 0, \dots, z_{255} \rightarrow 0,$$

contrary to Mantin–Shamir claim.

The single-byte biases:

2001 Mantin–Shamir:

$z_2 \rightarrow 0$.

2002 Mironov:

$z_1 \not\rightarrow 0$, $z_1 \not\rightarrow 1$, $z_1 \rightarrow 2$, etc.

2011 Maitra–Paul–Sen Gupta:

$z_3 \rightarrow 0$, $z_4 \rightarrow 0$, \dots , $z_{255} \rightarrow 0$,

contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–

Sarkar: $z_{16} \rightarrow 240$.

(This is specific to 128-bit keys.)

The single-byte biases:

2001 Mantin–Shamir:

$z_2 \rightarrow 0$.

2002 Mironov:

$z_1 \not\rightarrow 0$, $z_1 \not\rightarrow 1$, $z_1 \rightarrow 2$, etc.

2011 Maitra–Paul–Sen Gupta:

$z_3 \rightarrow 0$, $z_4 \rightarrow 0$, \dots , $z_{255} \rightarrow 0$,
contrary to Mantin–Shamir claim.

2011 Sen Gupta–Maitra–Paul–

Sarkar: $z_{16} \rightarrow 240$.

(This is specific to 128-bit keys.)

But wait: there's more!

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \dots, 256\}$, all j ;
found \approx **65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt:
accurately computed $\Pr[z_i = j]$
for all $i \in \{1, \dots, 256\}$, all j ;
found \approx **65536** single-byte biases;
used *all* of them in SSL attack
via proper Bayesian analysis.

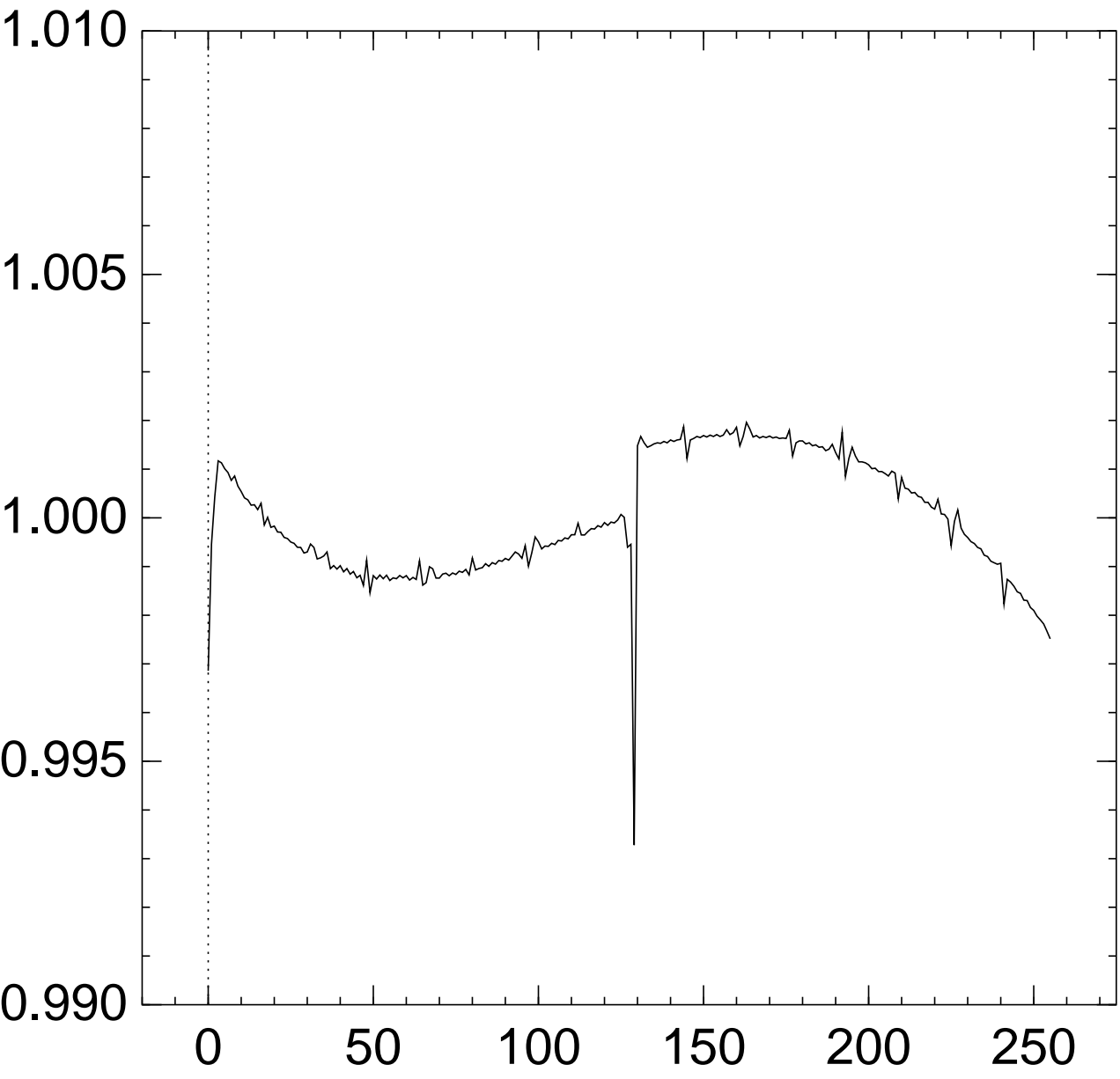
\approx 256 of these biases were found
independently (slightly earlier)

by 2013 Watanabe–Isobe–
Ohigashi–Morii, 2013 Isobe–
Ohigashi–Watanabe–Morii:

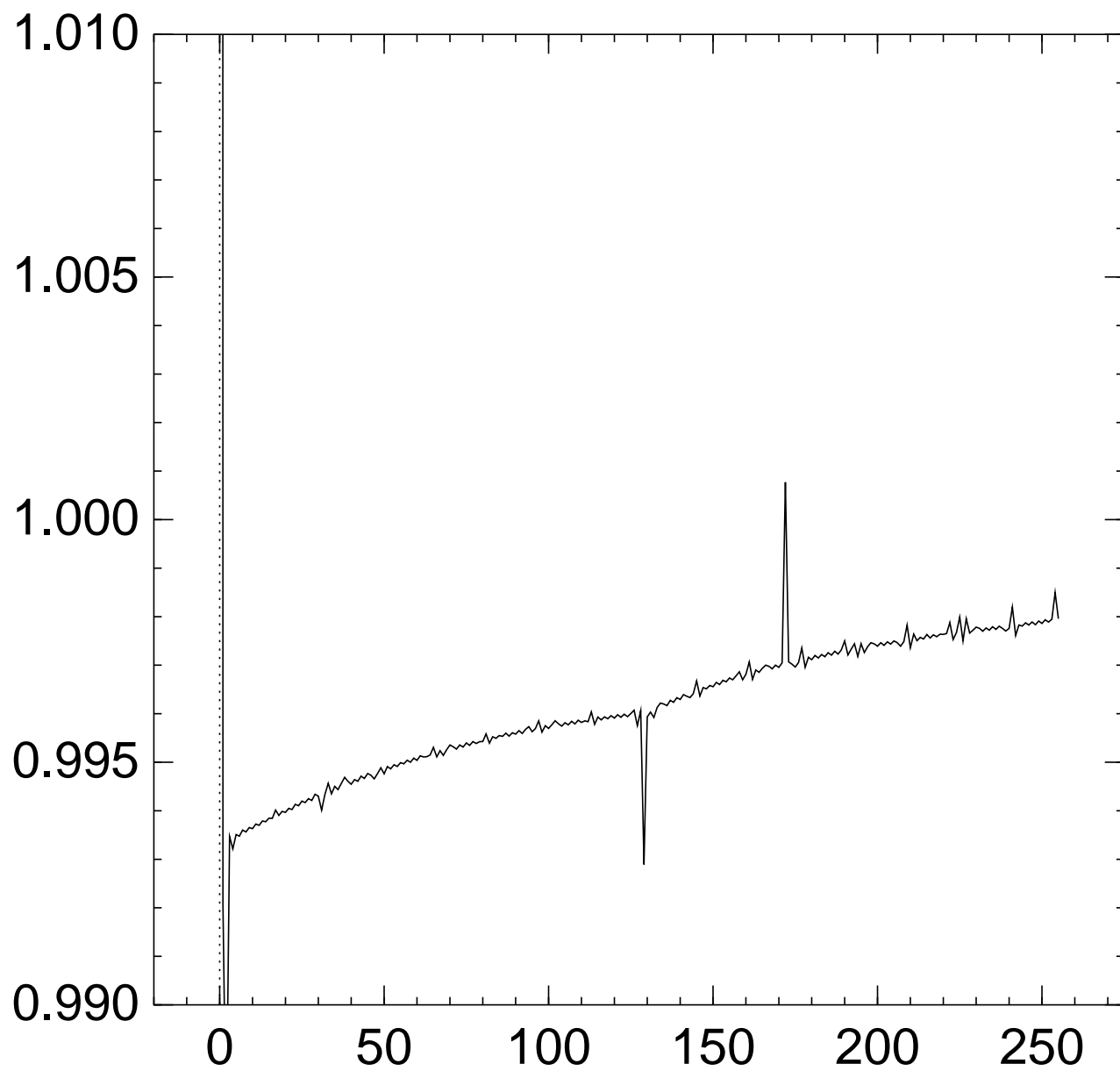
$z_{32} \rightarrow 224$, $z_{48} \rightarrow 208$, etc.;

$z_3 \rightarrow 131$; $z_i \rightarrow i$; $z_{256} \not\rightarrow 0$.

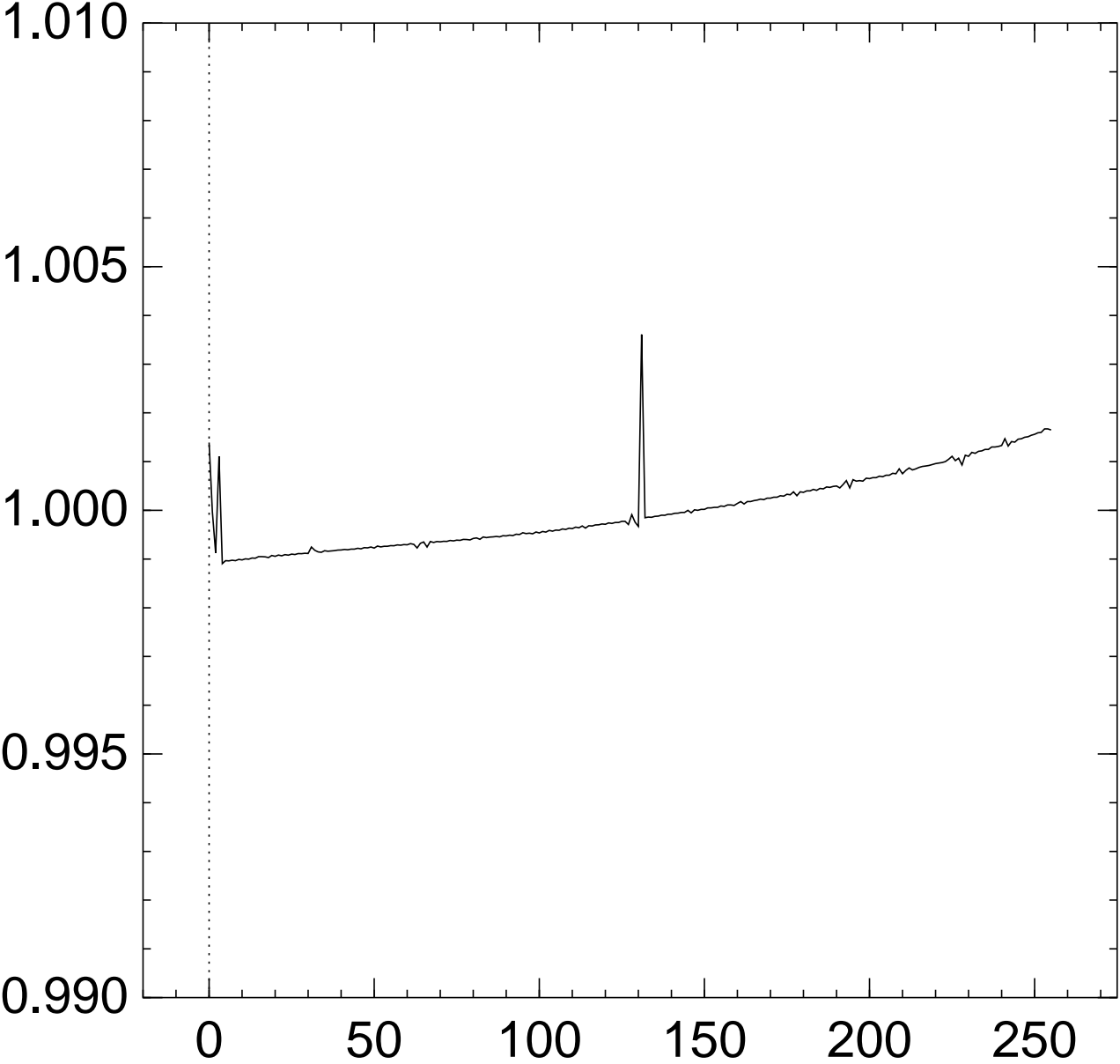
Graph of $256 \Pr[z_1 = x]$:



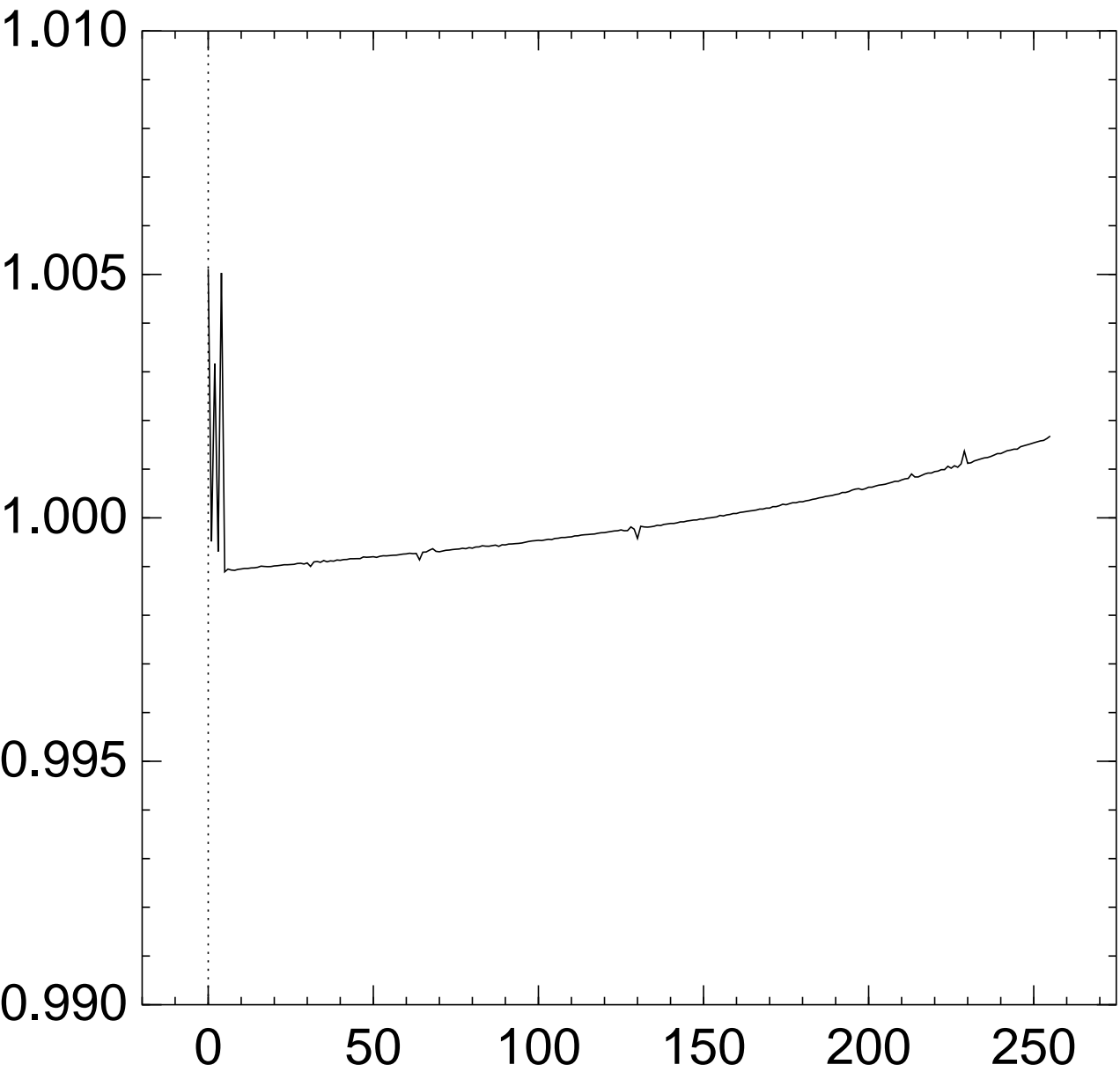
Graph of $256 \Pr[z_2 = x]$:



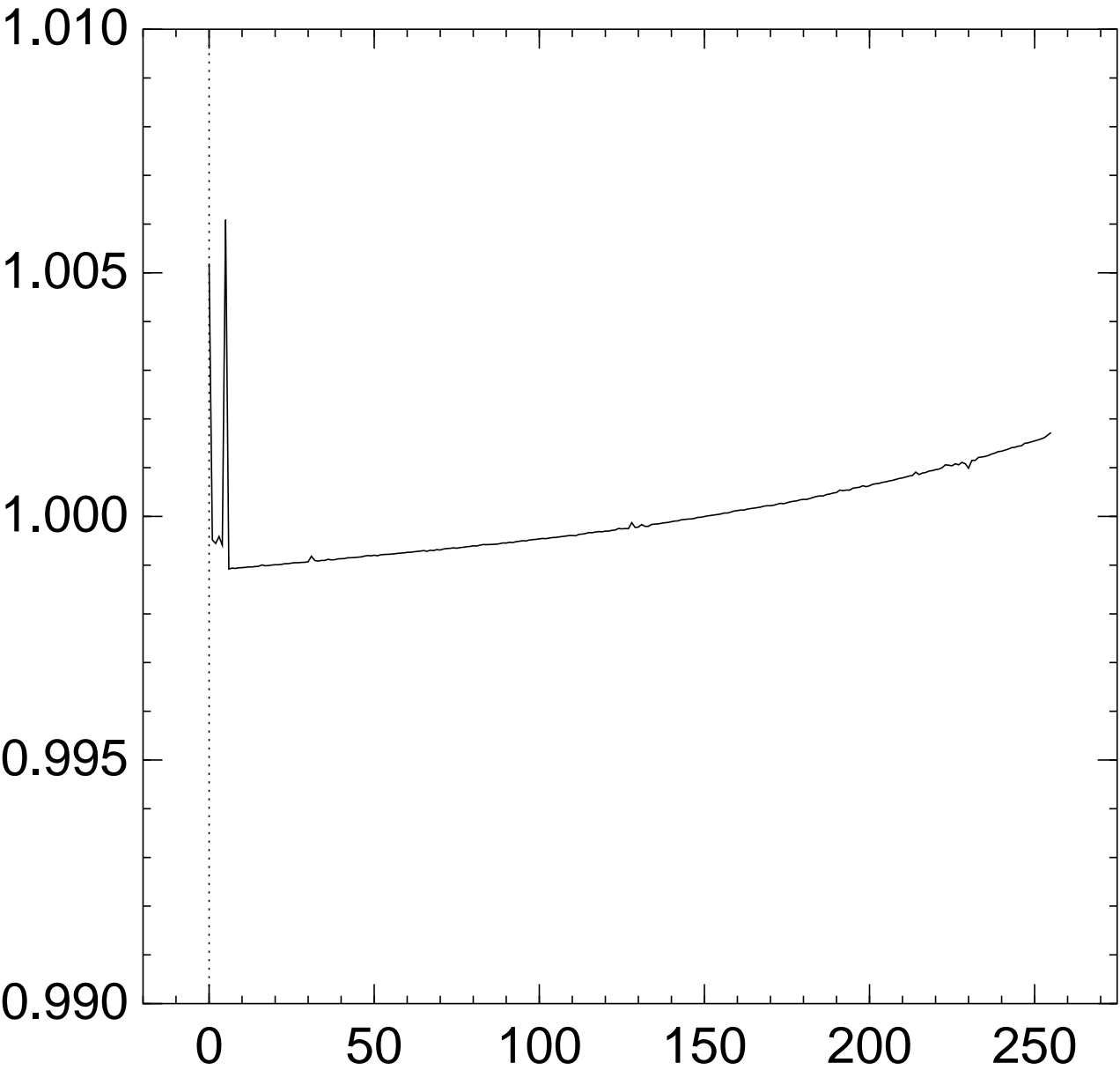
Graph of $256 \Pr[z_3 = x]$:



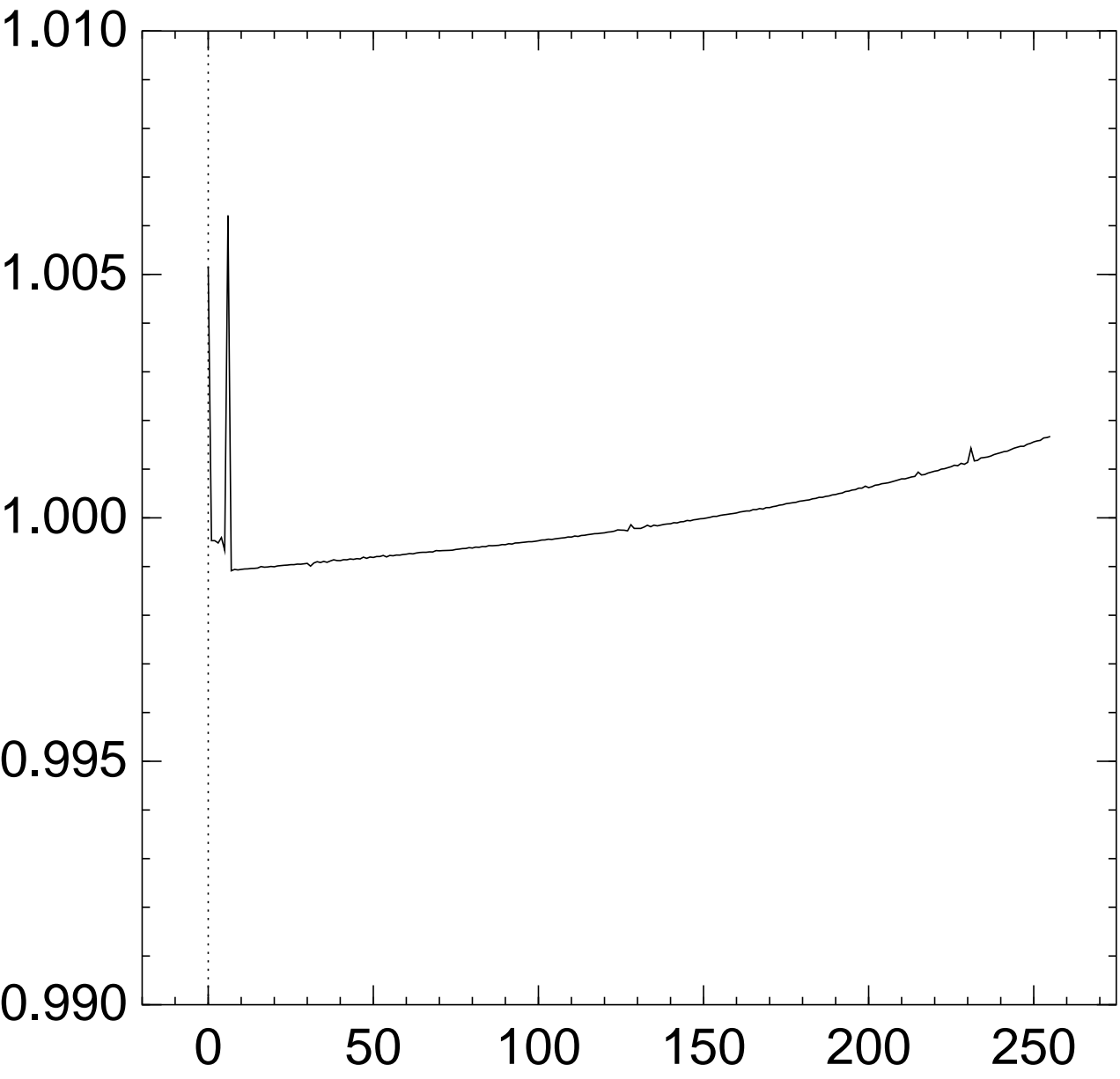
Graph of $256 \Pr[z_4 = x]$:



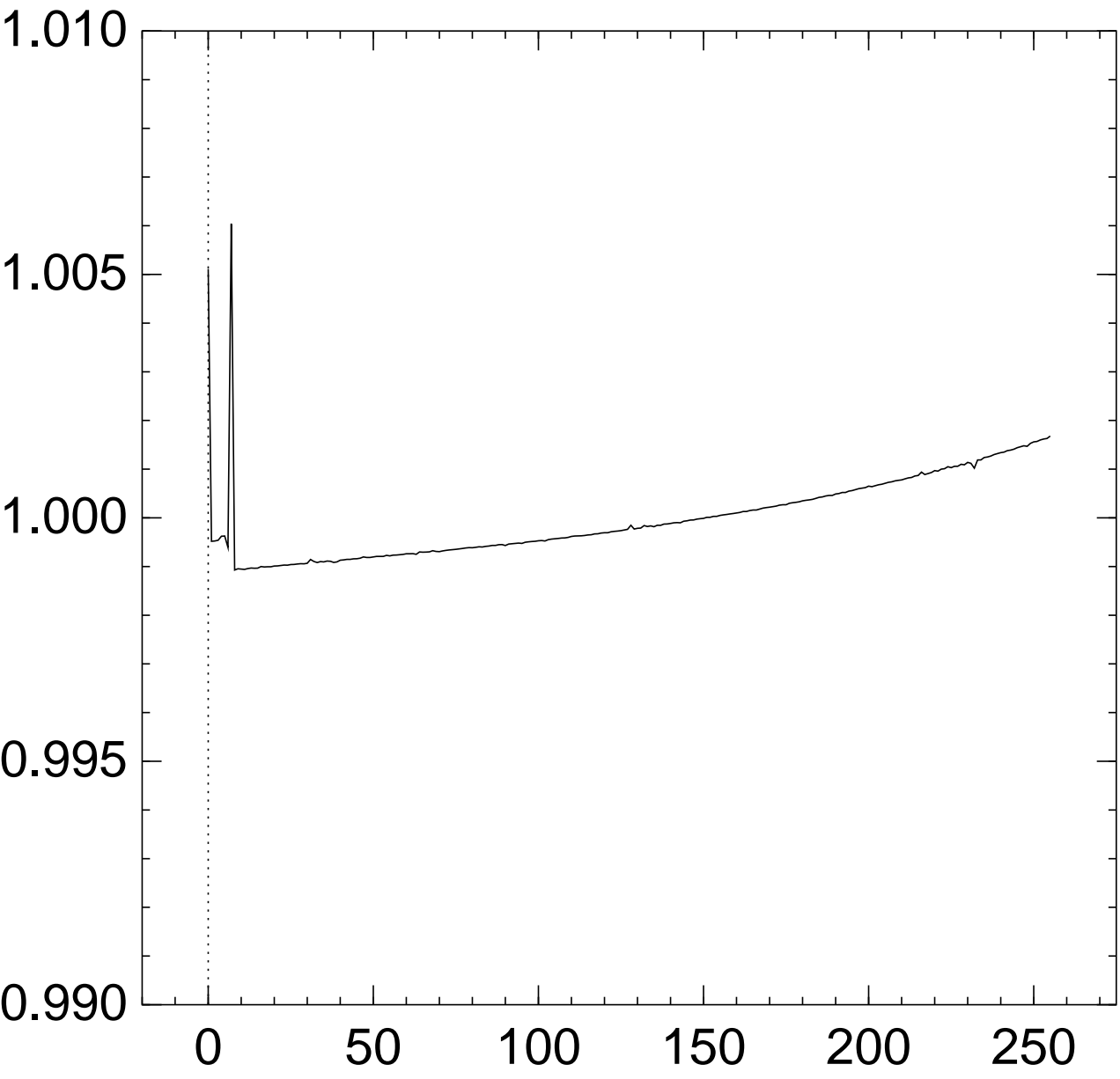
Graph of $256 \Pr[z_5 = x]$:



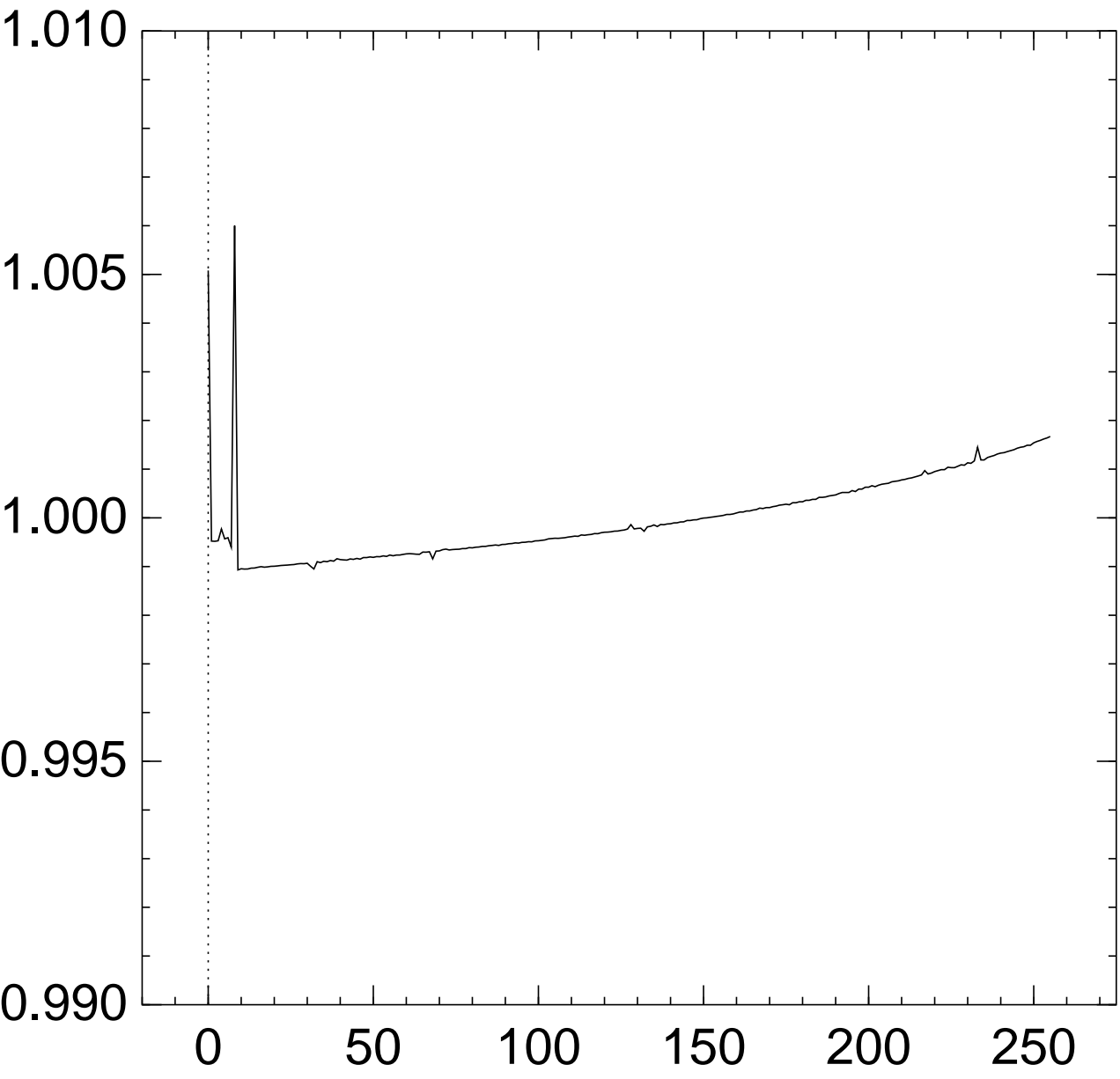
Graph of $256 \Pr[z_6 = x]$:



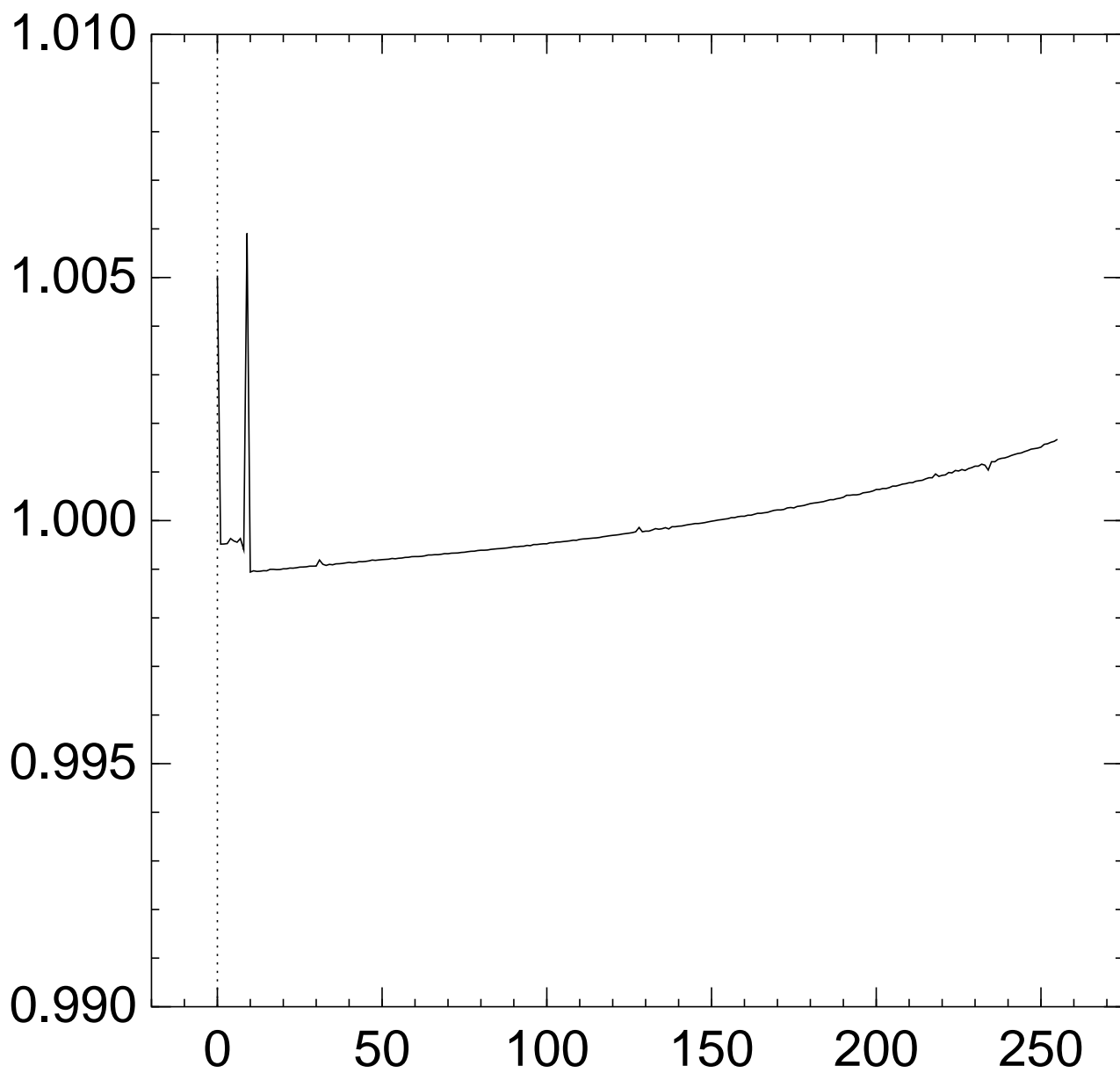
Graph of $256 \Pr[z_7 = x]$:



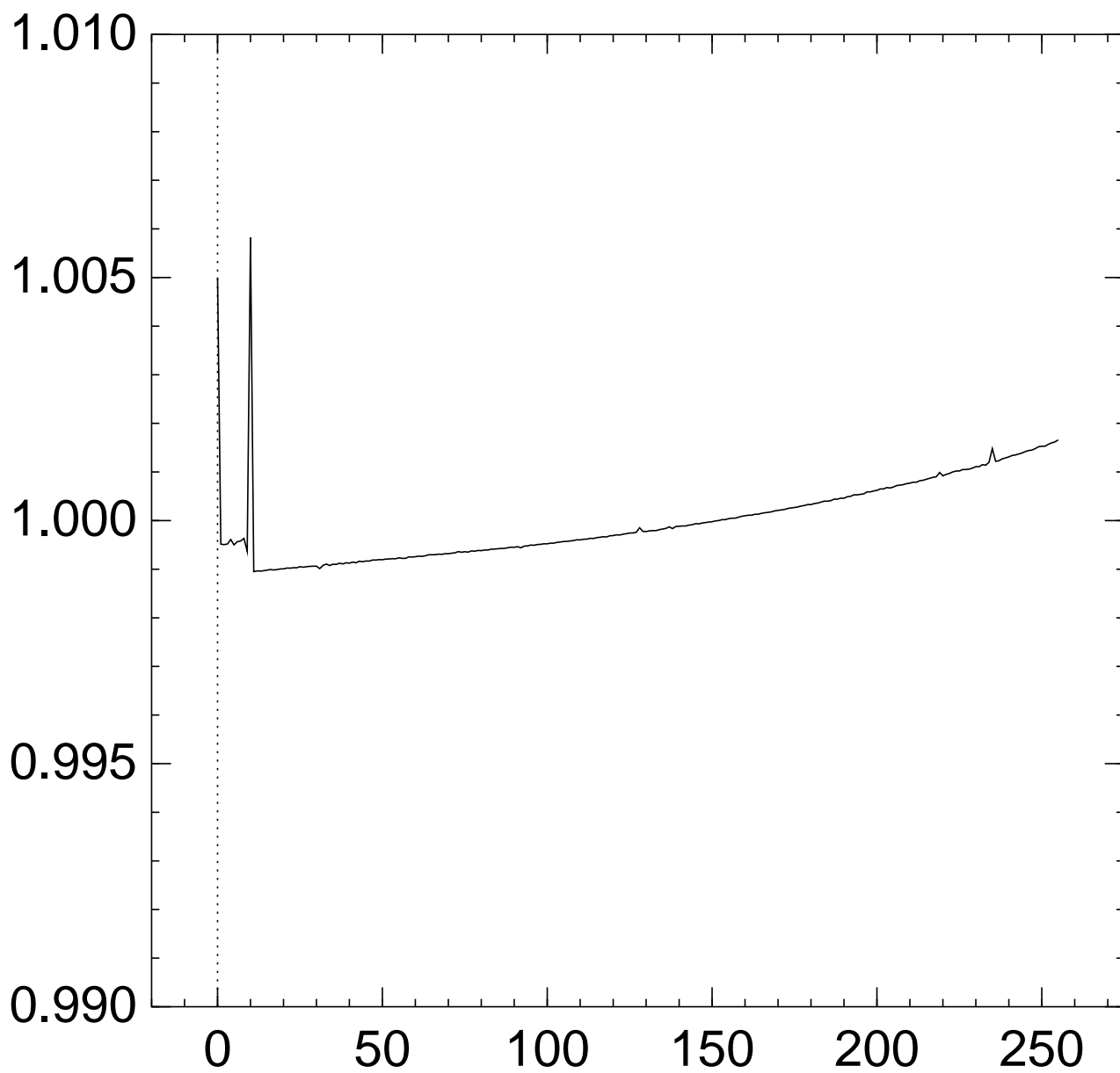
Graph of $256 \Pr[z_8 = x]$:



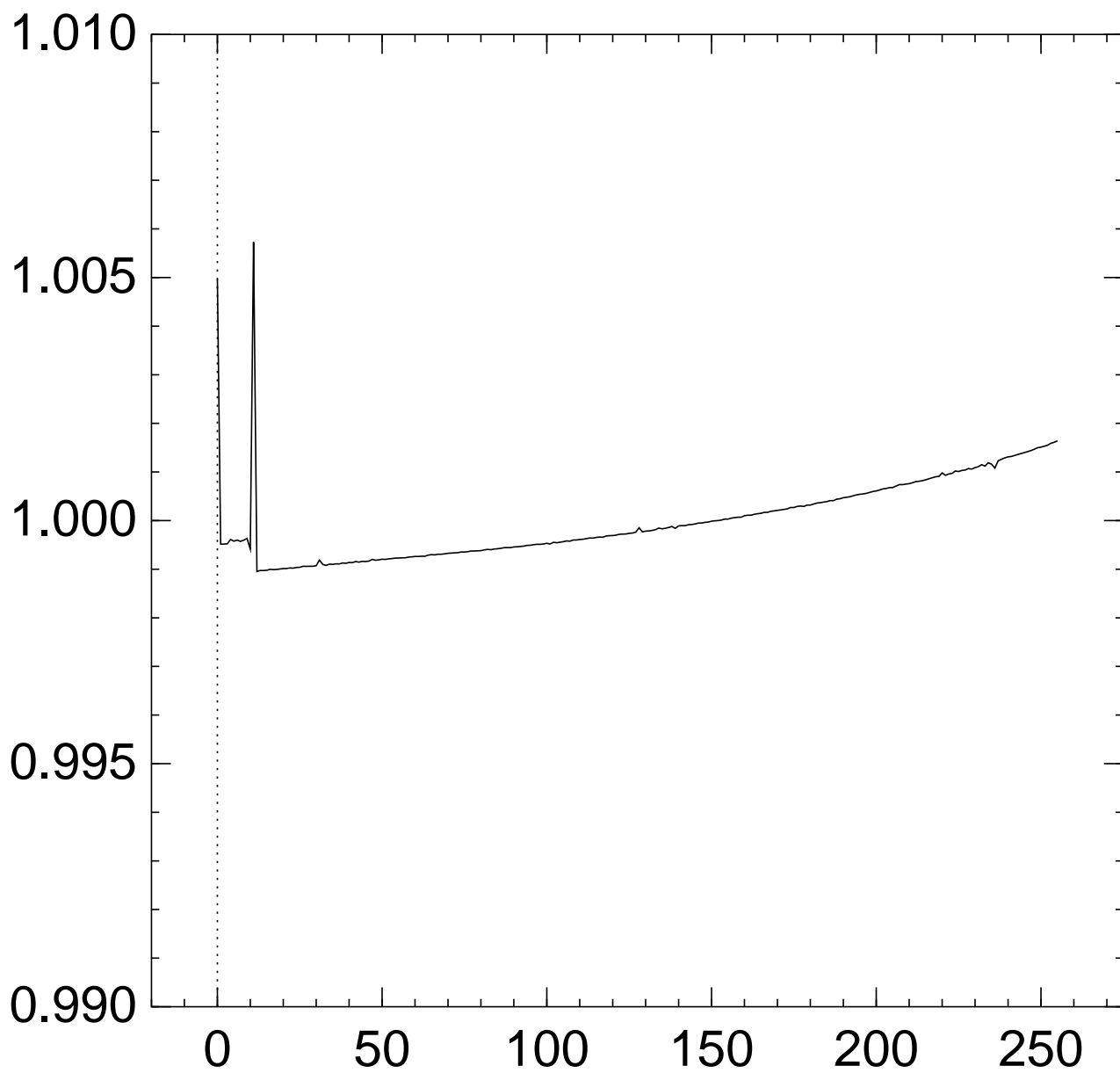
Graph of $256 \Pr[z_9 = x]$:



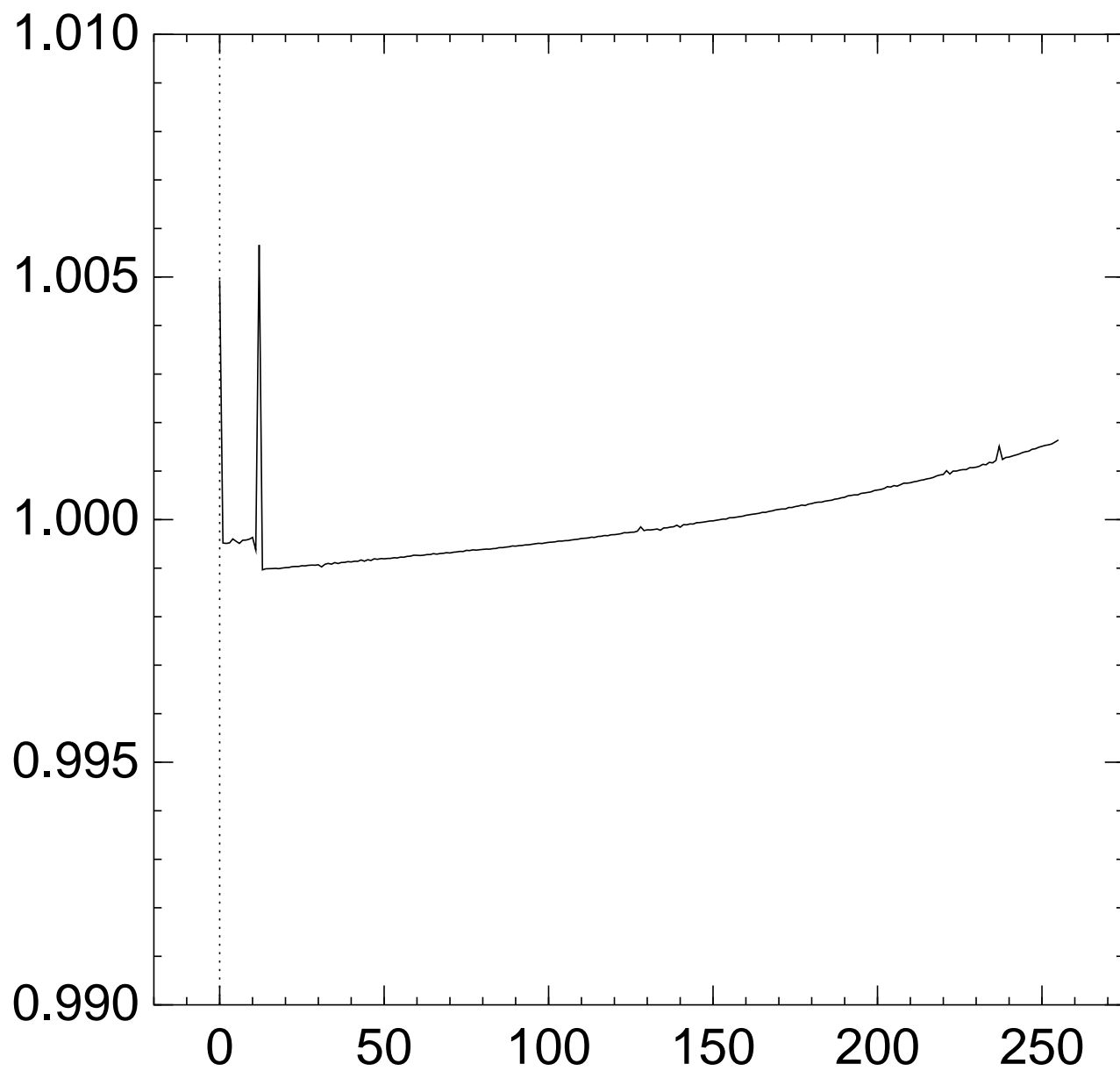
Graph of $256 \Pr[z_{10} = x]$:



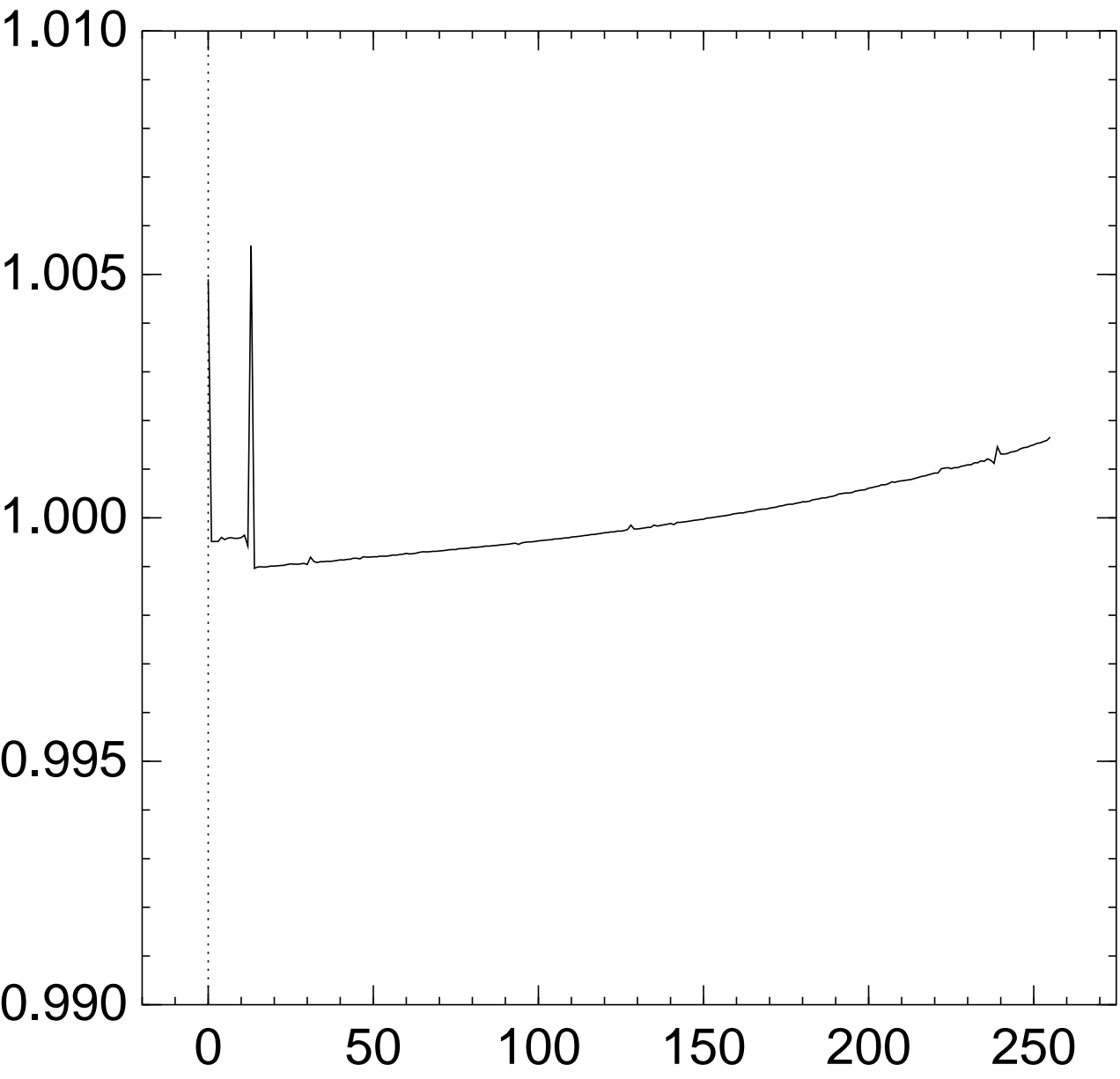
Graph of $256 \Pr[z_{11} = x]$:



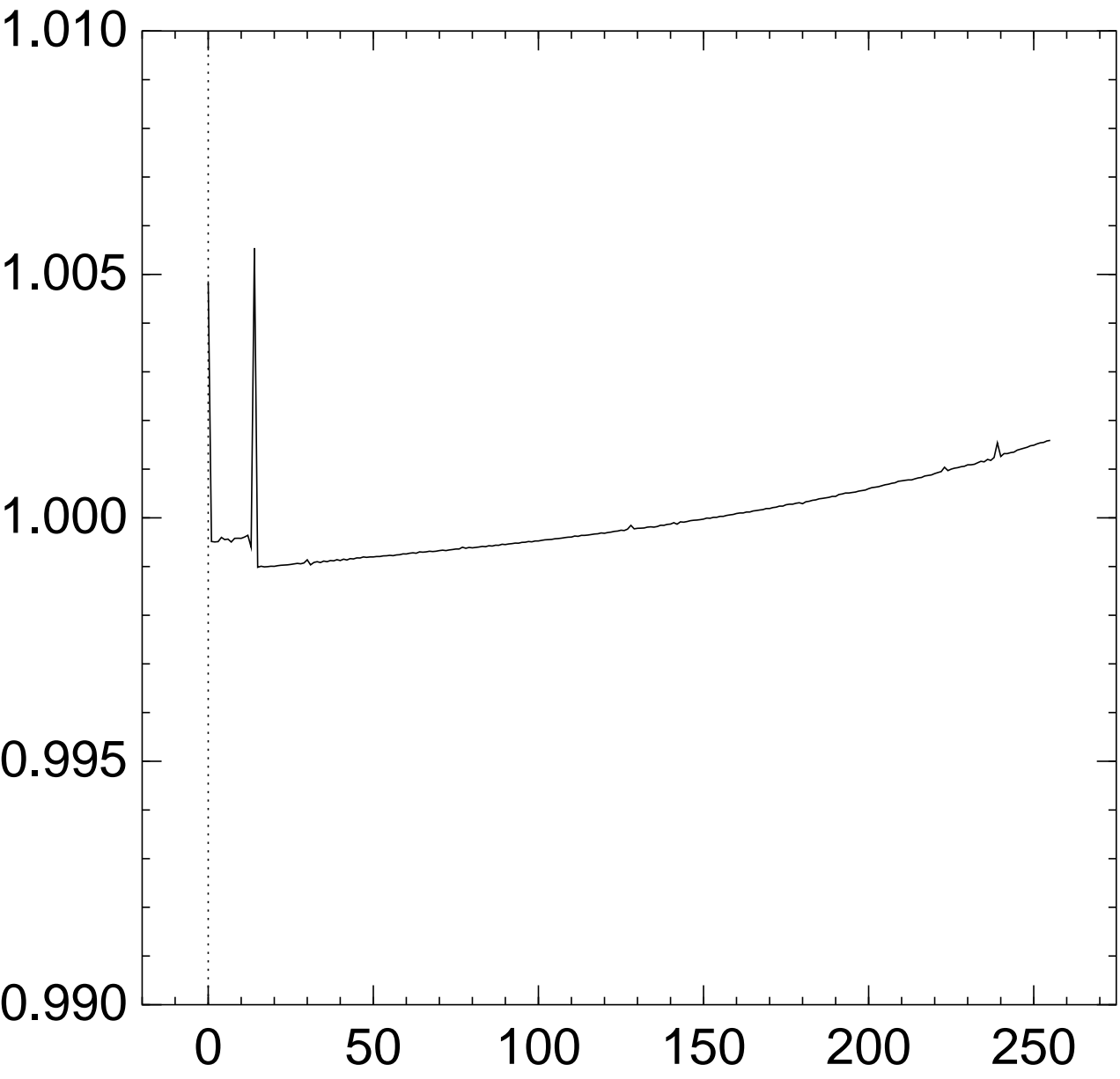
Graph of $256 \Pr[z_{12} = x]$:



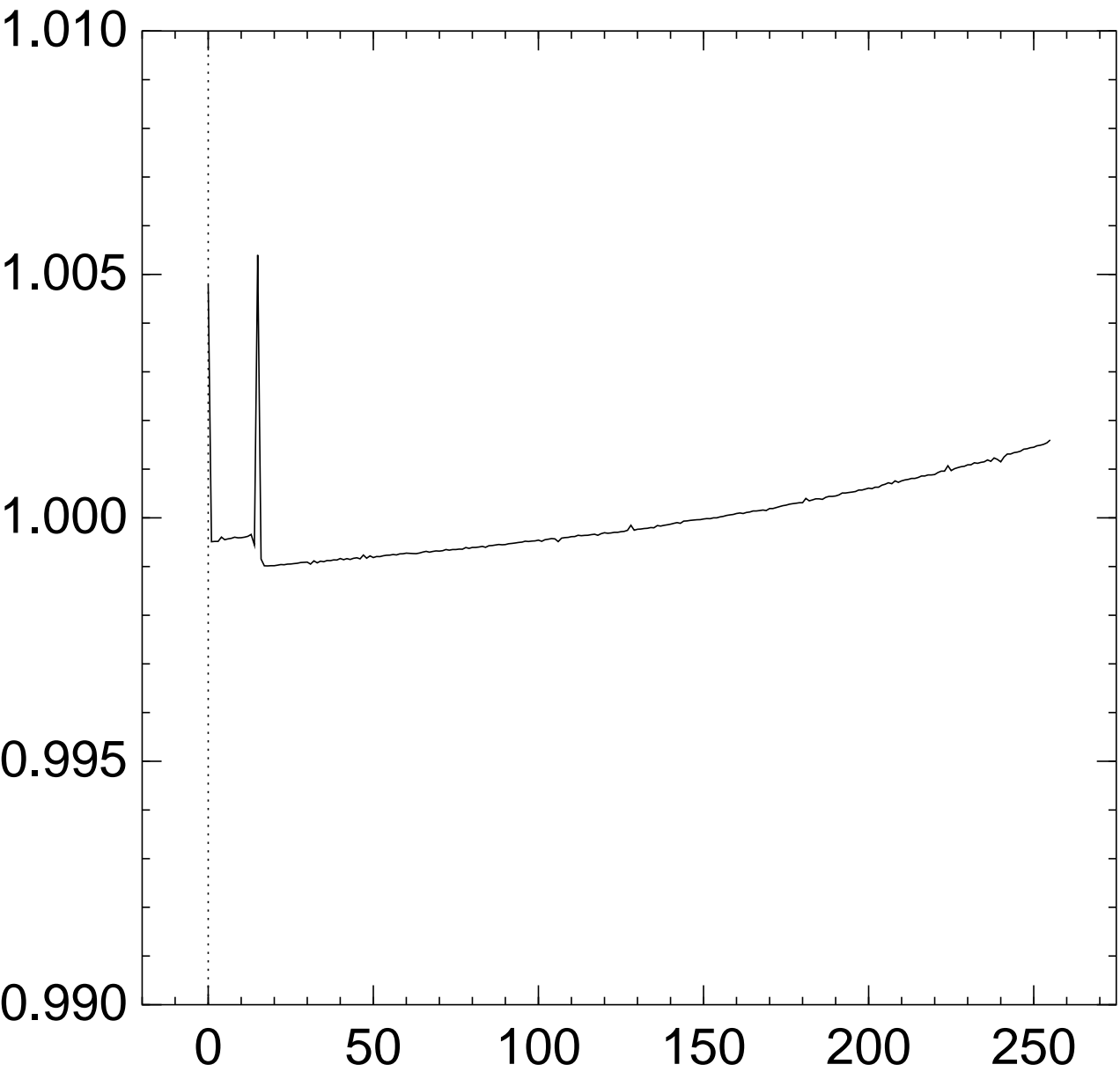
Graph of $256 \Pr[z_{13} = x]$:



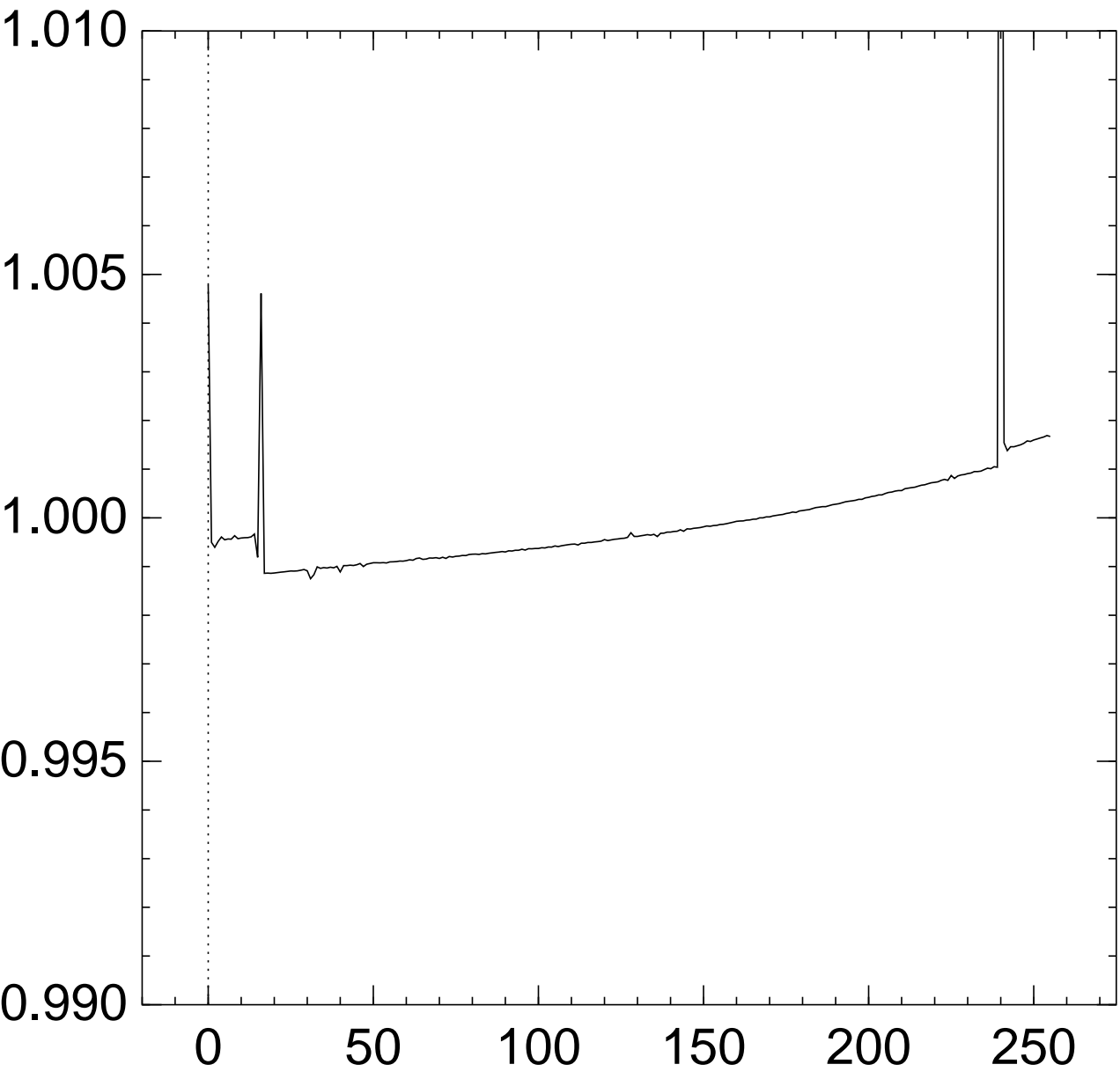
Graph of $256 \Pr[z_{14} = x]$:



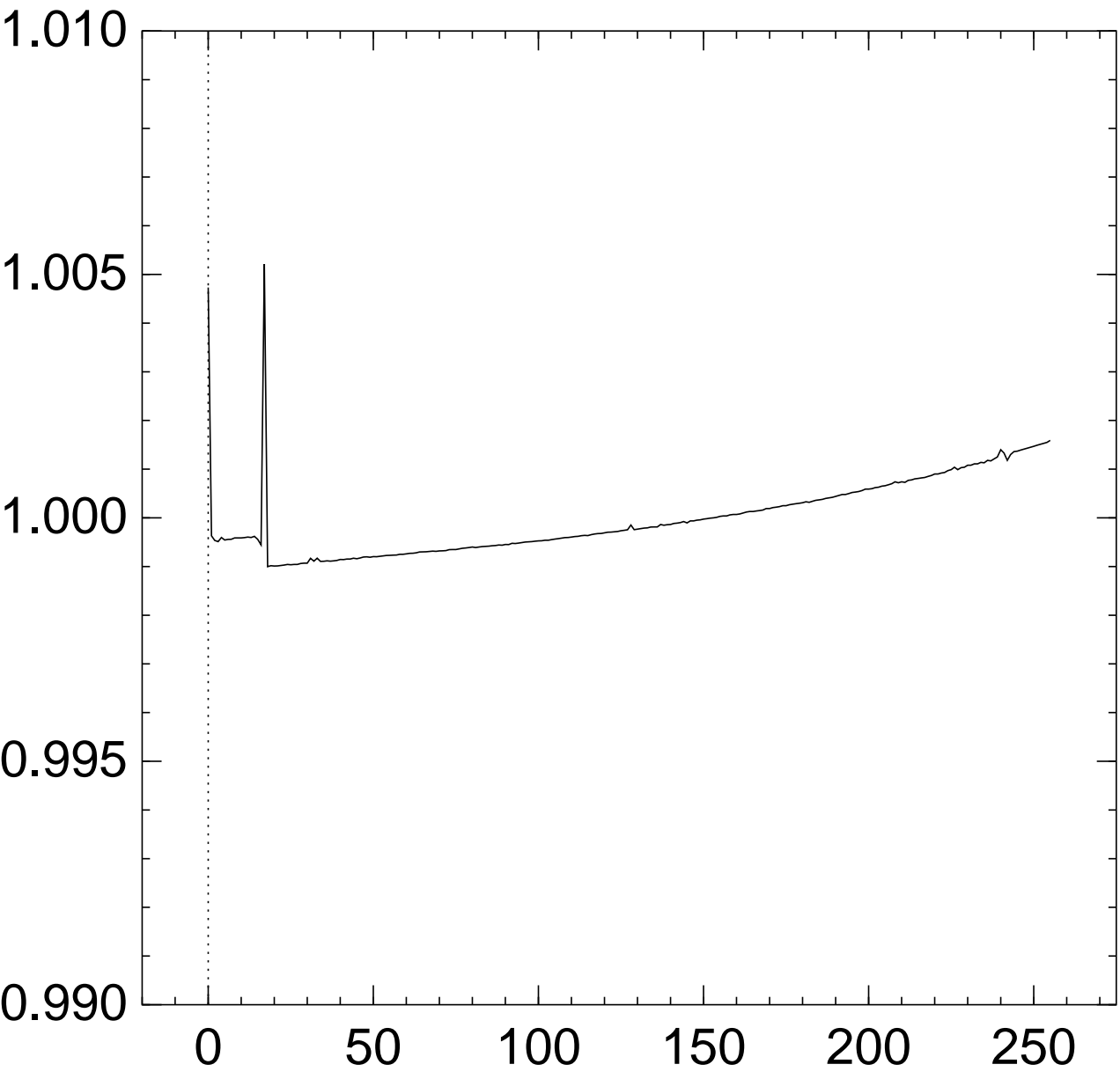
Graph of $256 \Pr[z_{15} = x]$:



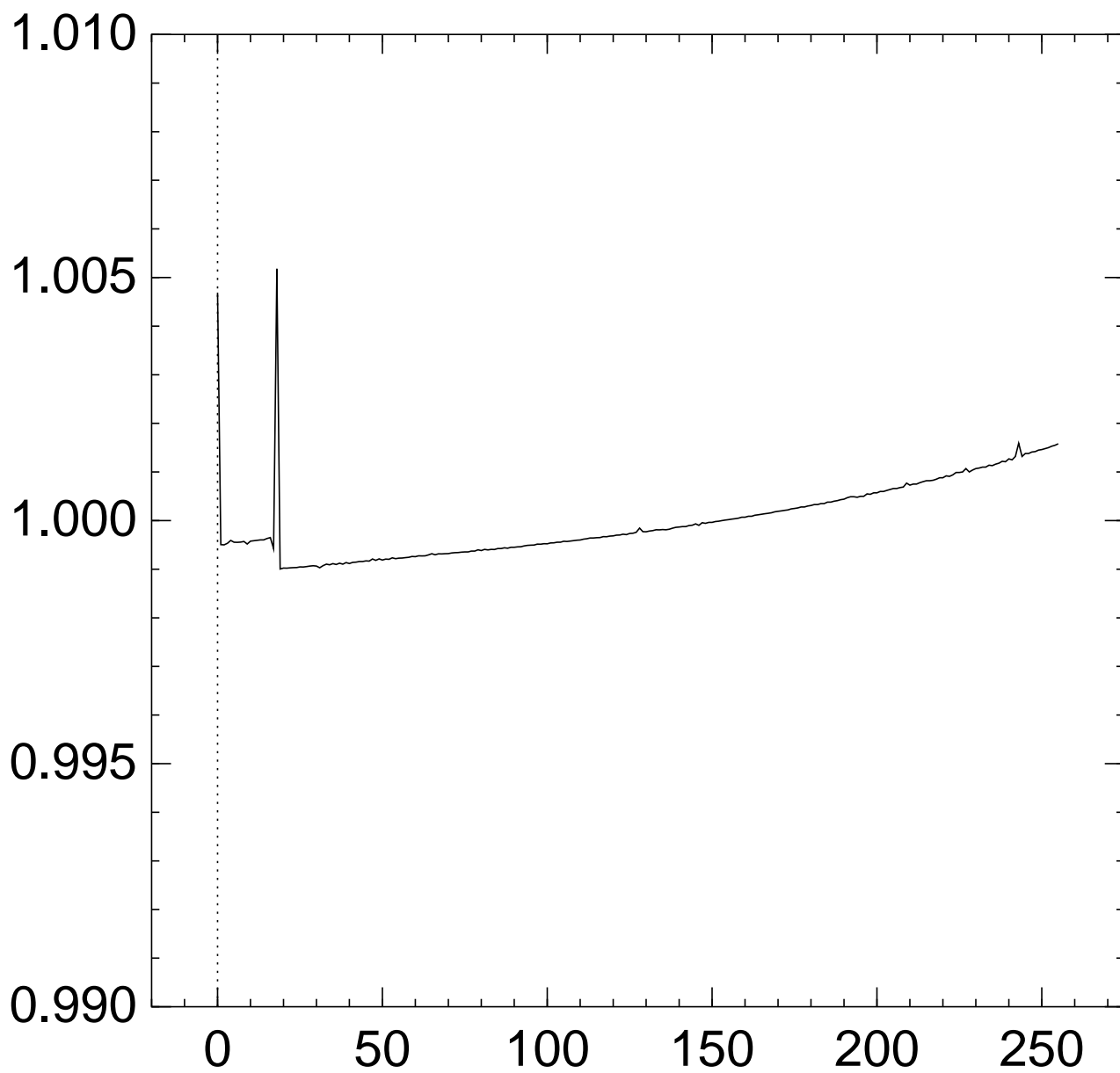
Graph of $256 \Pr[z_{16} = x]$:



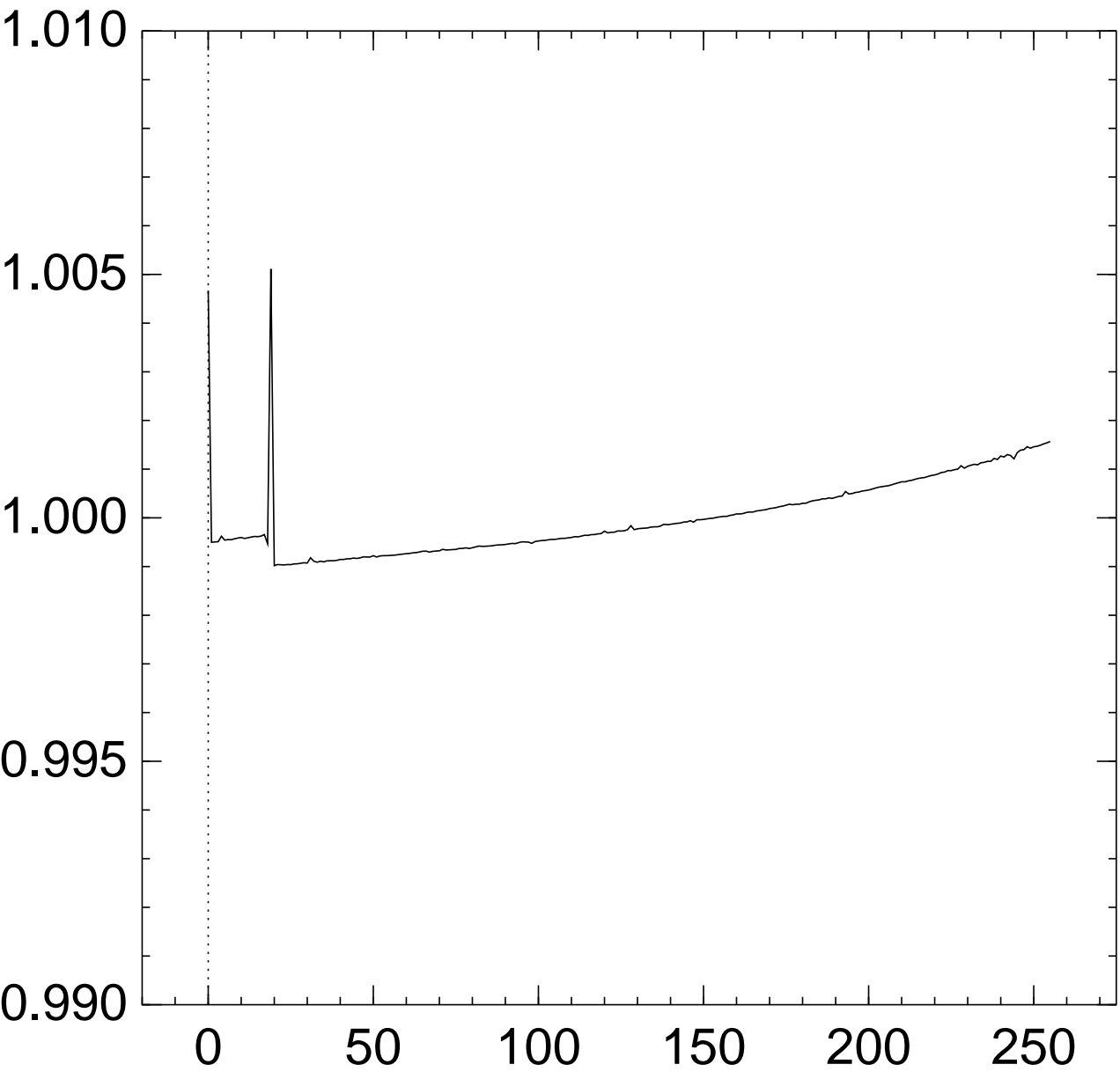
Graph of $256 \Pr[z_{17} = x]$:



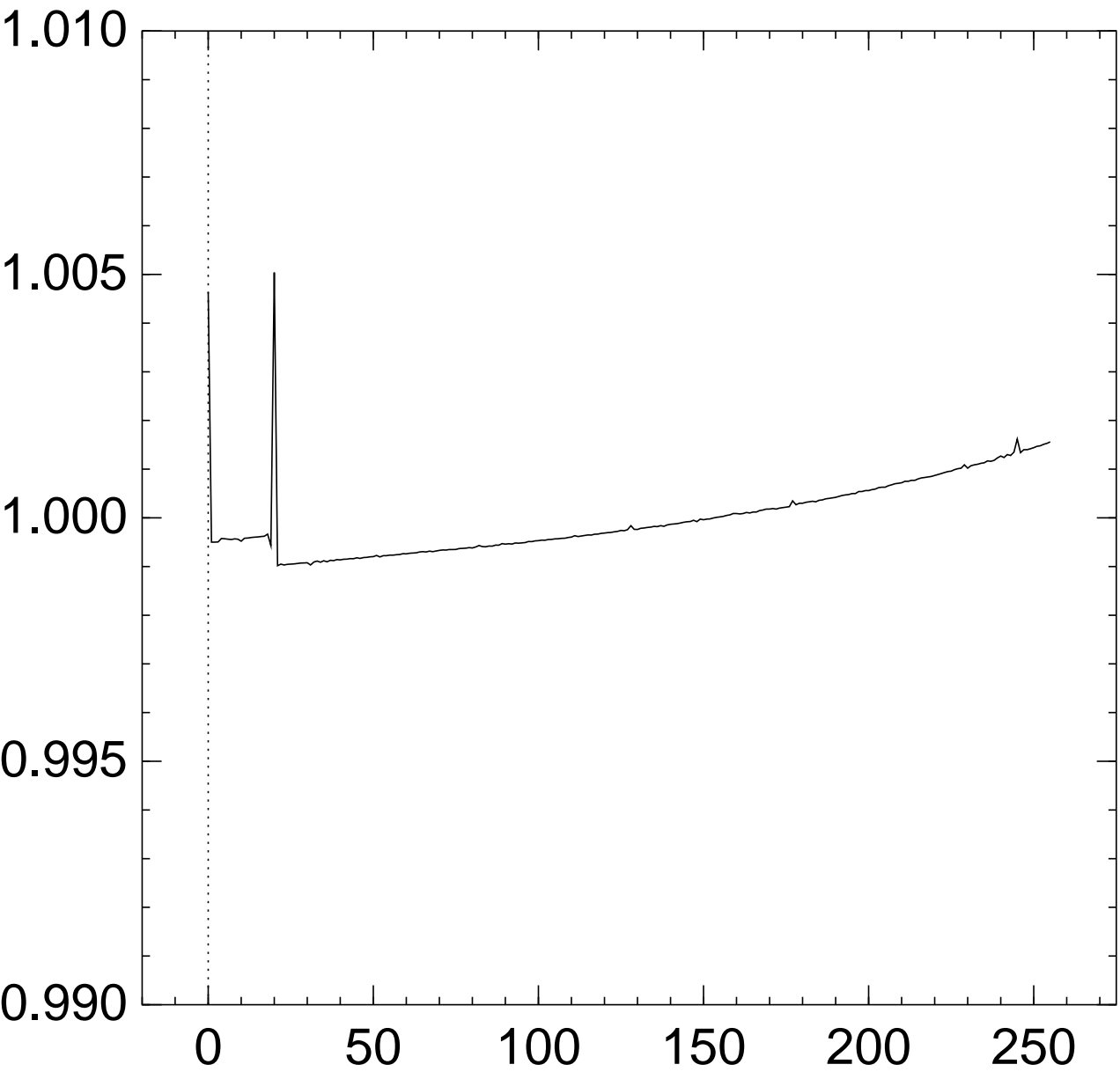
Graph of $256 \Pr[z_{18} = x]$:



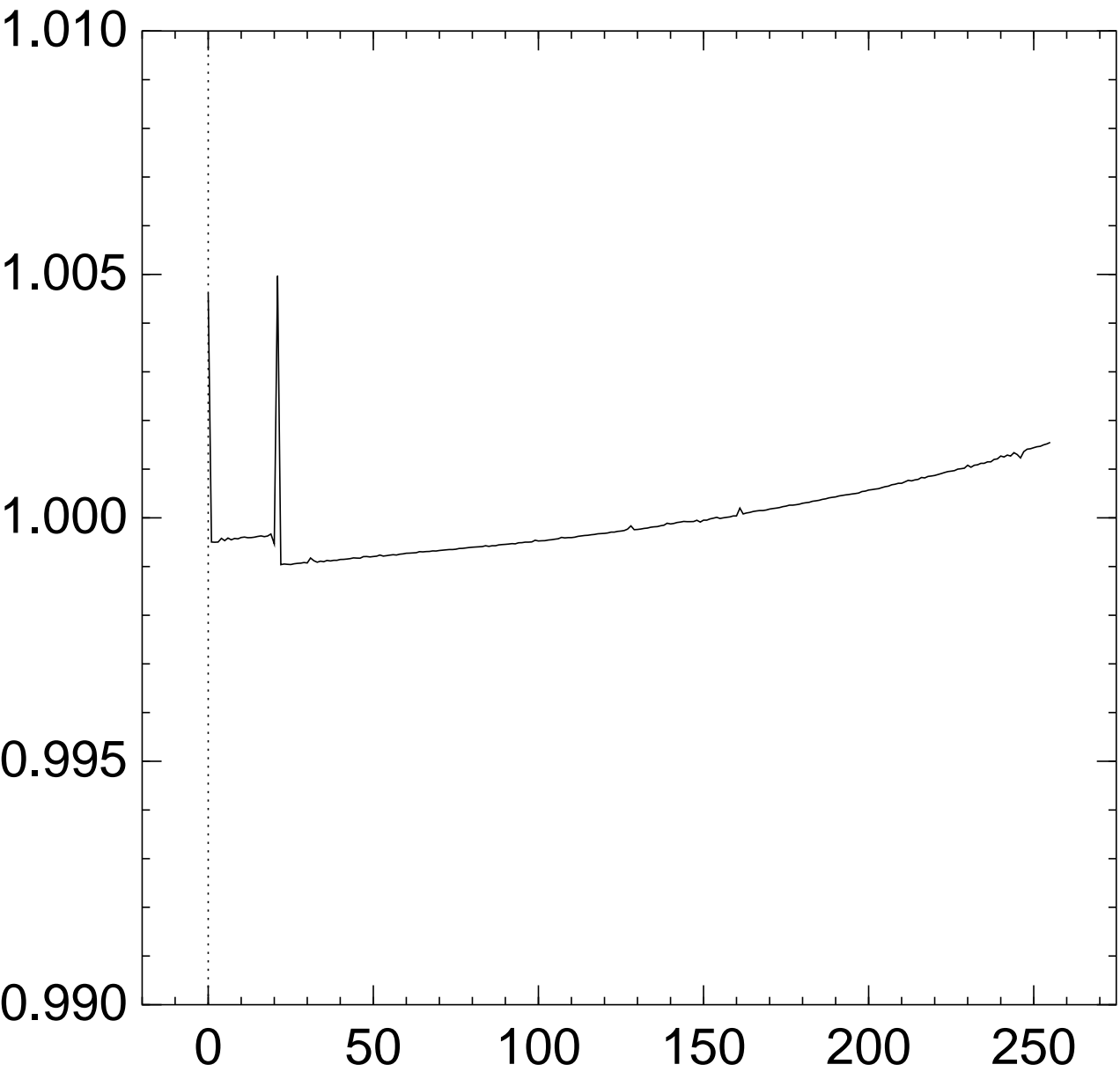
Graph of $256 \Pr[z_{19} = x]$:



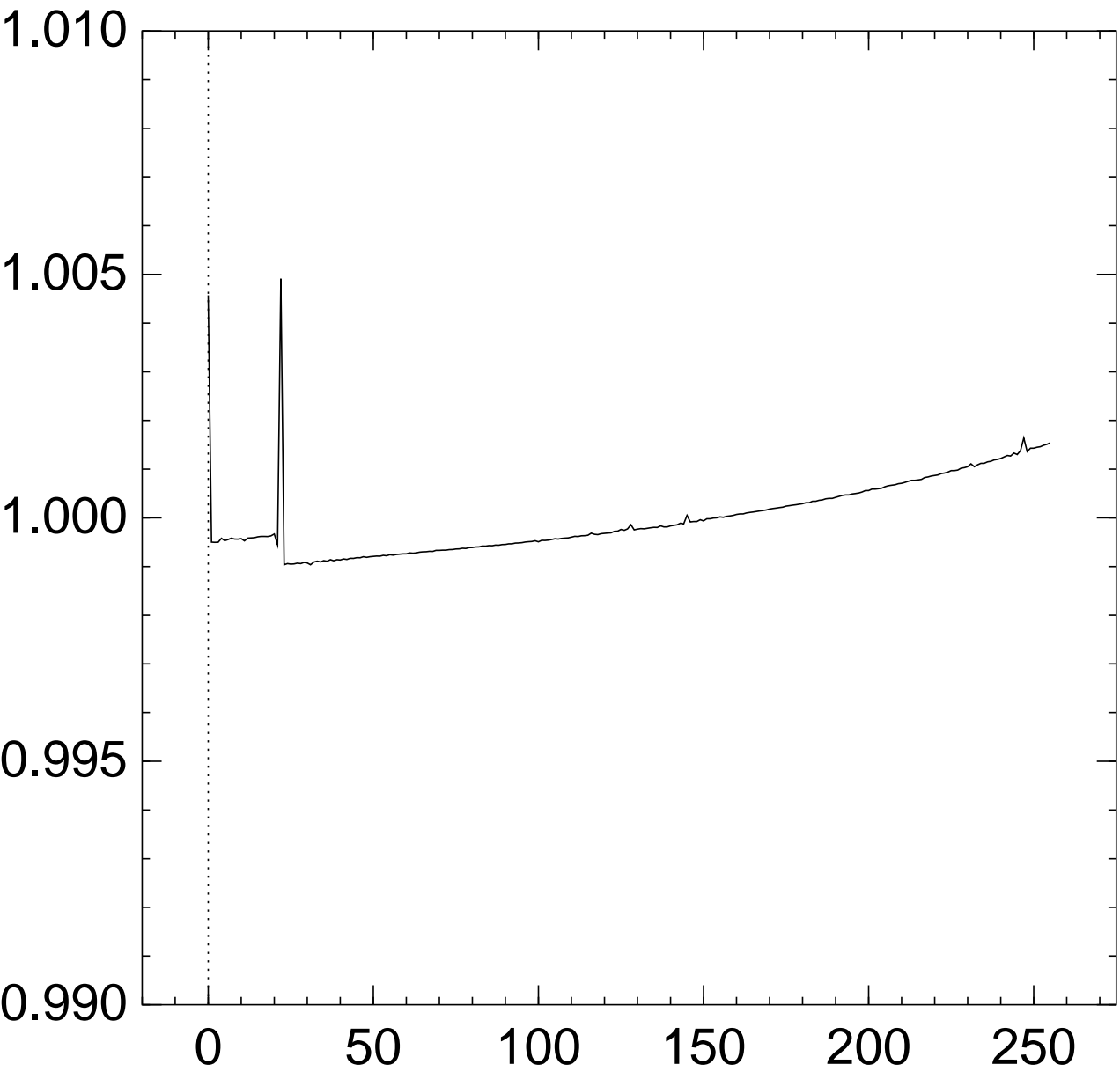
Graph of $256 \Pr[z_{20} = x]$:



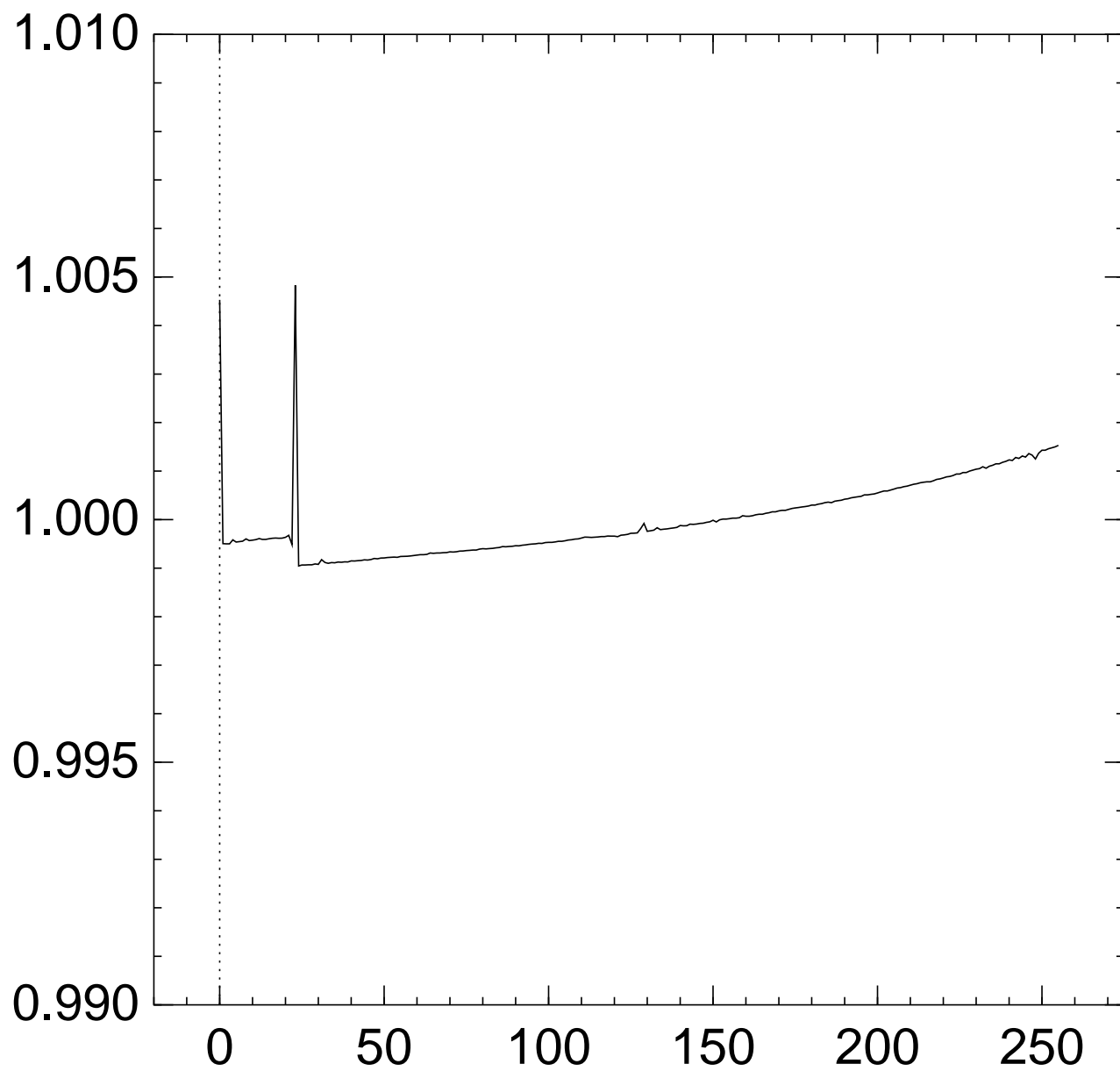
Graph of $256 \Pr[z_{21} = x]$:



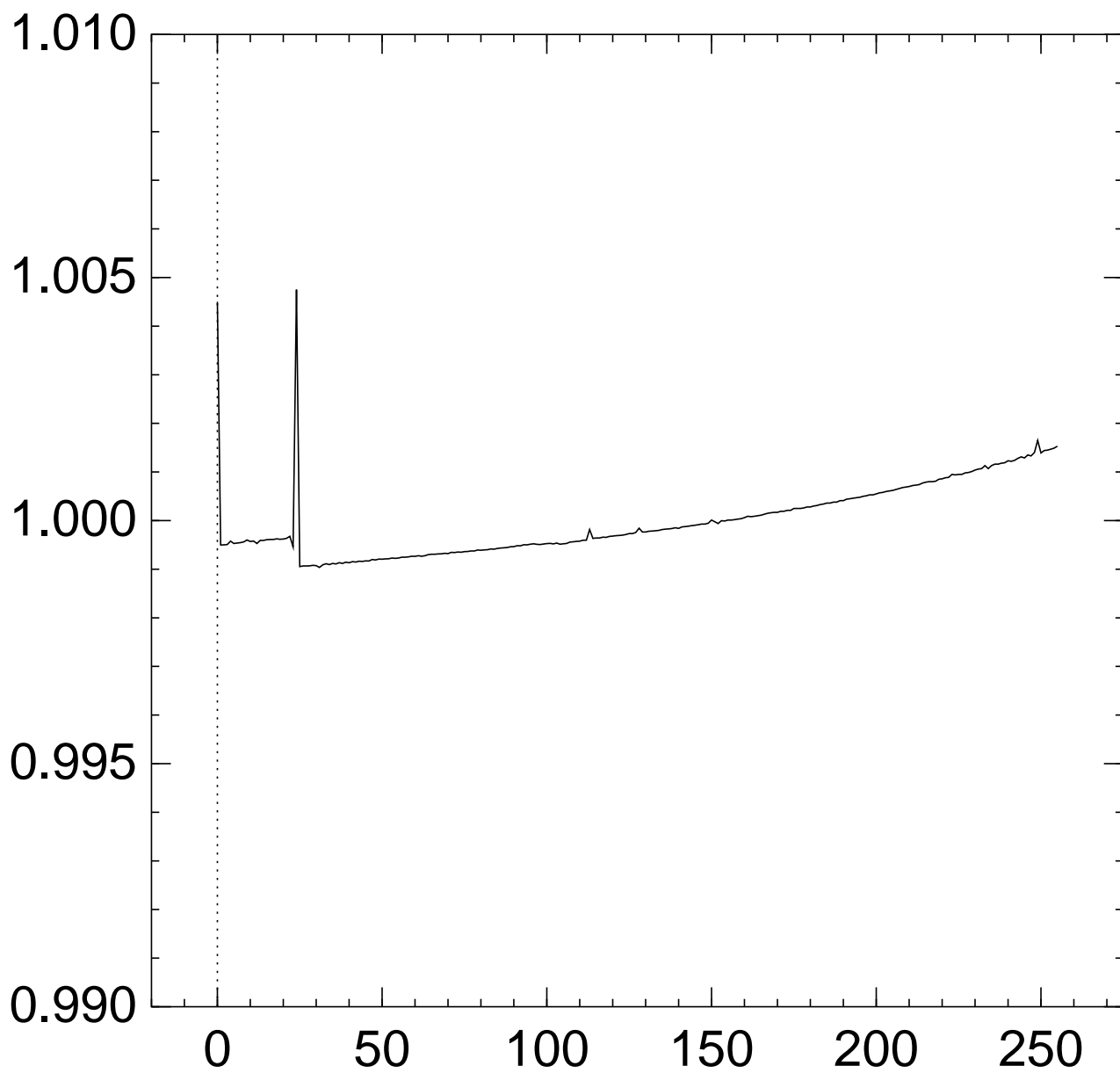
Graph of $256 \Pr[z_{22} = x]$:



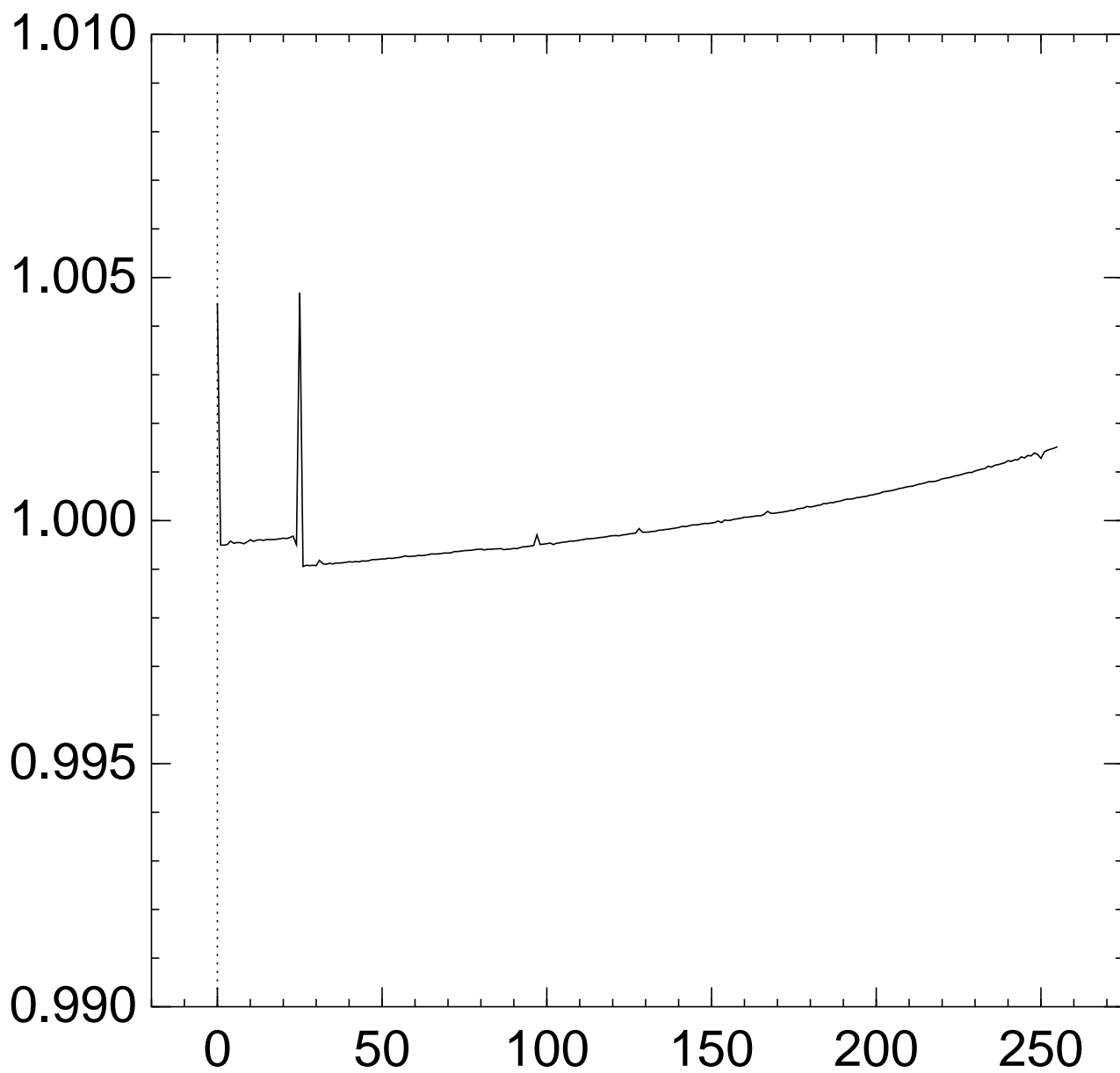
Graph of $256 \Pr[z_{23} = x]$:



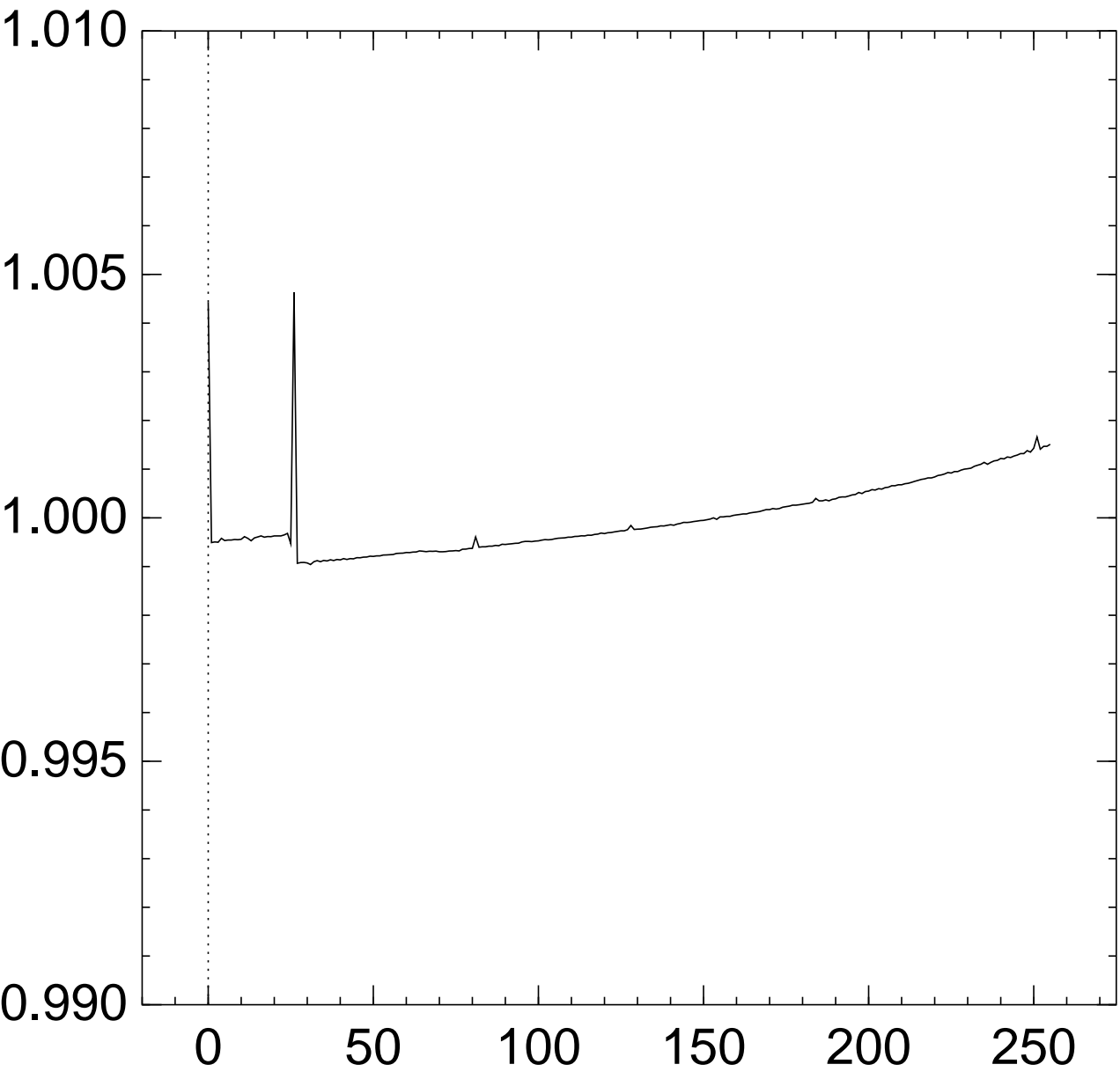
Graph of $256 \Pr[z_{24} = x]$:



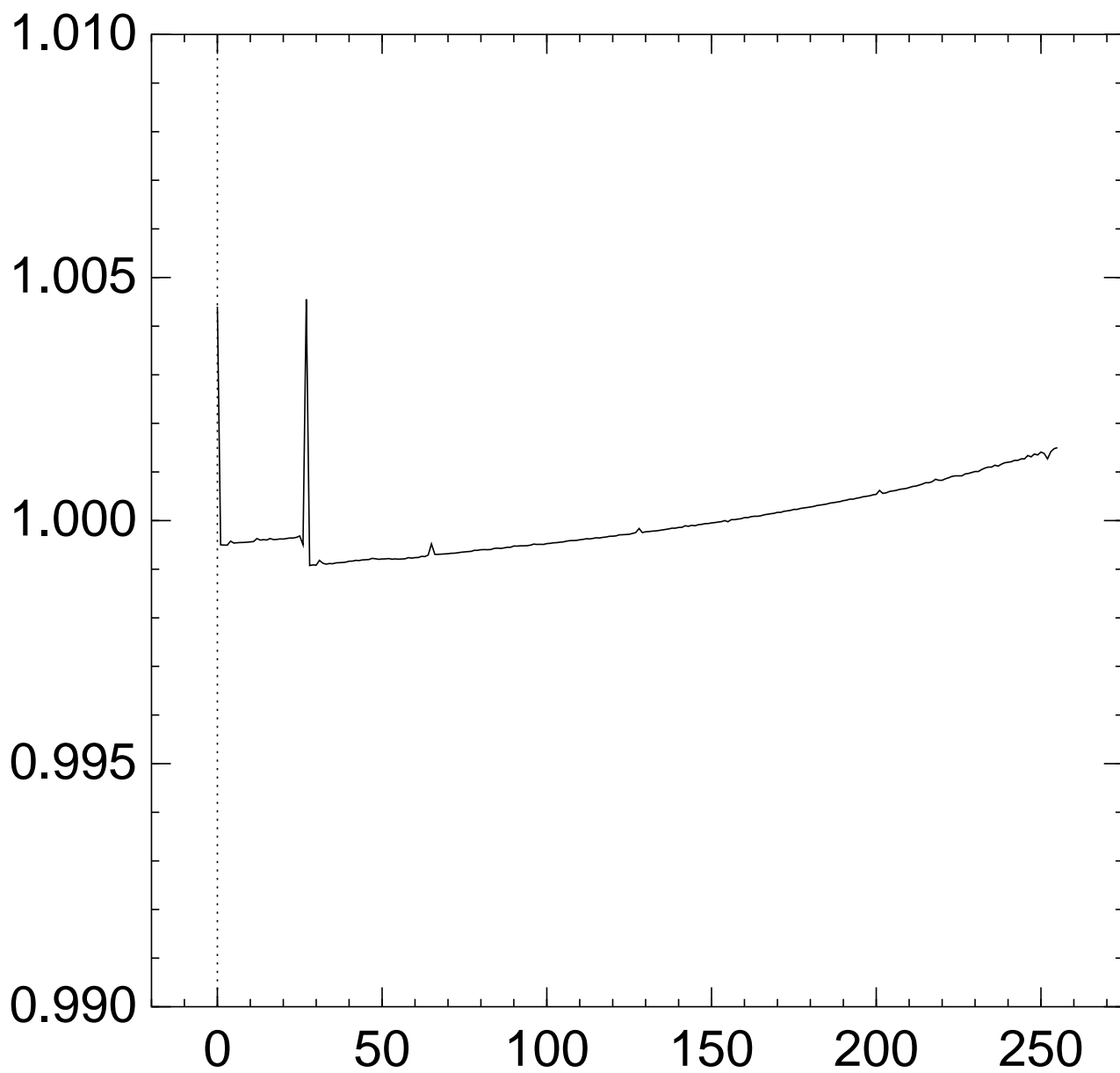
Graph of $256 \Pr[z_{25} = x]$:



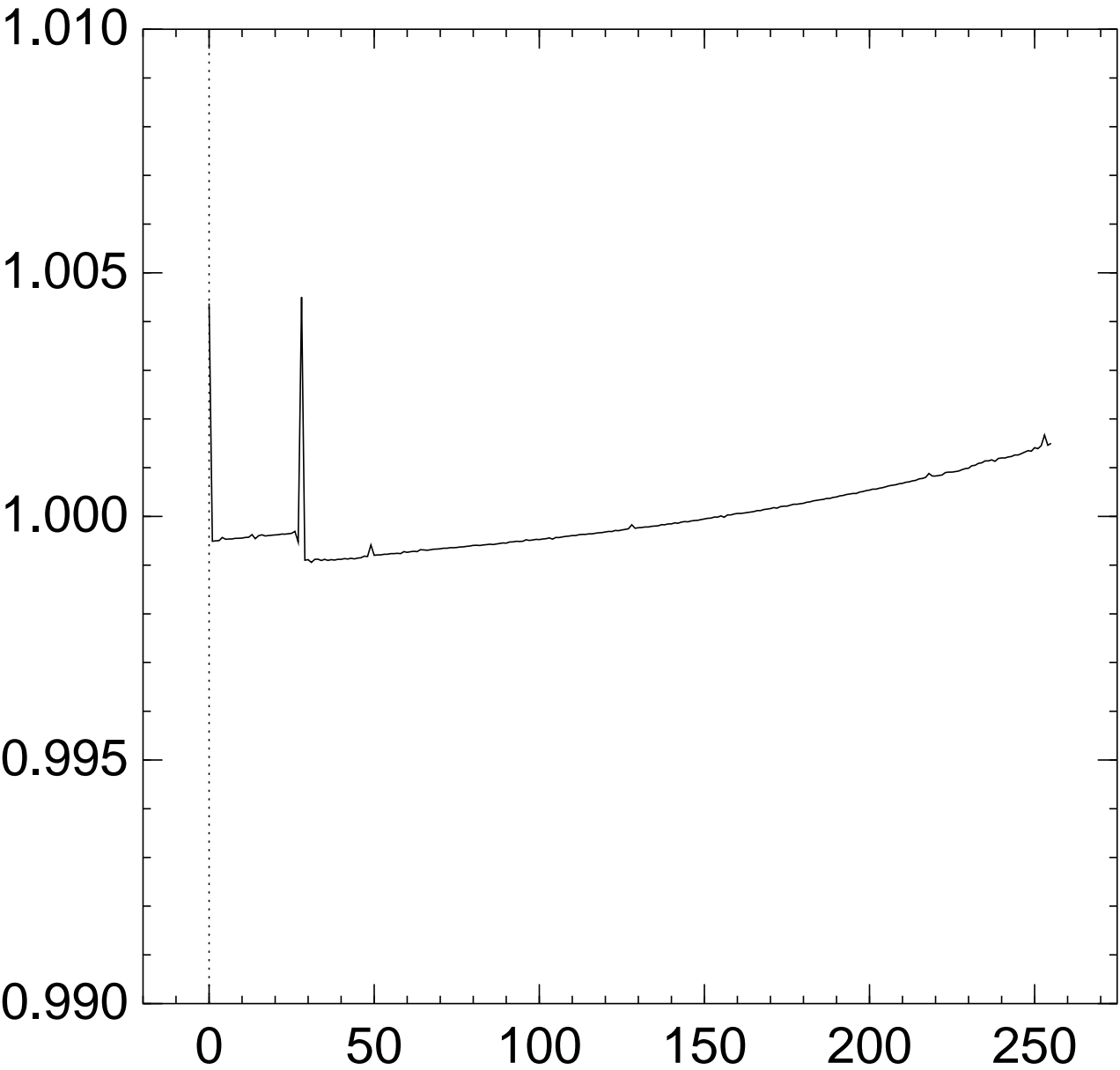
Graph of $256 \Pr[z_{26} = x]$:



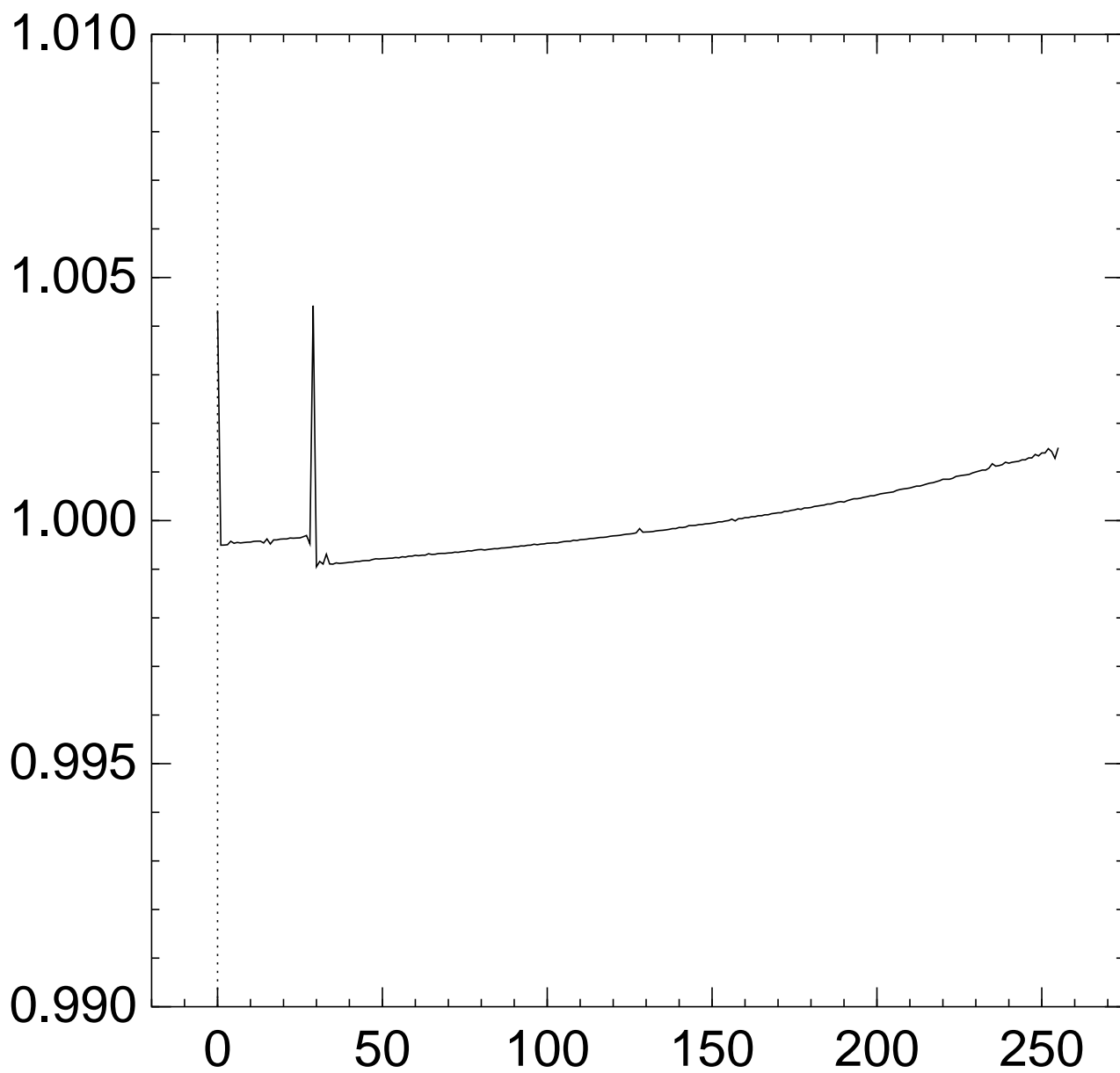
Graph of $256 \Pr[z_{27} = x]$:



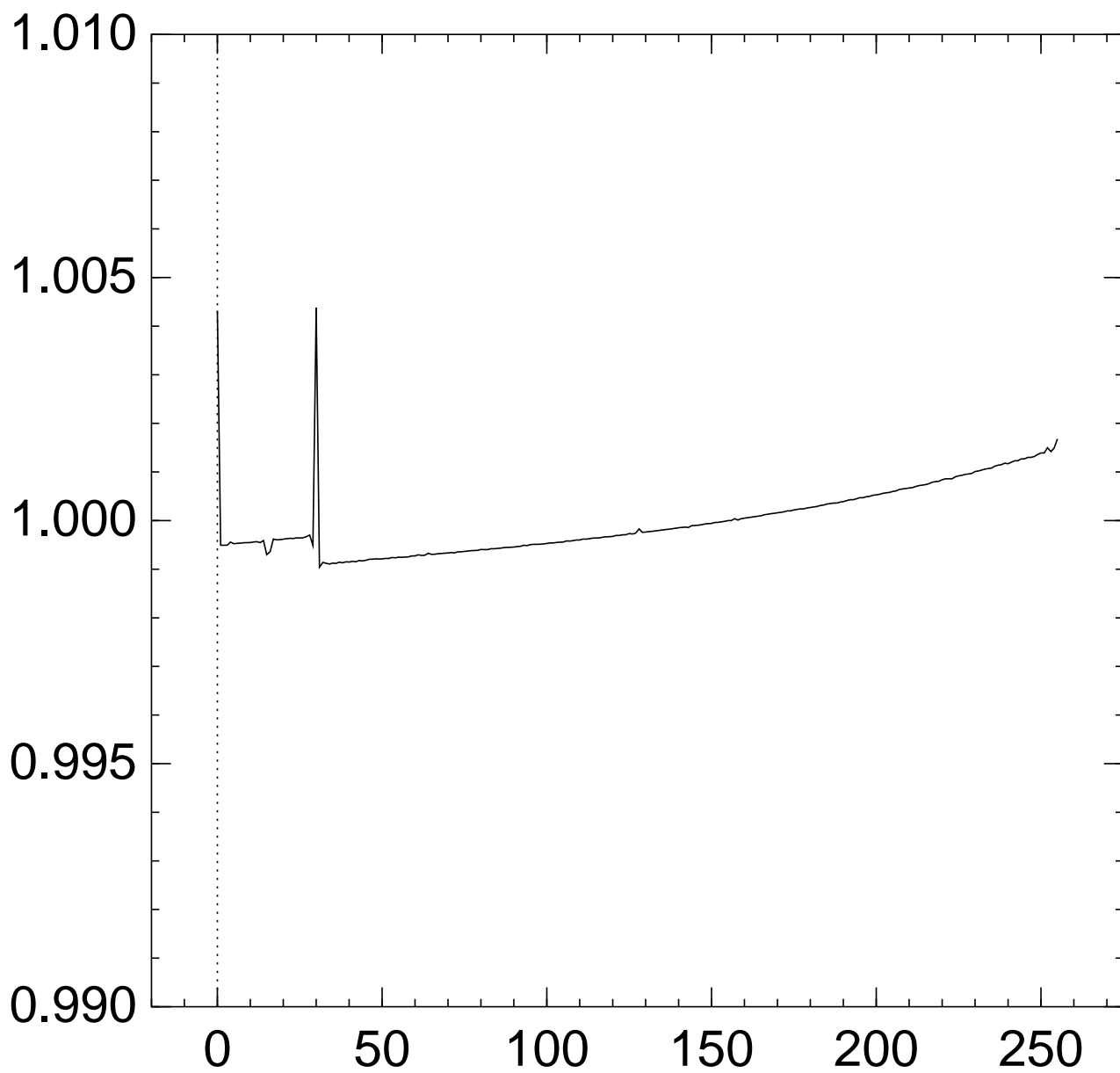
Graph of $256 \Pr[z_{28} = x]$:



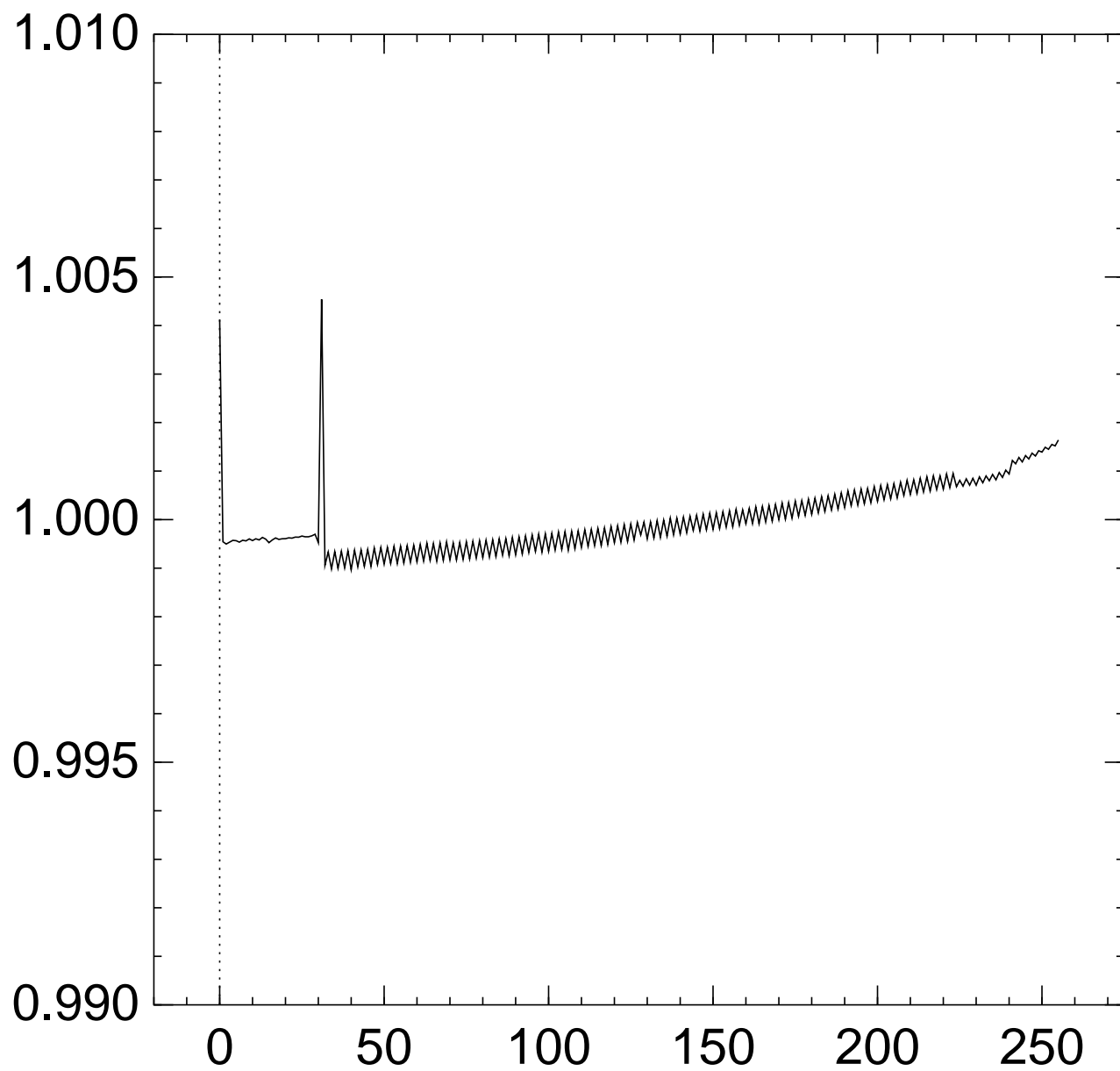
Graph of $256 \Pr[z_{29} = x]$:



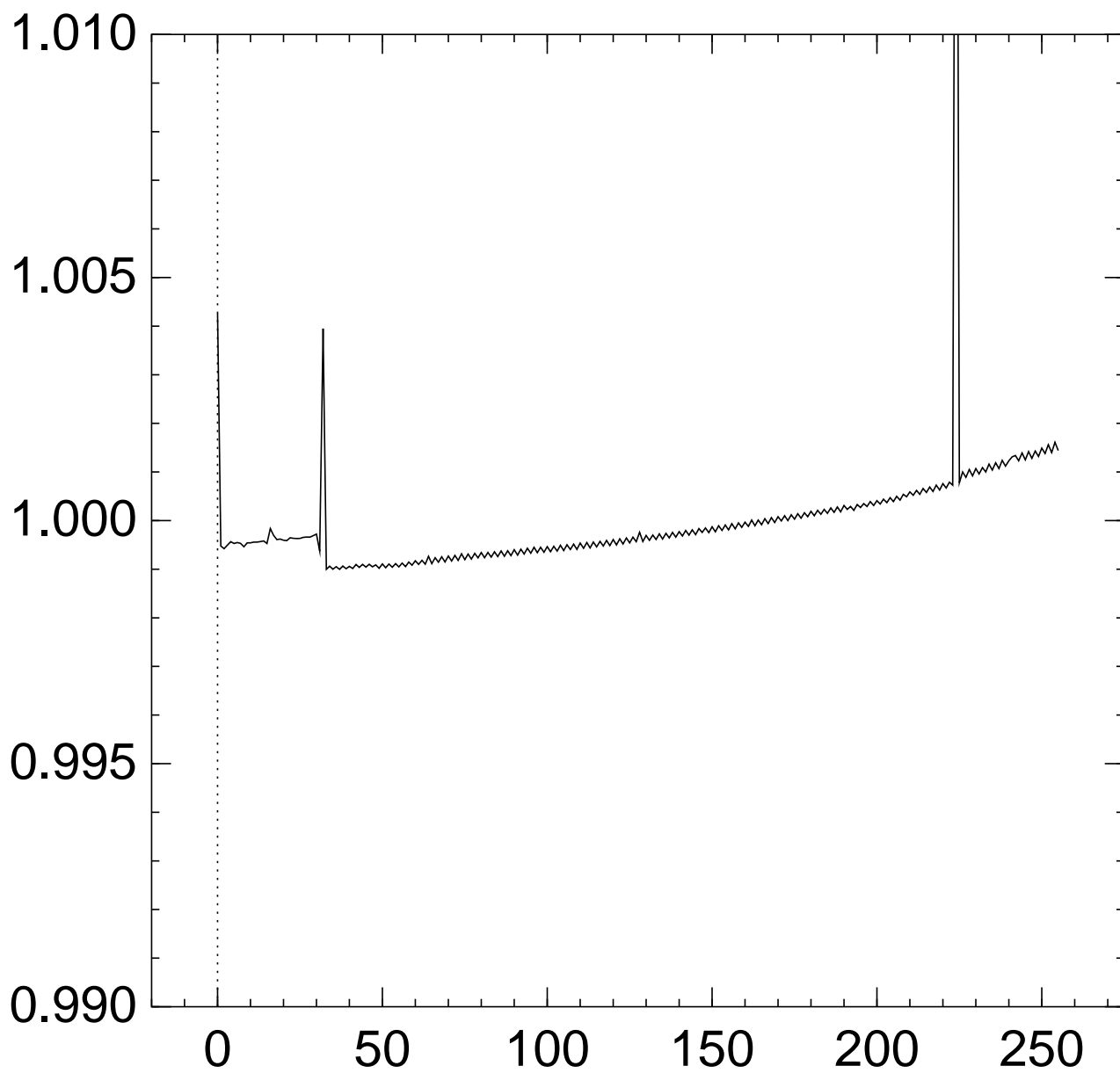
Graph of $256 \Pr[z_{30} = x]$:



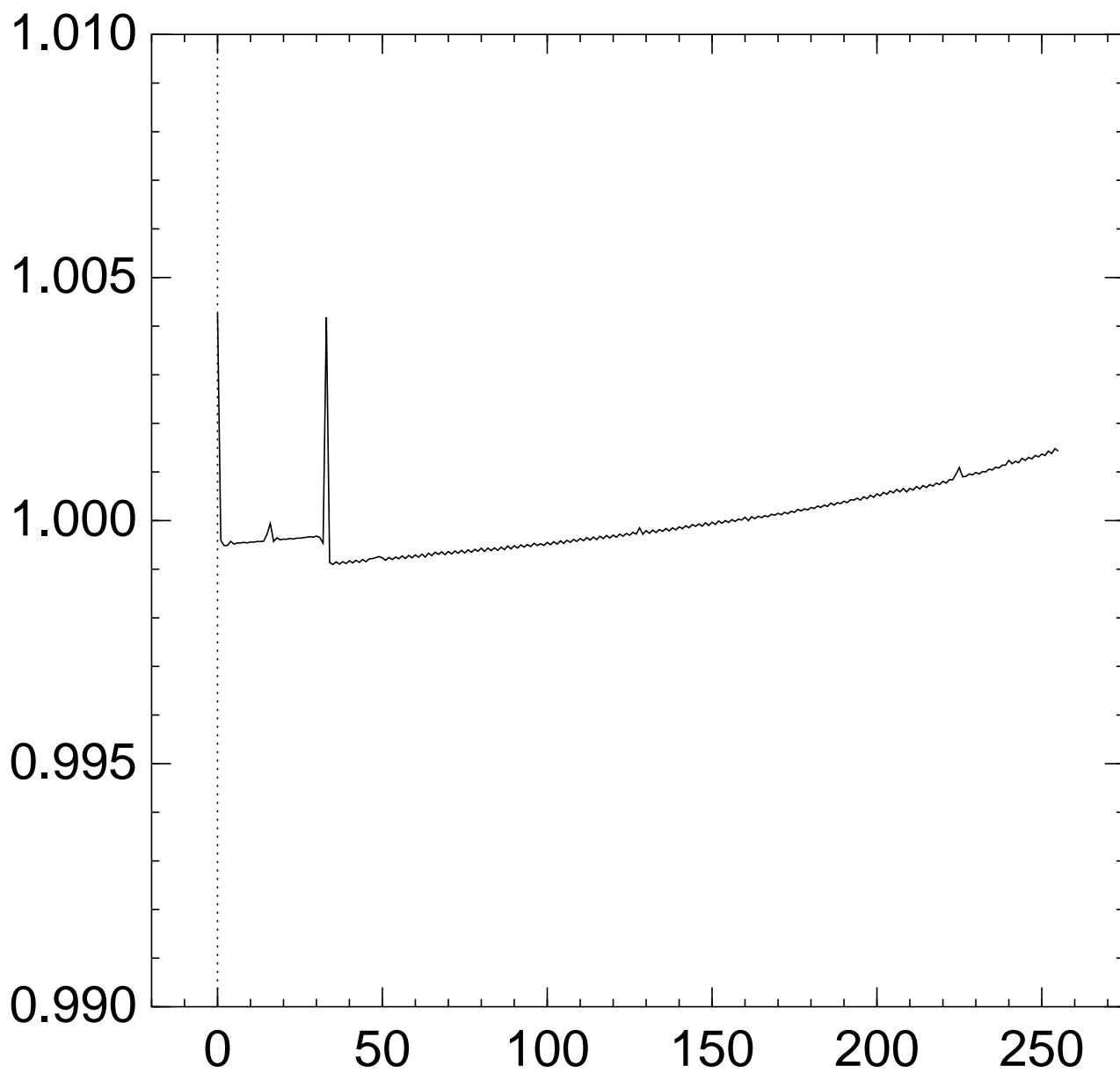
Graph of $256 \Pr[z_{31} = x]$:



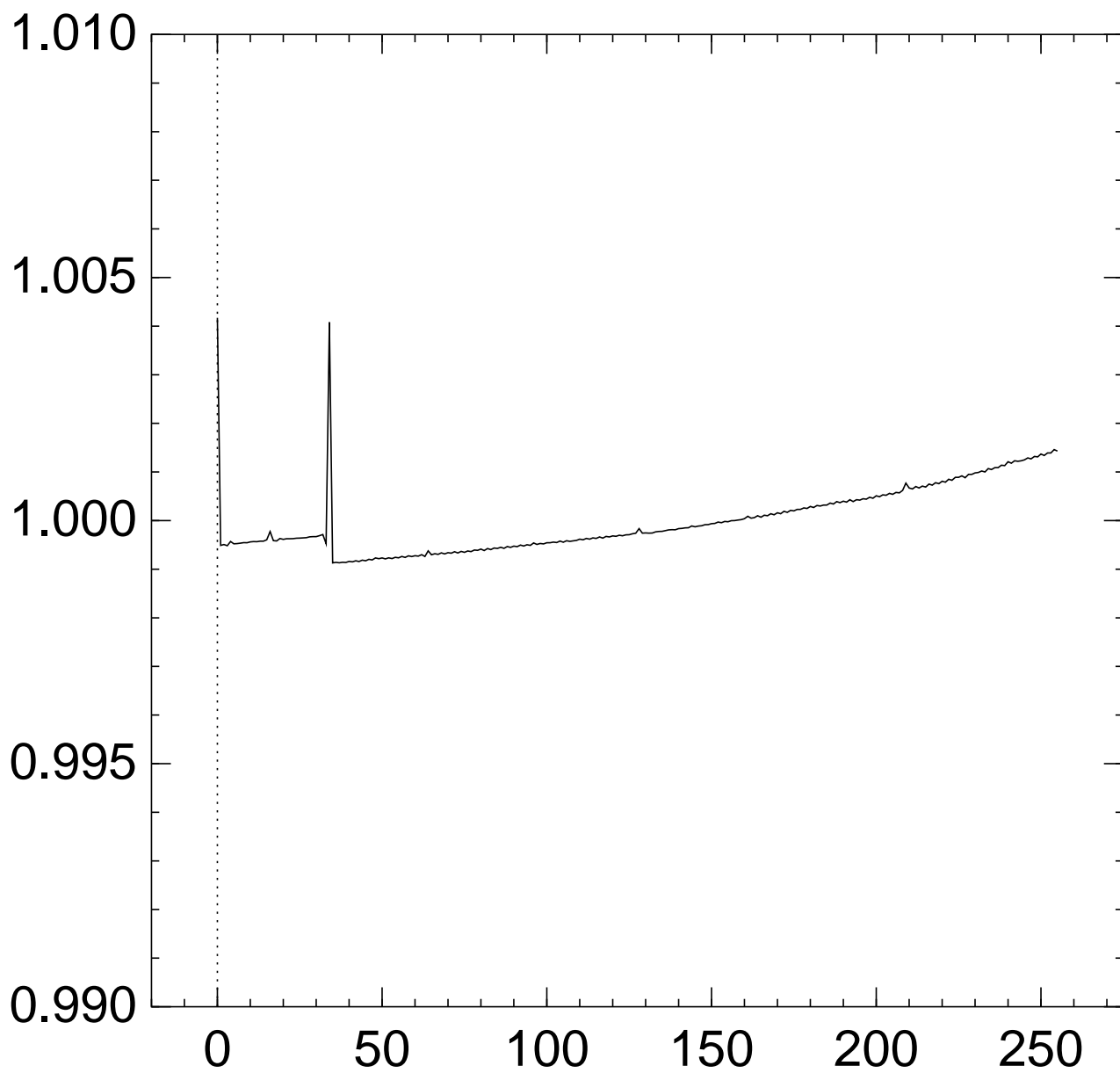
Graph of $256 \Pr[z_{32} = x]$:



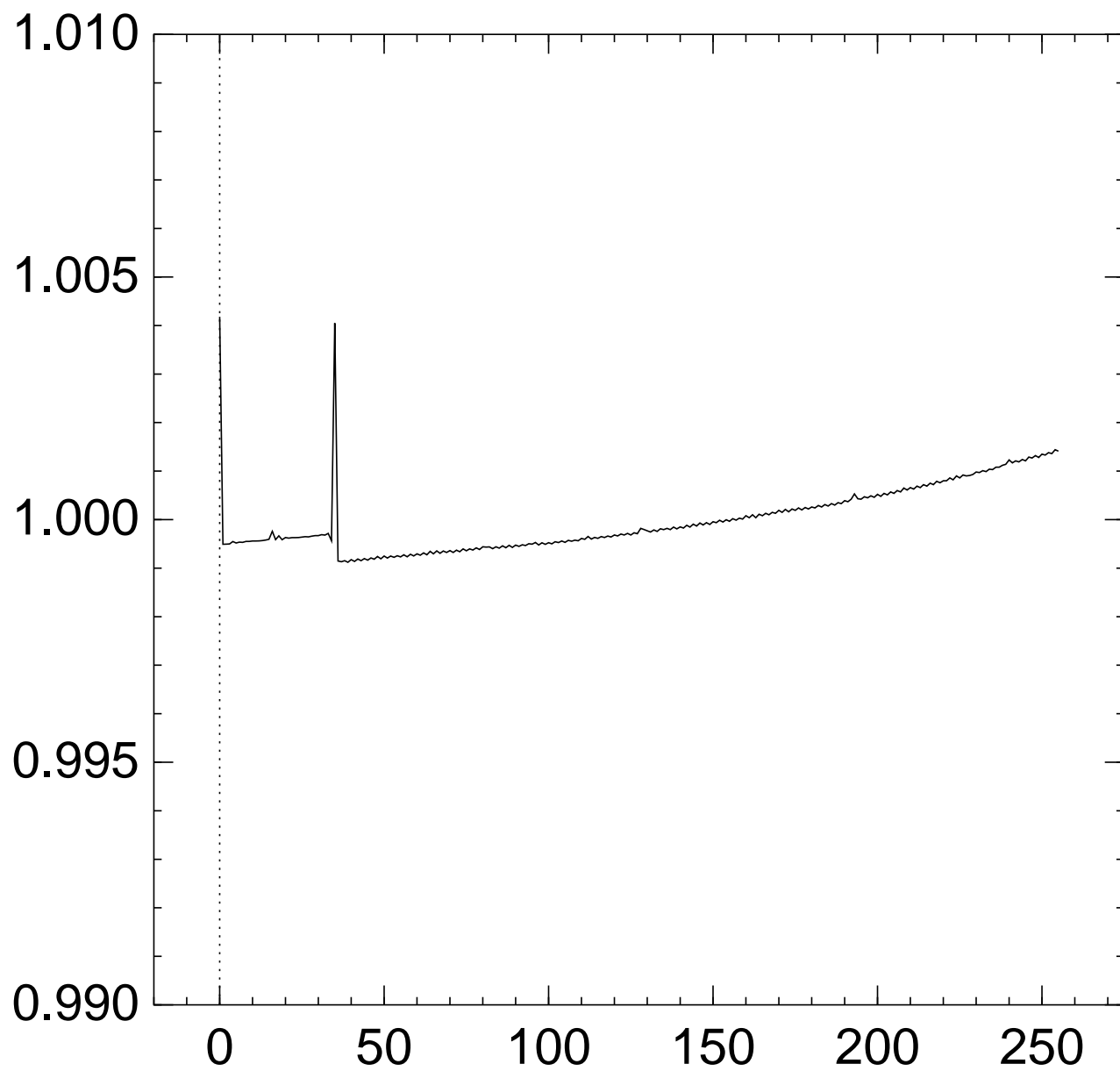
Graph of $256 \Pr[z_{33} = x]$:



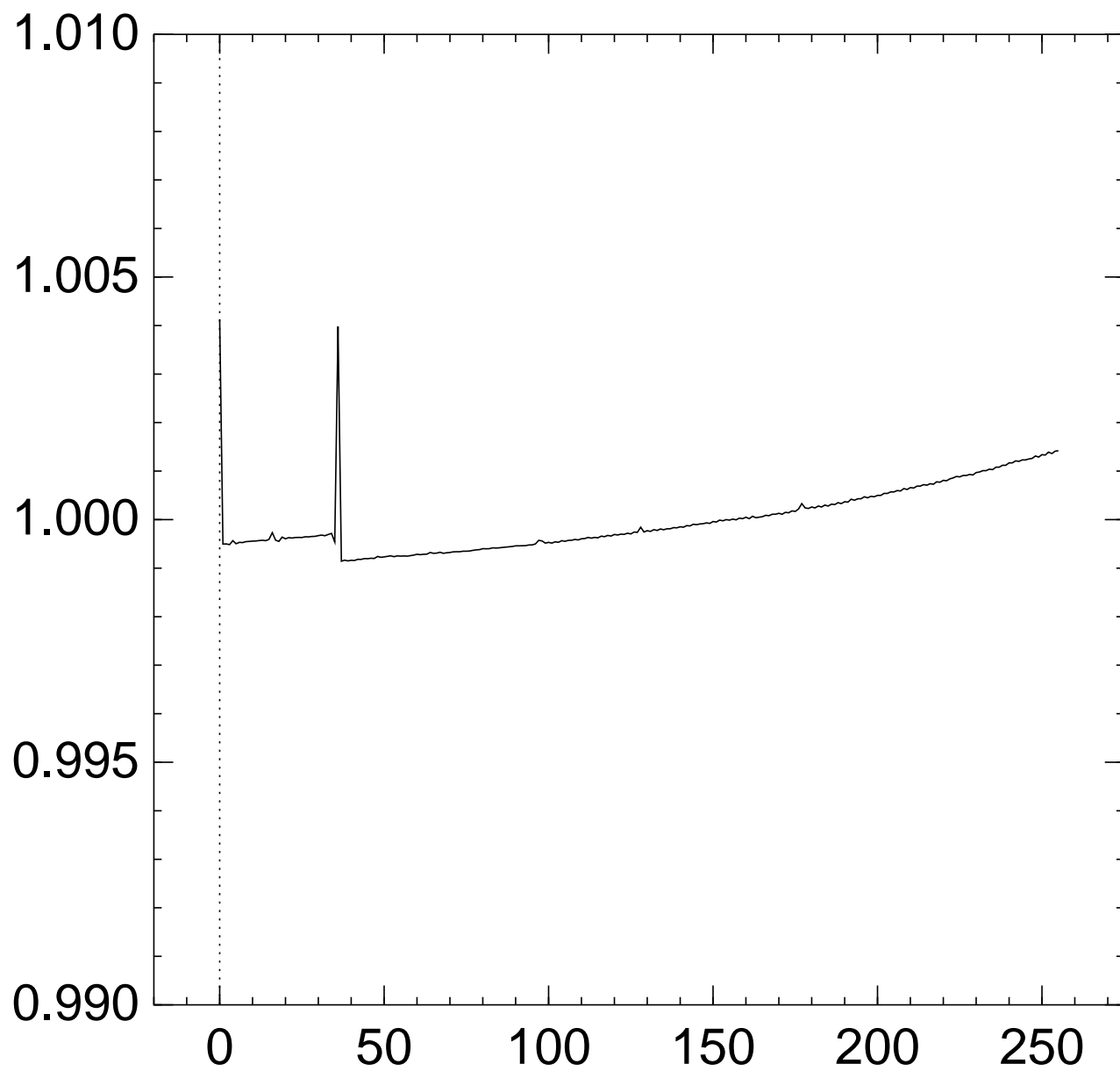
Graph of $256 \Pr[z_{34} = x]$:



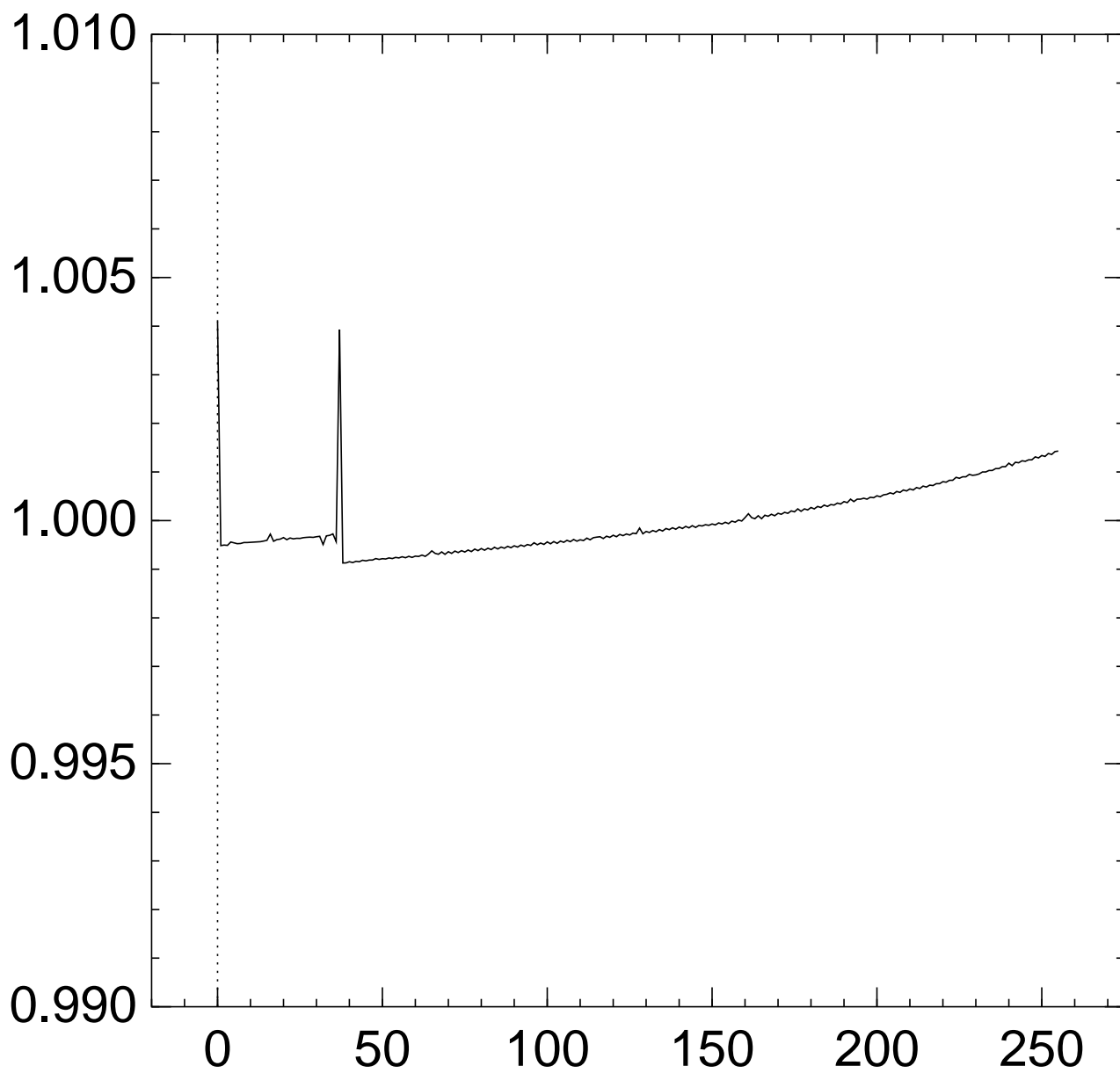
Graph of $256 \Pr[z_{35} = x]$:



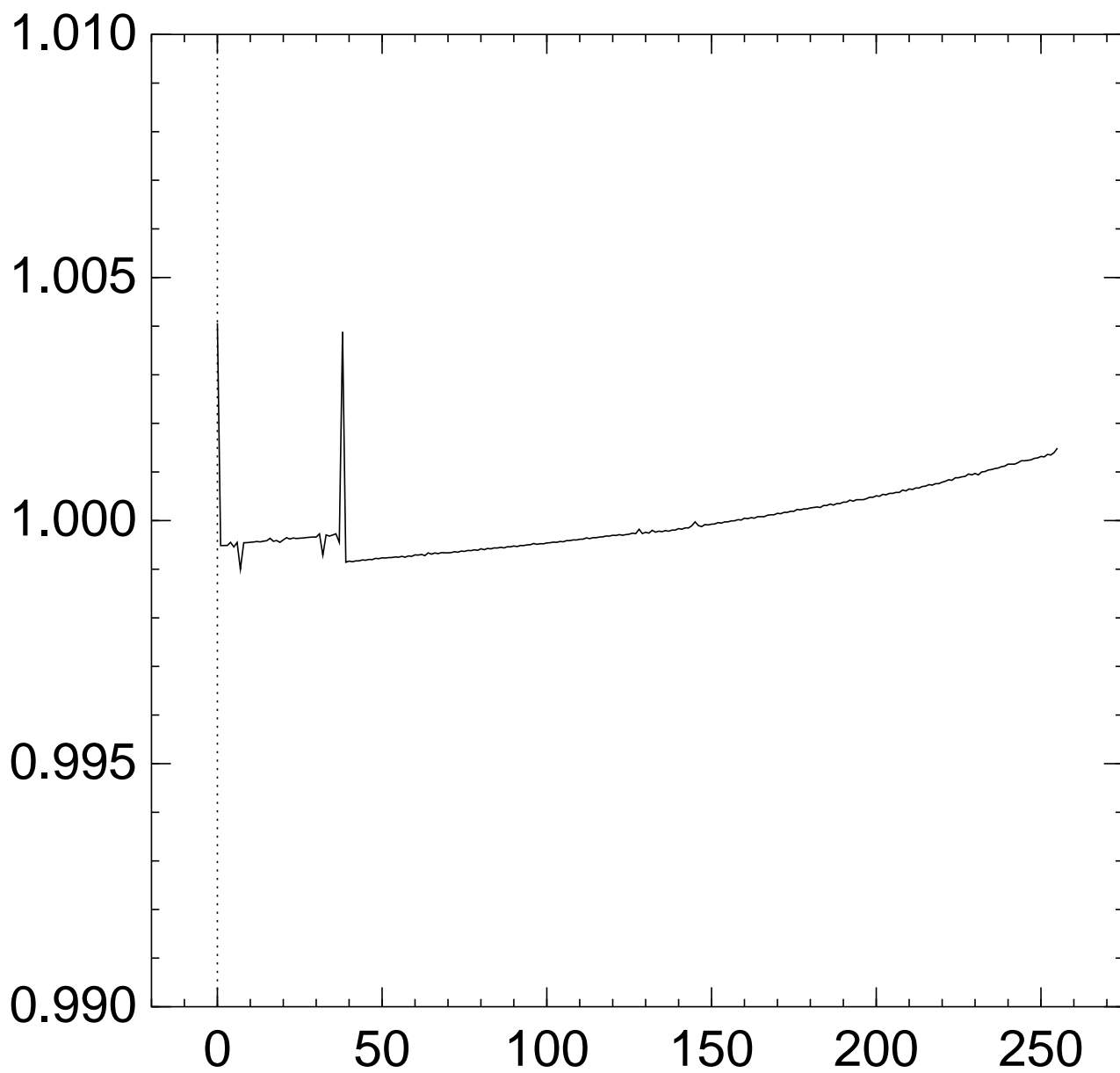
Graph of $256 \Pr[z_{36} = x]$:



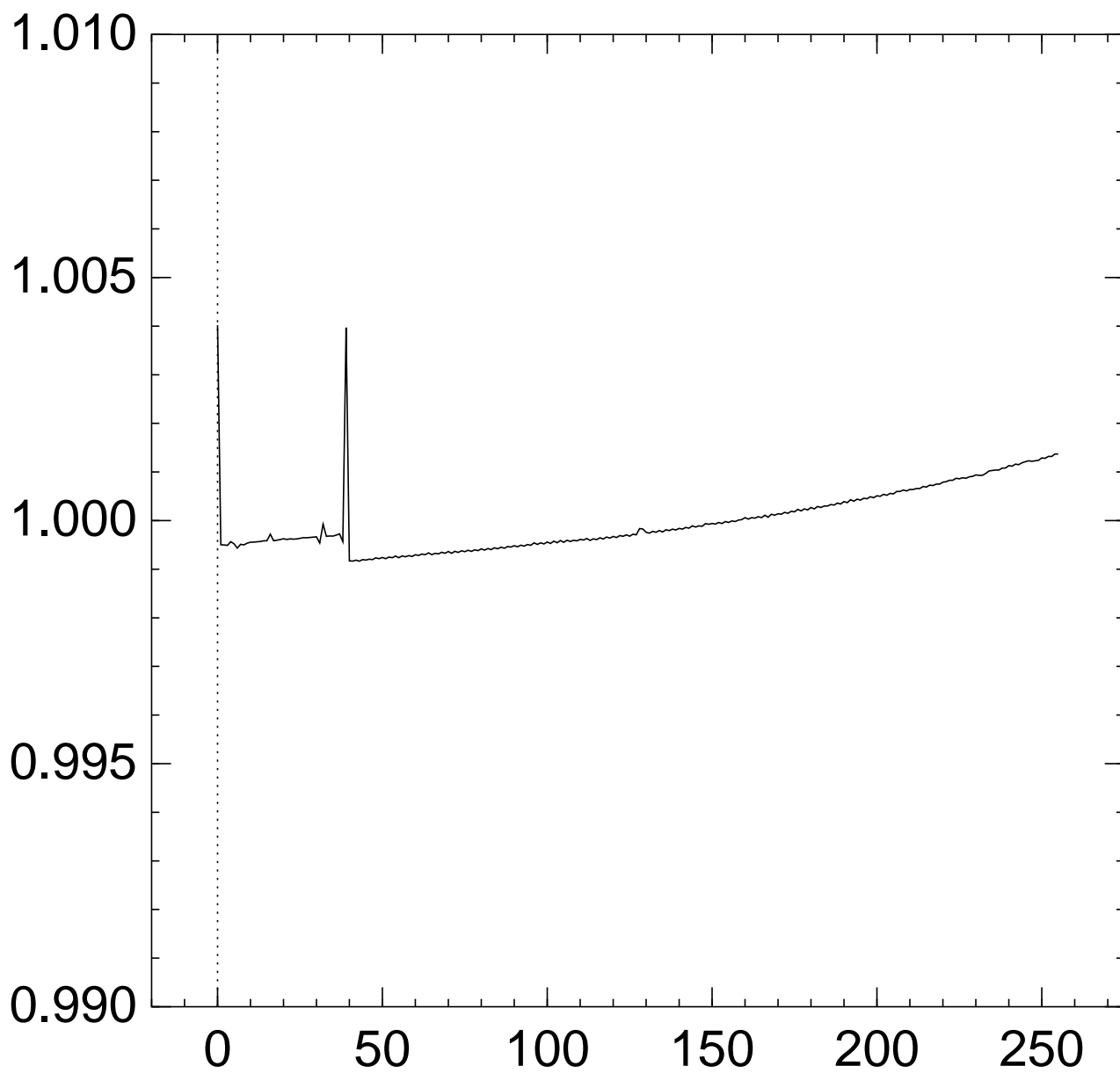
Graph of $256 \Pr[z_{37} = x]$:



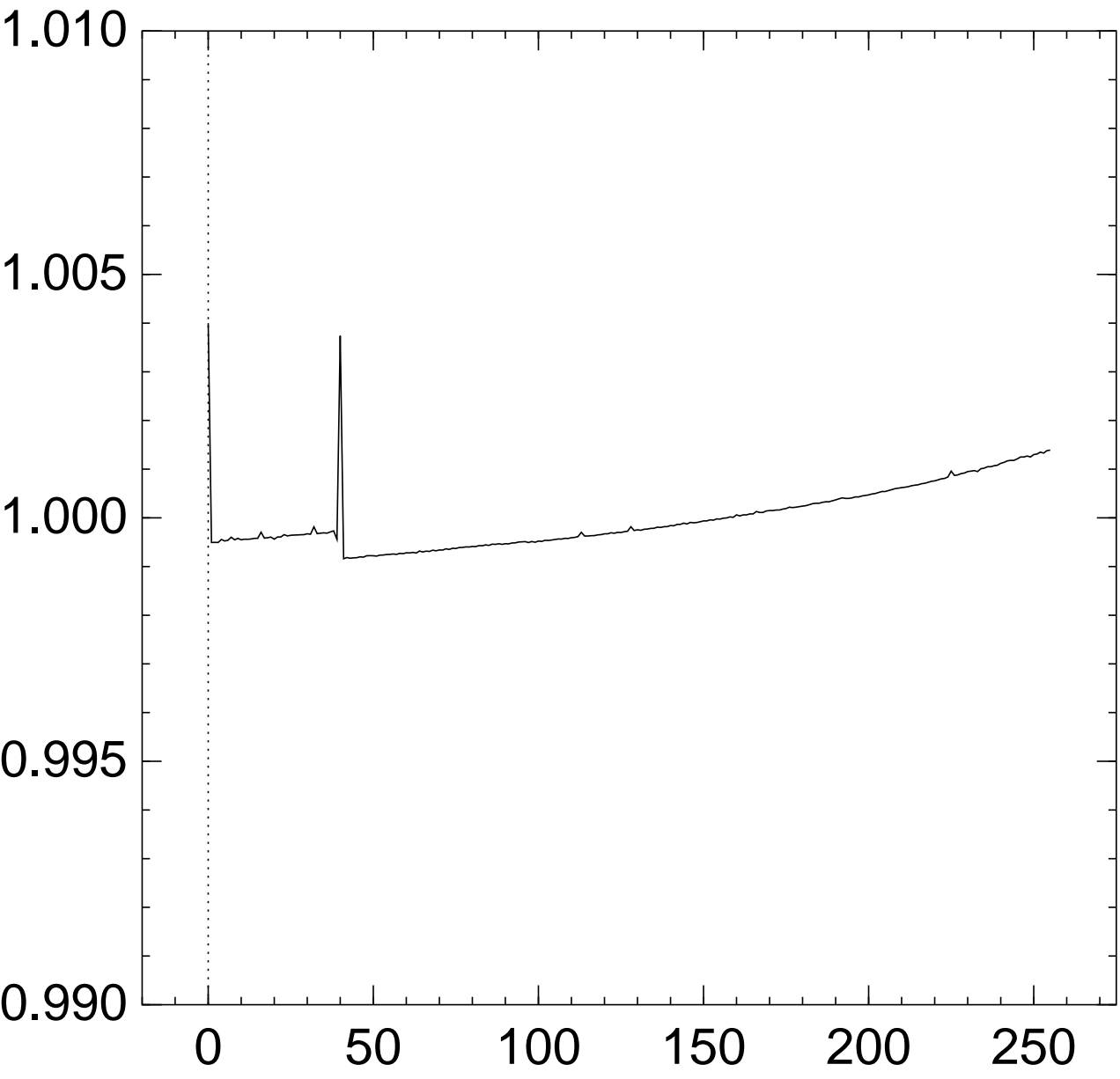
Graph of $256 \Pr[z_{38} = x]$:



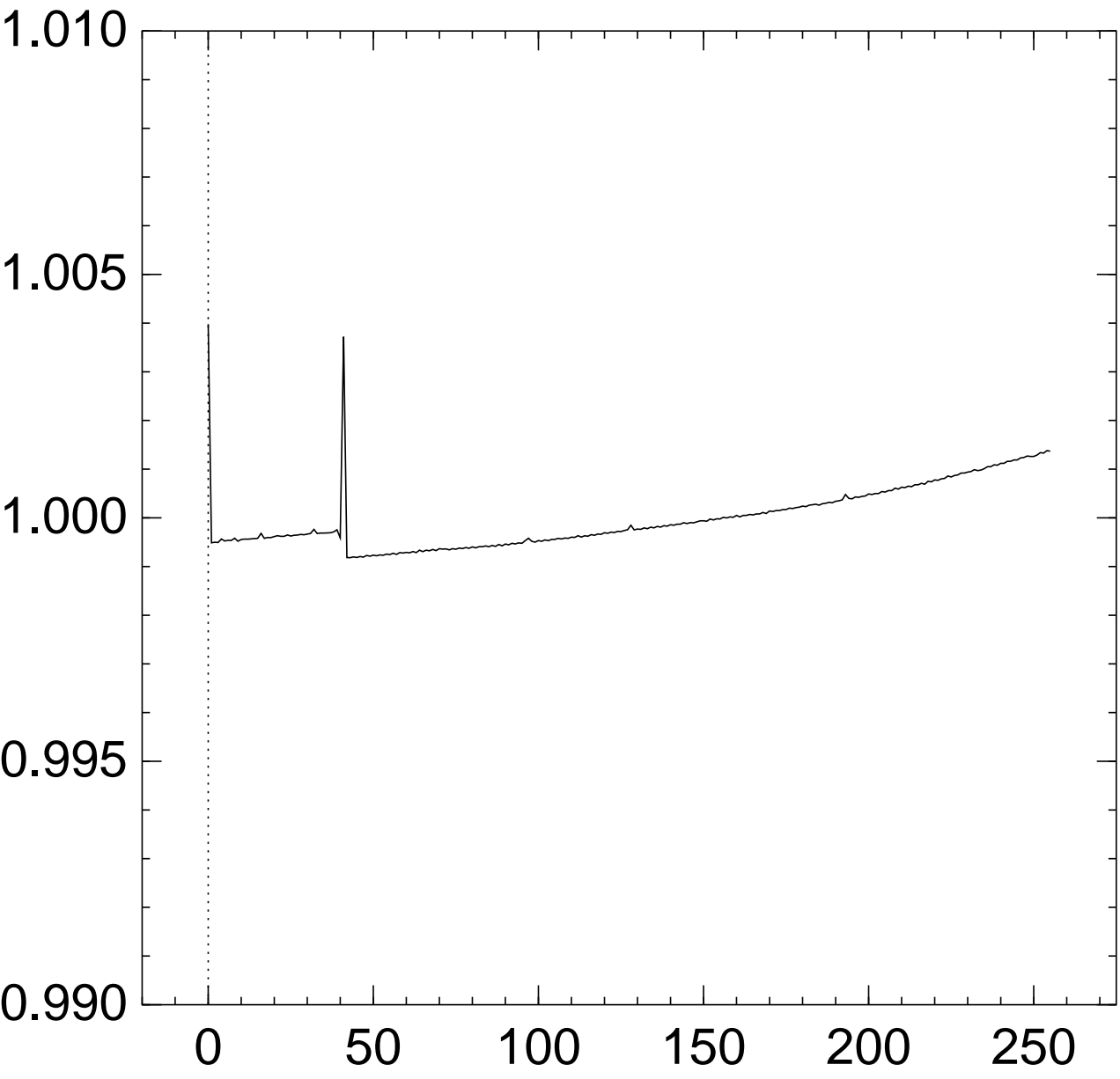
Graph of $256 \Pr[z_{39} = x]$:



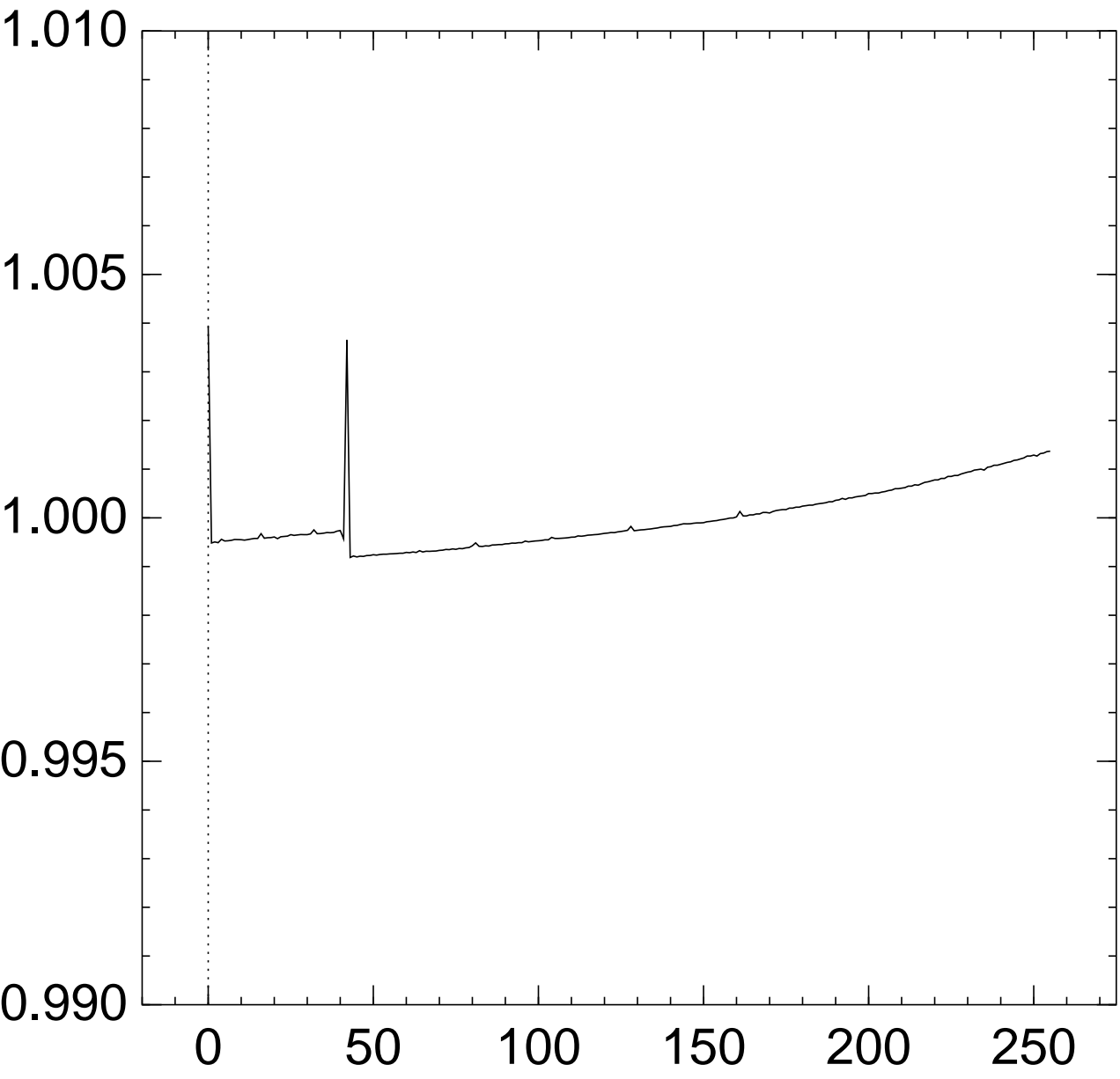
Graph of $256 \Pr[z_{40} = x]$:



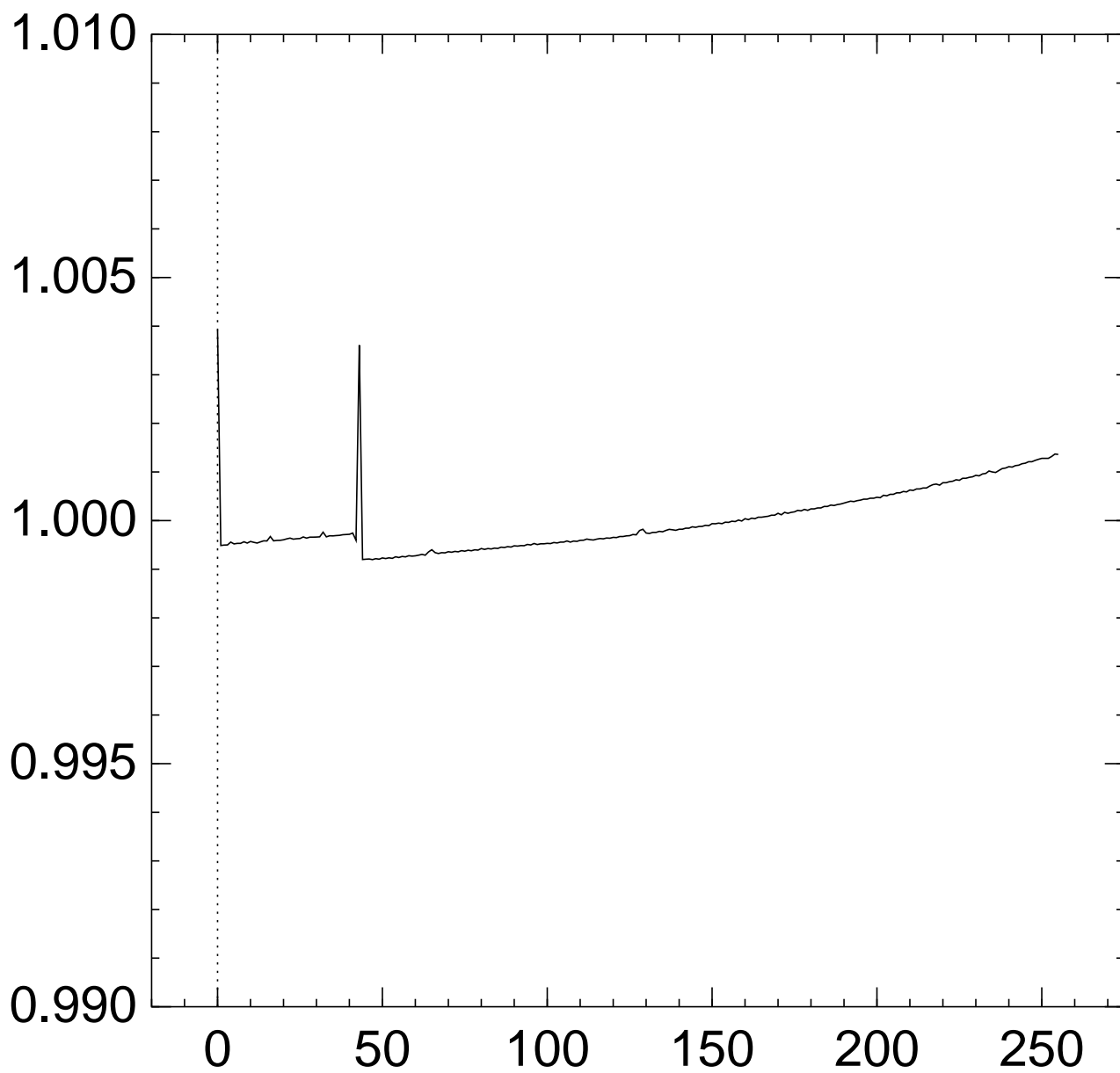
Graph of $256 \Pr[z_{41} = x]$:



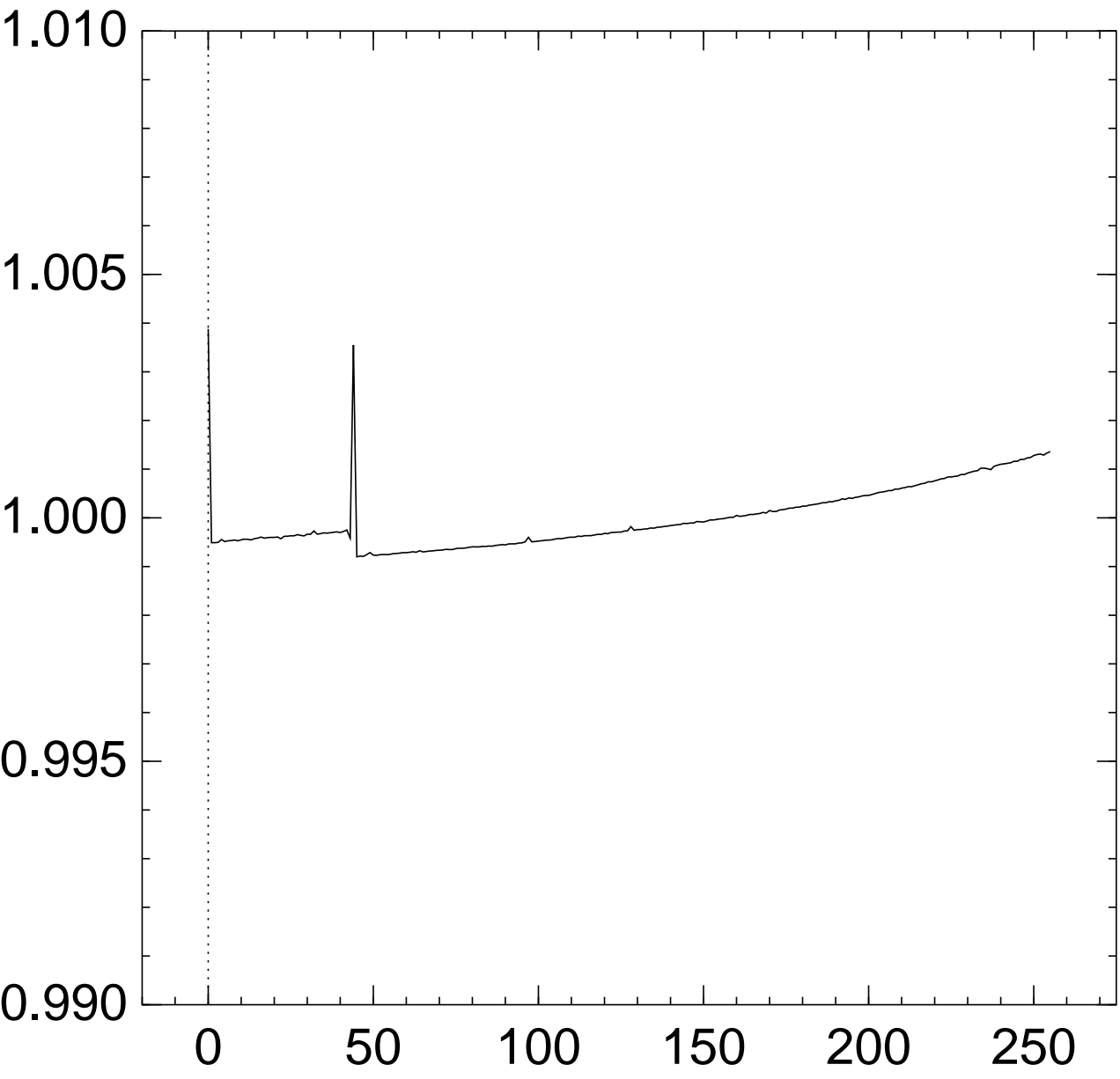
Graph of $256 \Pr[z_{42} = x]$:



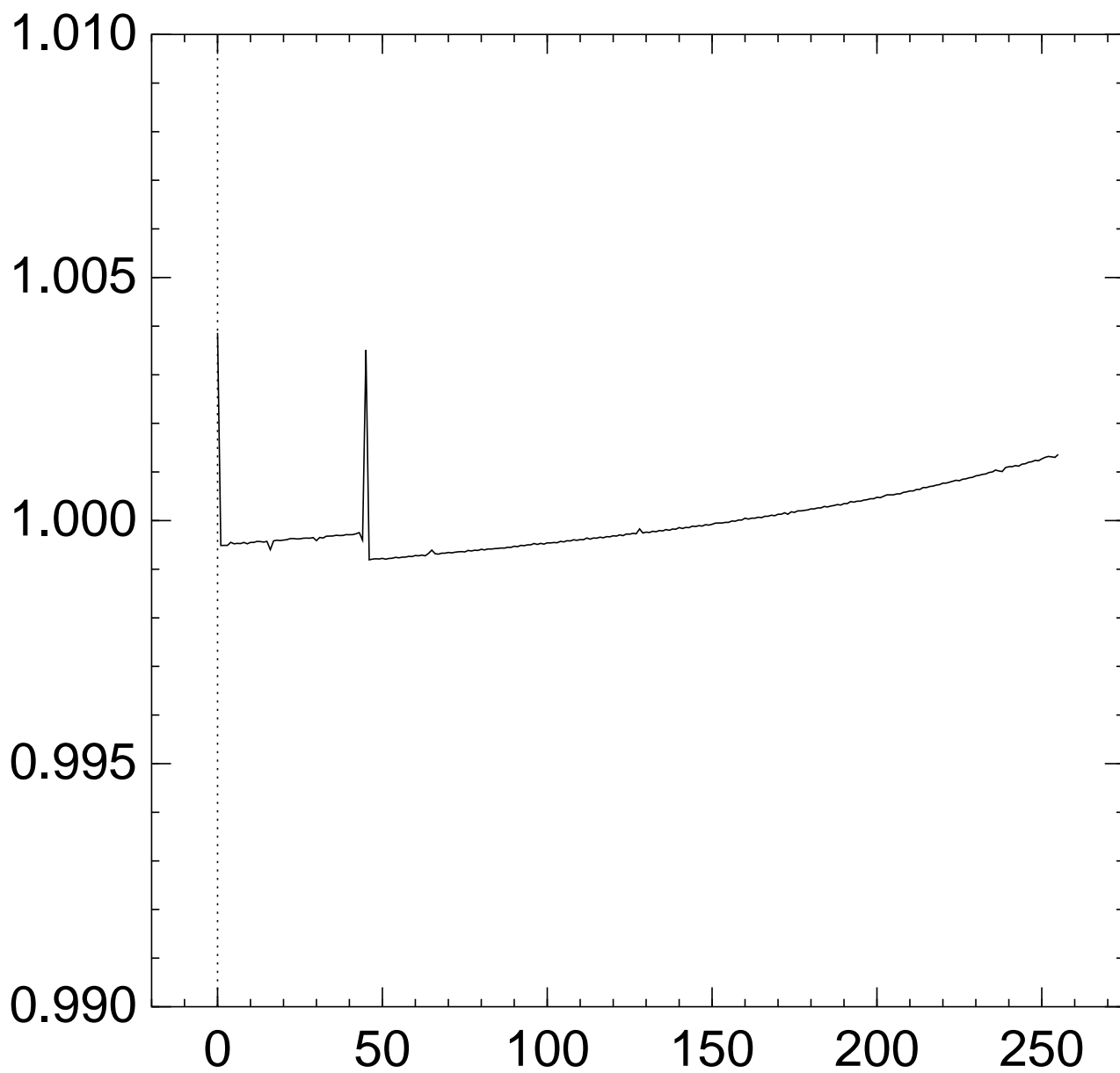
Graph of $256 \Pr[z_{43} = x]$:



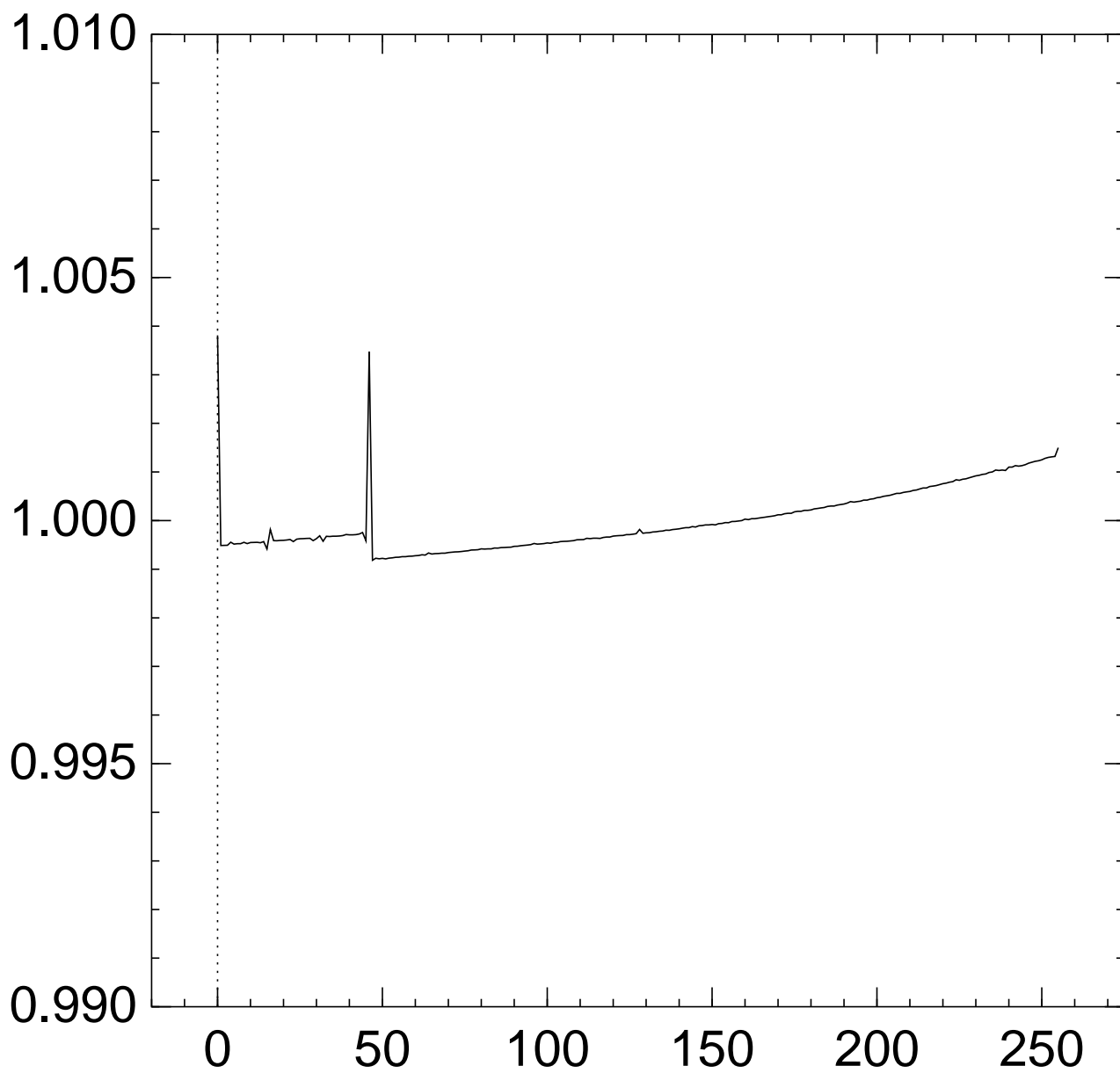
Graph of $256 \Pr[z_{44} = x]$:



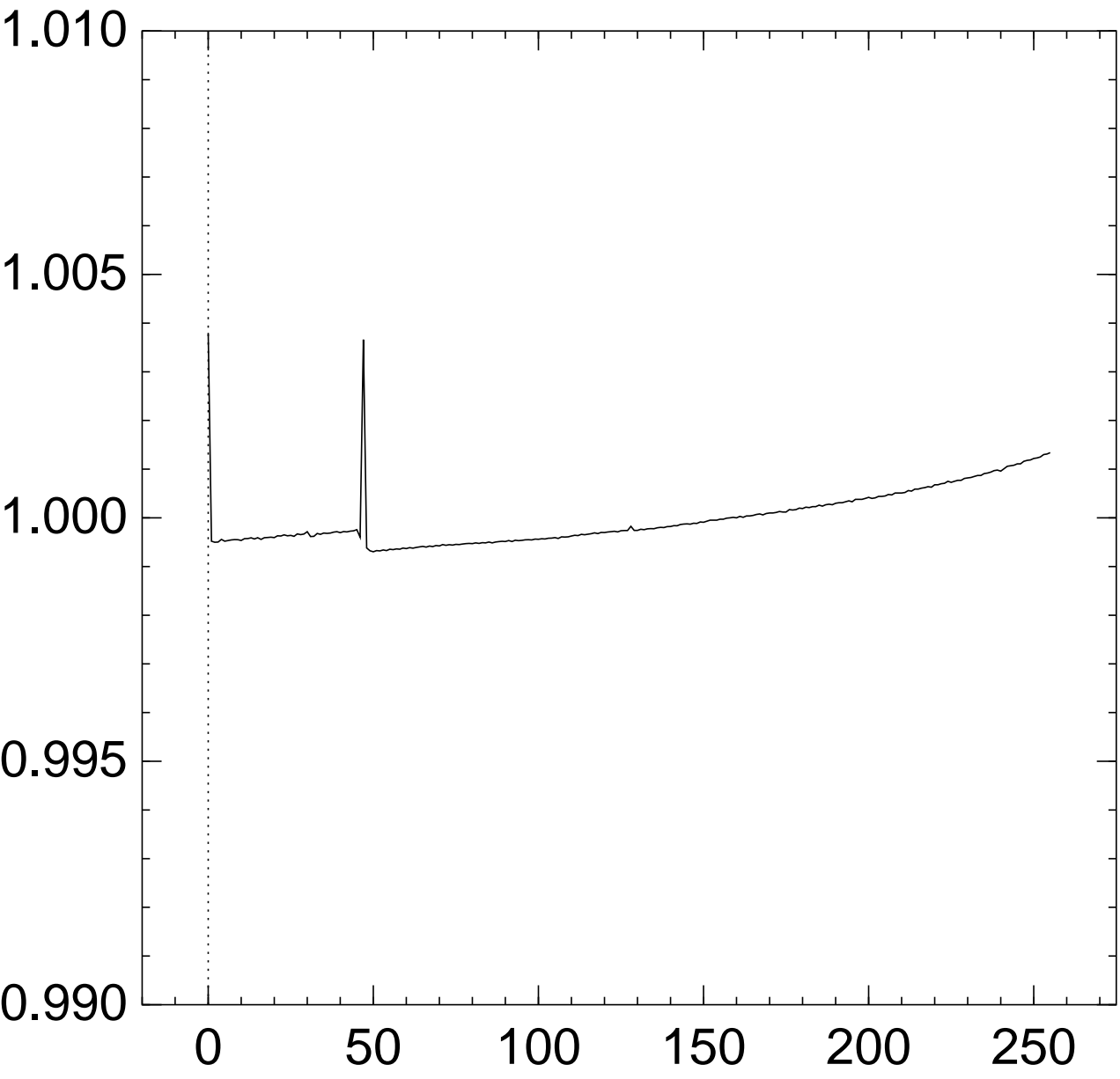
Graph of $256 \Pr[z_{45} = x]$:



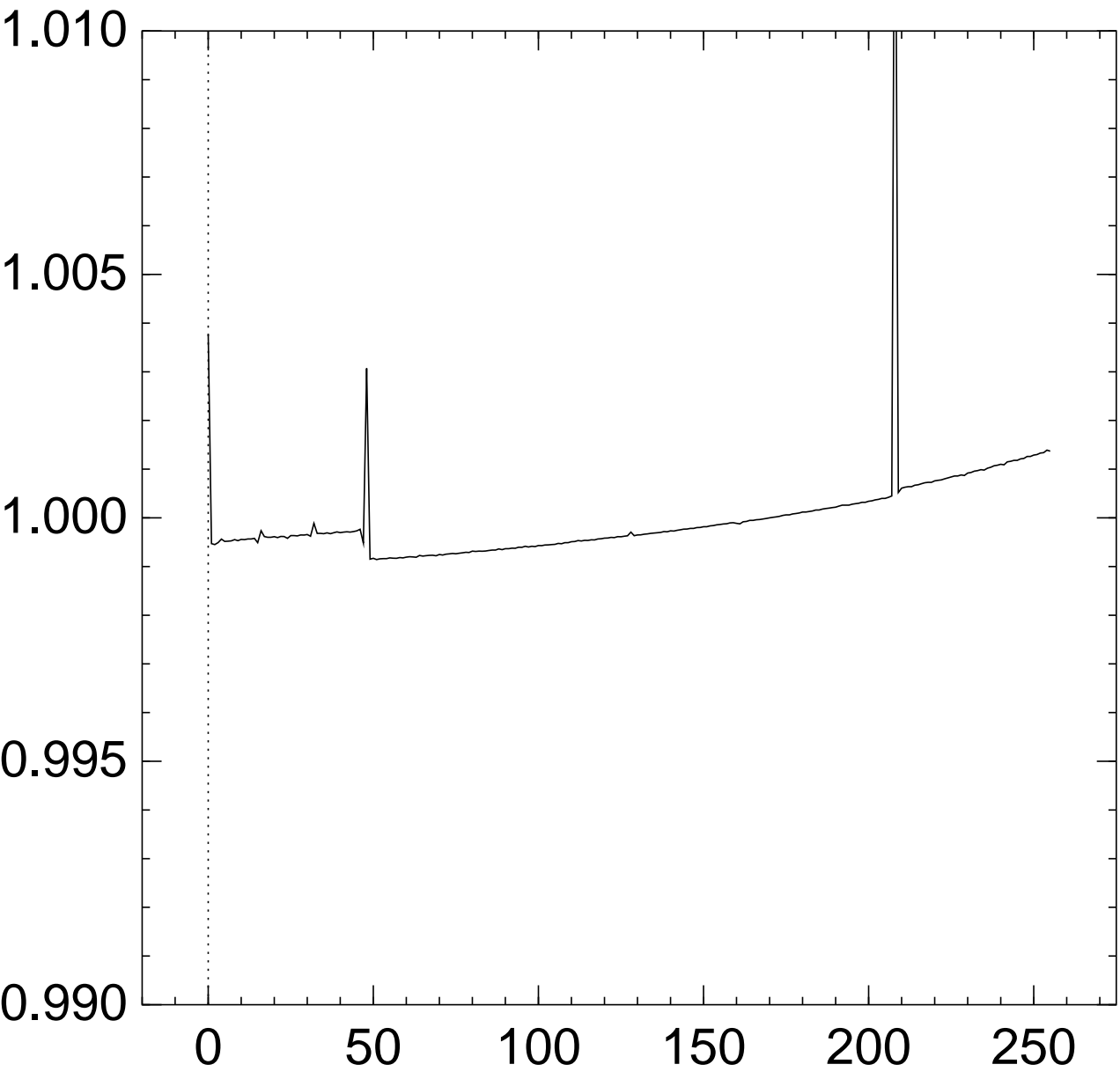
Graph of $256 \Pr[z_{46} = x]$:



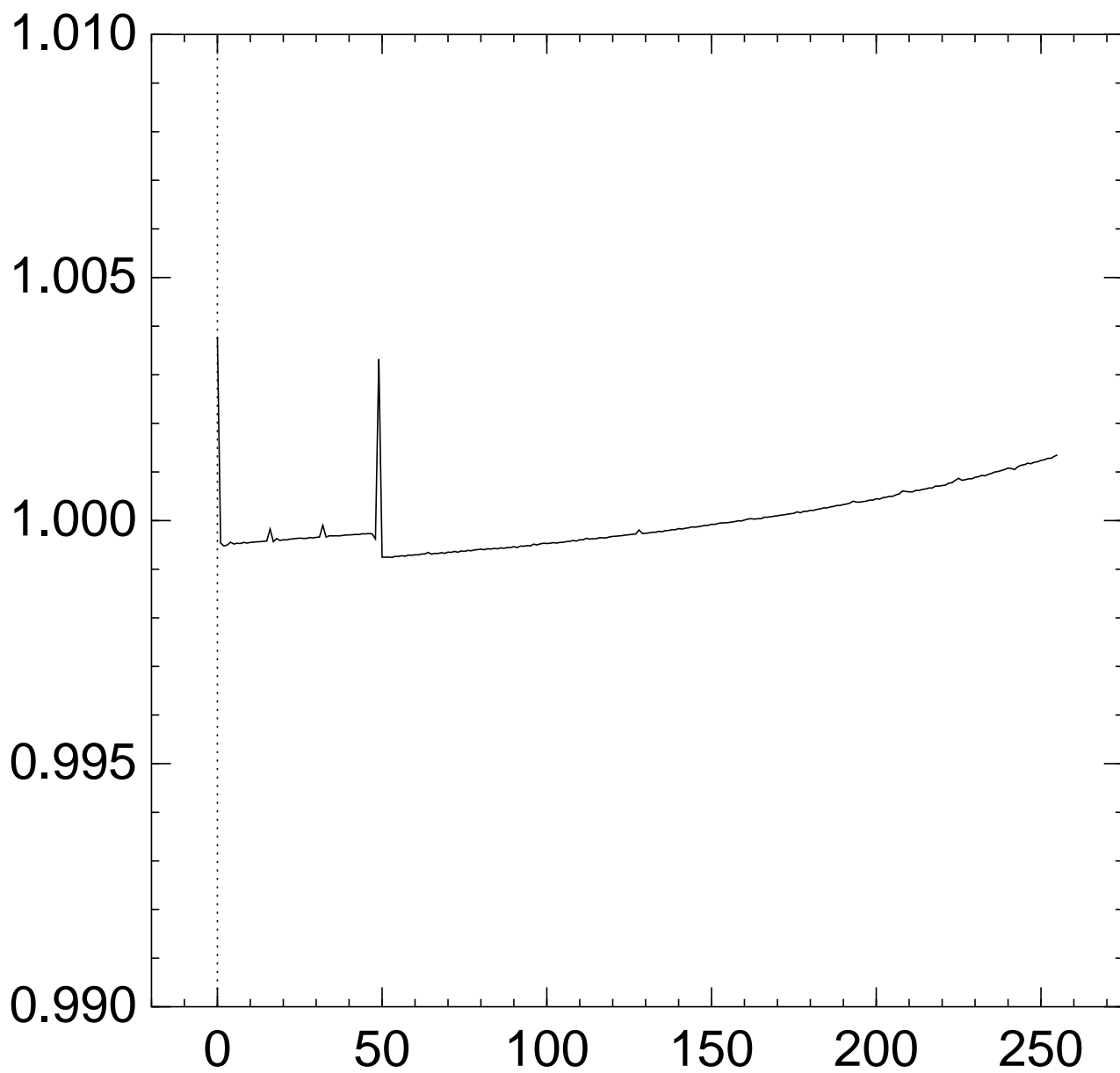
Graph of $256 \Pr[z_{47} = x]$:



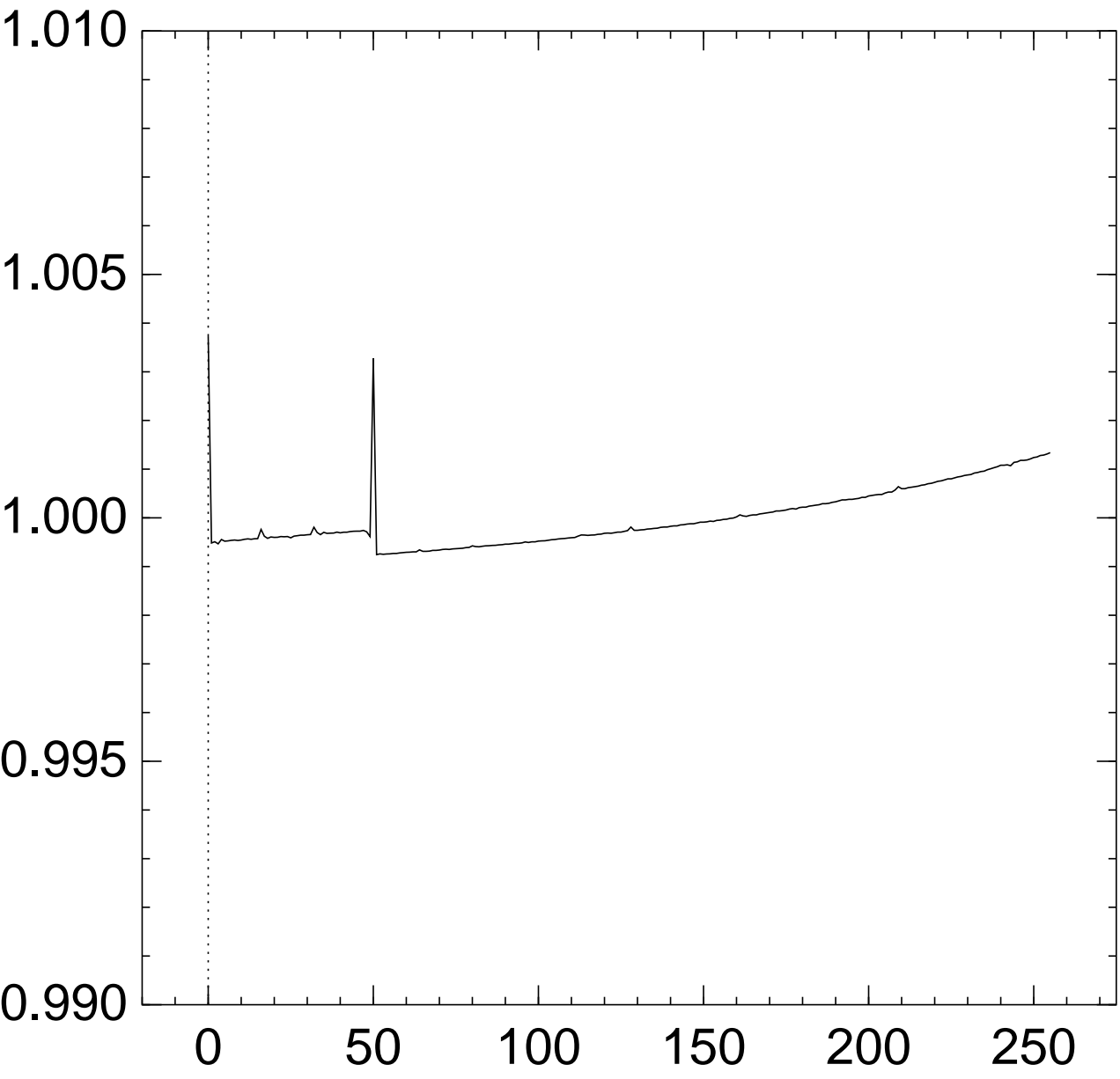
Graph of $256 \Pr[z_{48} = x]$:



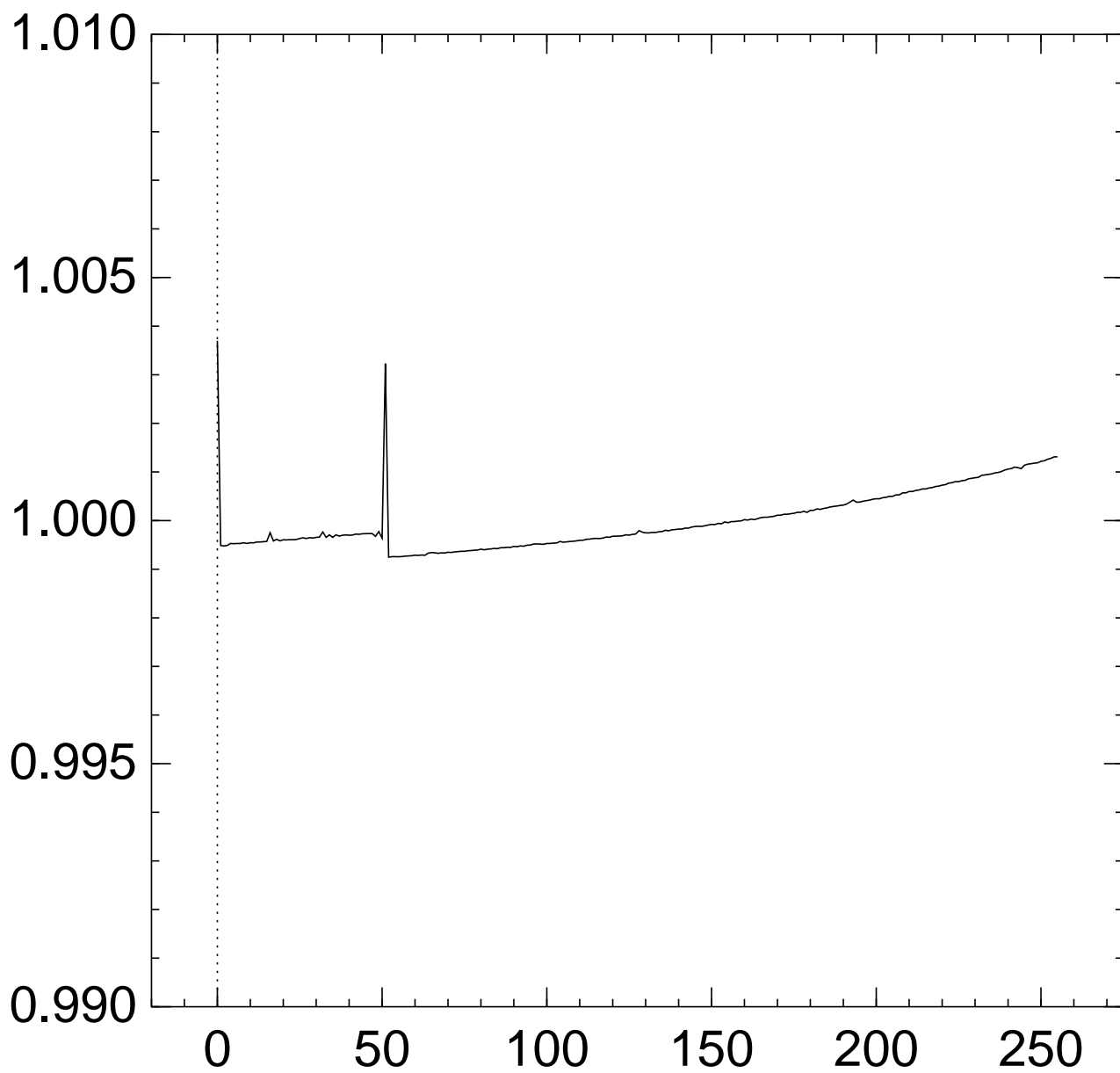
Graph of $256 \Pr[z_{49} = x]$:



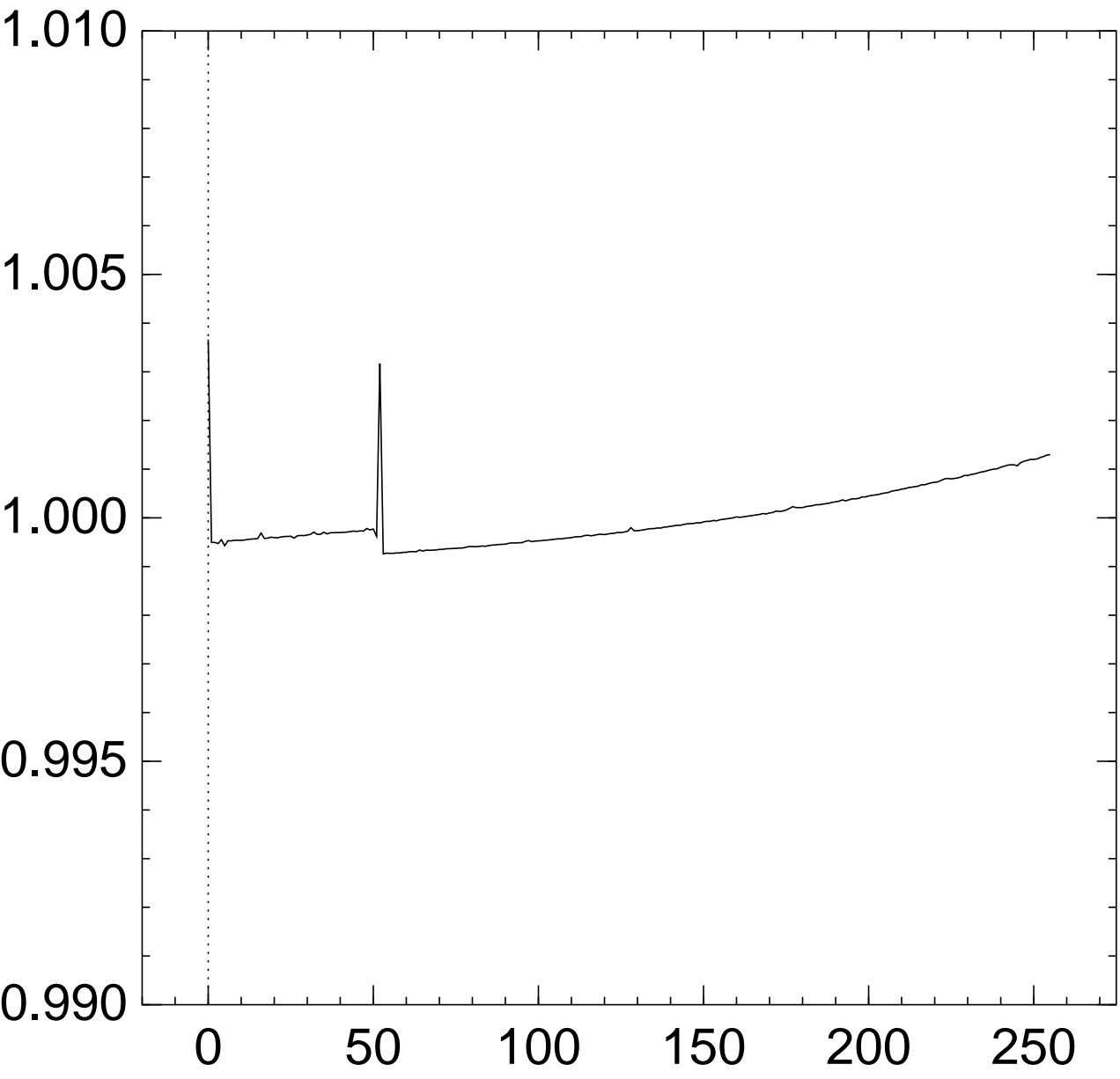
Graph of $256 \Pr[z_{50} = x]$:



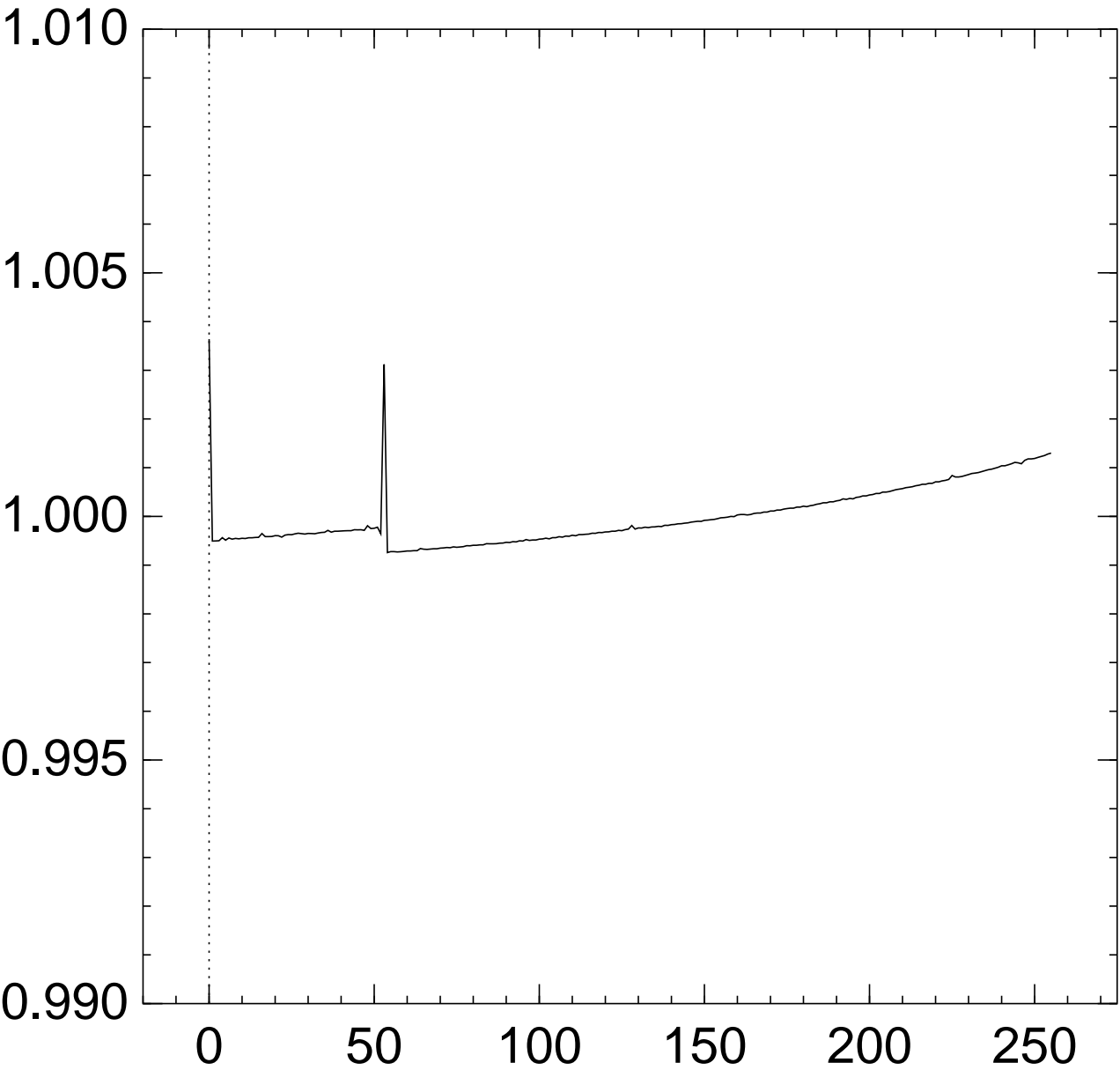
Graph of $256 \Pr[z_{51} = x]$:



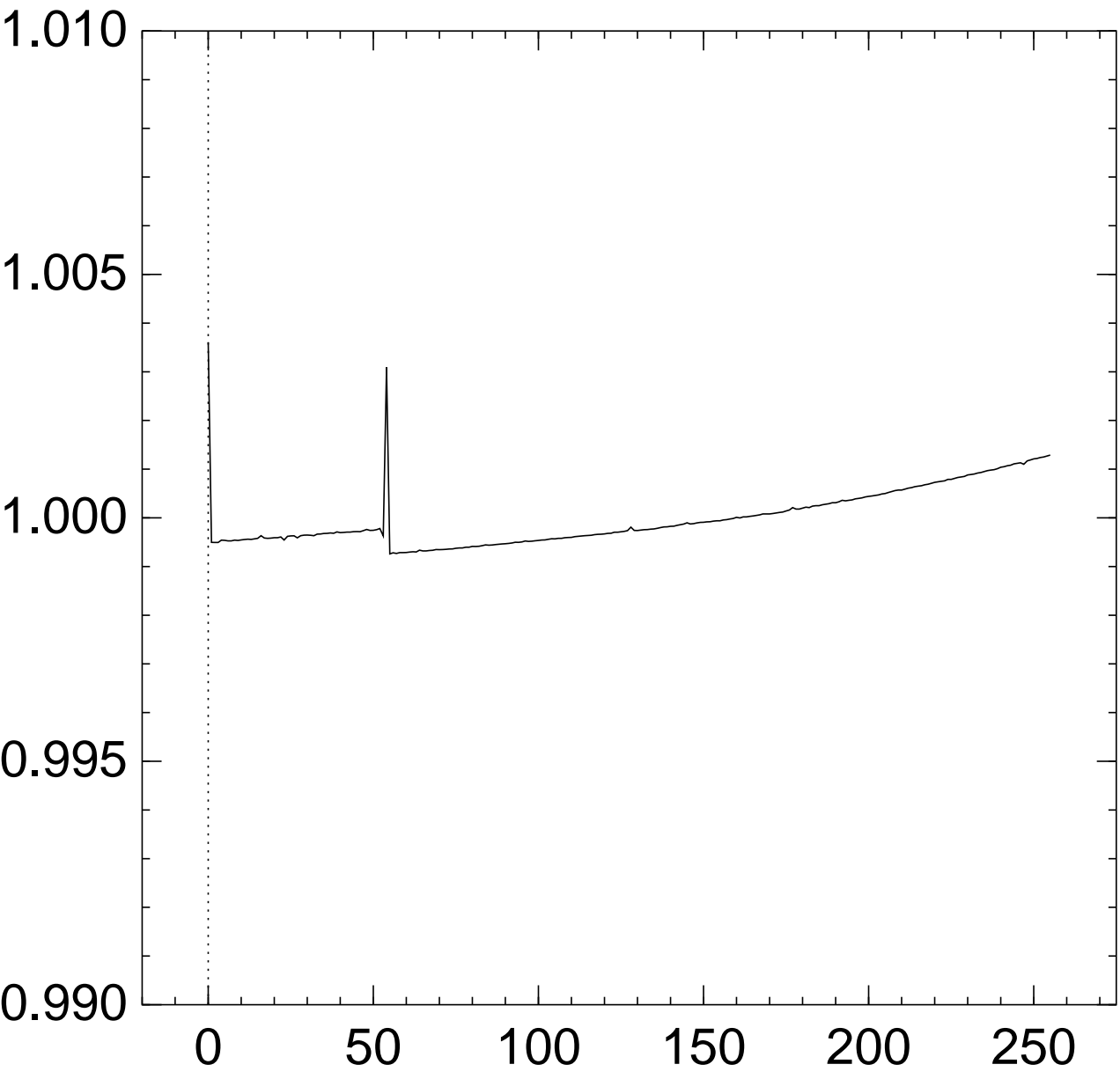
Graph of $256 \Pr[z_{52} = x]$:



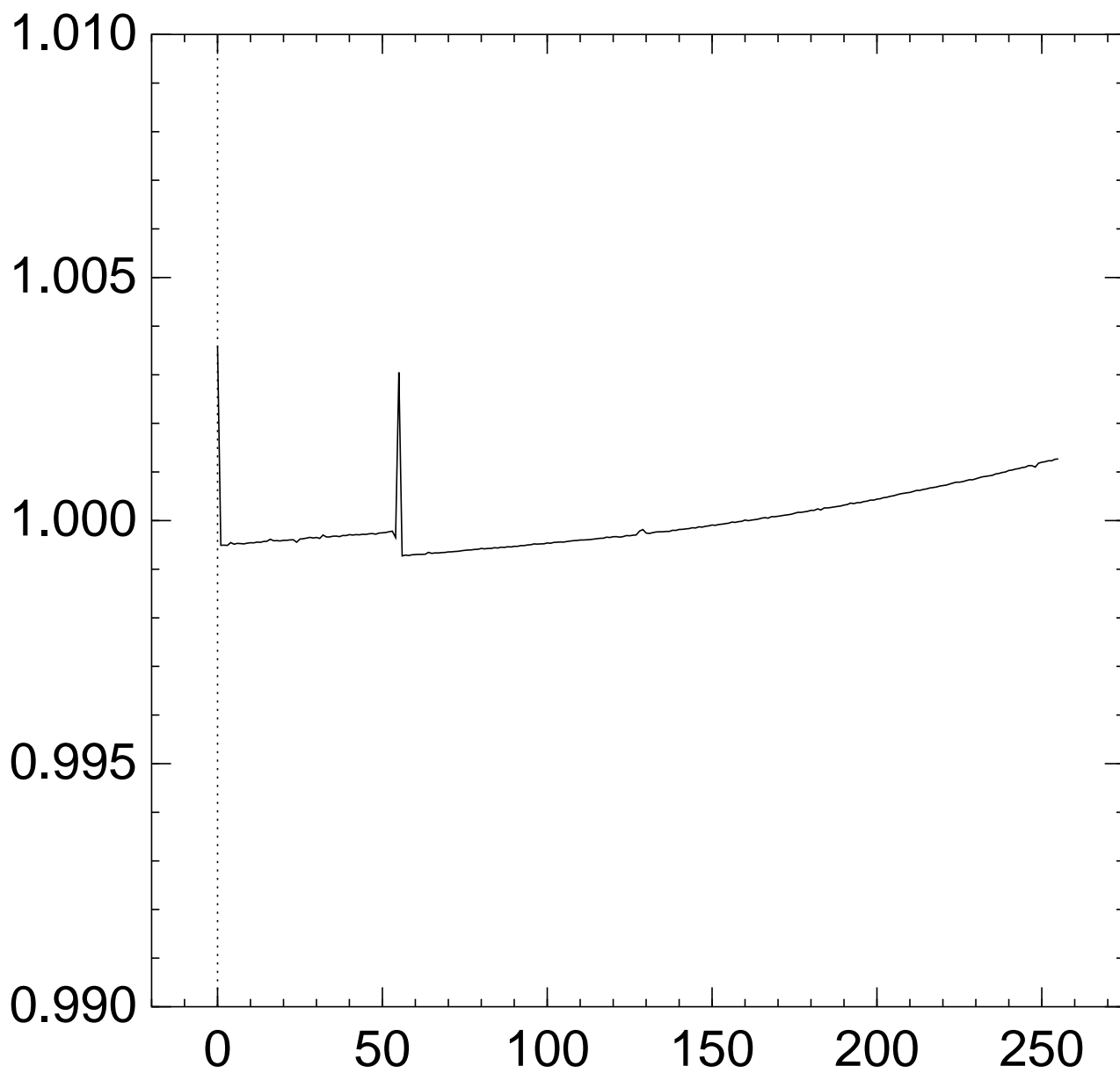
Graph of $256 \Pr[z_{53} = x]$:



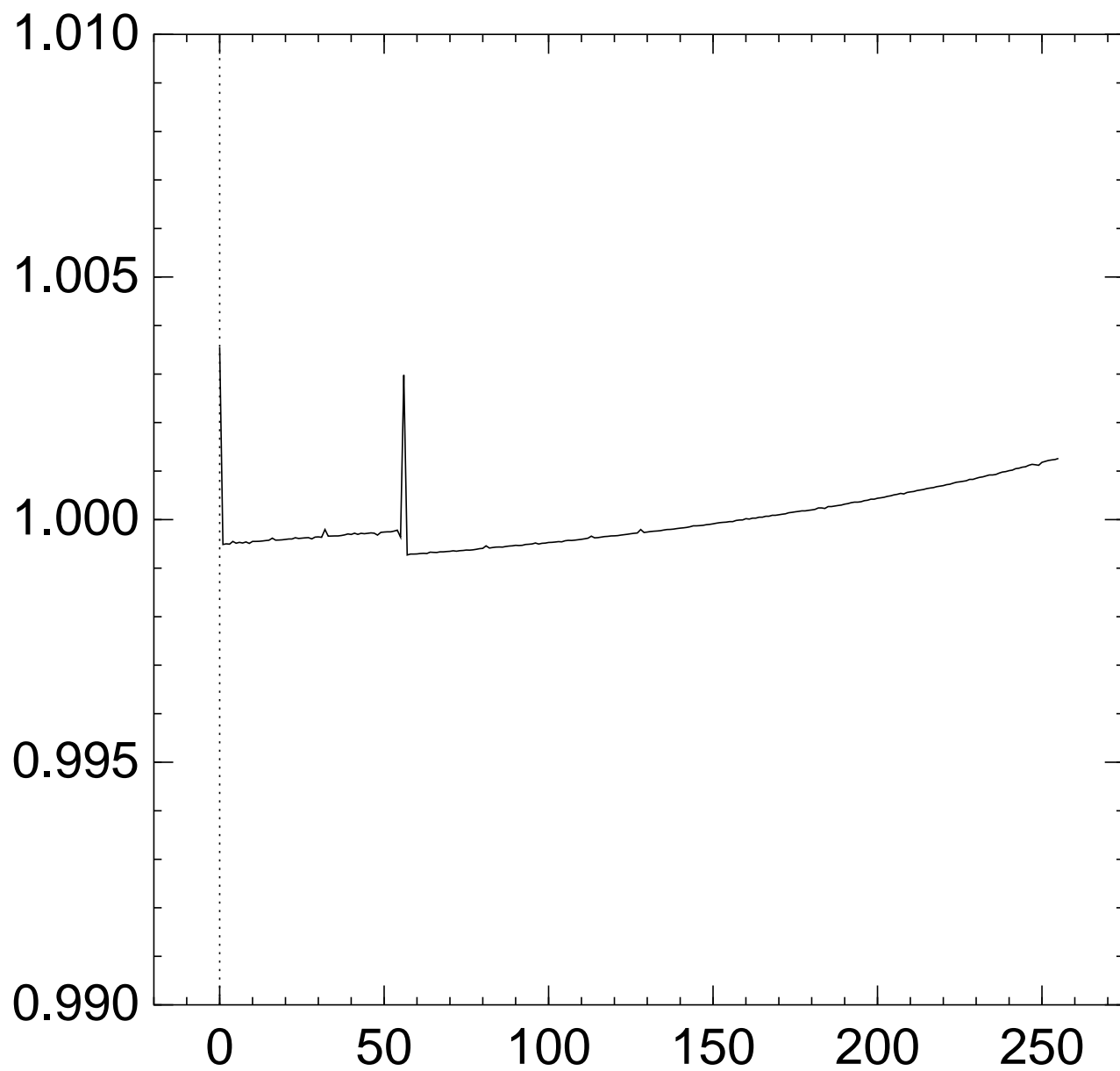
Graph of $256 \Pr[z_{54} = x]$:



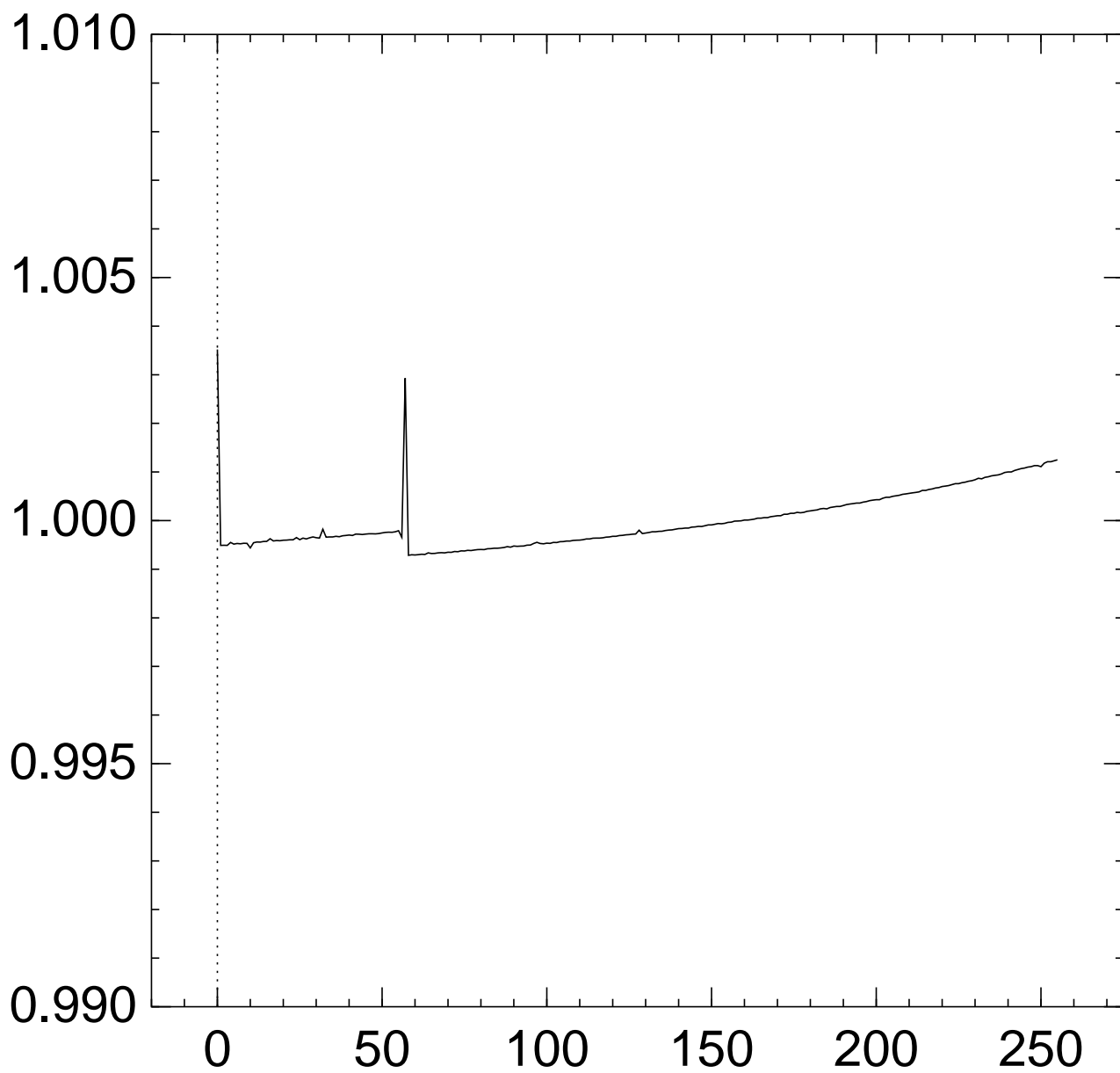
Graph of $256 \Pr[z_{55} = x]$:



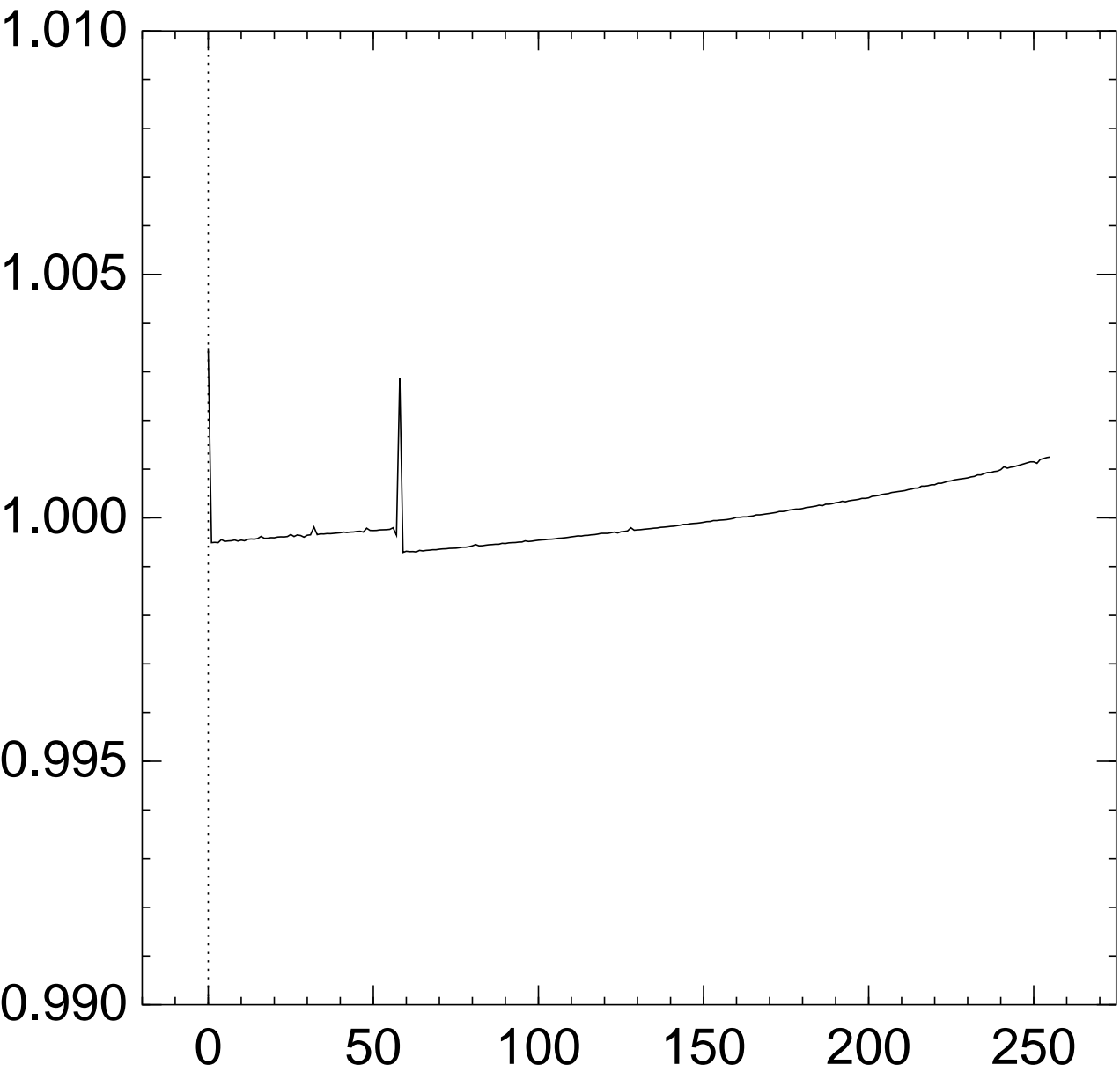
Graph of $256 \Pr[z_{56} = x]$:



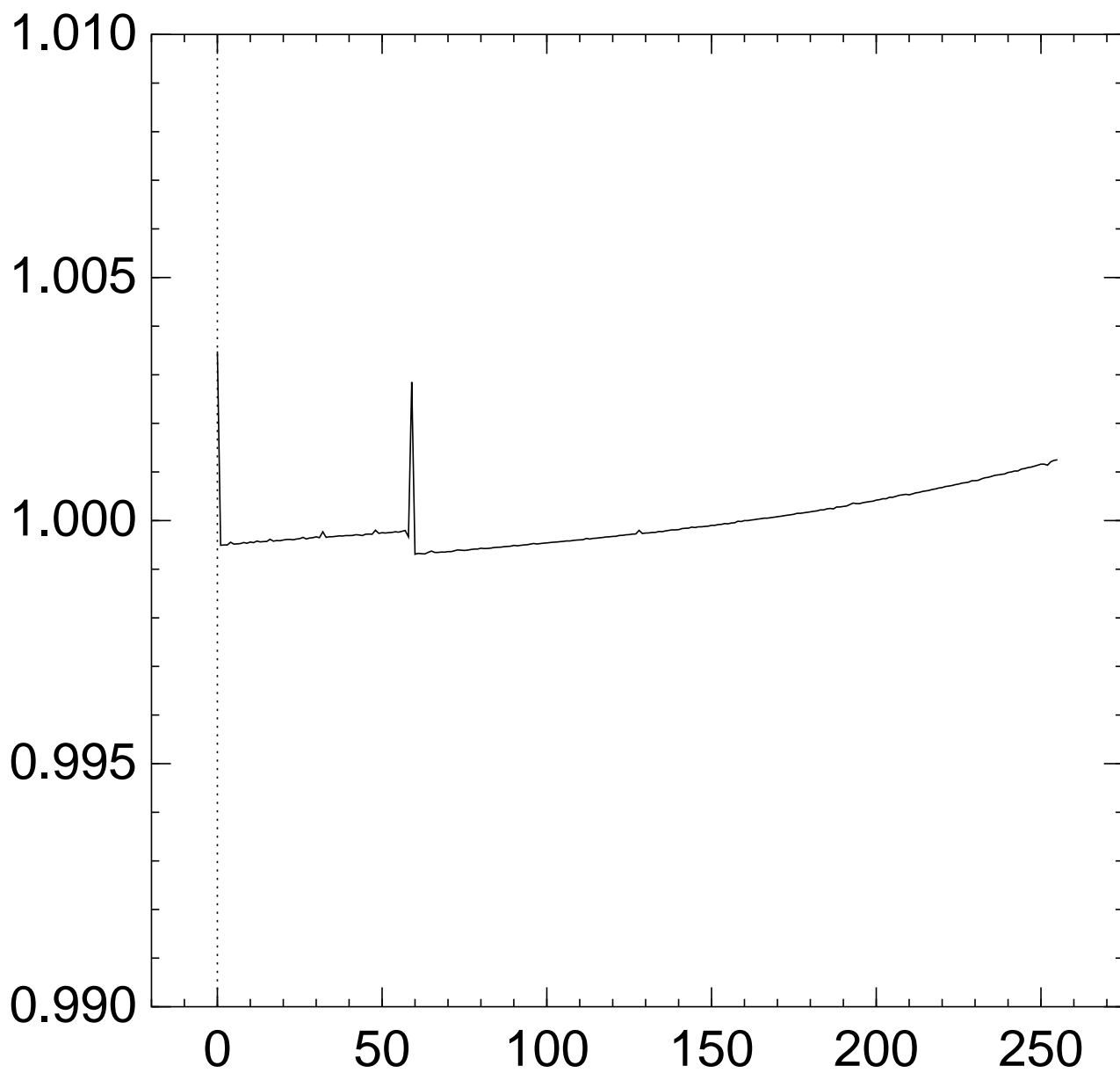
Graph of $256 \Pr[z_{57} = x]$:



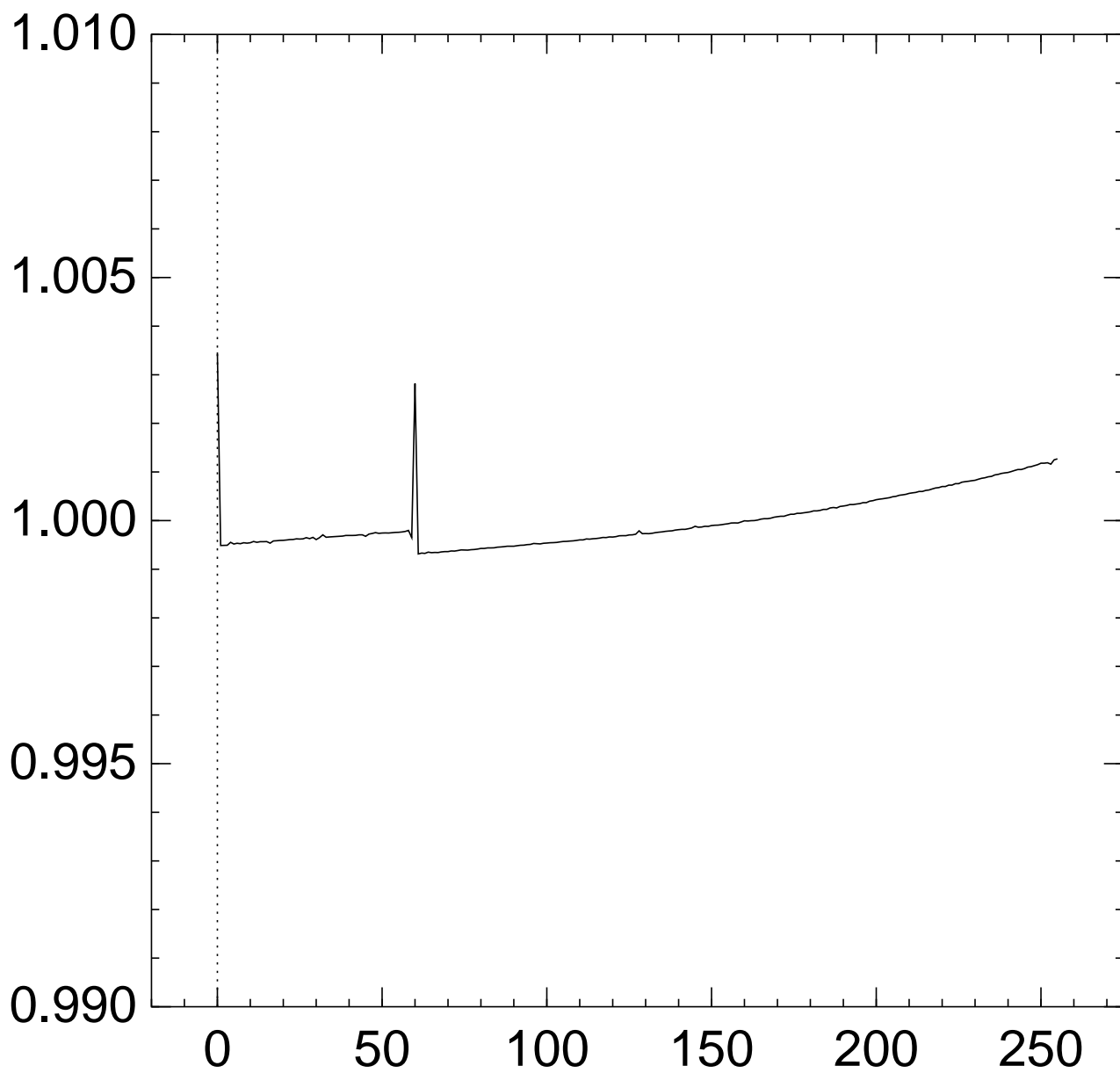
Graph of $256 \Pr[z_{58} = x]$:



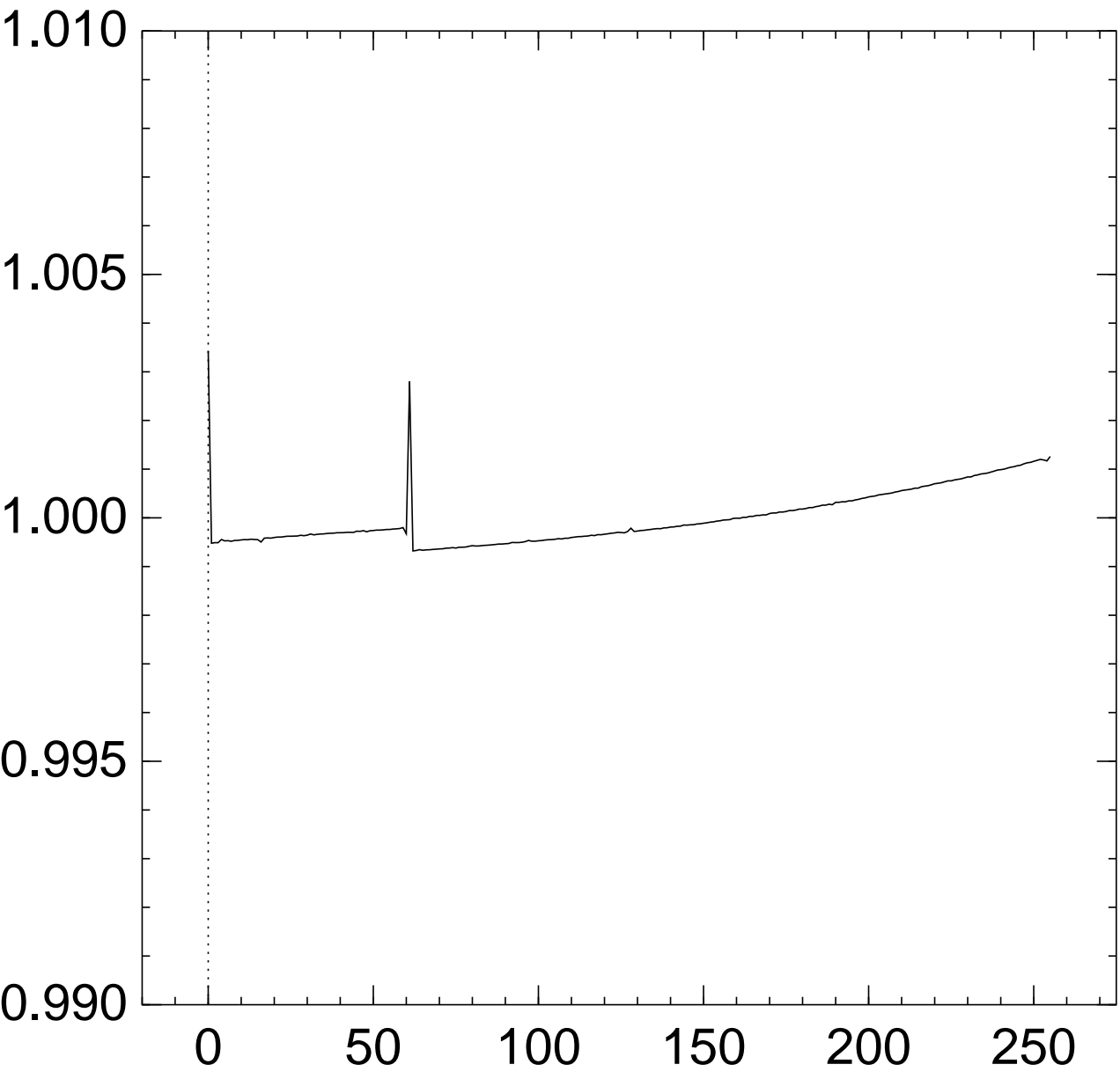
Graph of $256 \Pr[z_{59} = x]$:



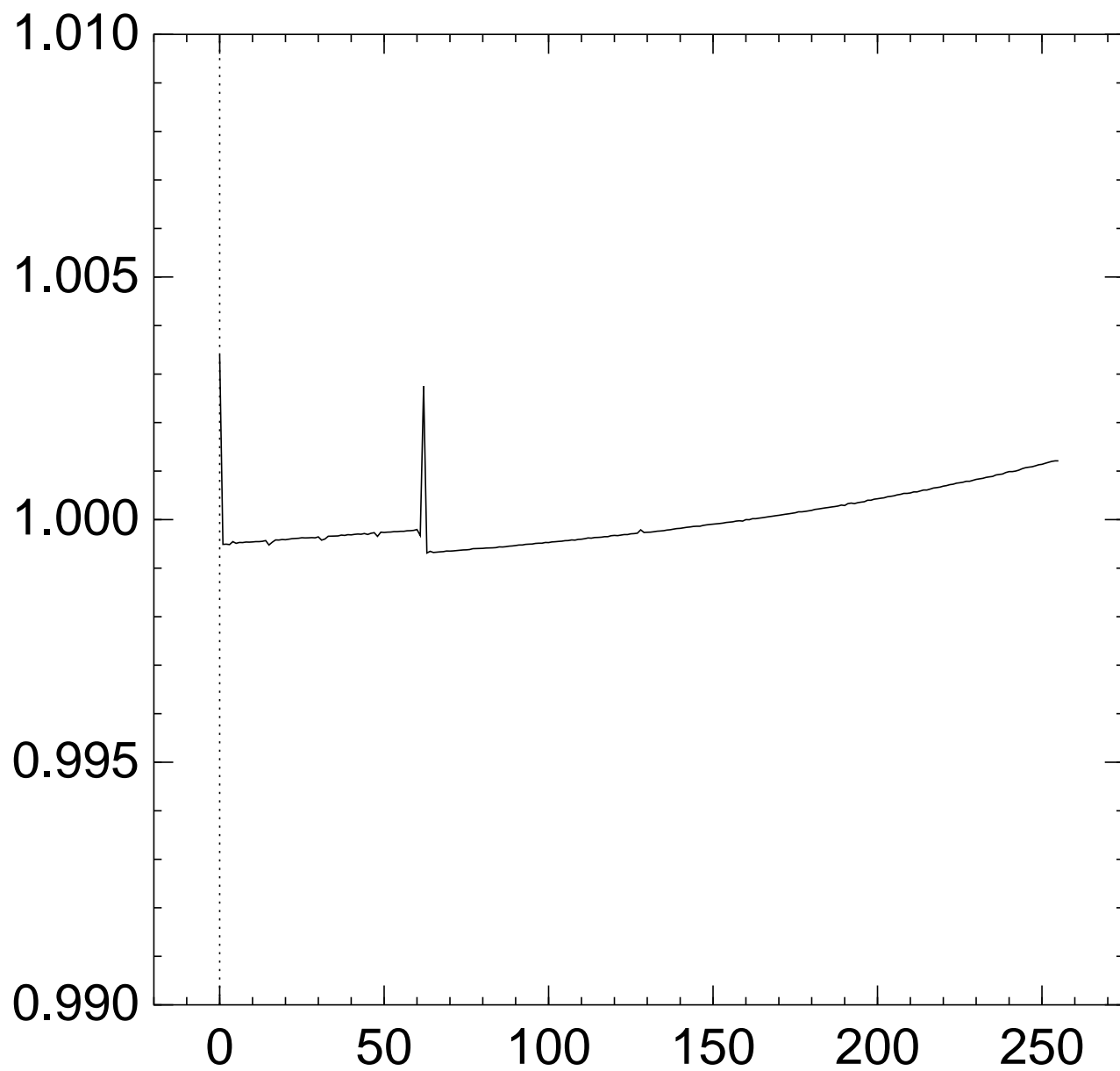
Graph of $256 \Pr[z_{60} = x]$:



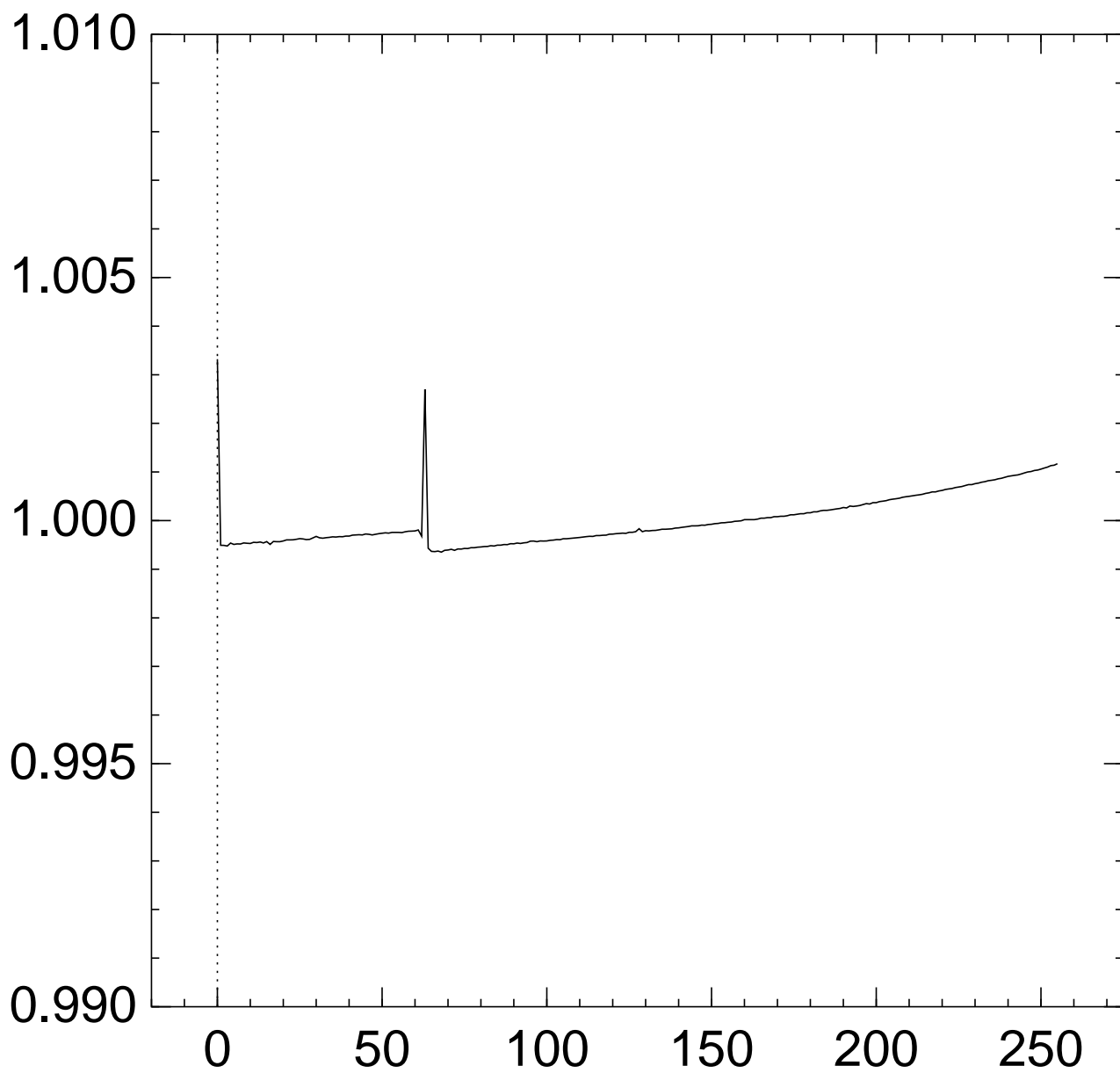
Graph of $256 \Pr[z_{61} = x]$:



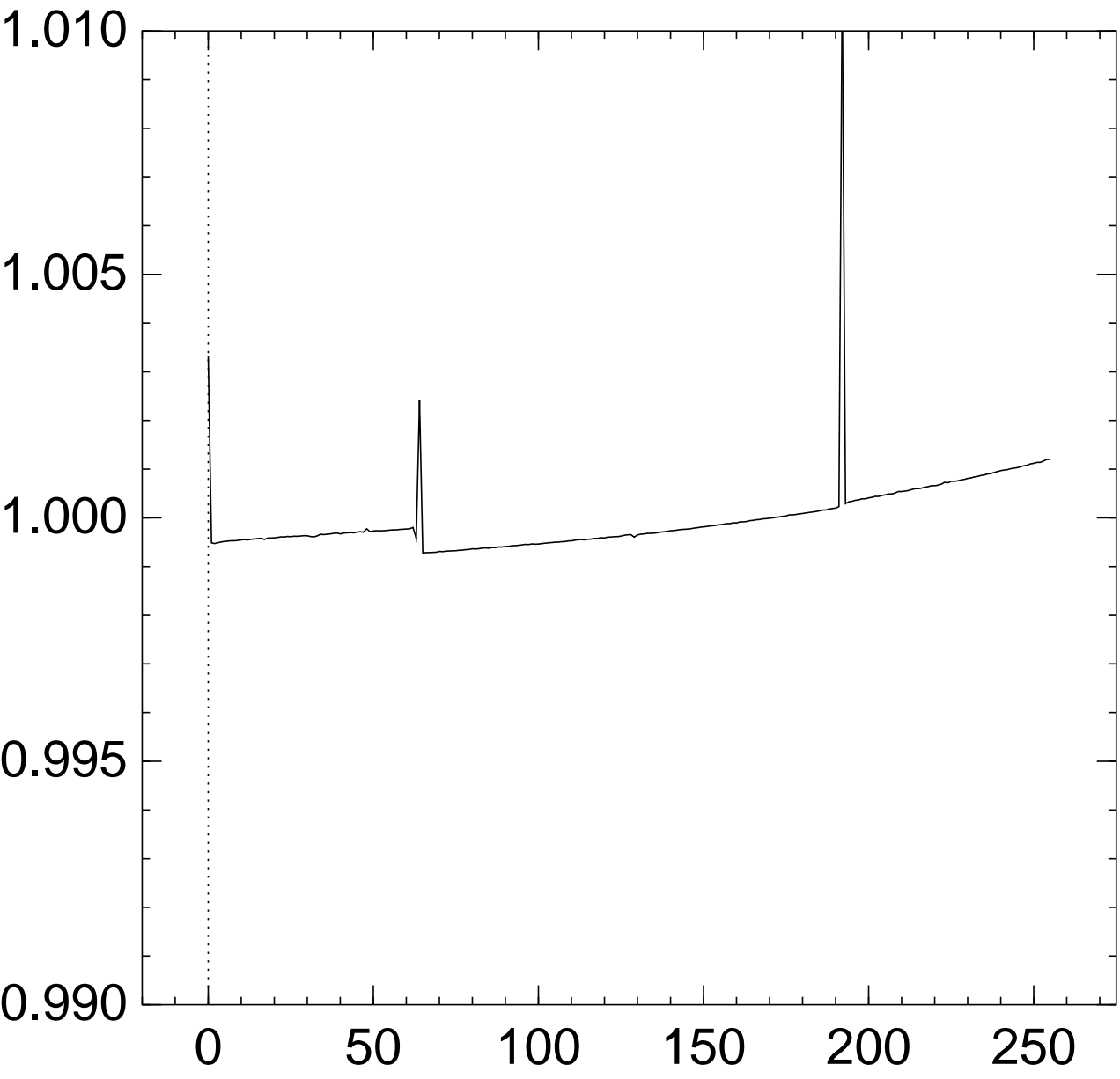
Graph of $256 \Pr[z_{62} = x]$:



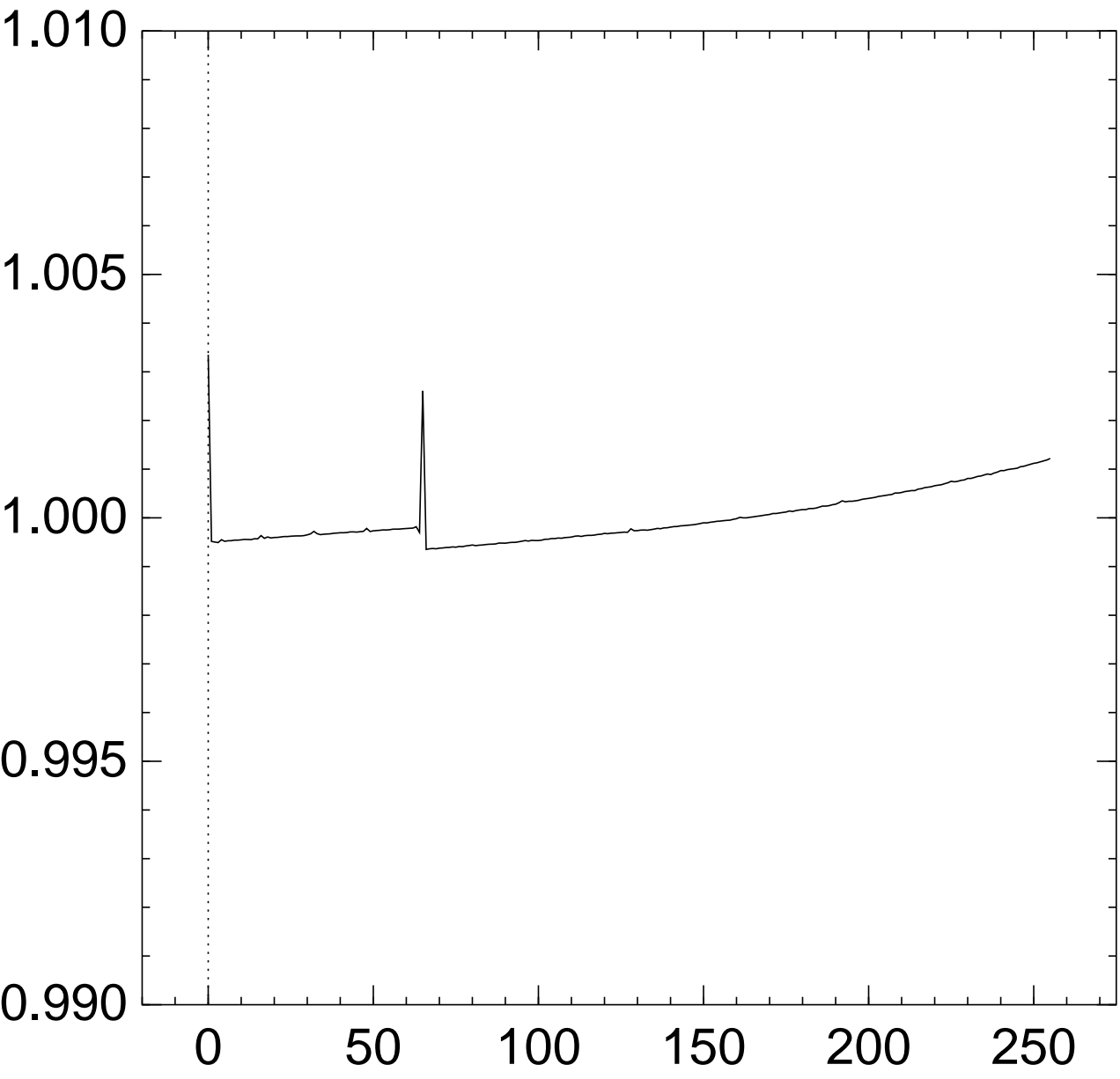
Graph of $256 \Pr[z_{63} = x]$:



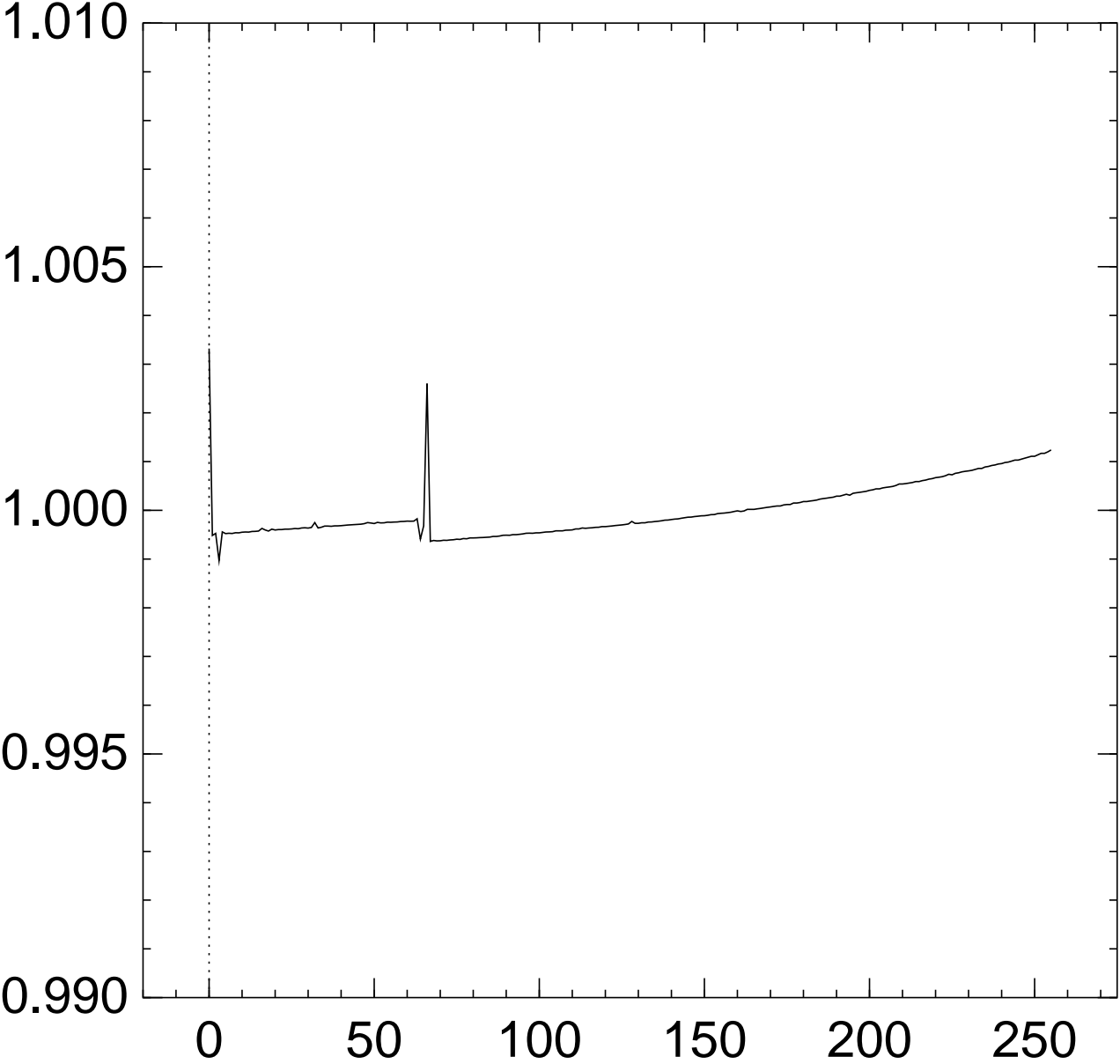
Graph of $256 \Pr[z_{64} = x]$:



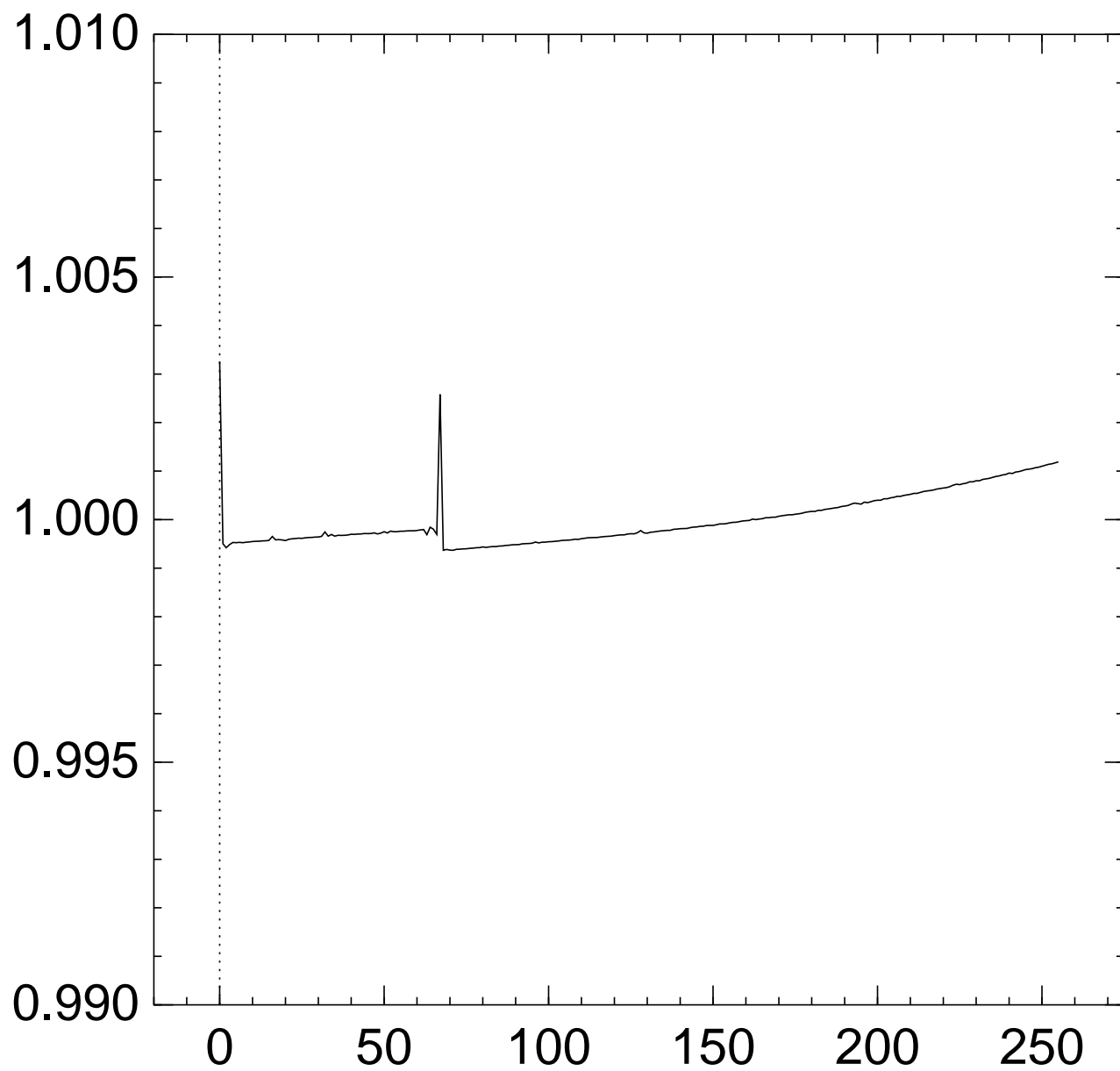
Graph of $256 \Pr[z_{65} = x]$:



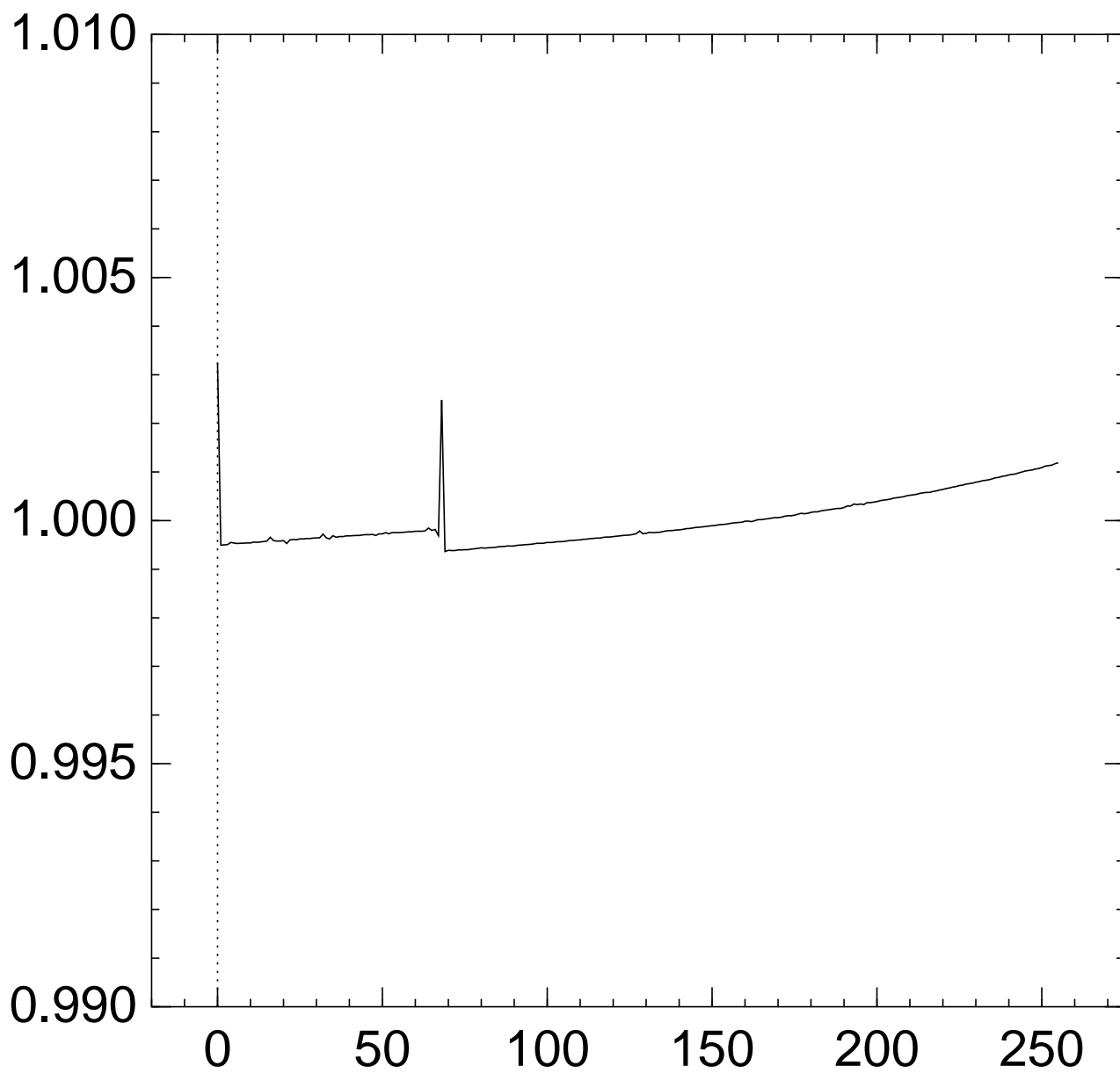
Graph of $256 \Pr[z_{66} = x]$:



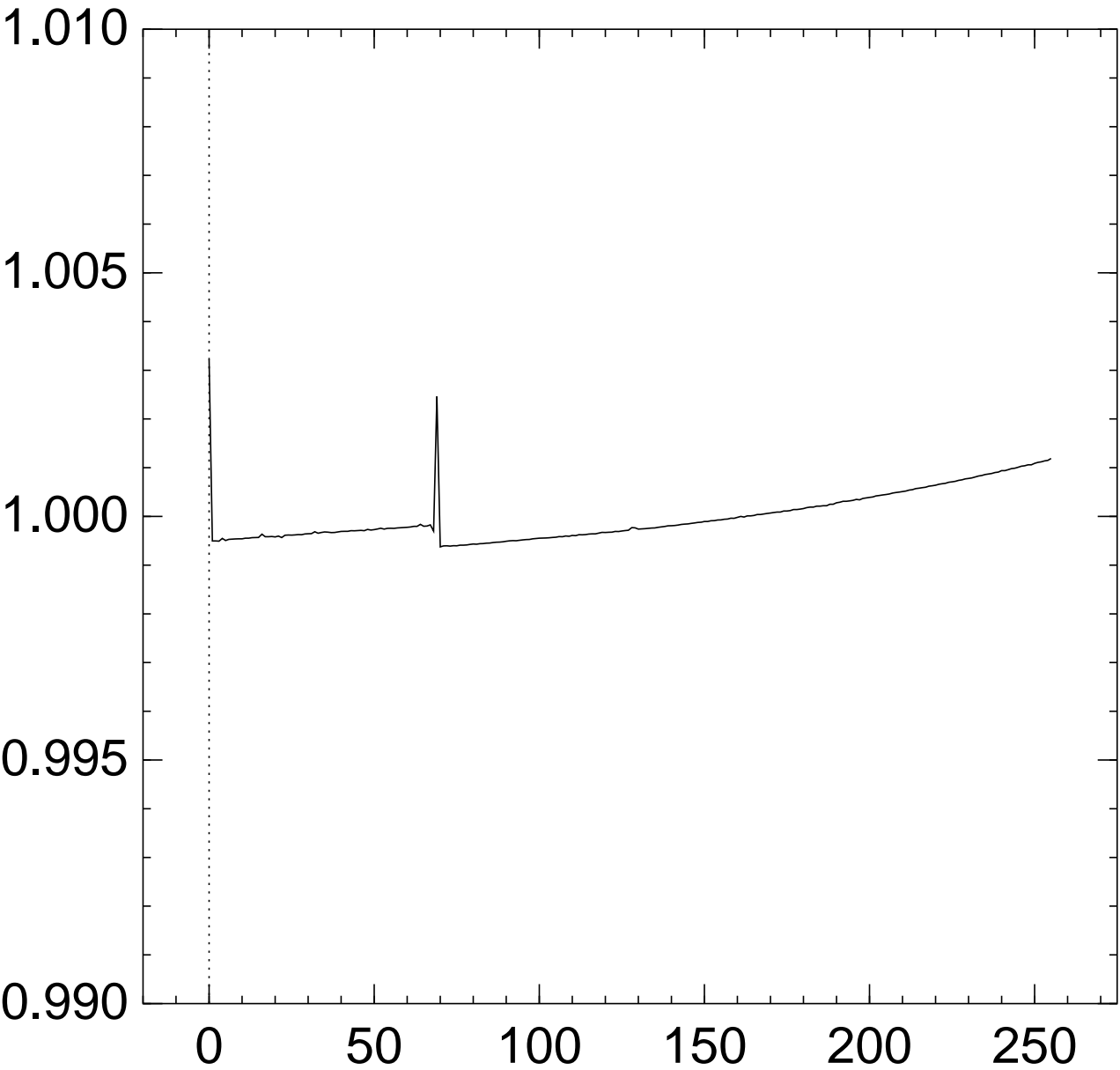
Graph of $256 \Pr[z_{67} = x]$:



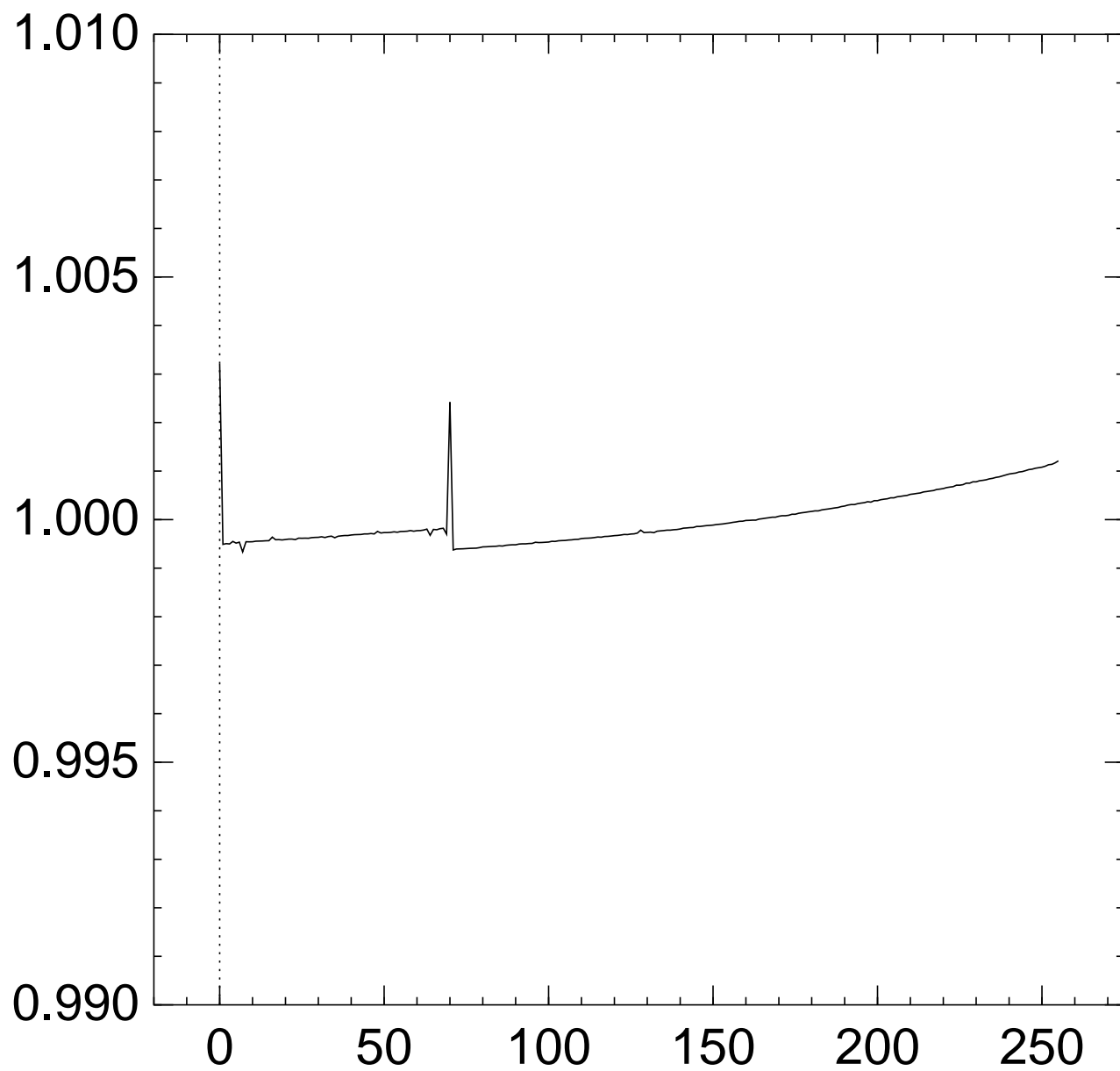
Graph of $256 \Pr[z_{68} = x]$:



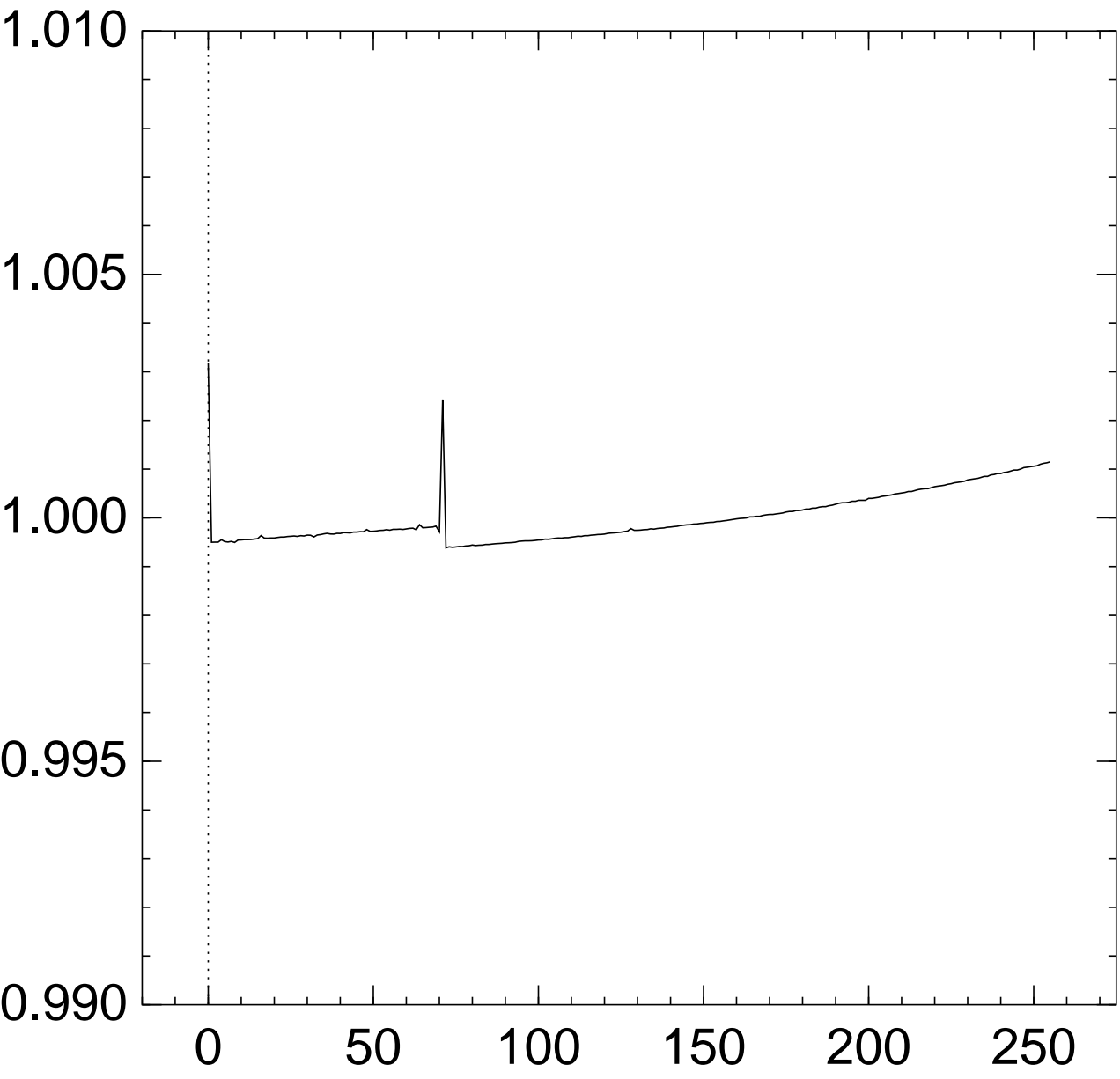
Graph of $256 \Pr[z_{69} = x]$:



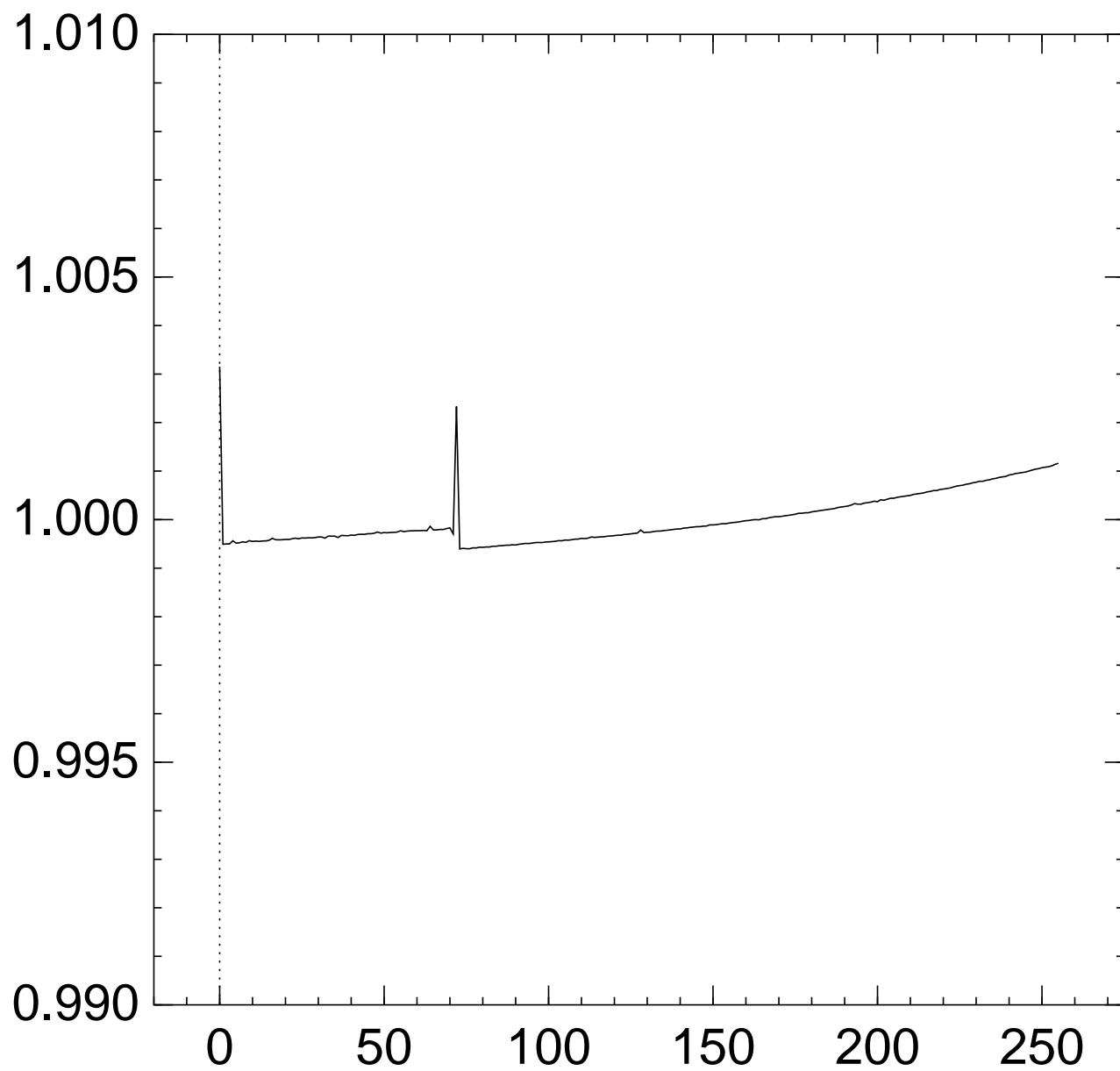
Graph of $256 \Pr[z_{70} = x]$:



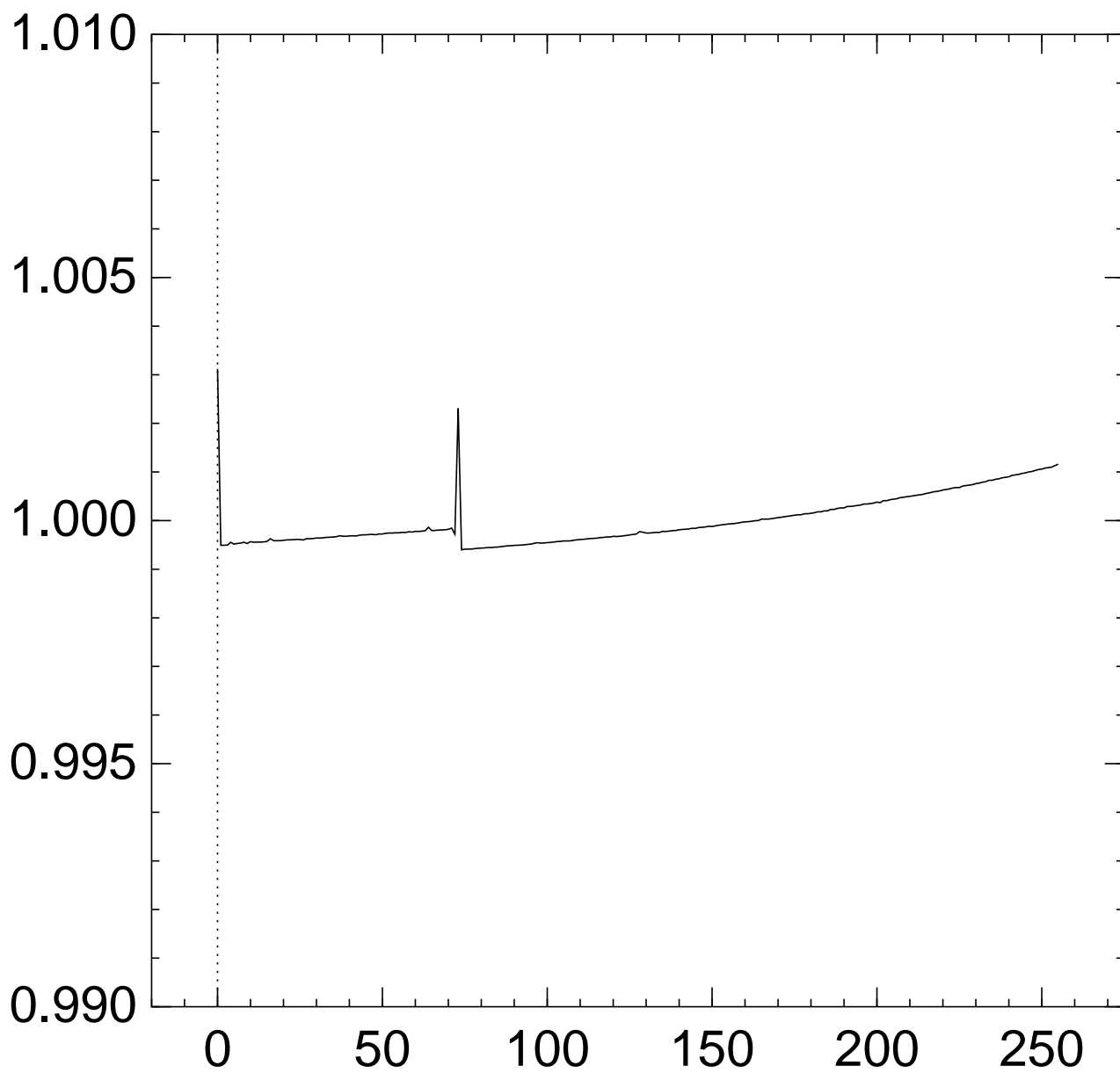
Graph of $256 \Pr[z_{71} = x]$:



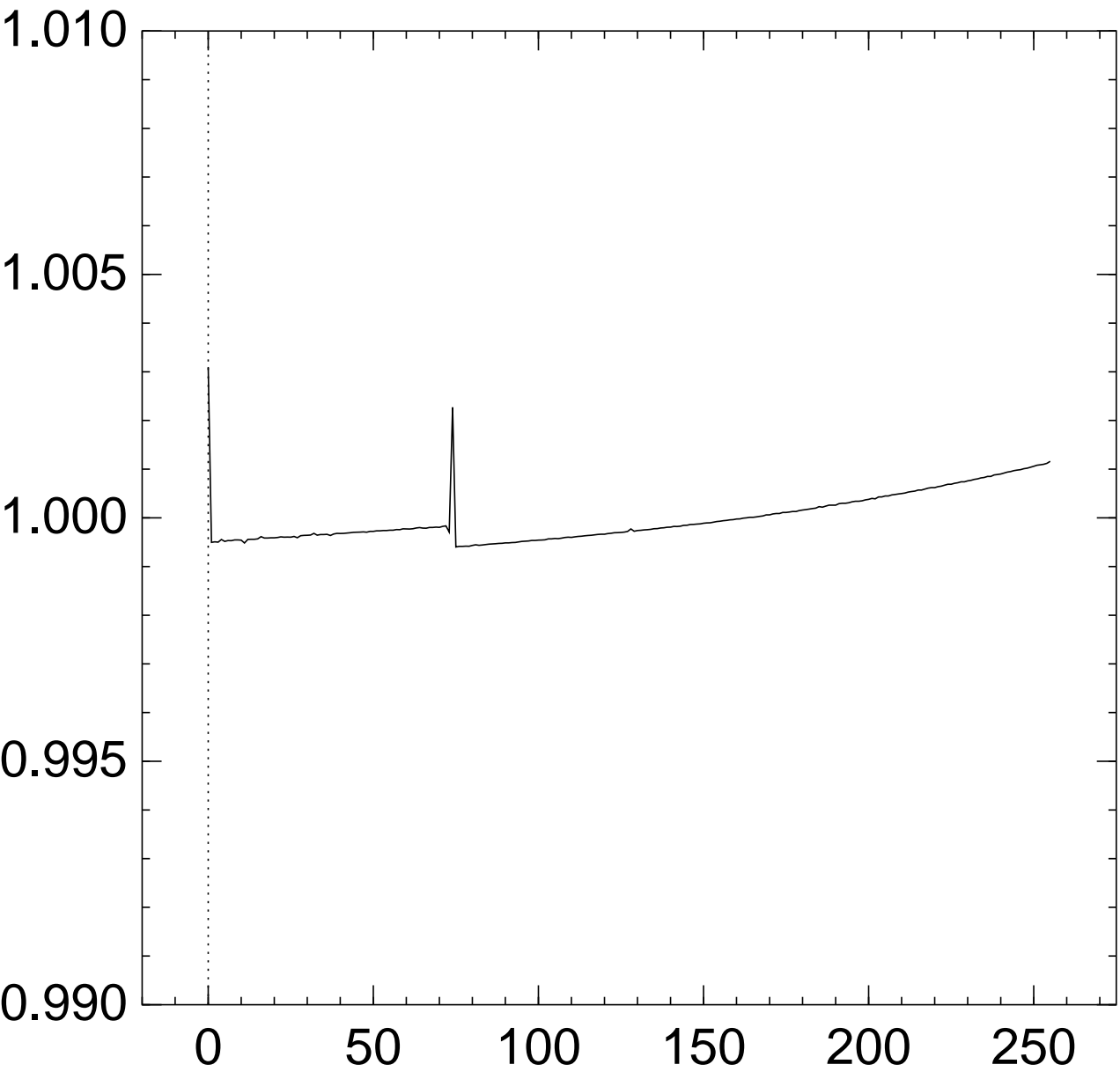
Graph of $256 \Pr[z_{72} = x]$:



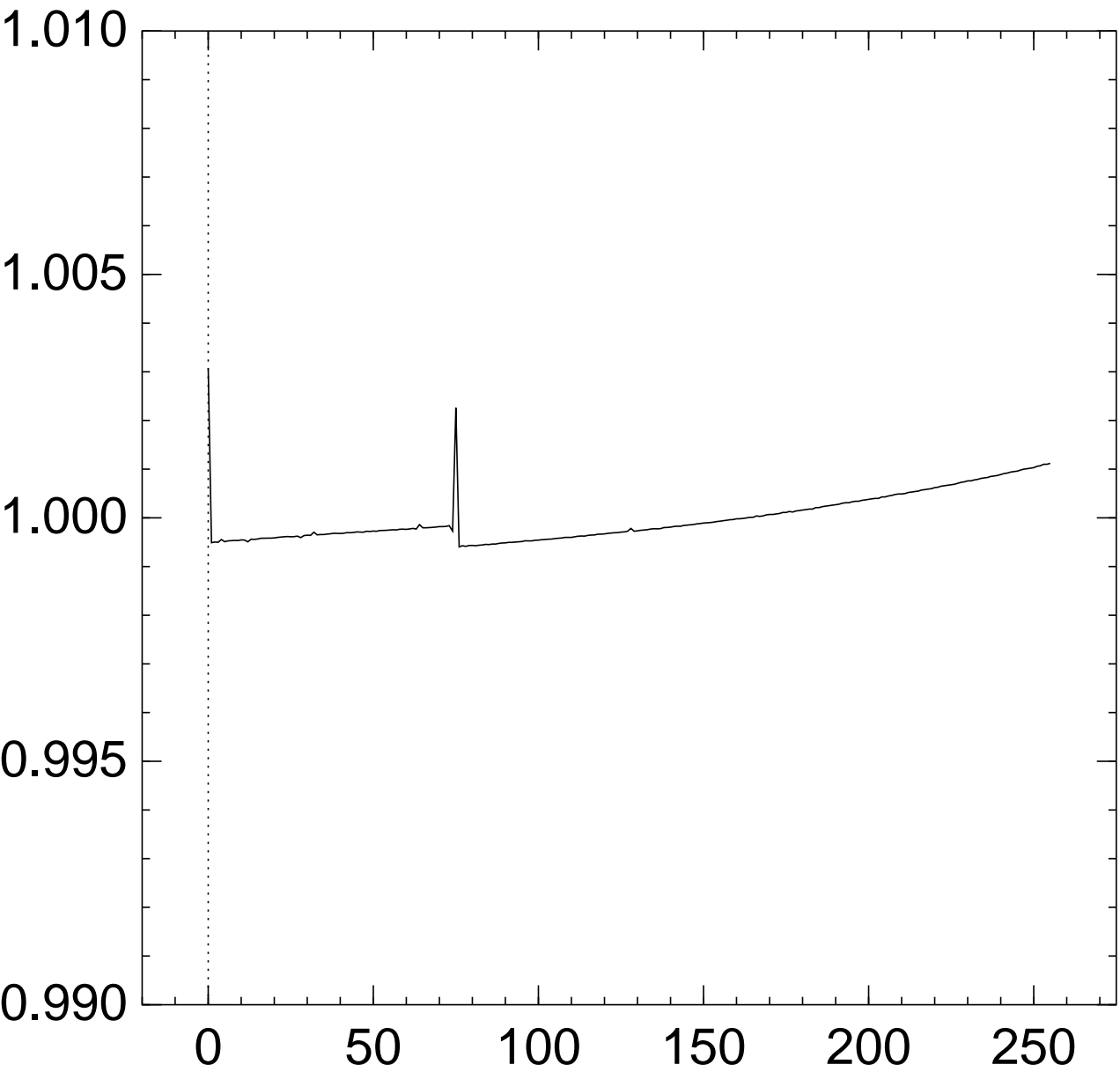
Graph of $256 \Pr[z_{73} = x]$:



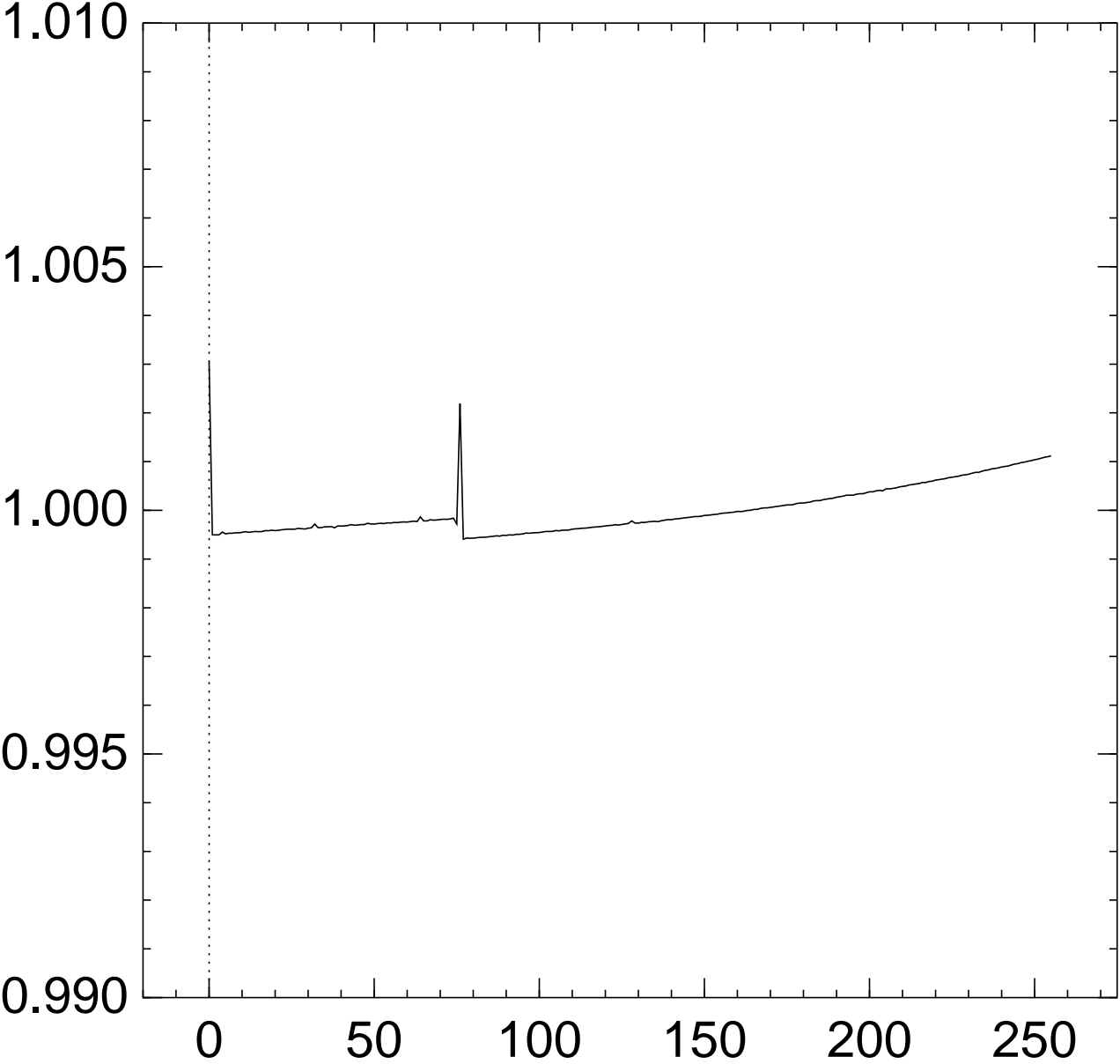
Graph of $256 \Pr[z_{74} = x]$:



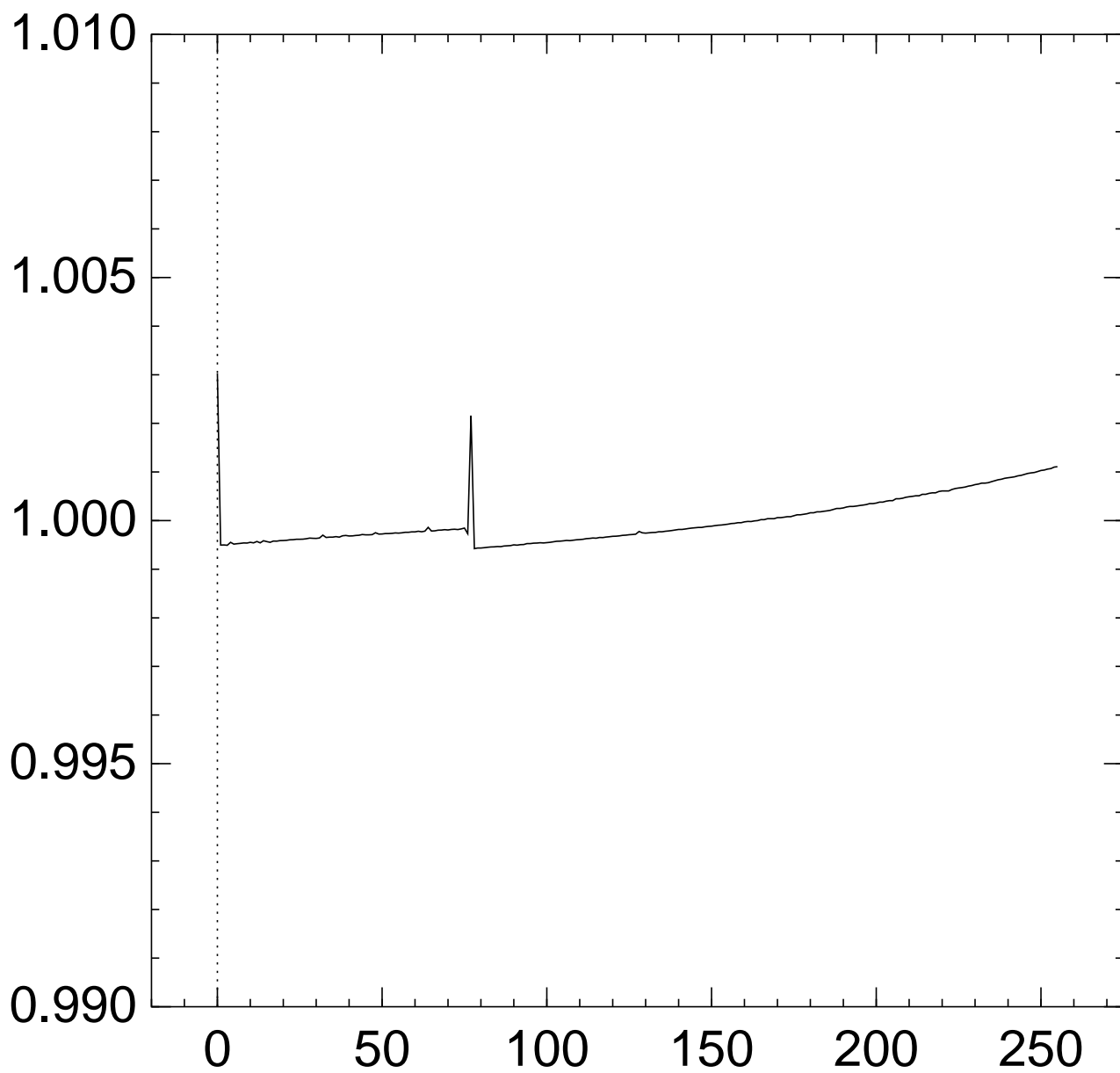
Graph of $256 \Pr[z_{75} = x]$:



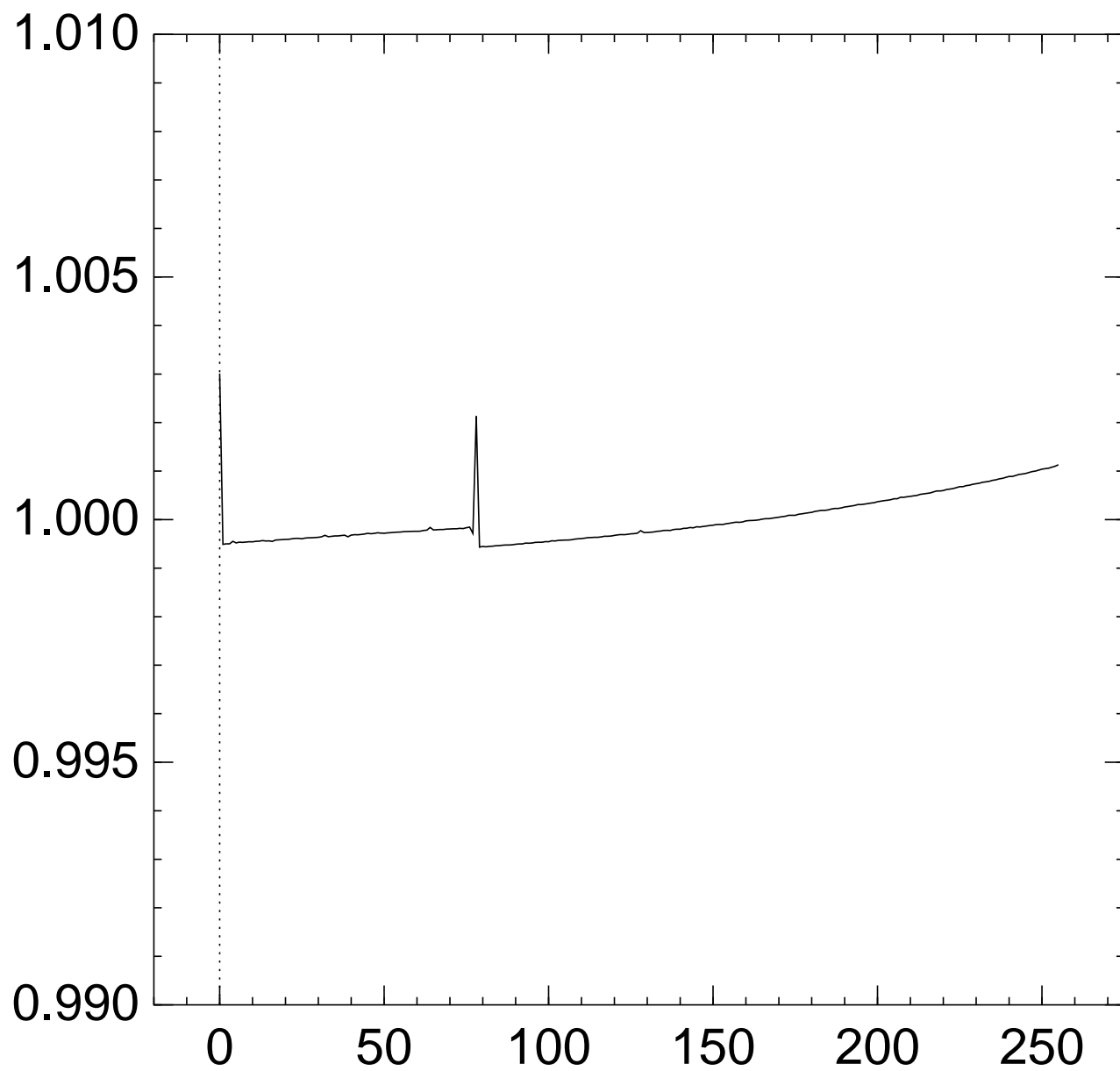
Graph of $256 \Pr[z_{76} = x]$:



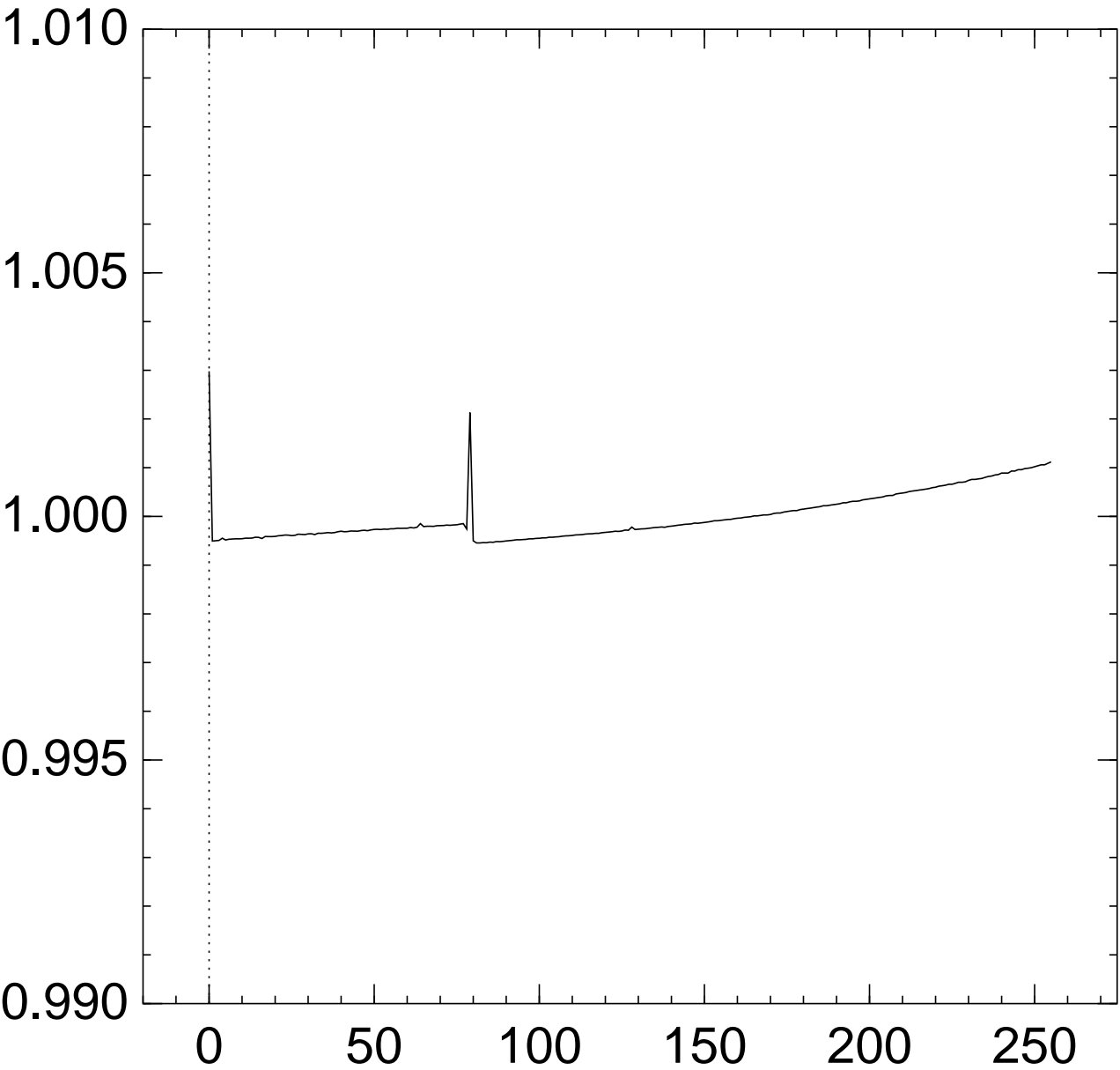
Graph of $256 \Pr[z_{77} = x]$:



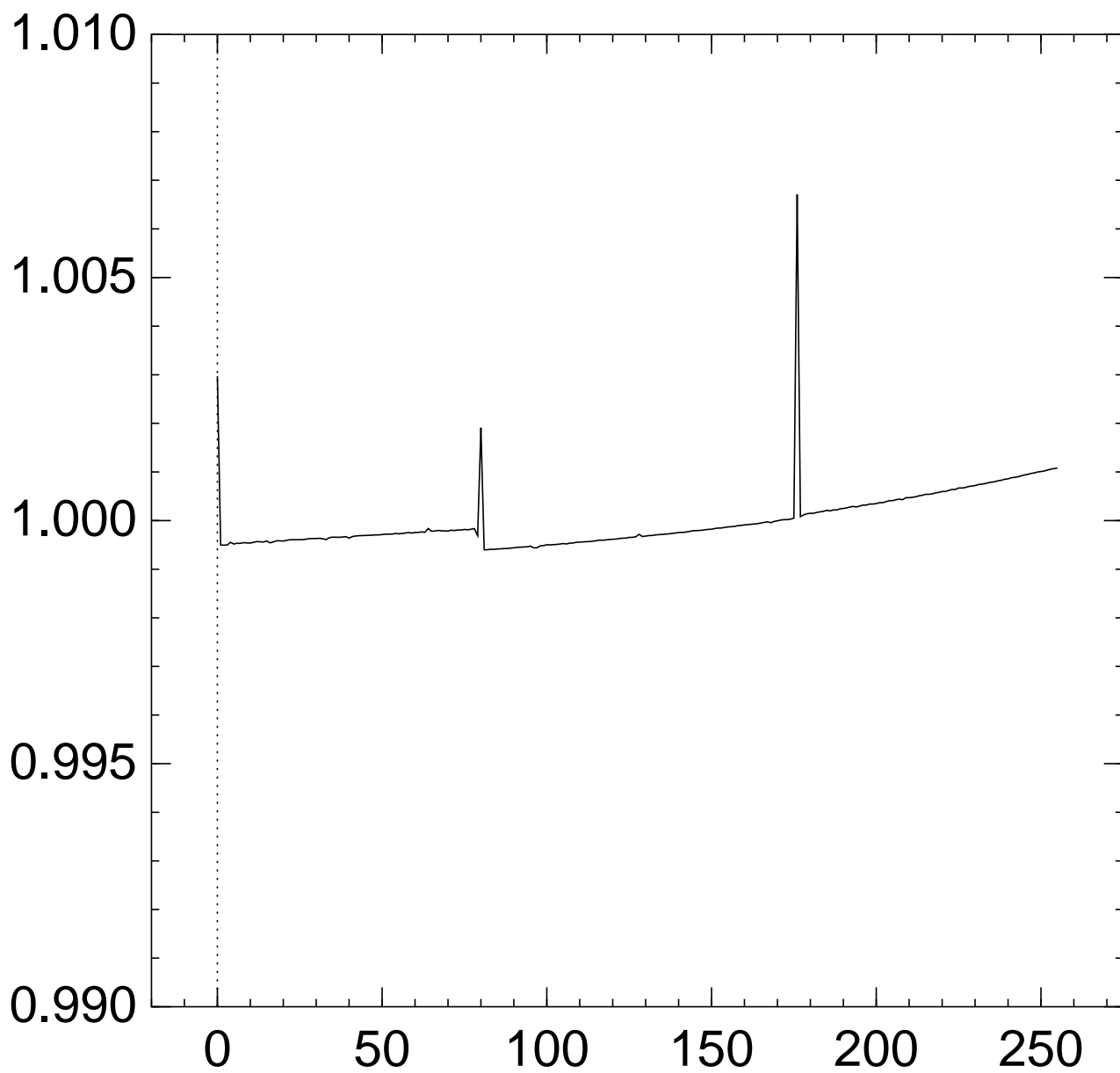
Graph of $256 \Pr[z_{78} = x]$:



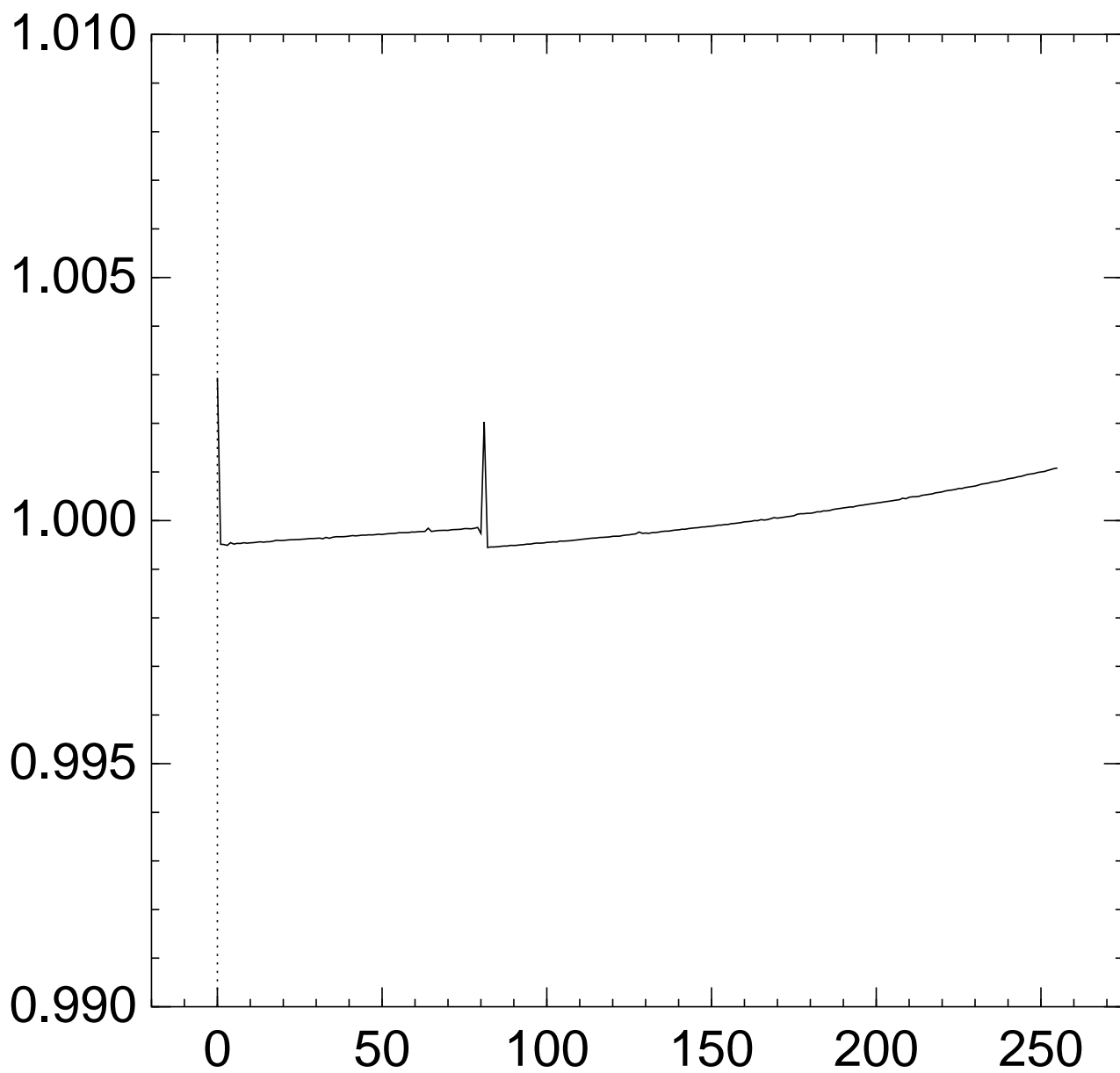
Graph of $256 \Pr[z_{79} = x]$:



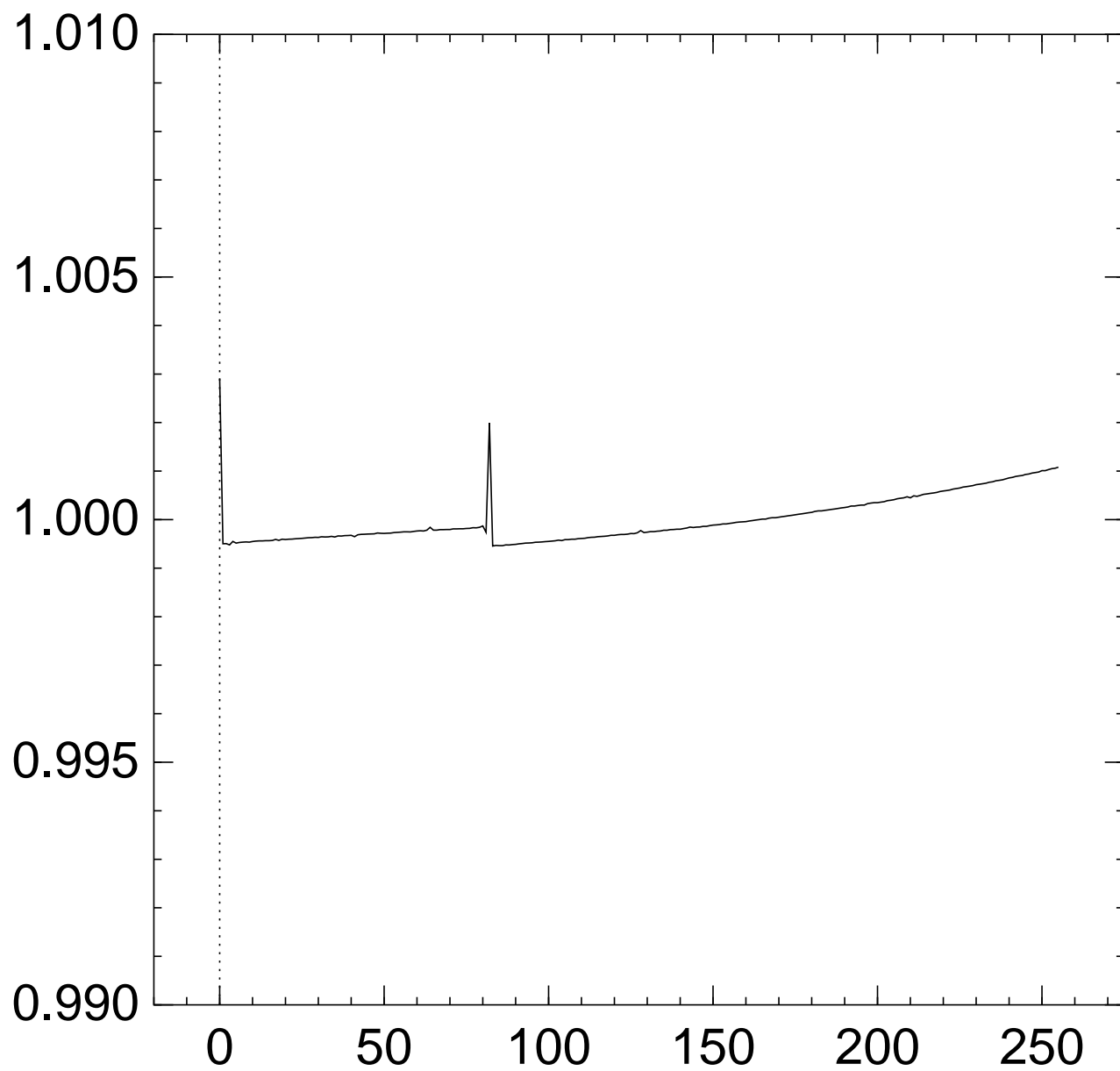
Graph of $256 \Pr[z_{80} = x]$:



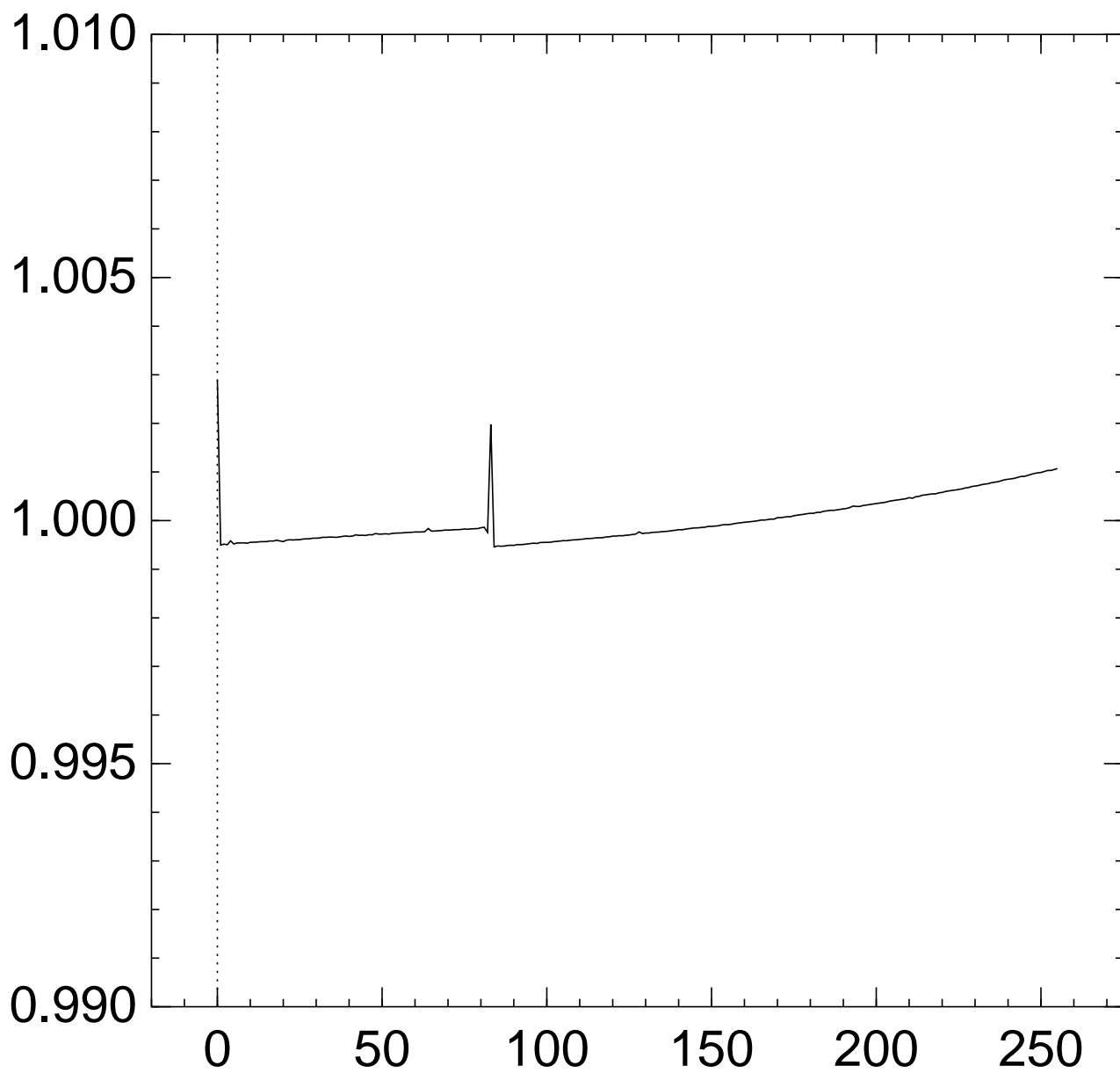
Graph of $256 \Pr[z_{81} = x]$:



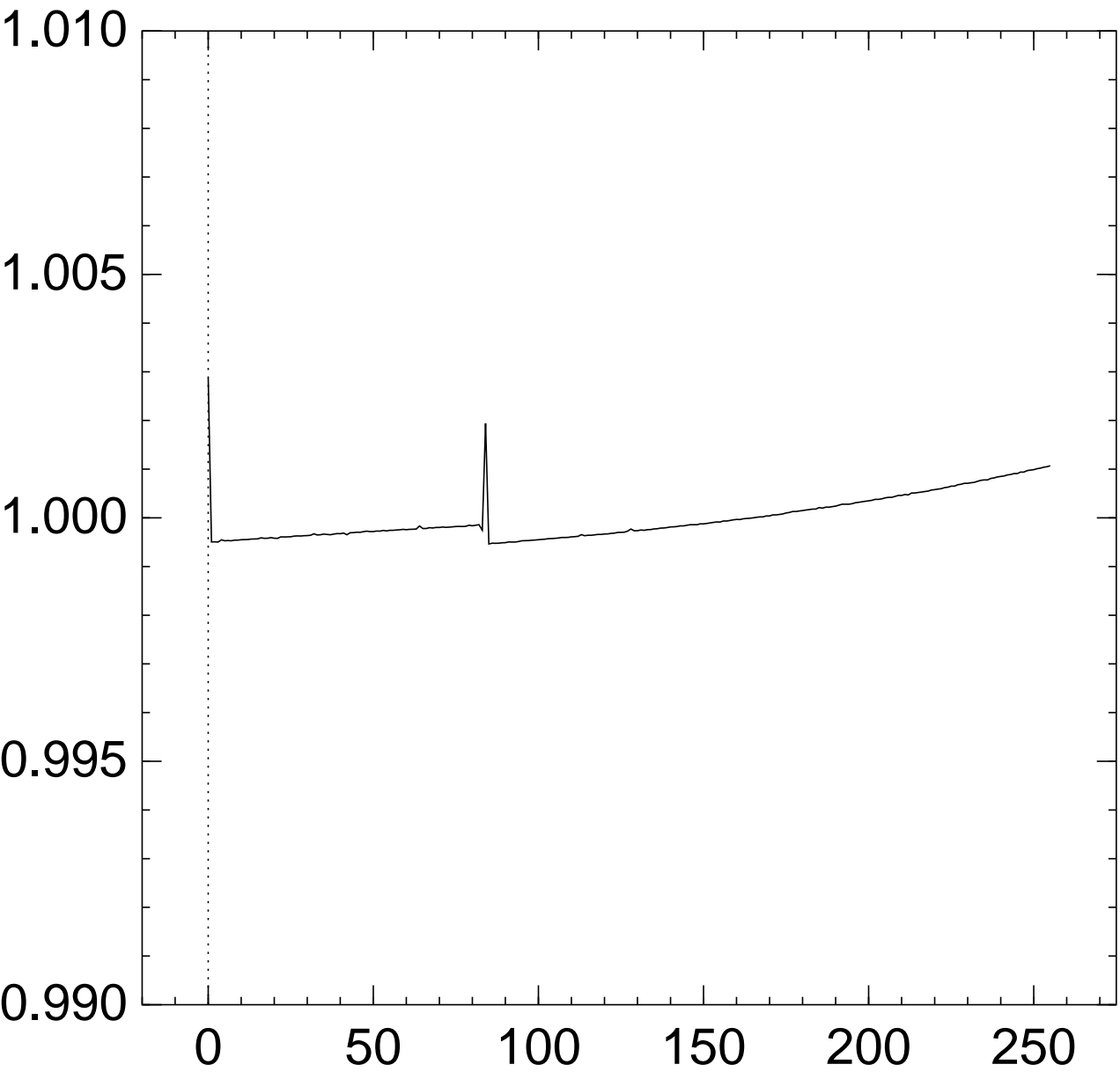
Graph of $256 \Pr[z_{82} = x]$:



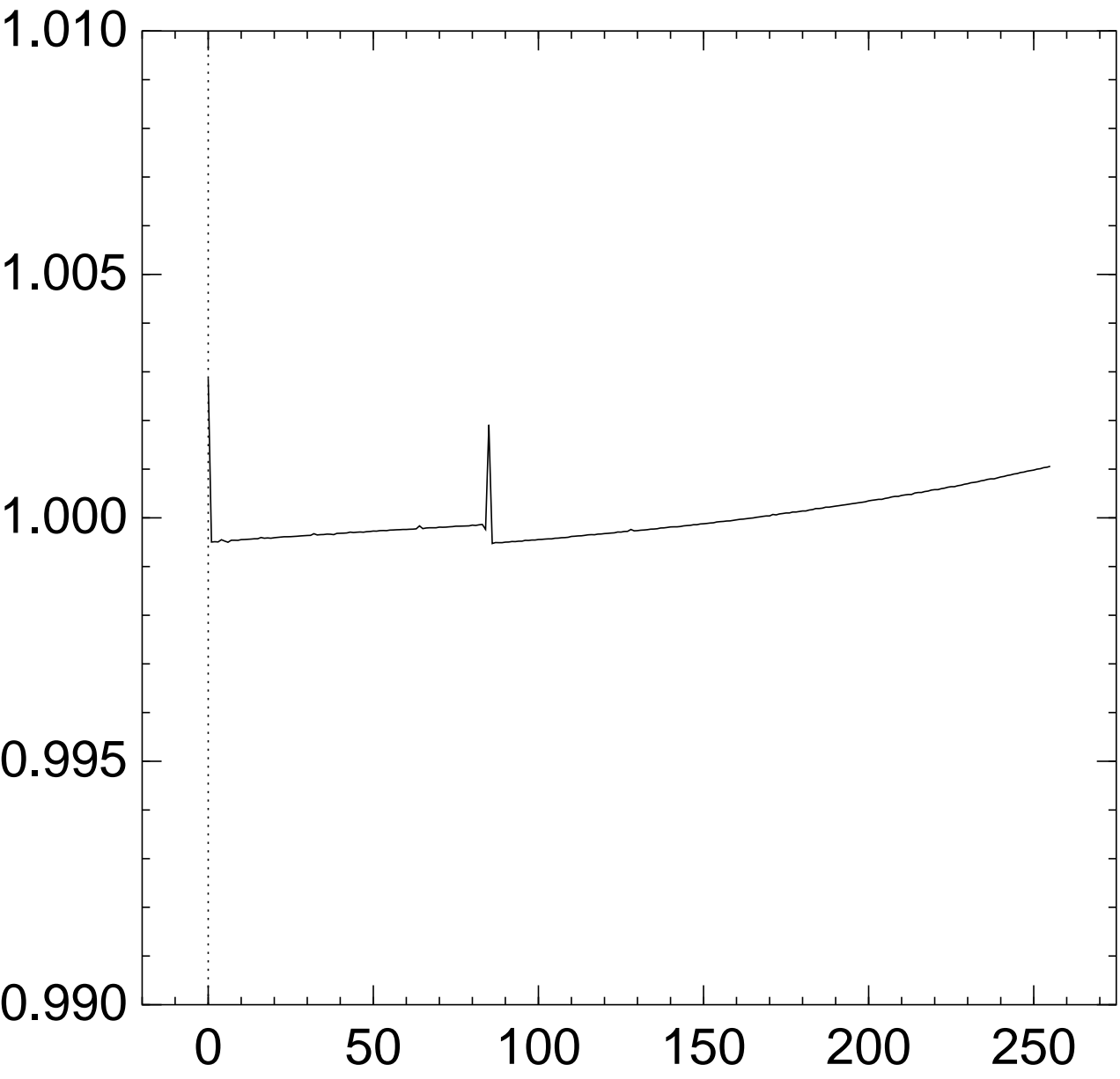
Graph of $256 \Pr[z_{83} = x]$:



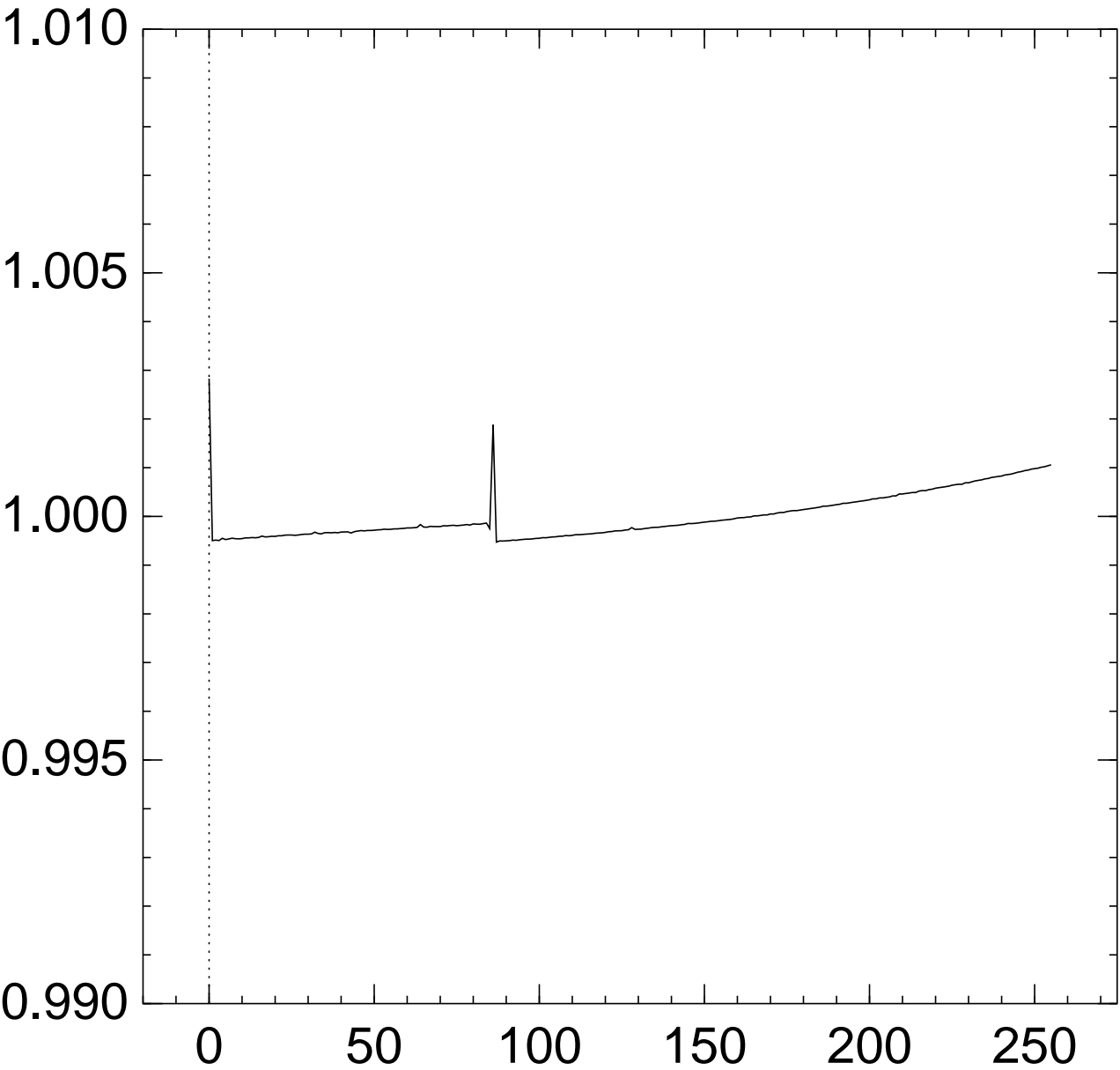
Graph of $256 \Pr[z_{84} = x]$:



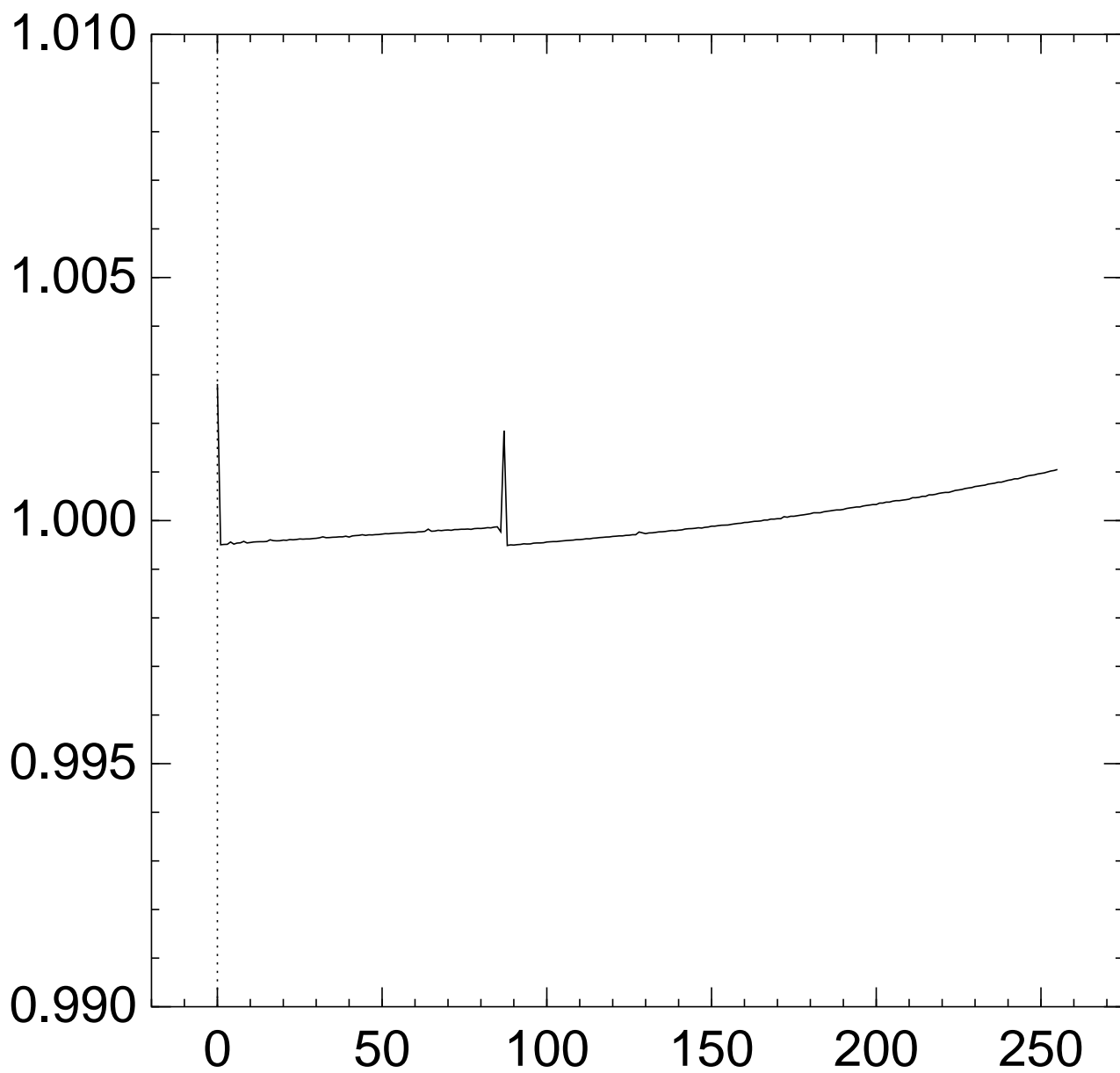
Graph of $256 \Pr[z_{85} = x]$:



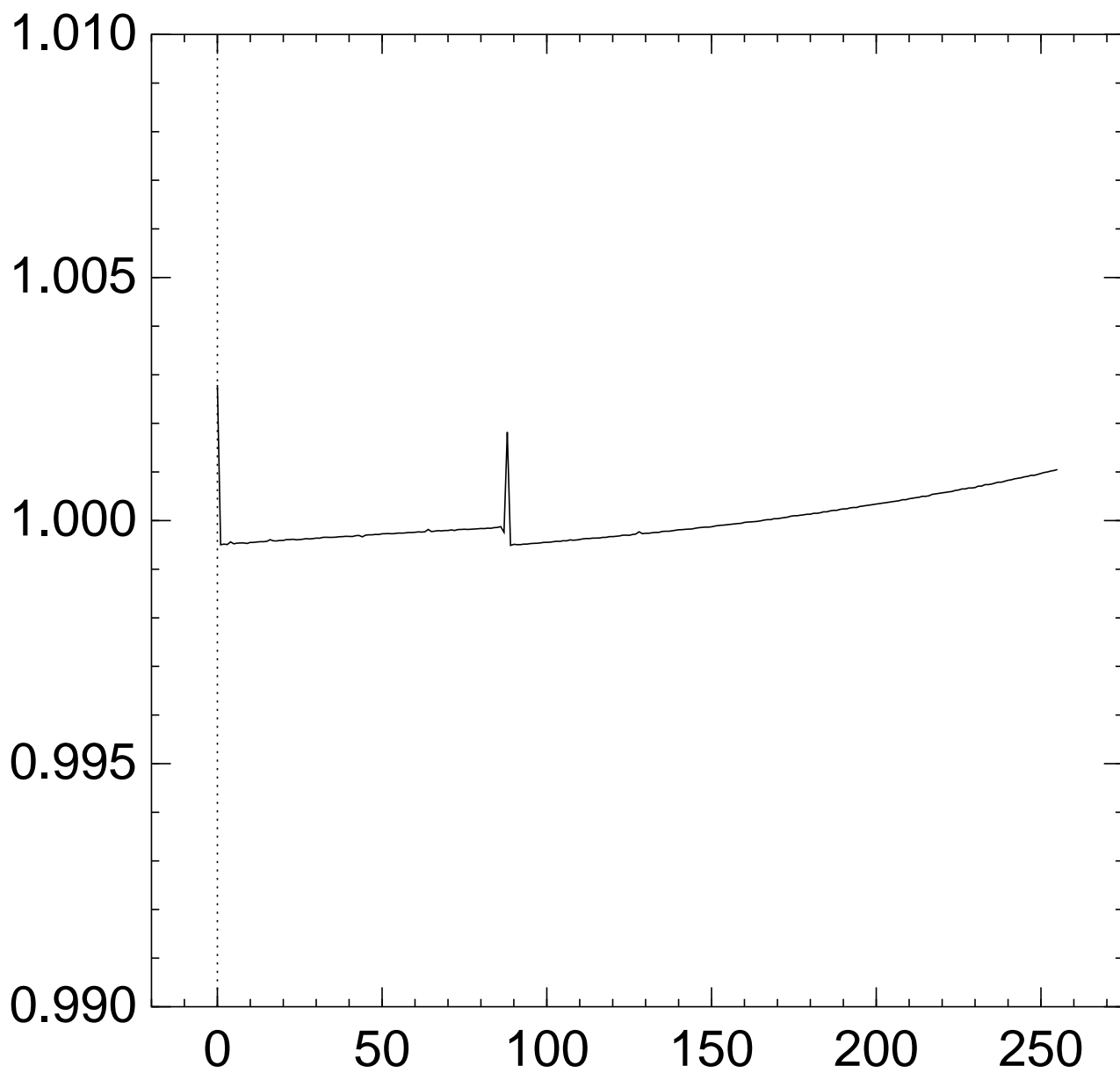
Graph of $256 \Pr[z_{86} = x]$:



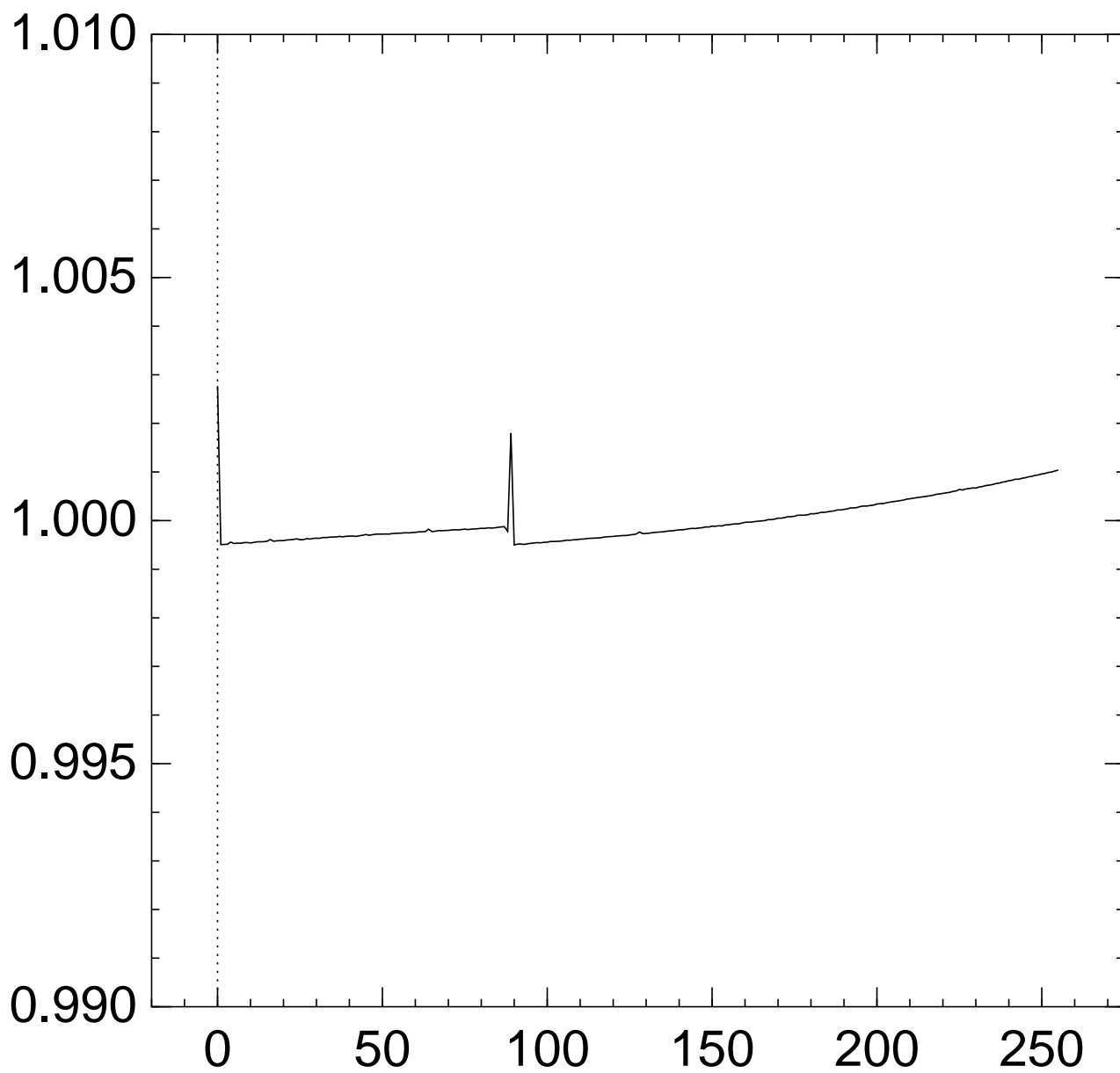
Graph of $256 \Pr[z_{87} = x]$:



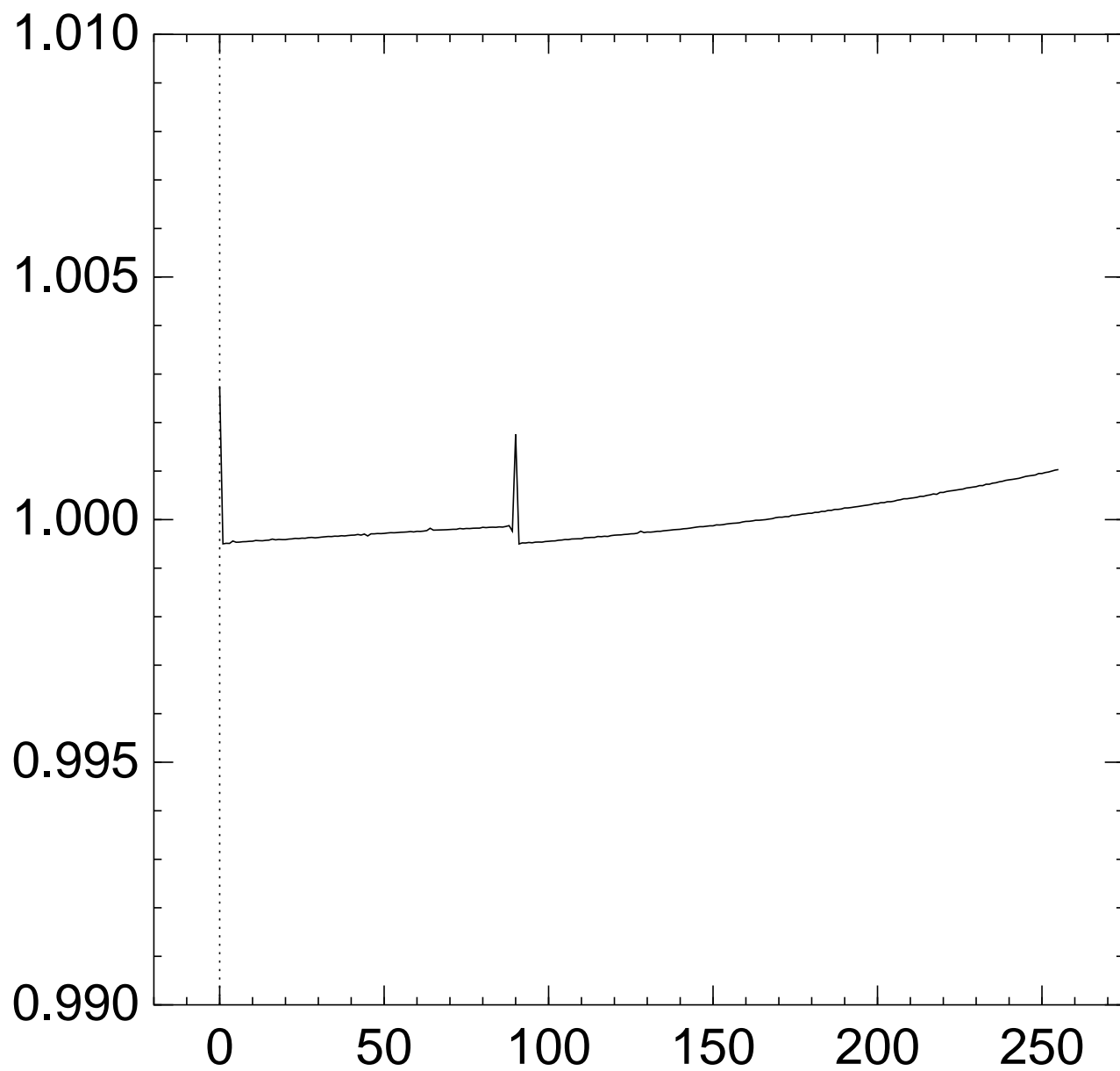
Graph of $256 \Pr[z_{88} = x]$:



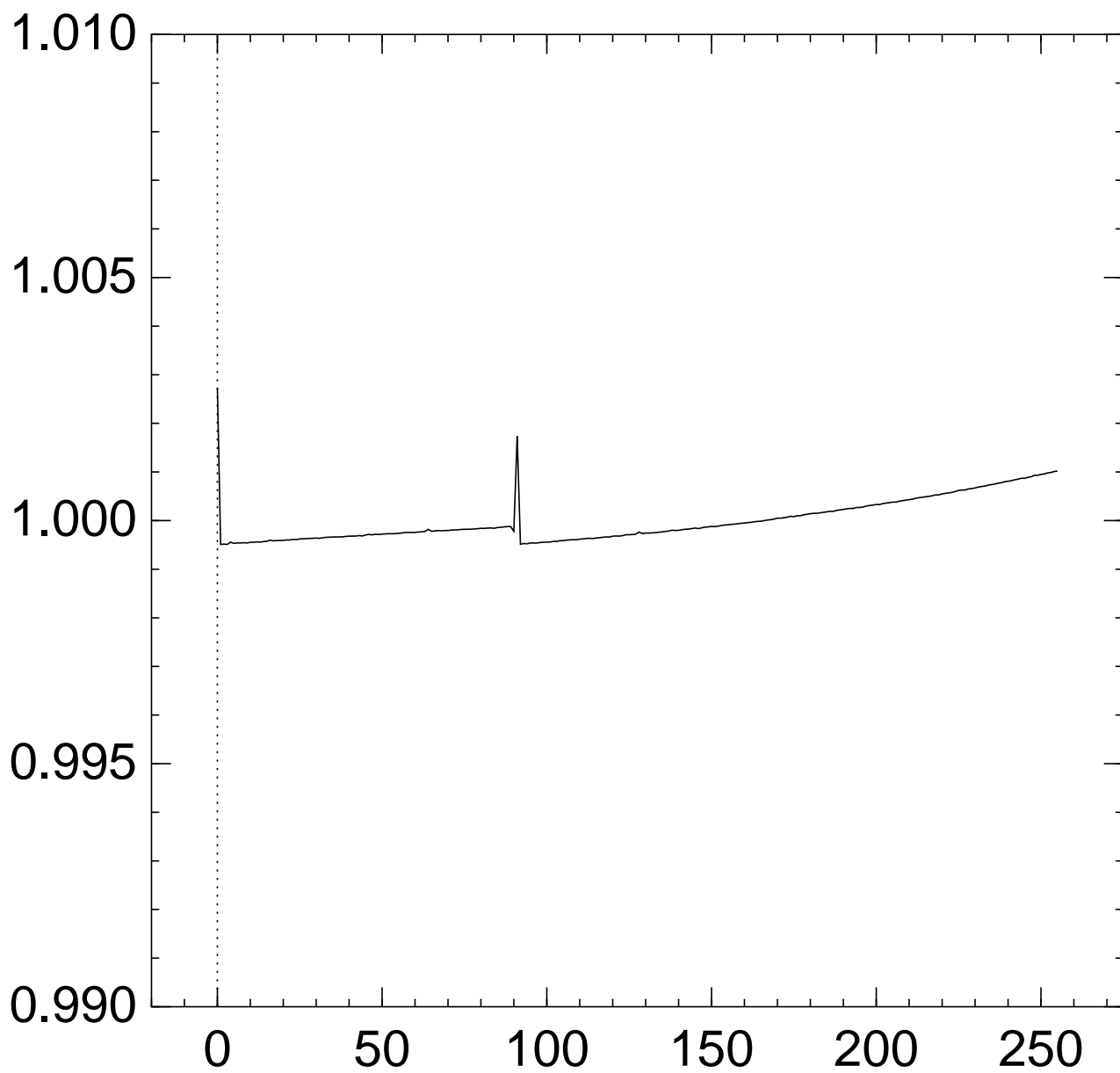
Graph of $256 \Pr[z_{89} = x]$:



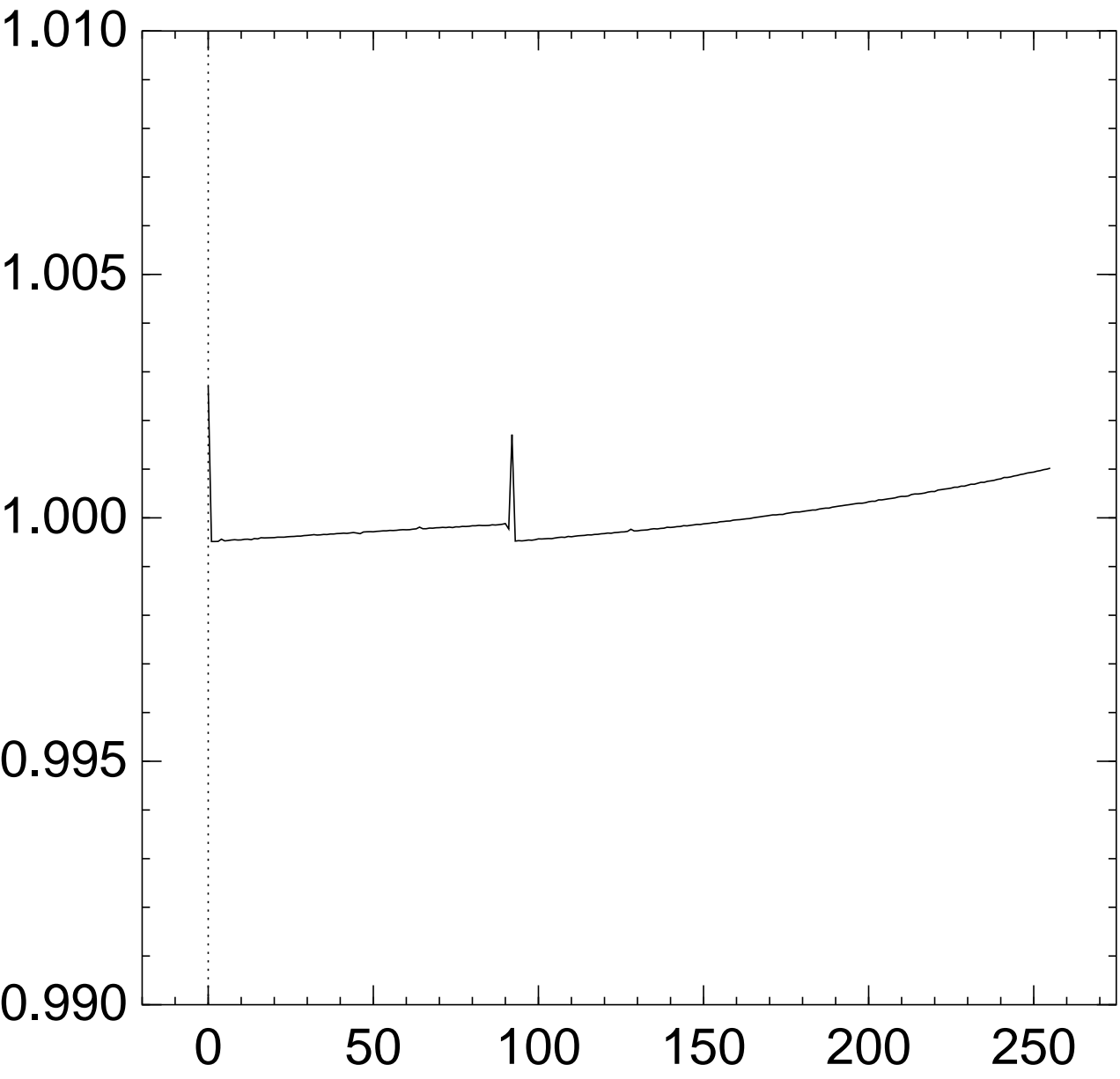
Graph of $256 \Pr[z_{90} = x]$:



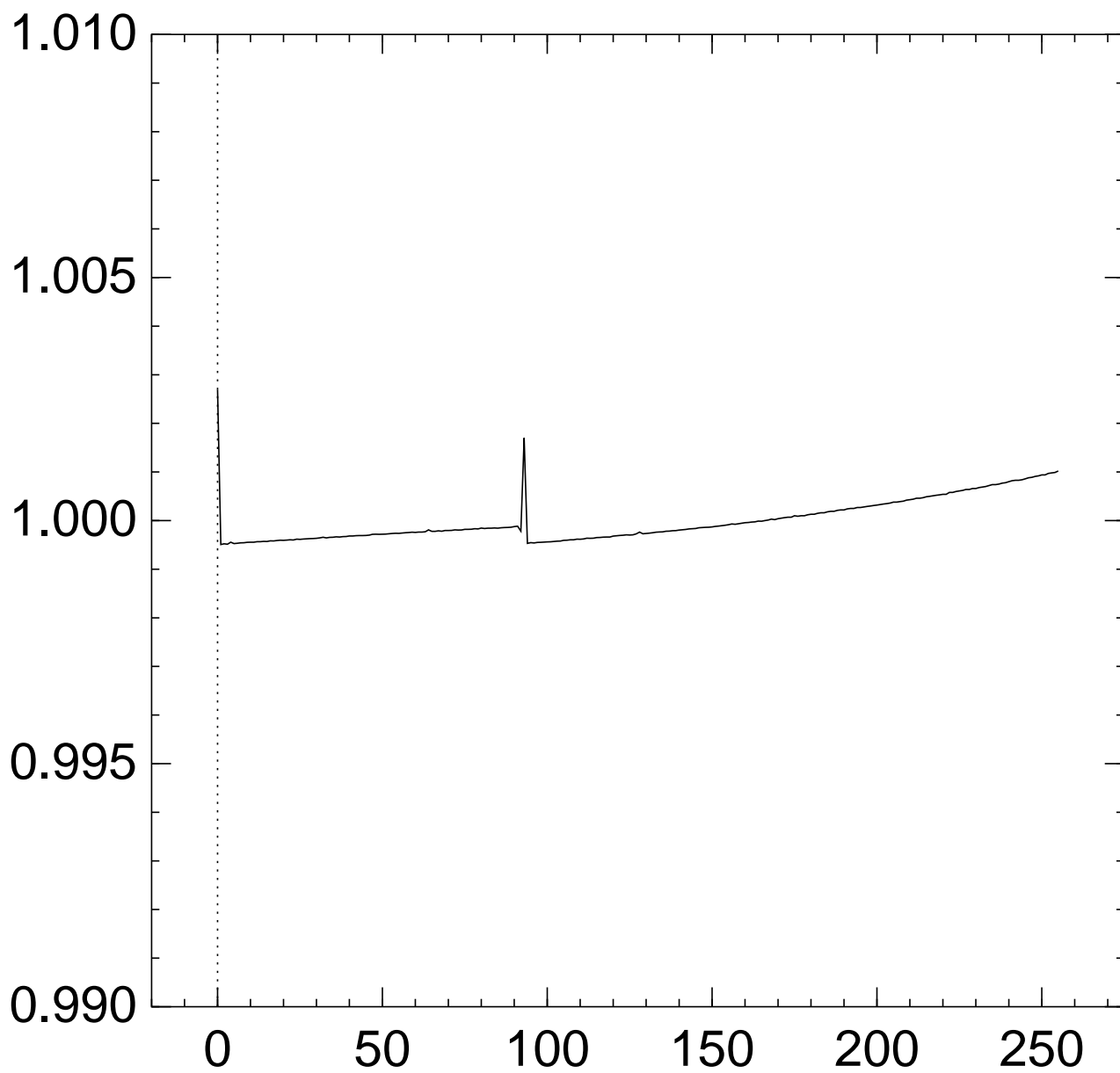
Graph of $256 \Pr[z_{91} = x]$:



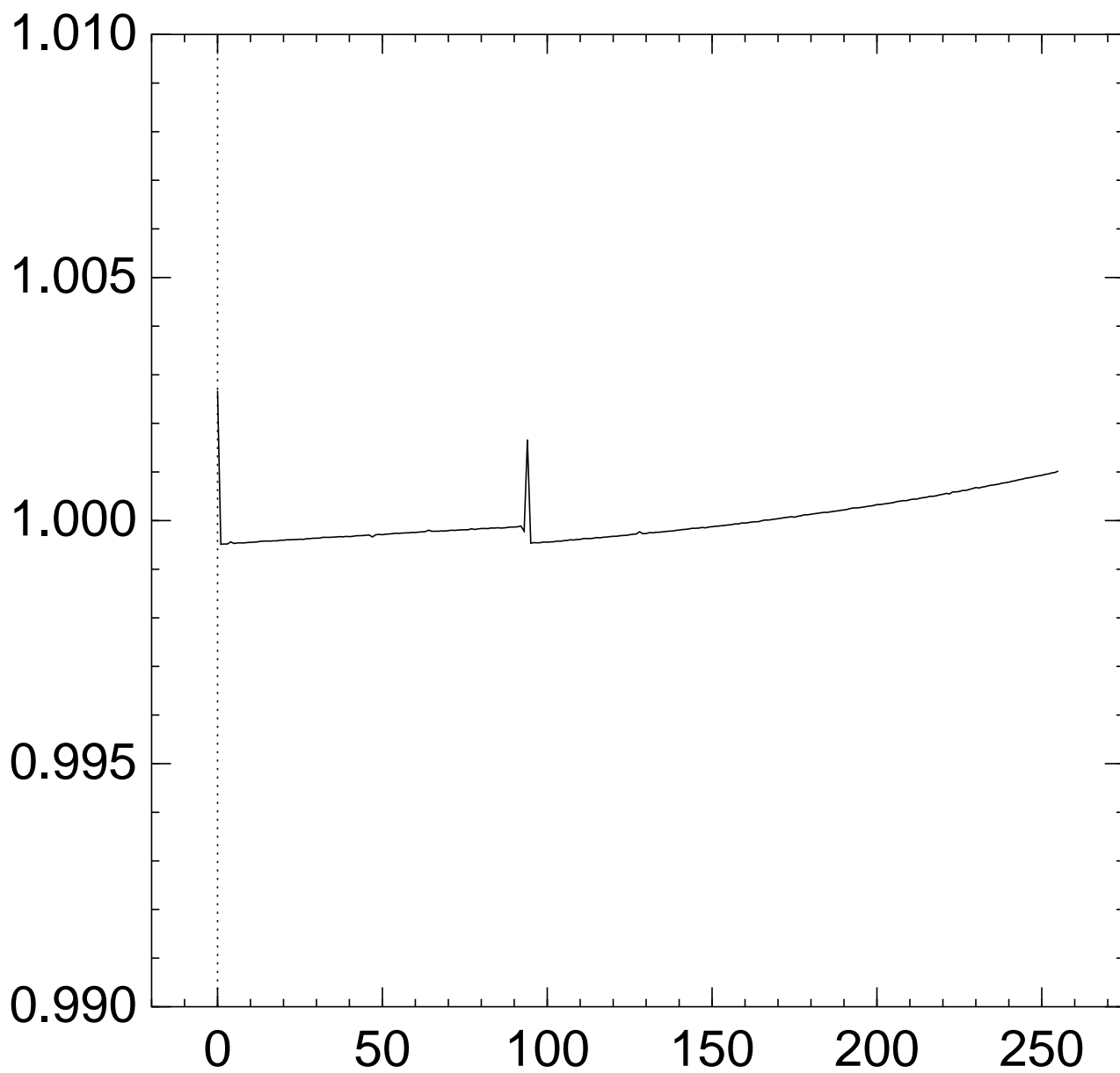
Graph of $256 \Pr[z_{92} = x]$:



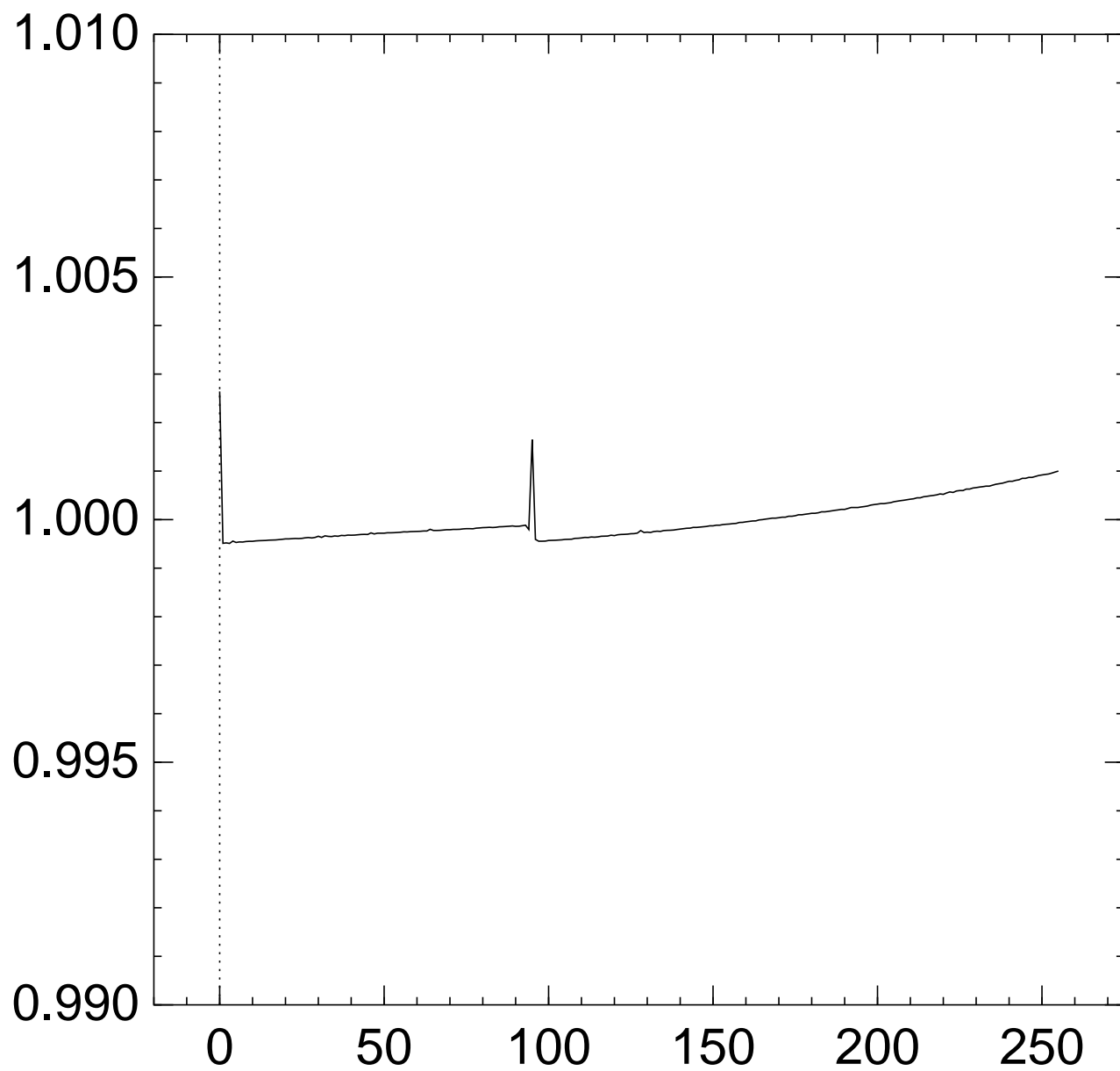
Graph of $256 \Pr[z_{93} = x]$:



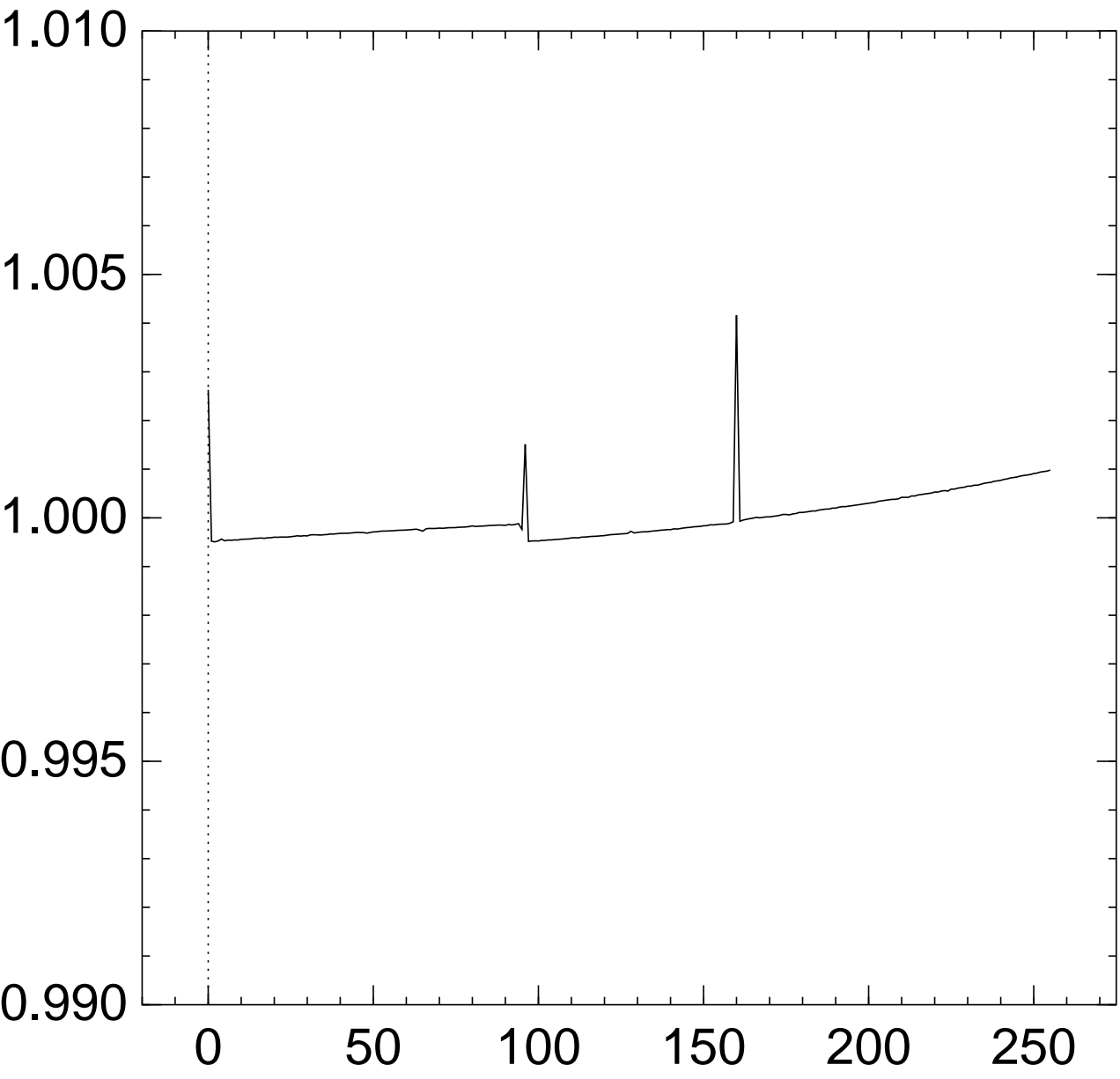
Graph of $256 \Pr[z_{94} = x]$:



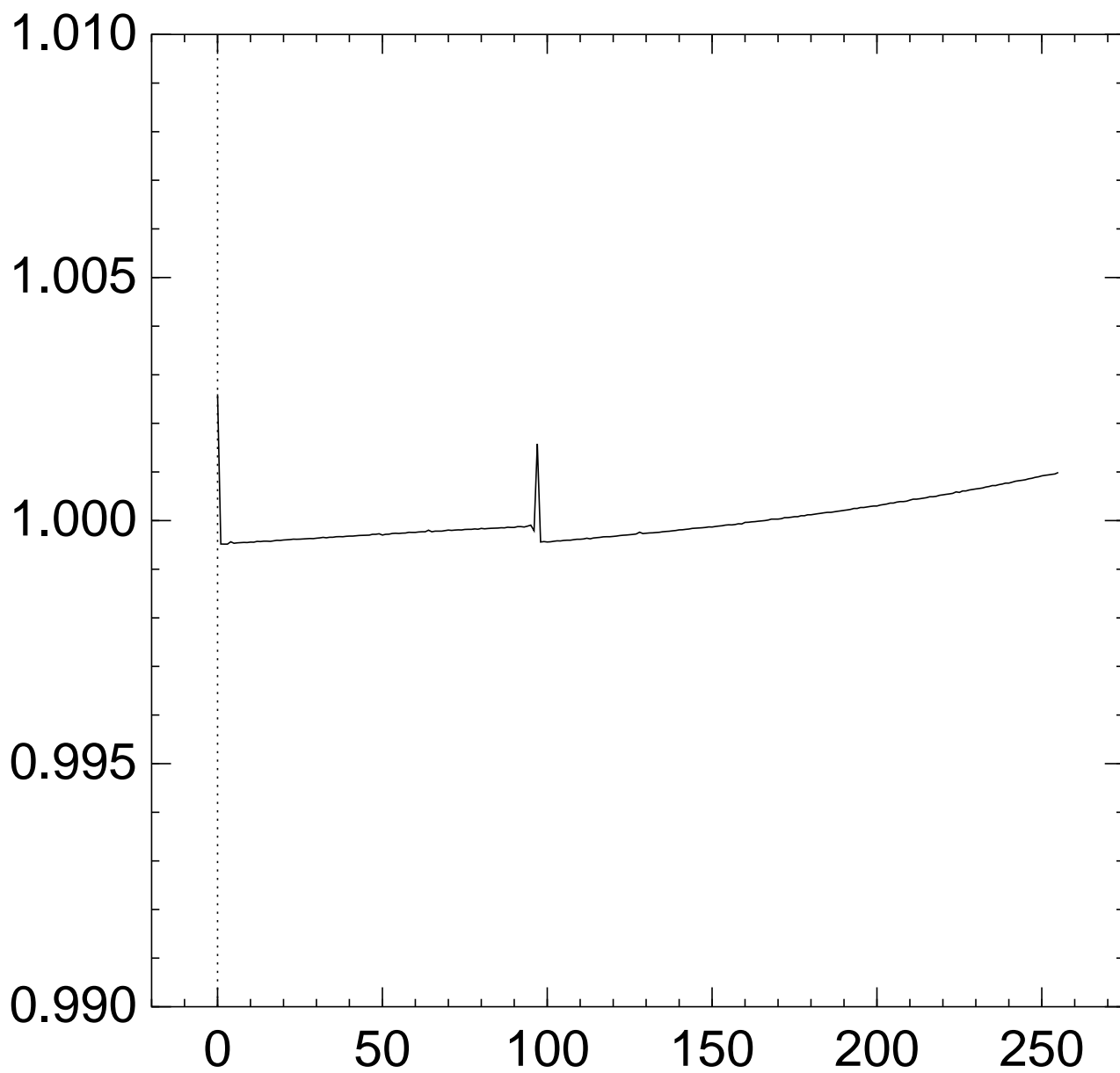
Graph of $256 \Pr[z_{95} = x]$:



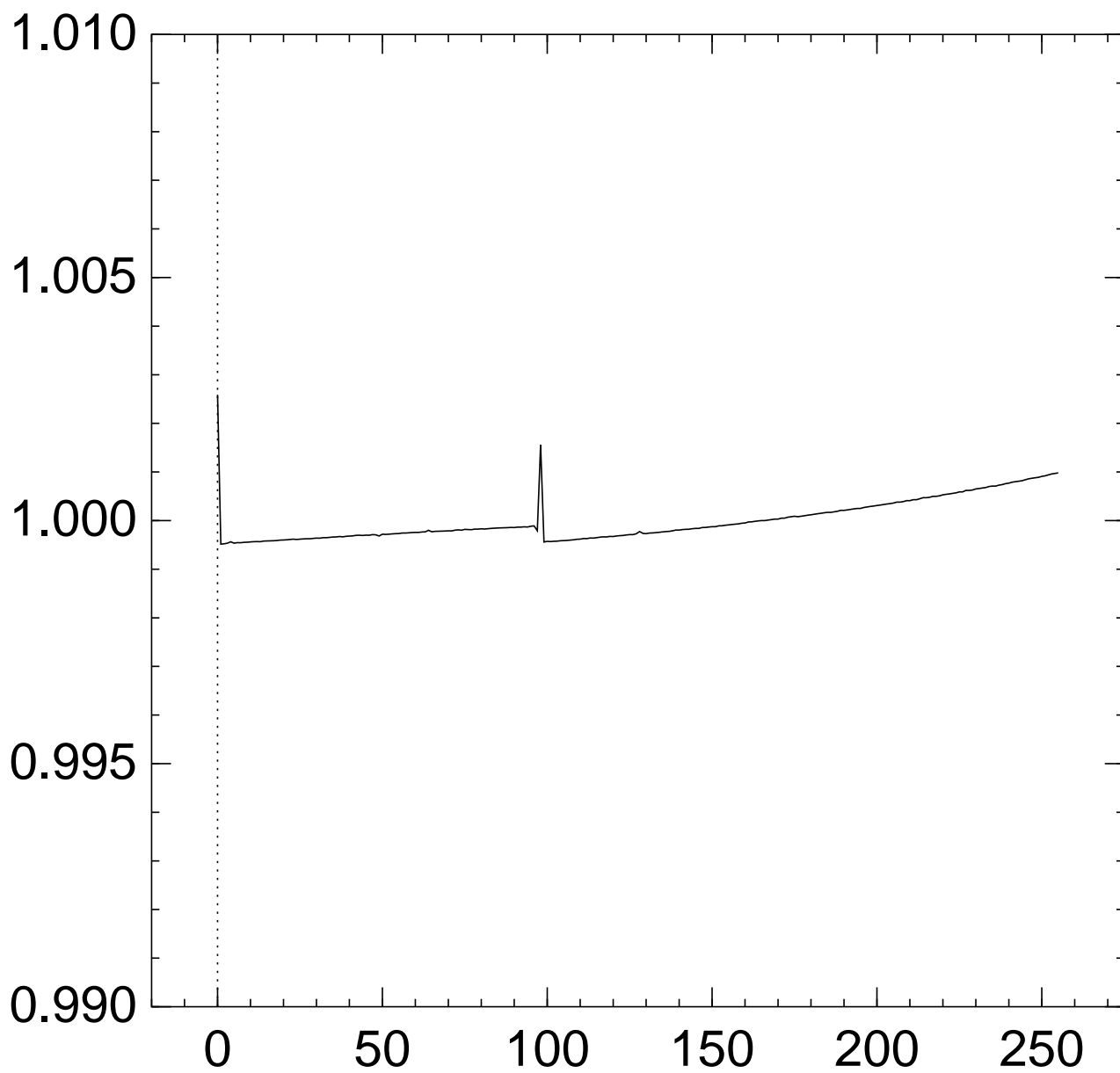
Graph of $256 \Pr[z_{96} = x]$:



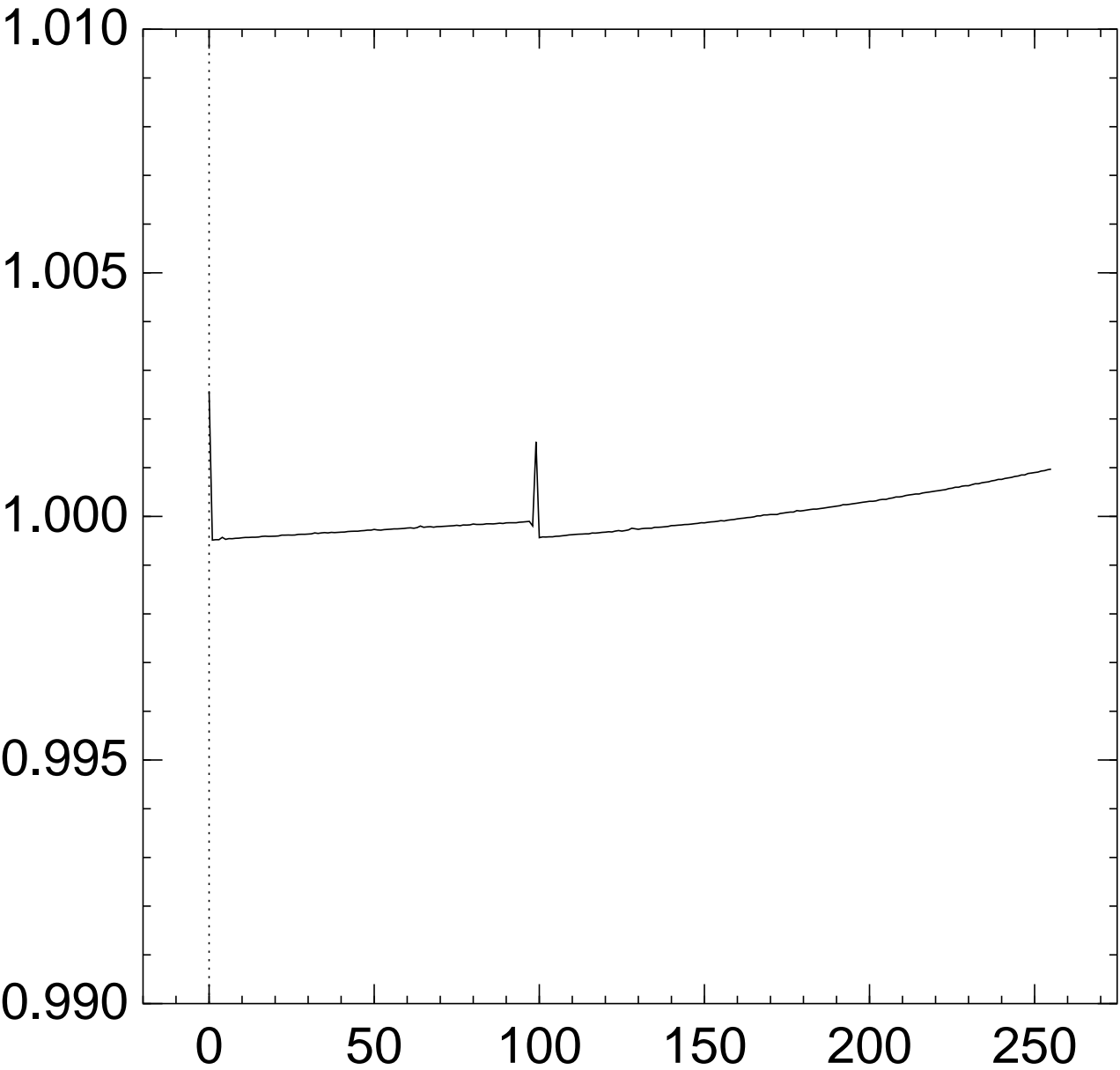
Graph of $256 \Pr[z_{97} = x]$:



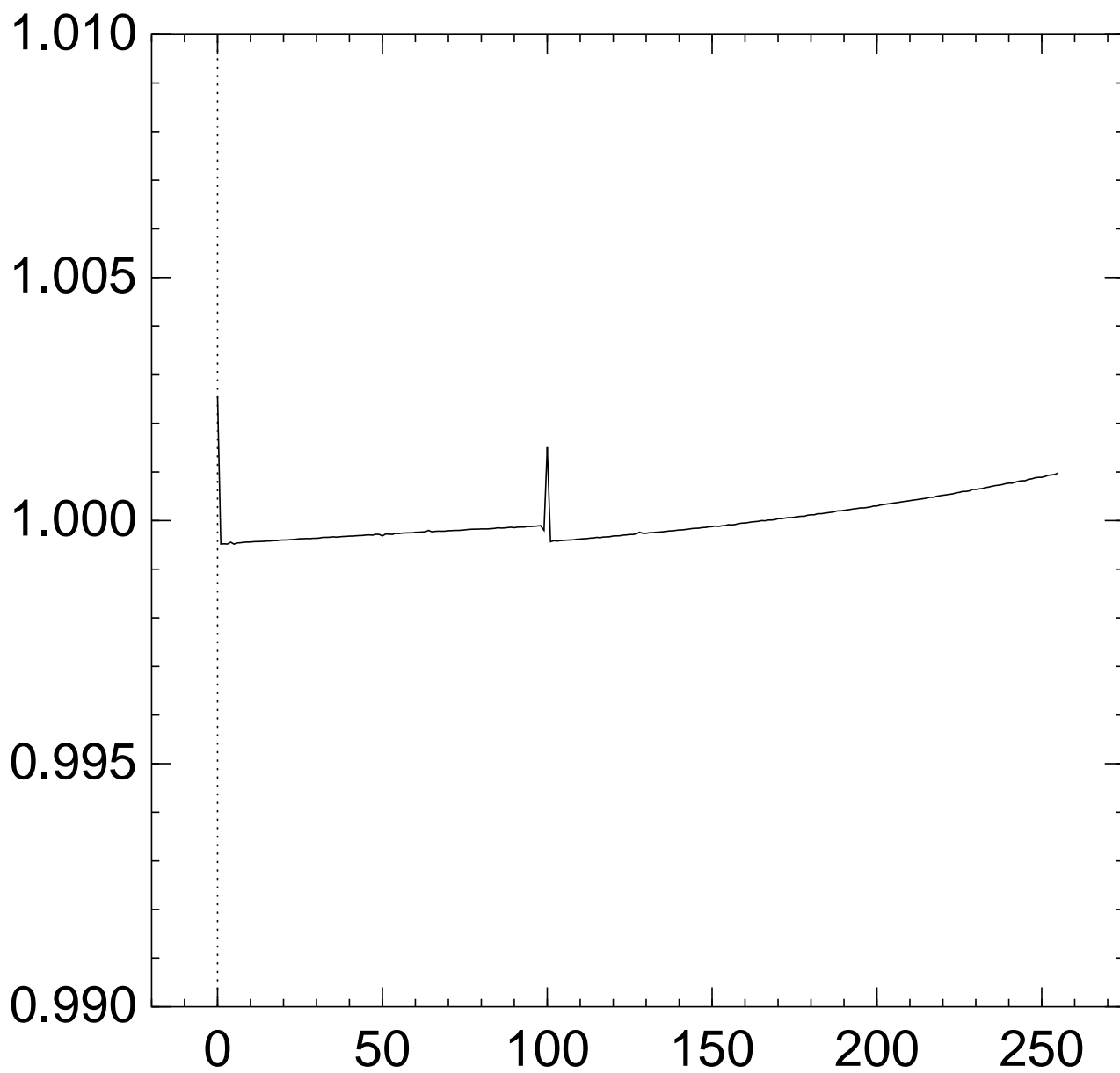
Graph of $256 \Pr[z_{98} = x]$:



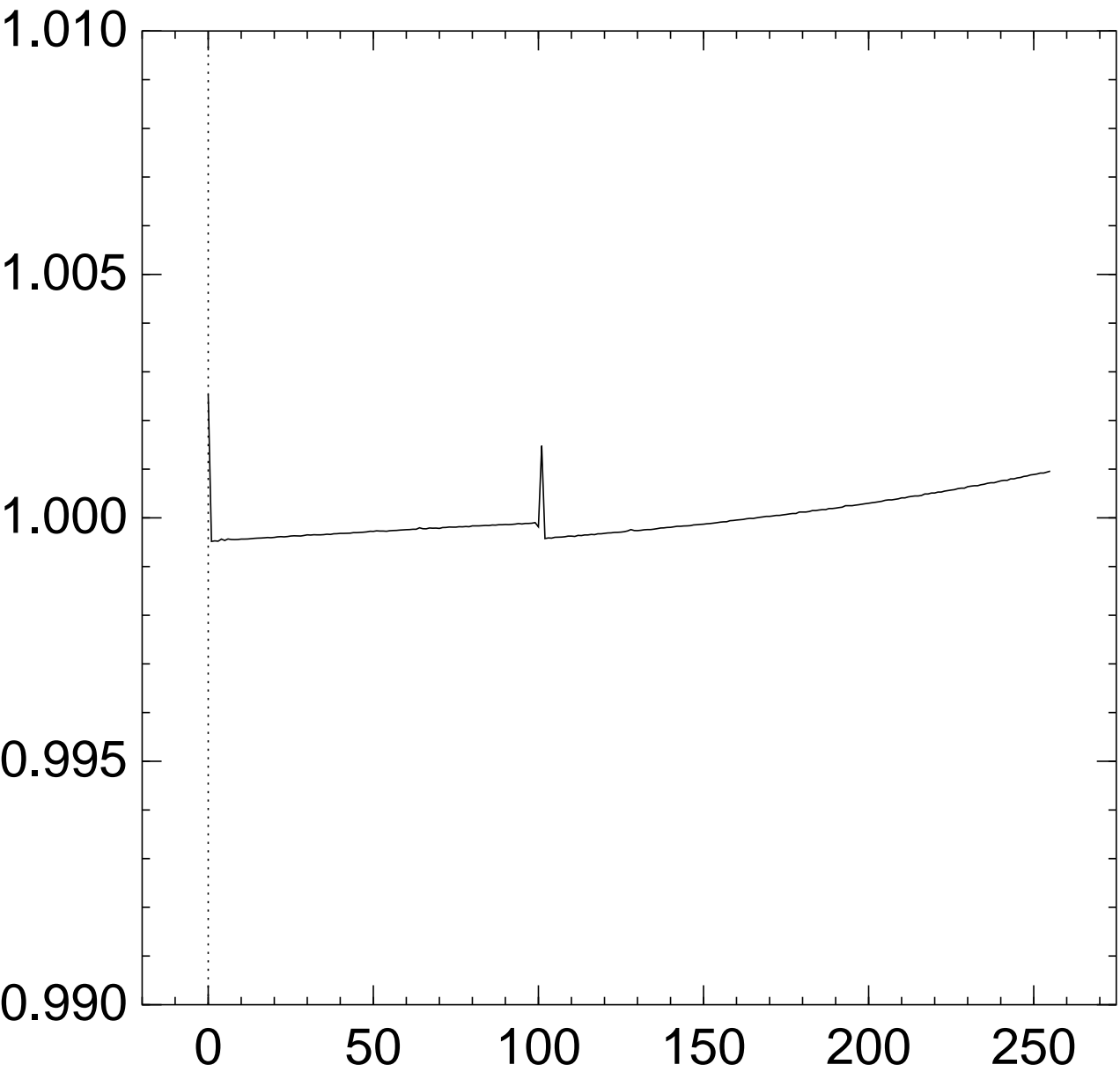
Graph of $256 \Pr[z_{99} = x]$:



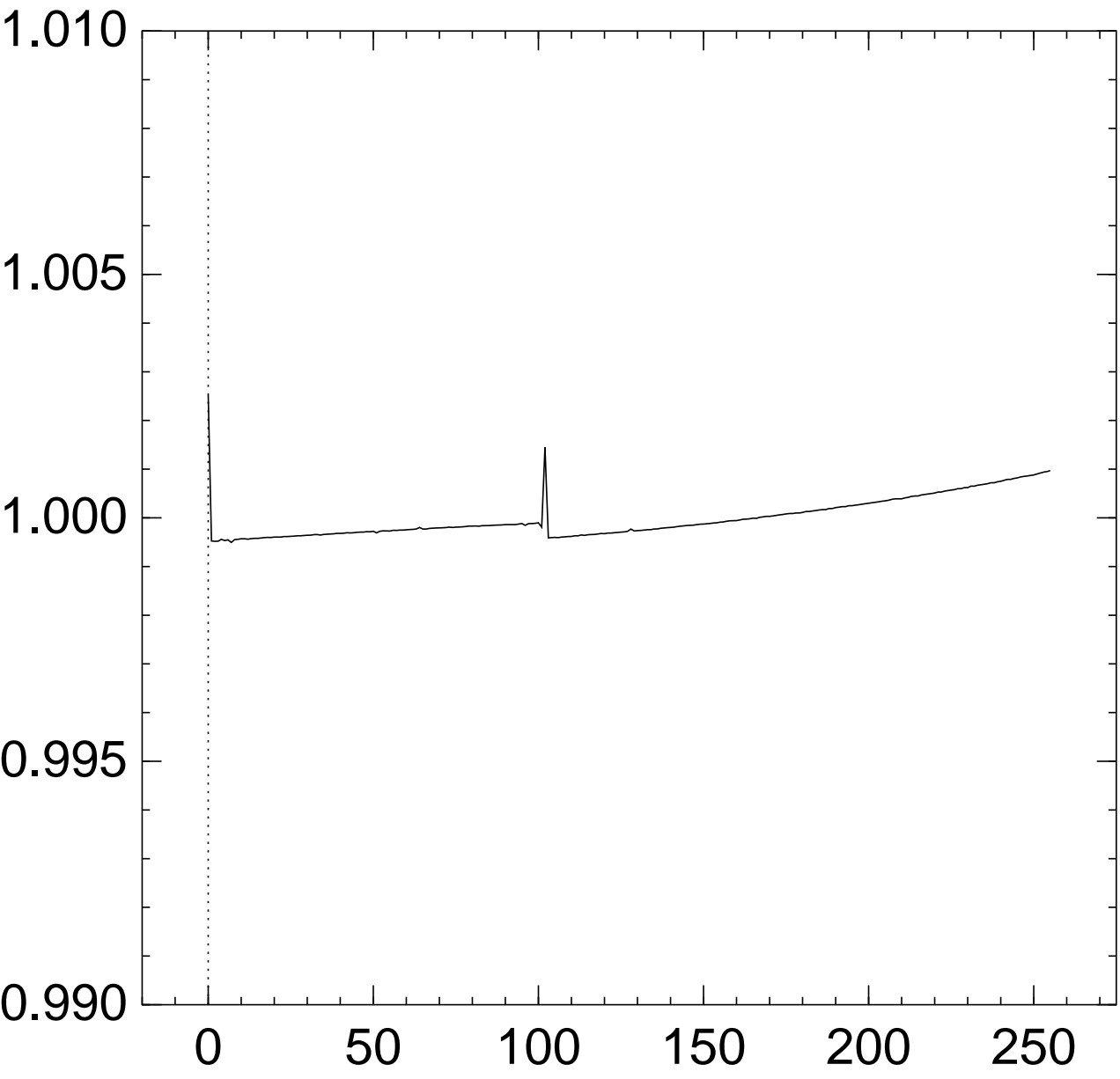
Graph of $256 \Pr[z_{100} = x]$:



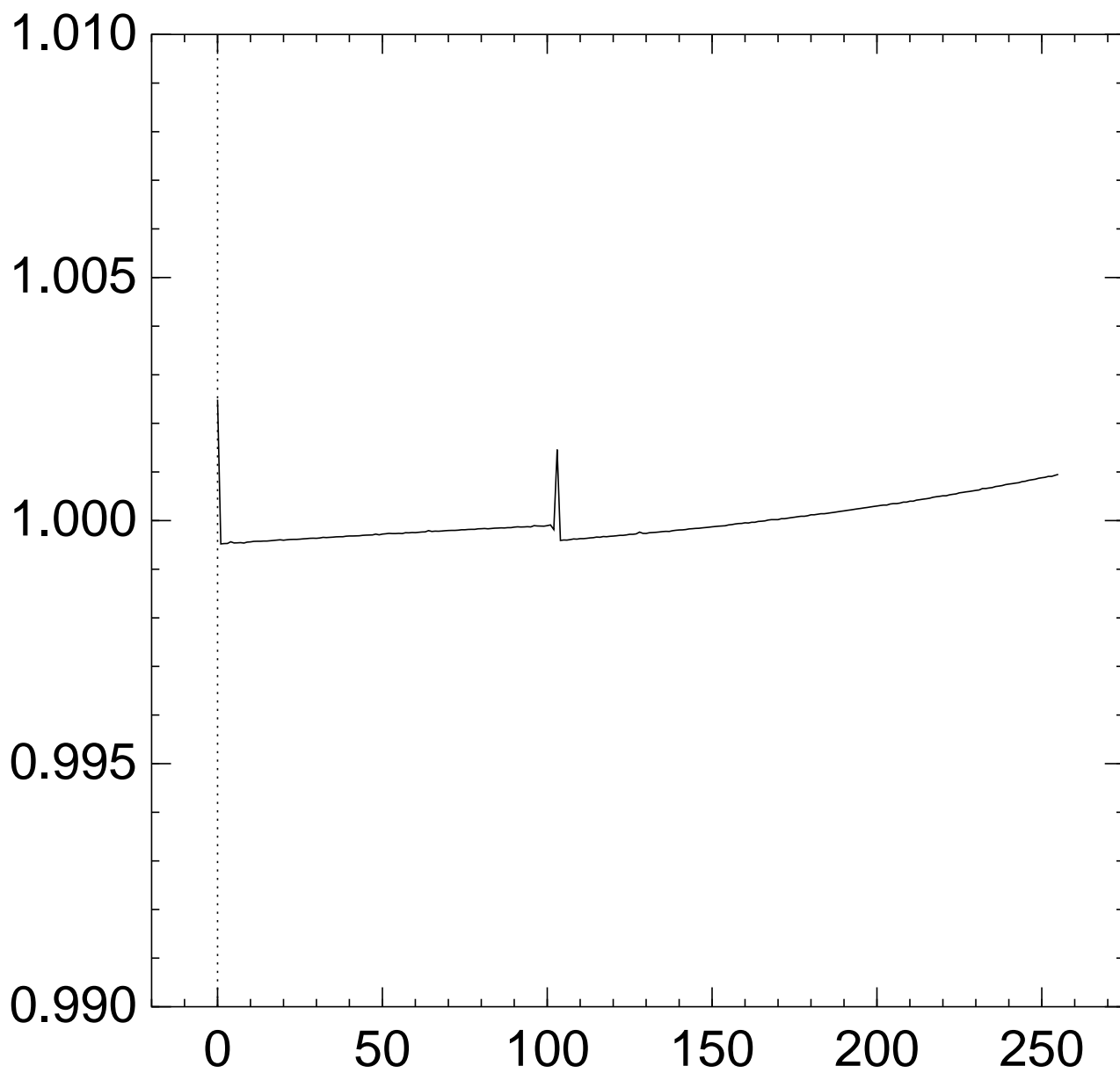
Graph of $256 \Pr[z_{101} = x]$:



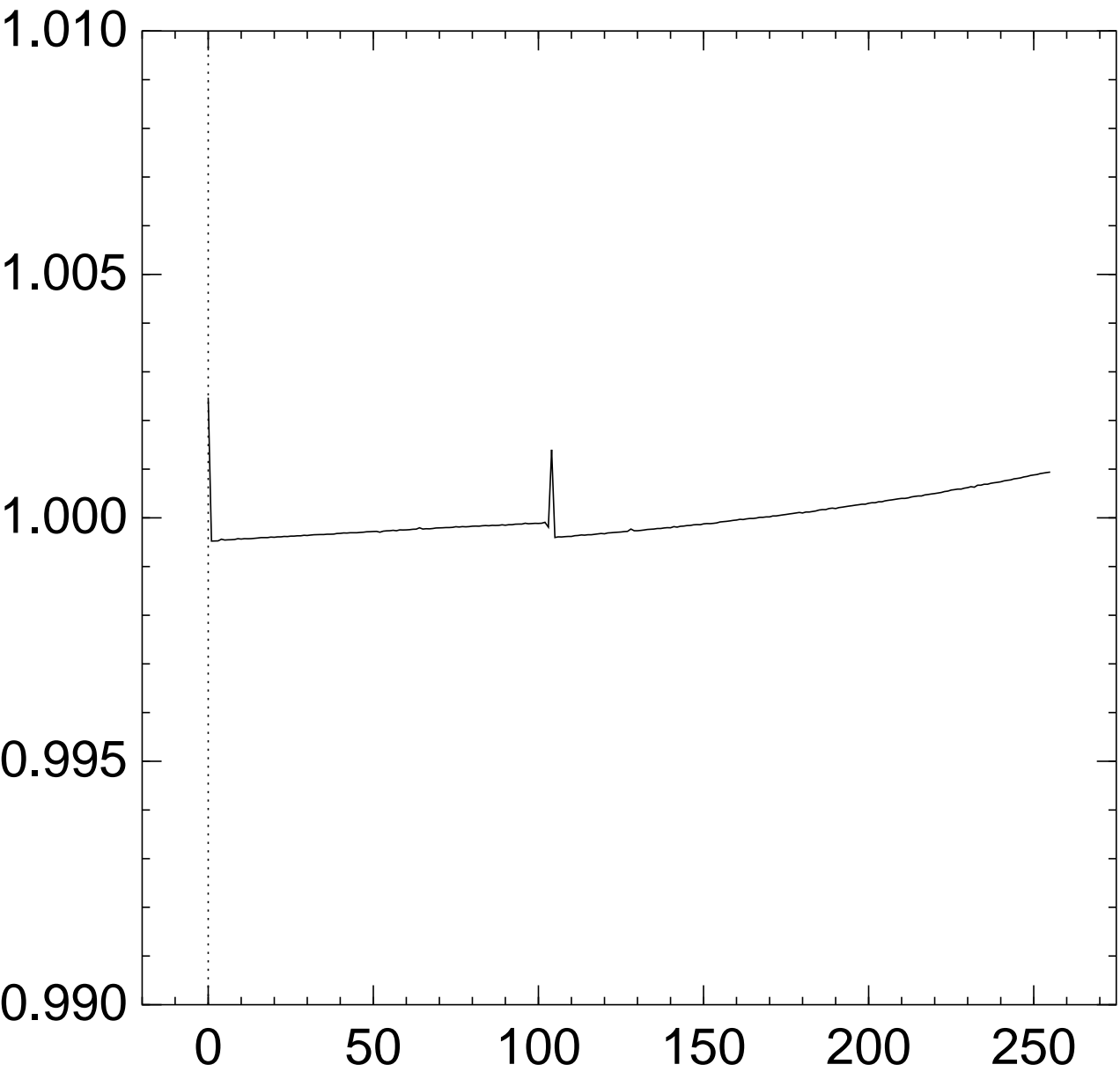
Graph of $256 \Pr[z_{102} = x]$:



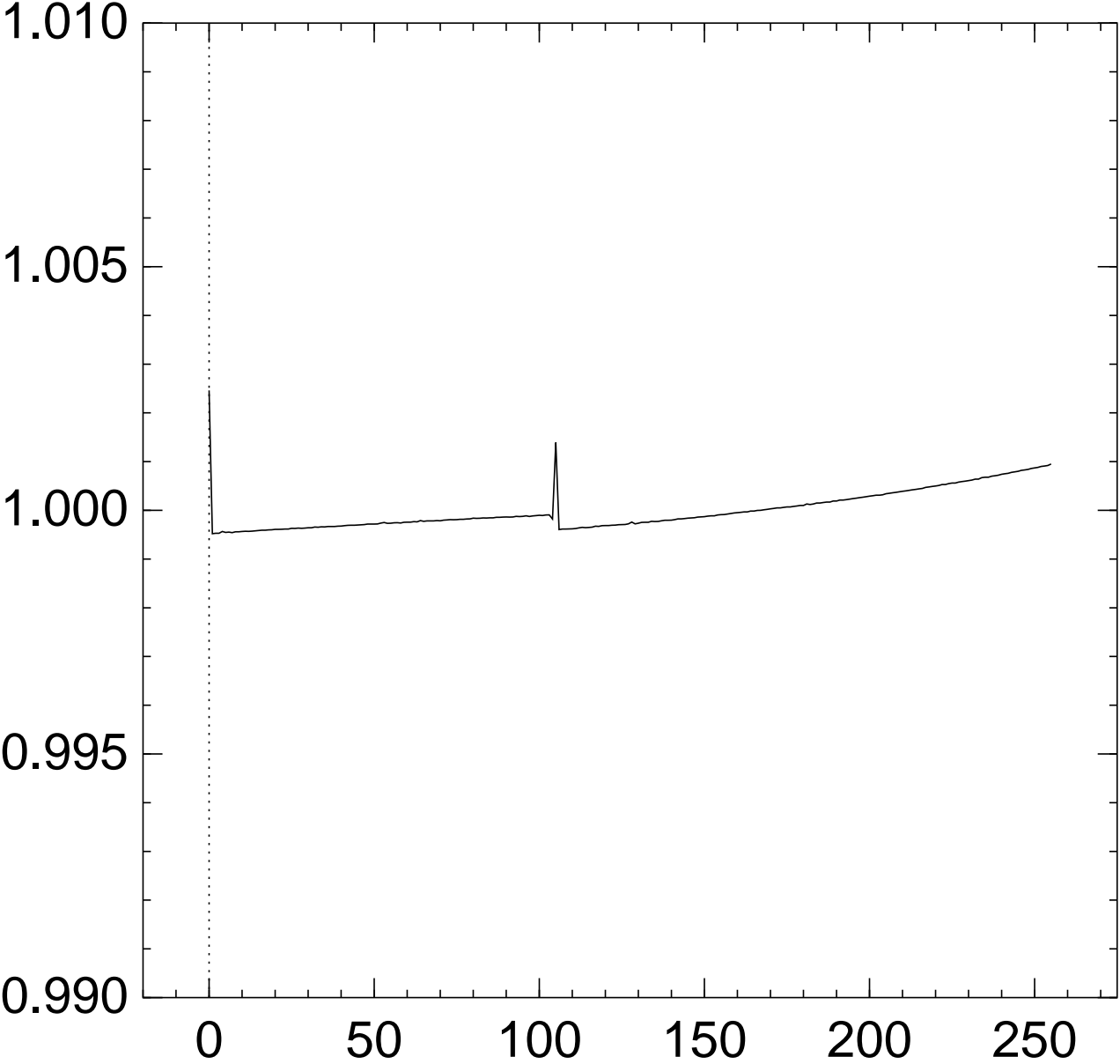
Graph of $256 \Pr[z_{103} = x]$:



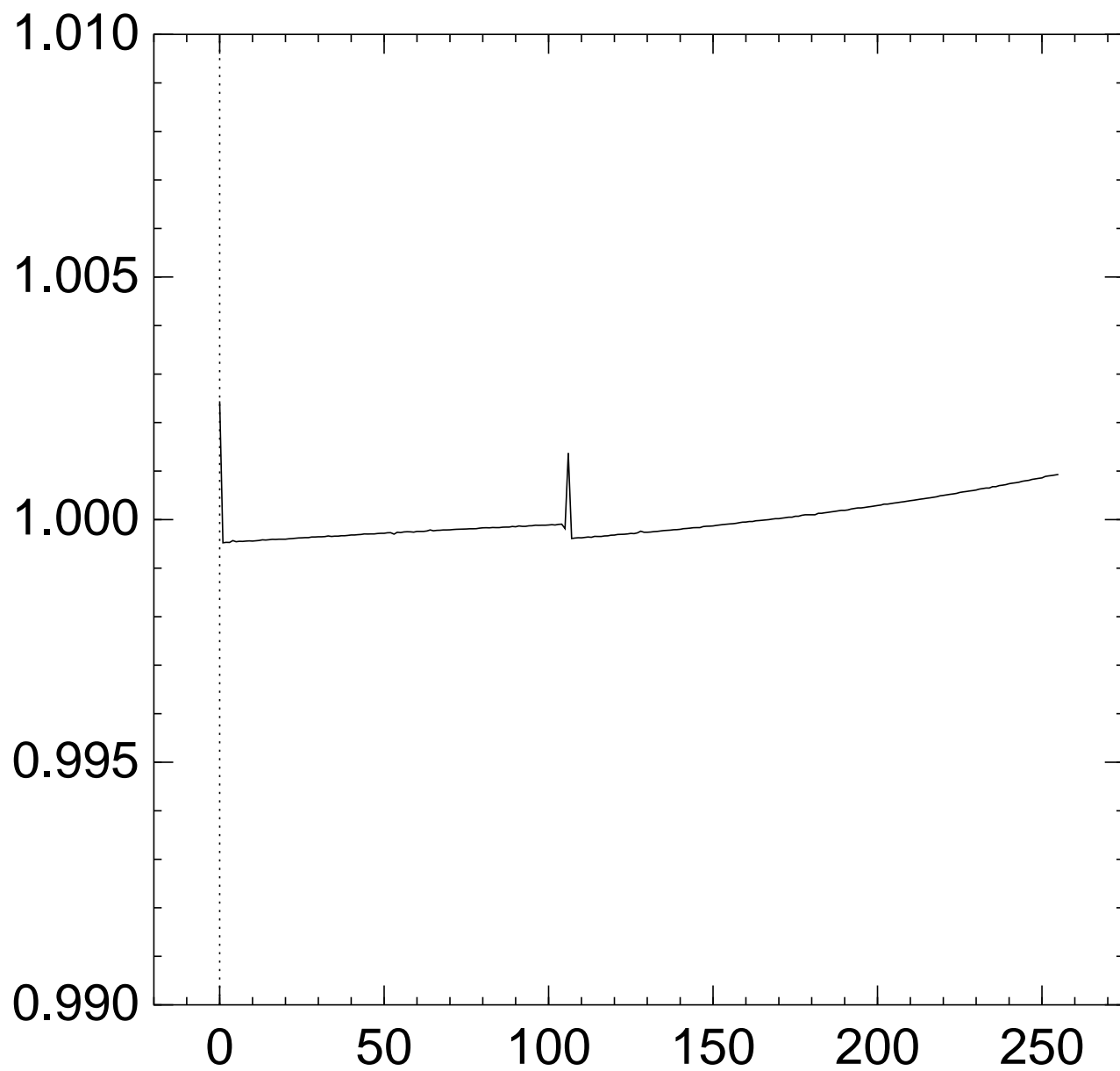
Graph of $256 \Pr[z_{104} = x]$:



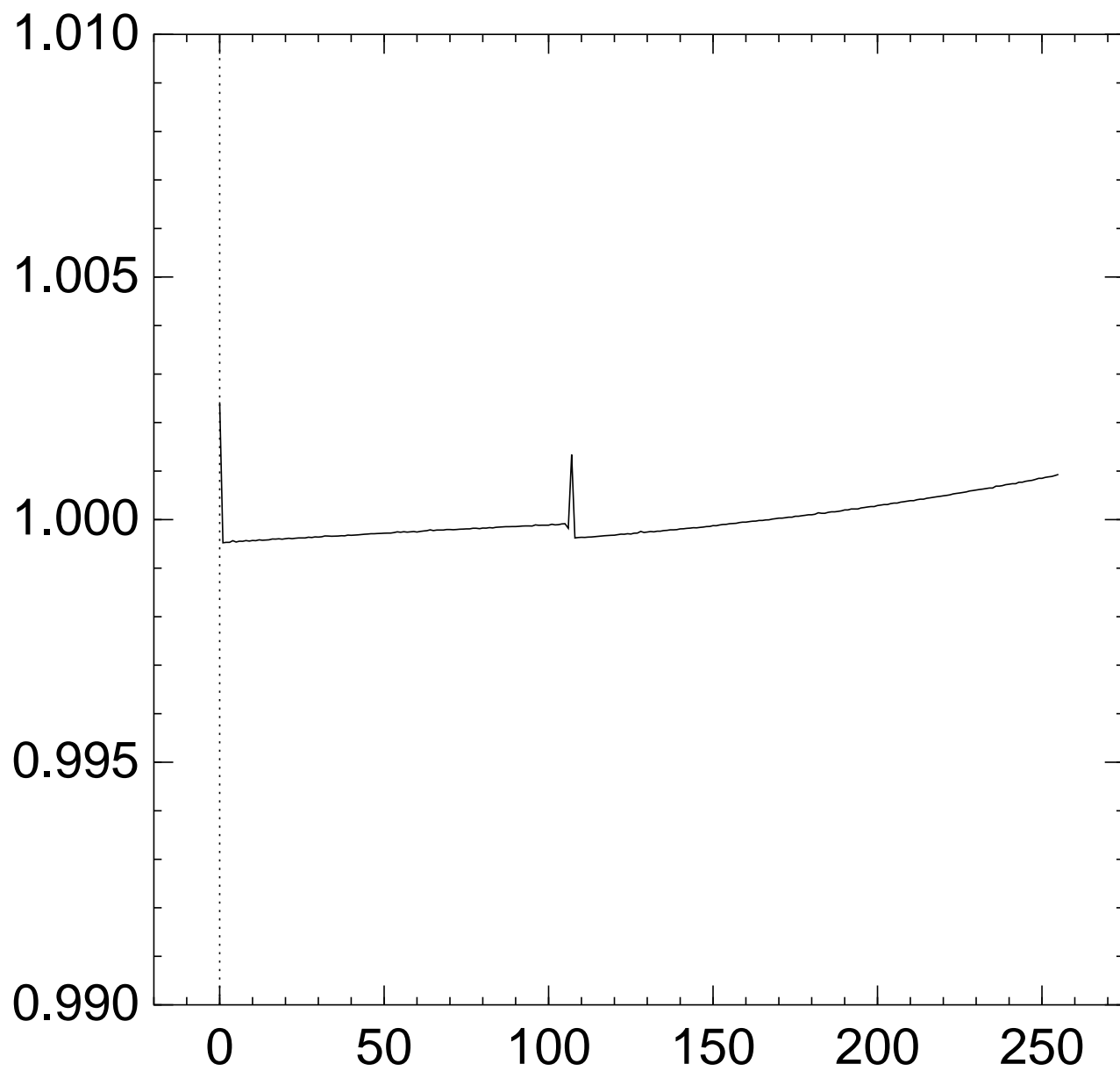
Graph of $256 \Pr[z_{105} = x]$:



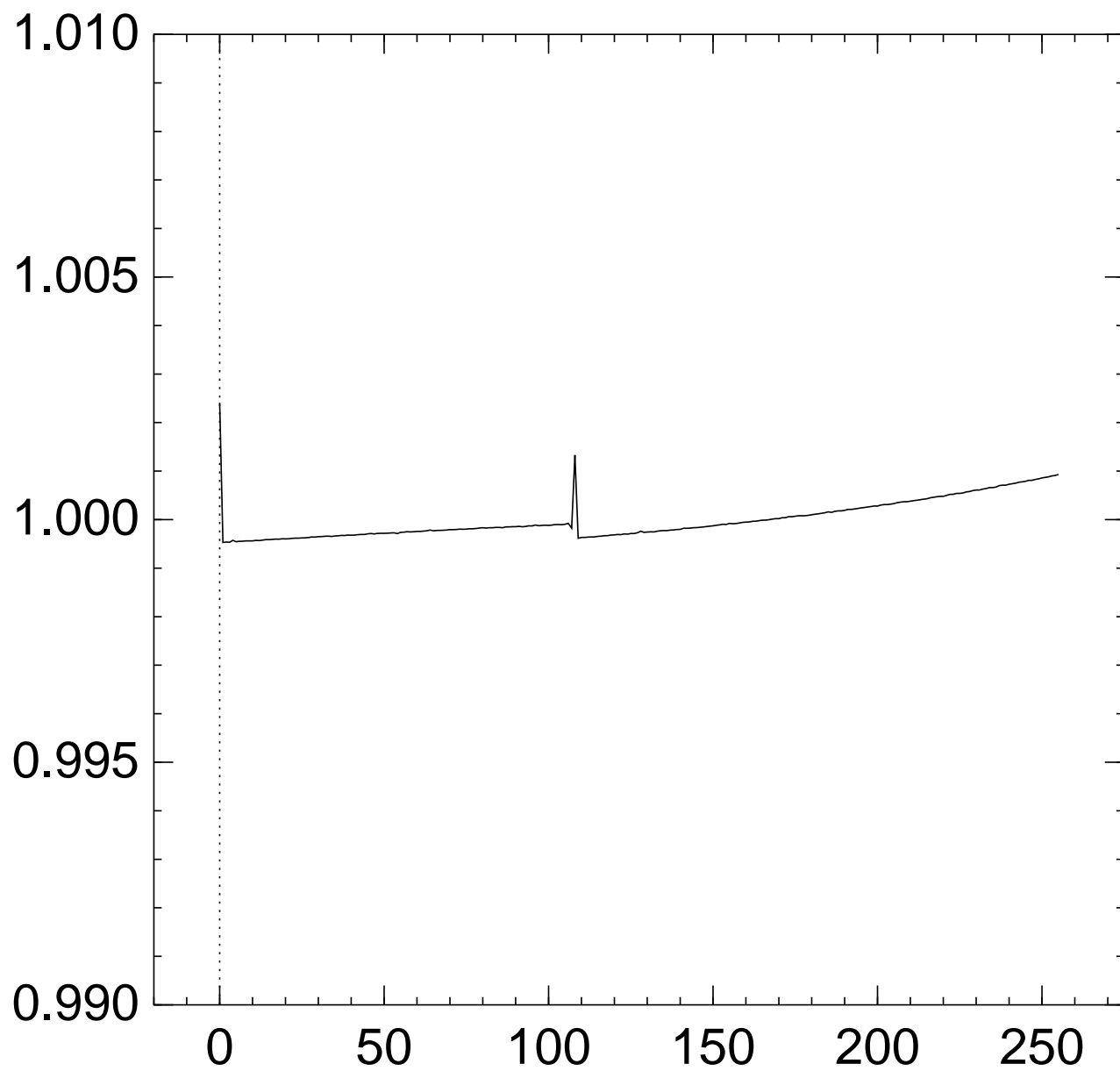
Graph of $256 \Pr[z_{106} = x]$:



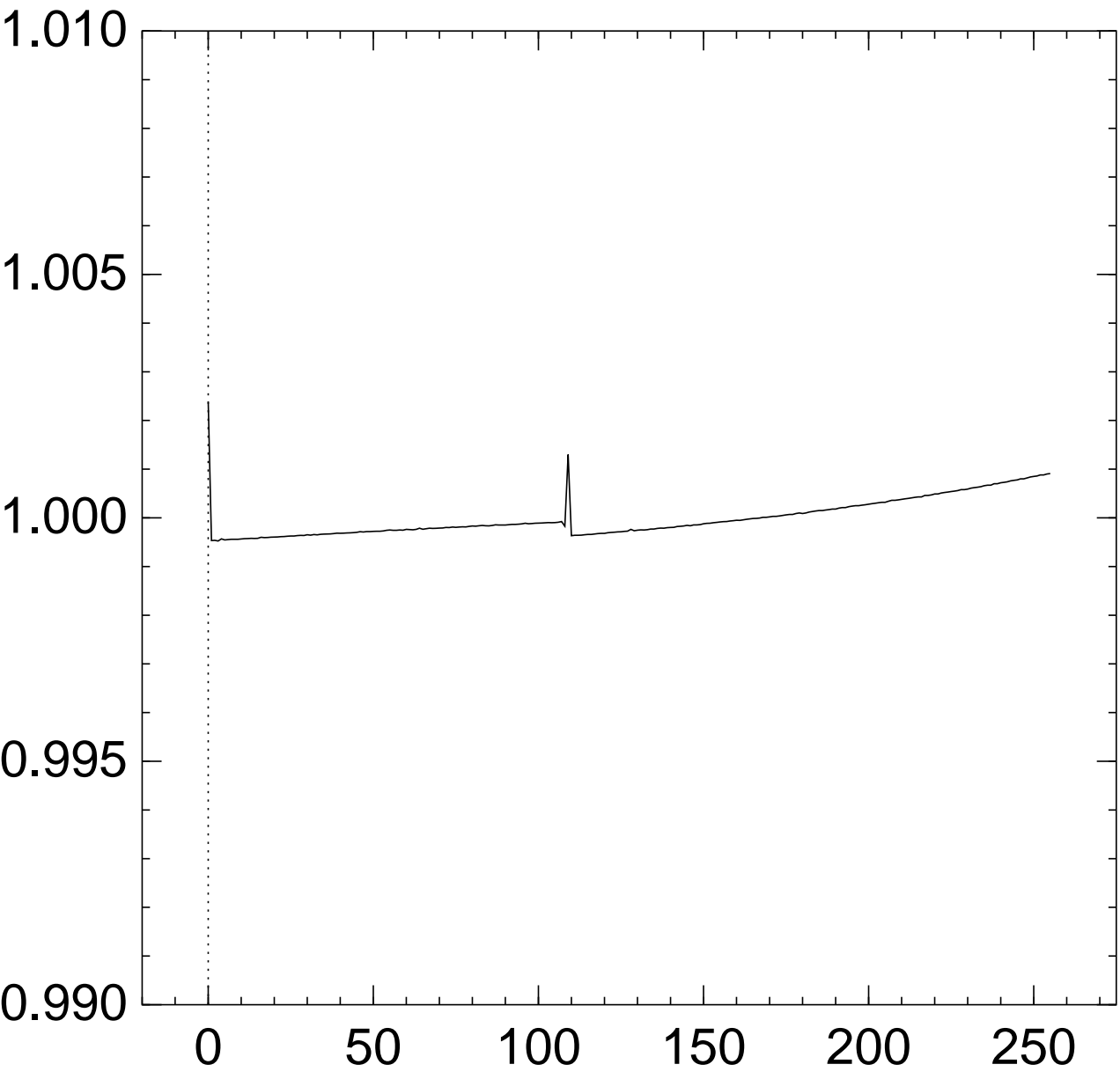
Graph of $256 \Pr[z_{107} = x]$:



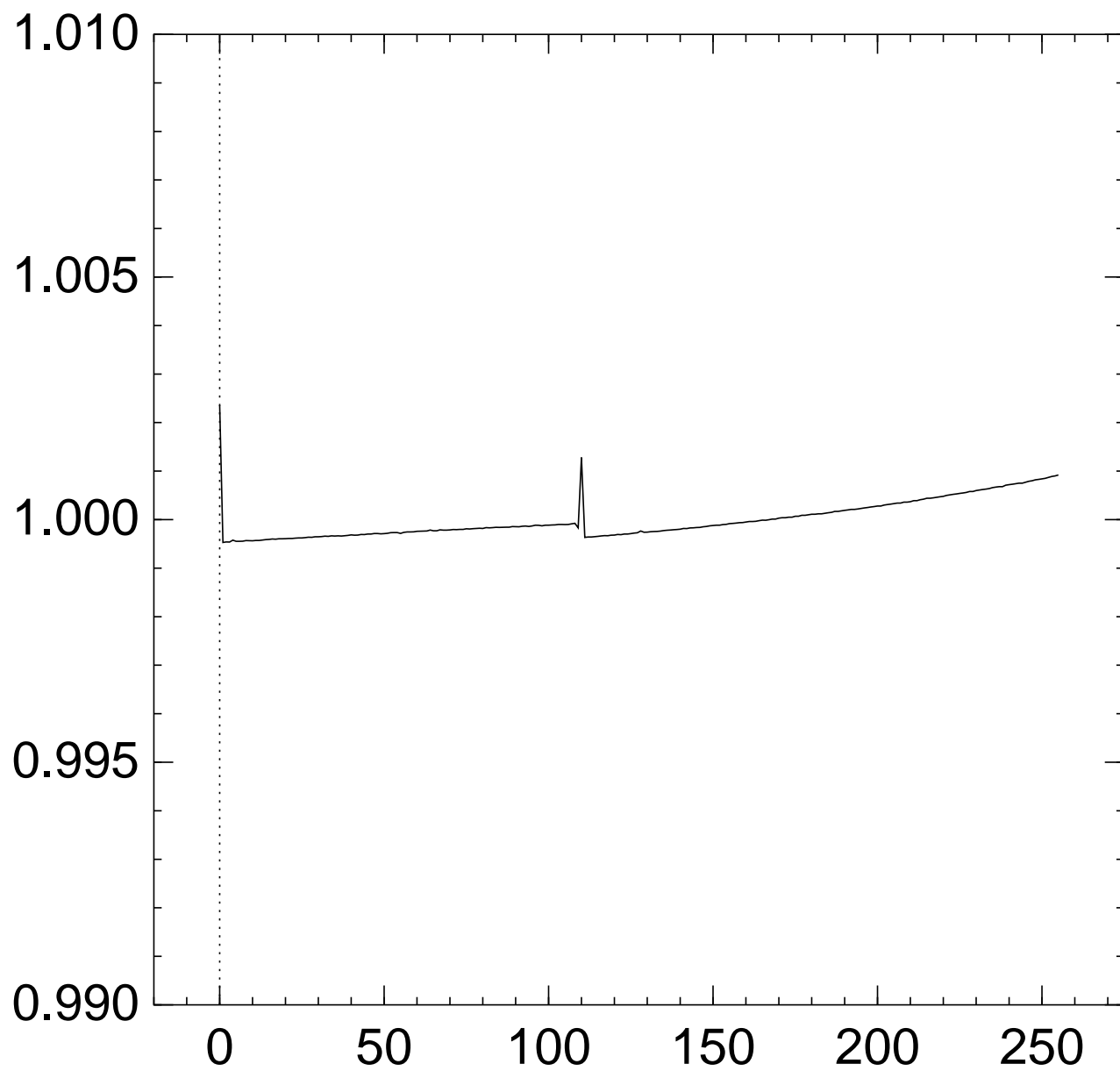
Graph of $256 \Pr[z_{108} = x]$:



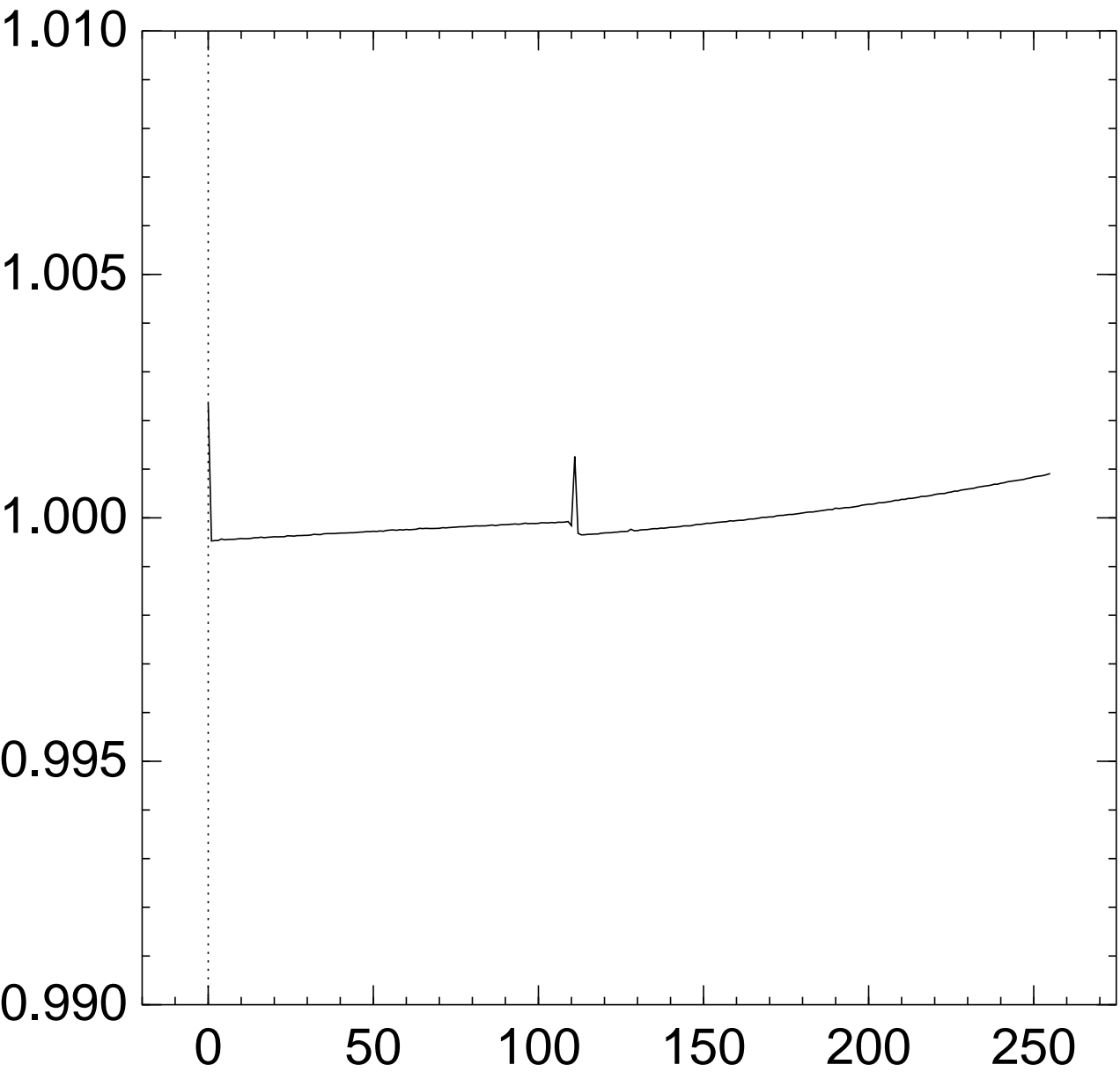
Graph of $256 \Pr[z_{109} = x]$:



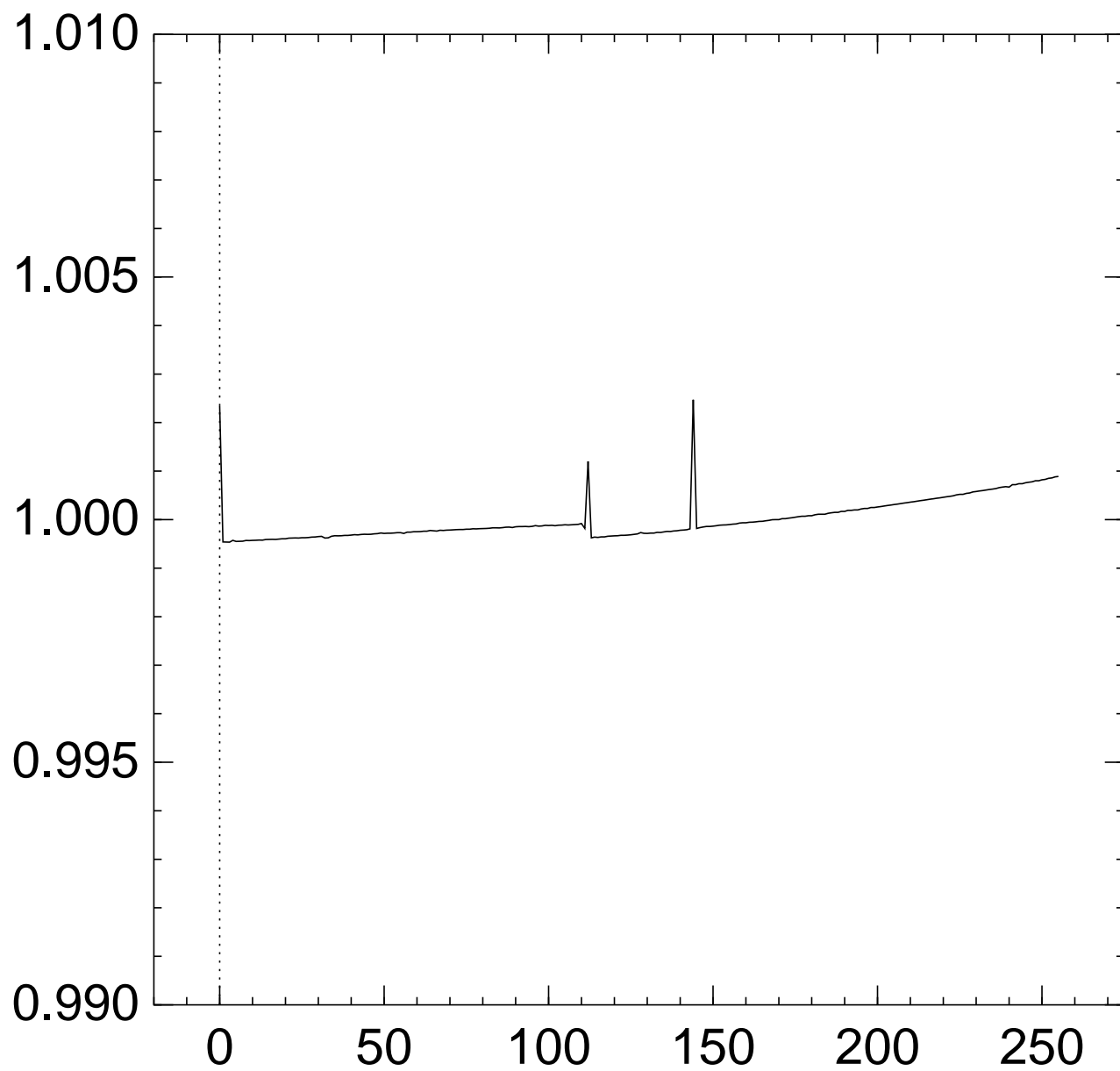
Graph of $256 \Pr[z_{110} = x]$:



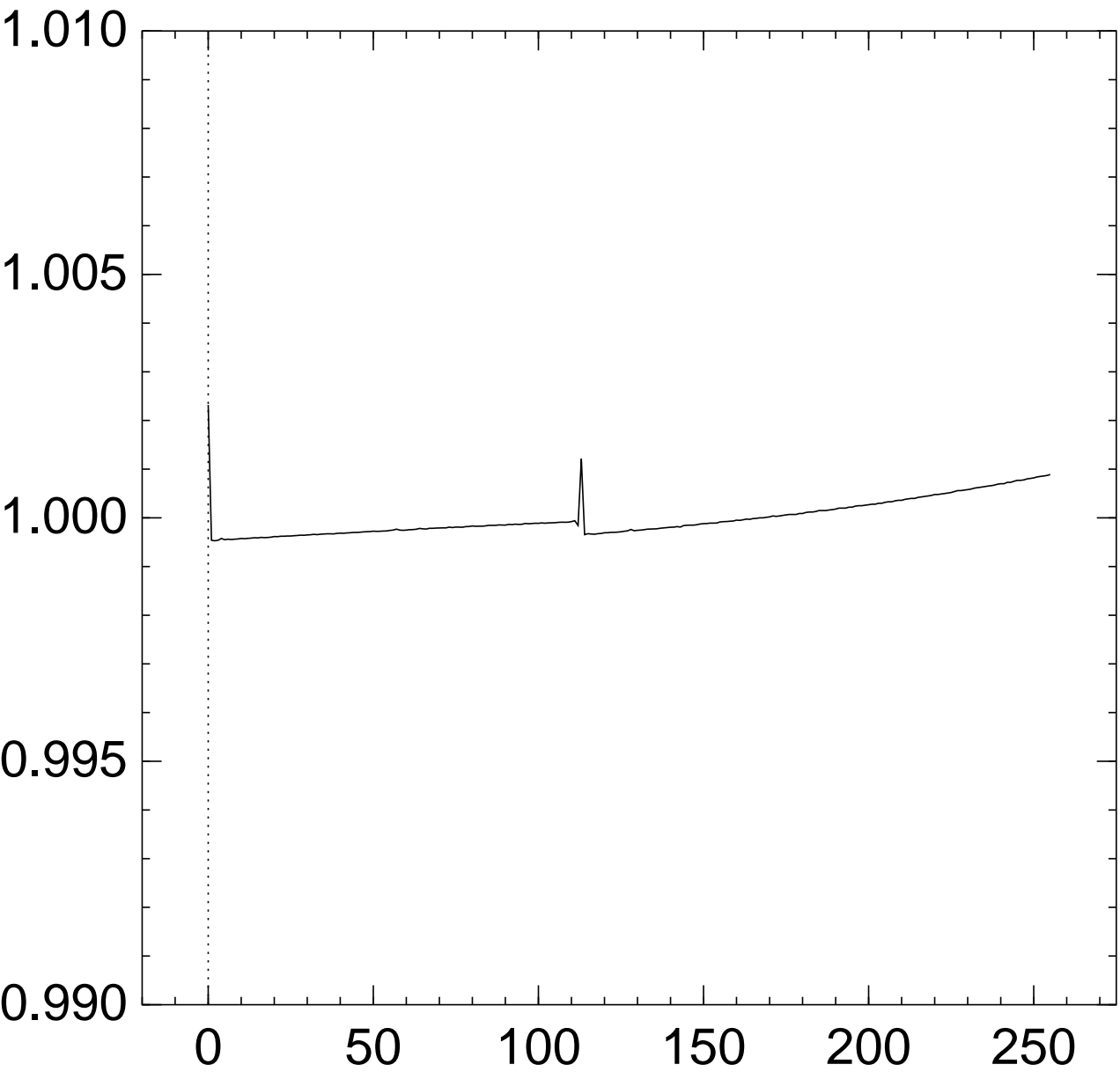
Graph of $256 \Pr[z_{111} = x]$:



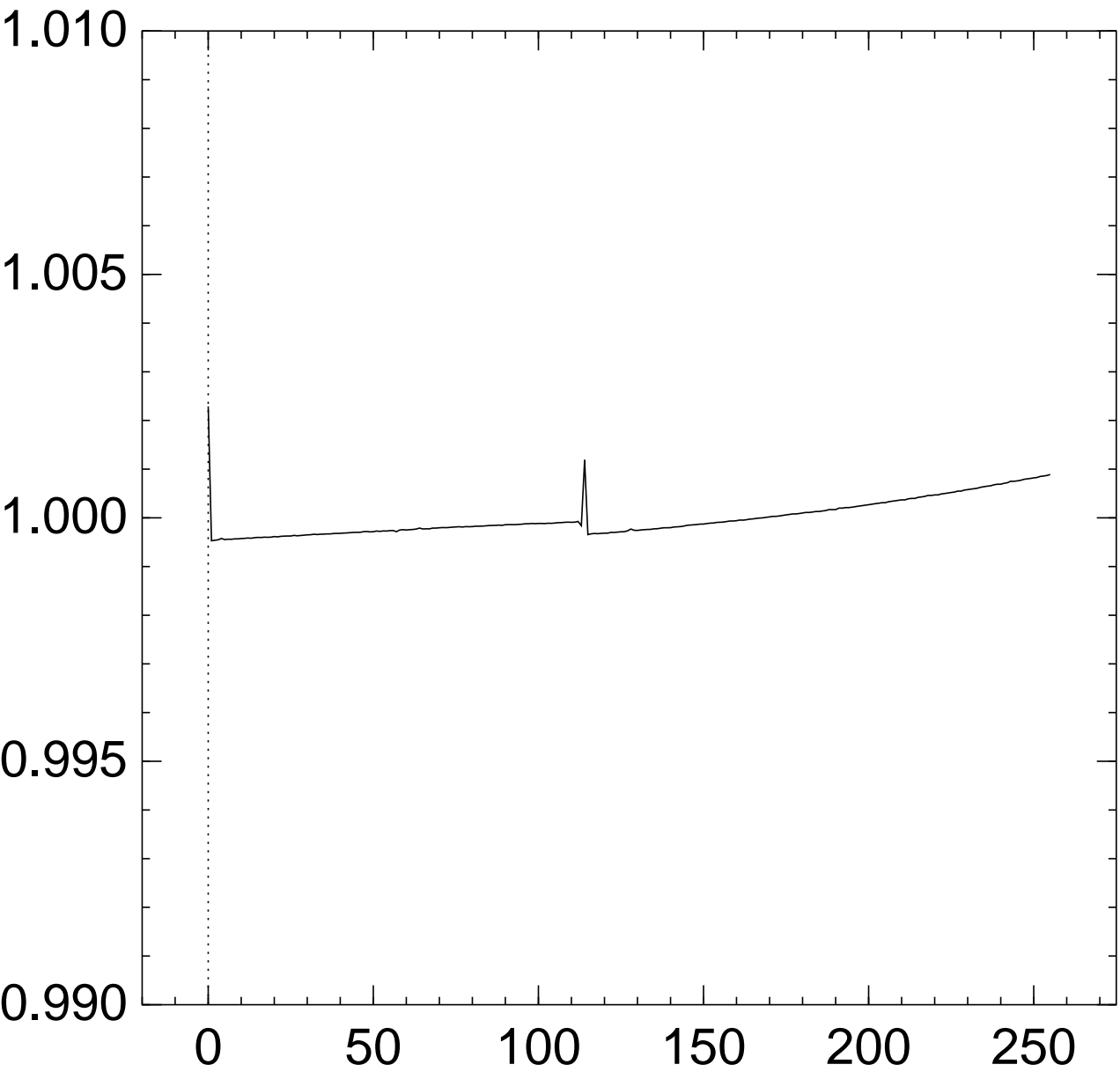
Graph of $256 \Pr[z_{112} = x]$:



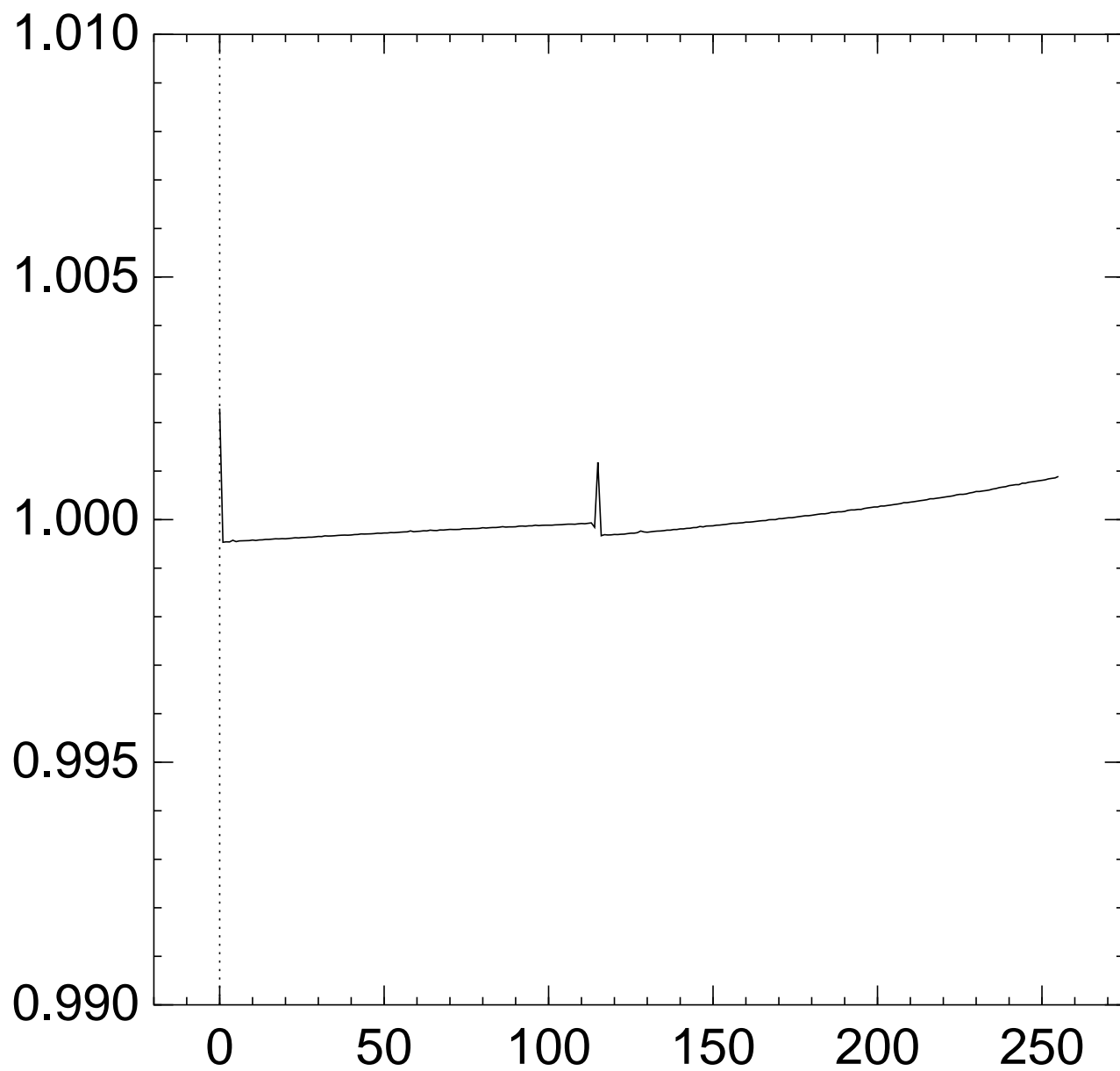
Graph of $256 \Pr[z_{113} = x]$:



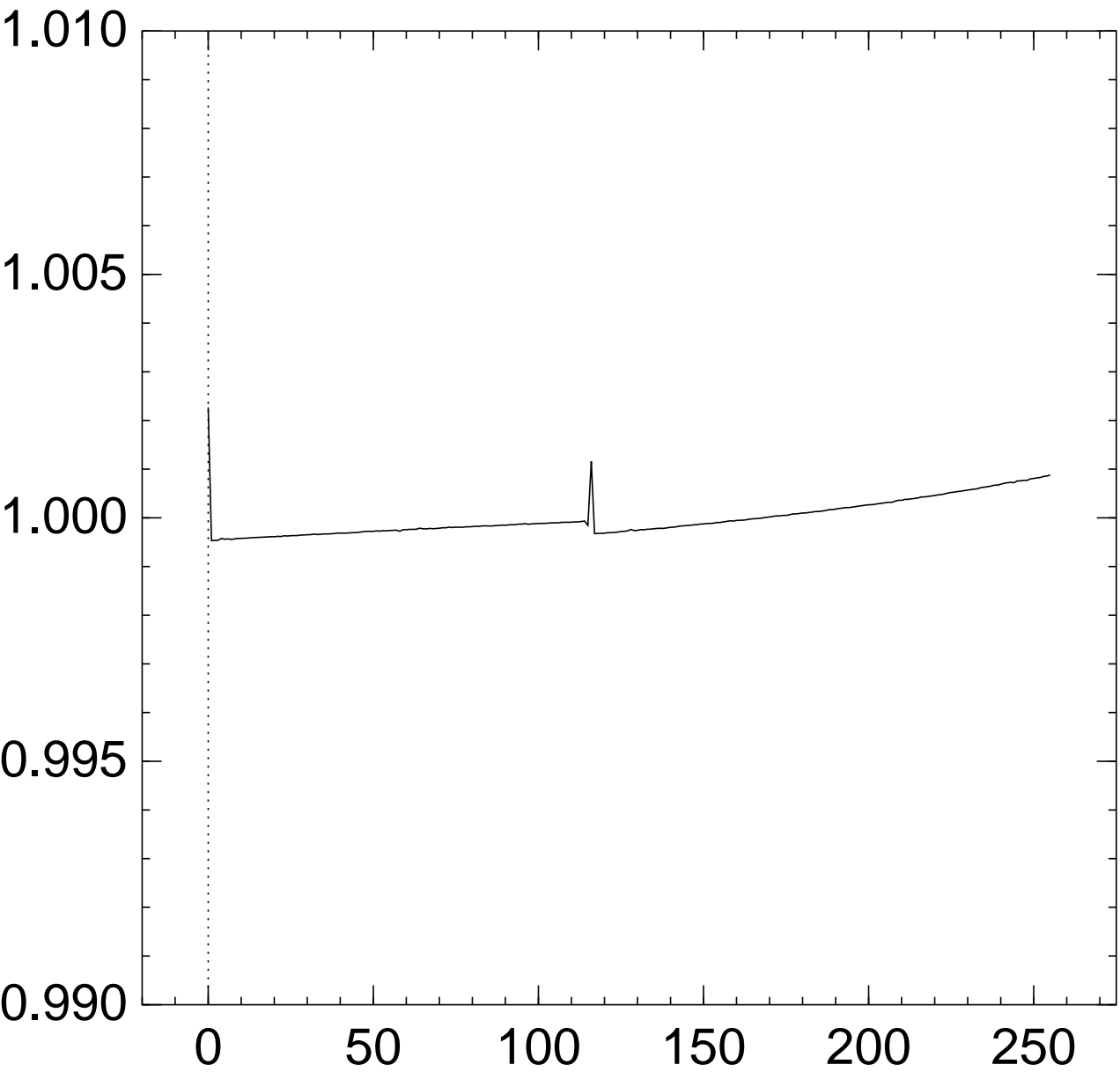
Graph of $256 \Pr[z_{114} = x]$:



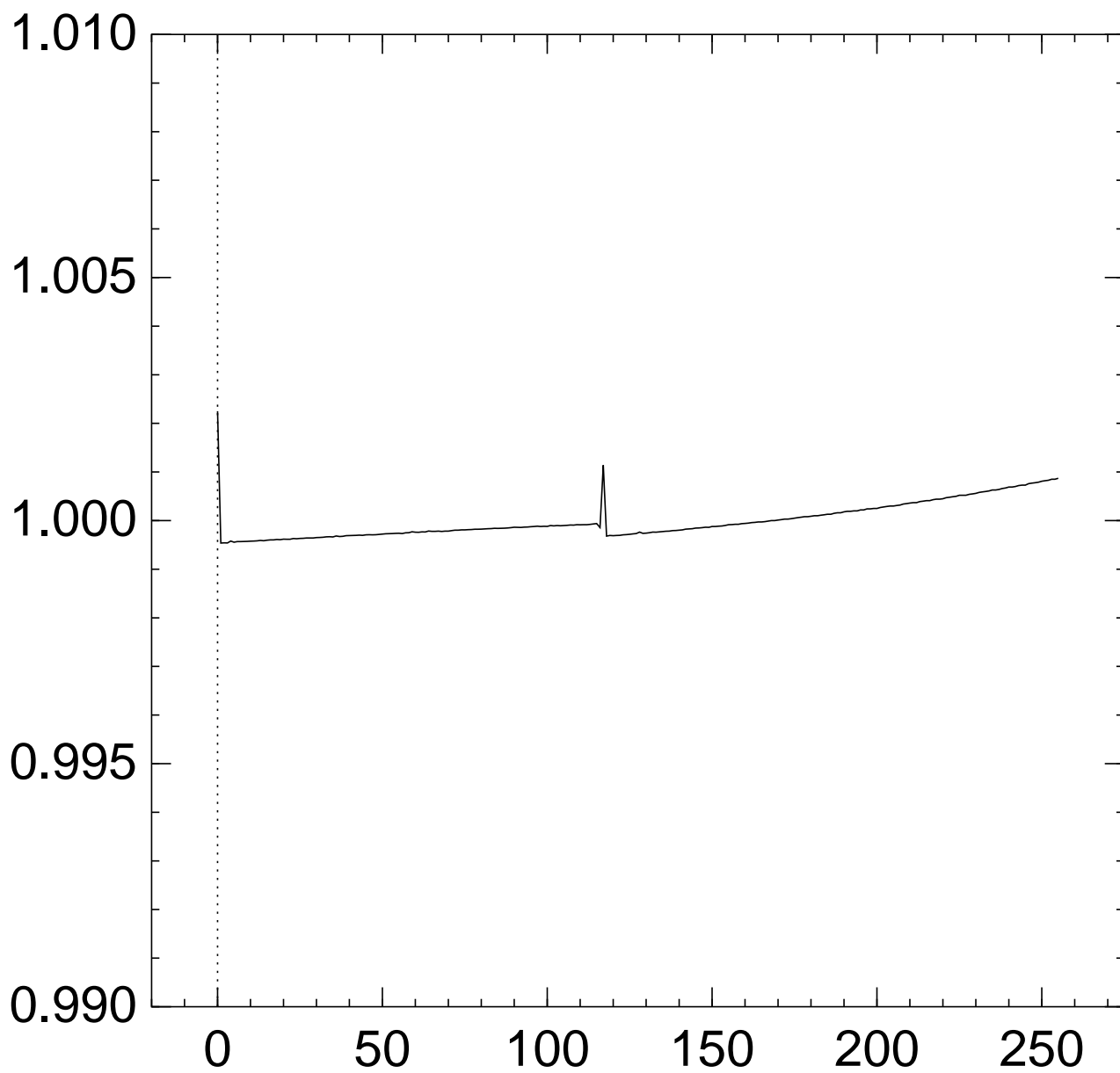
Graph of $256 \Pr[z_{115} = x]$:



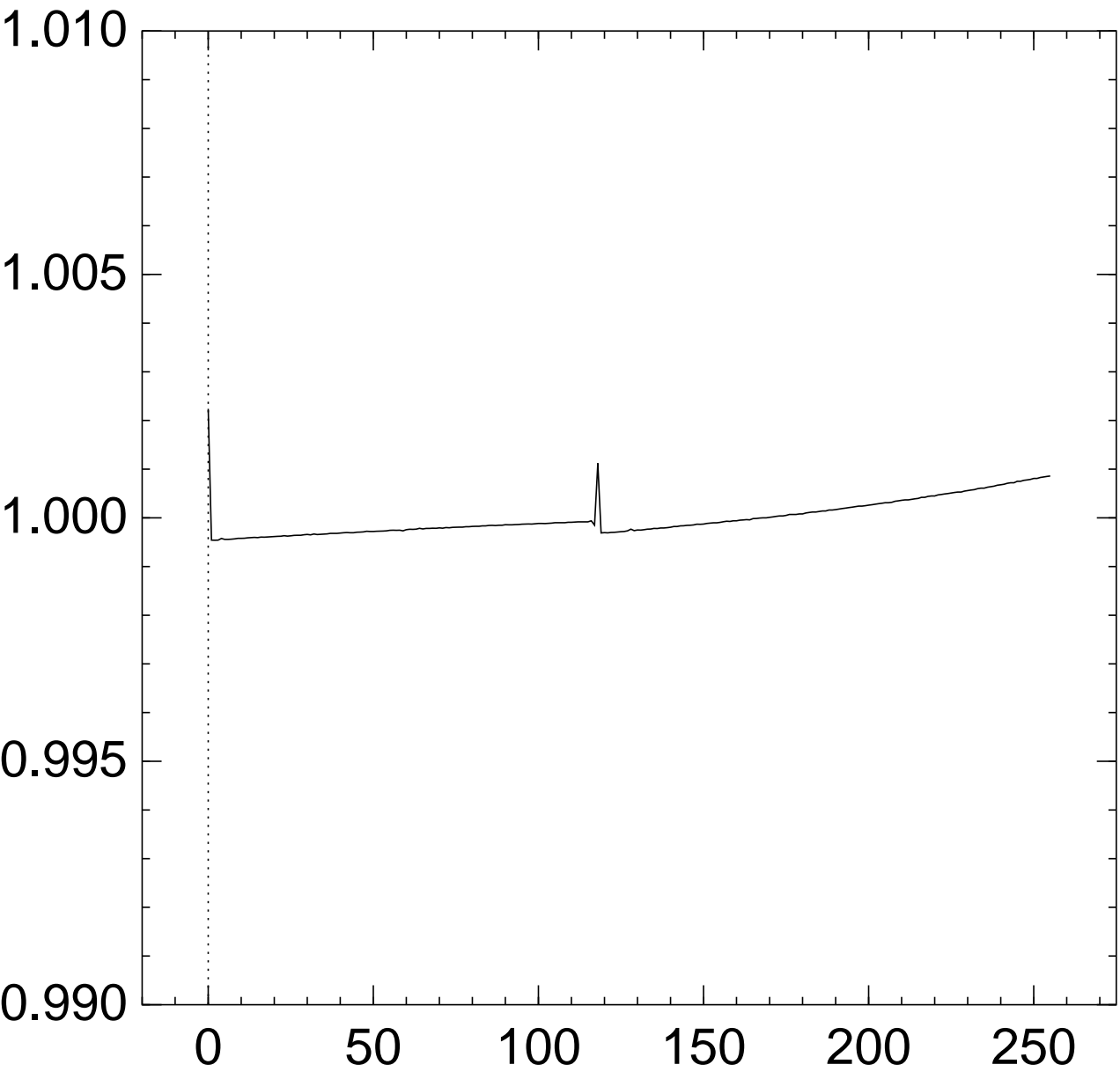
Graph of $256 \Pr[z_{116} = x]$:



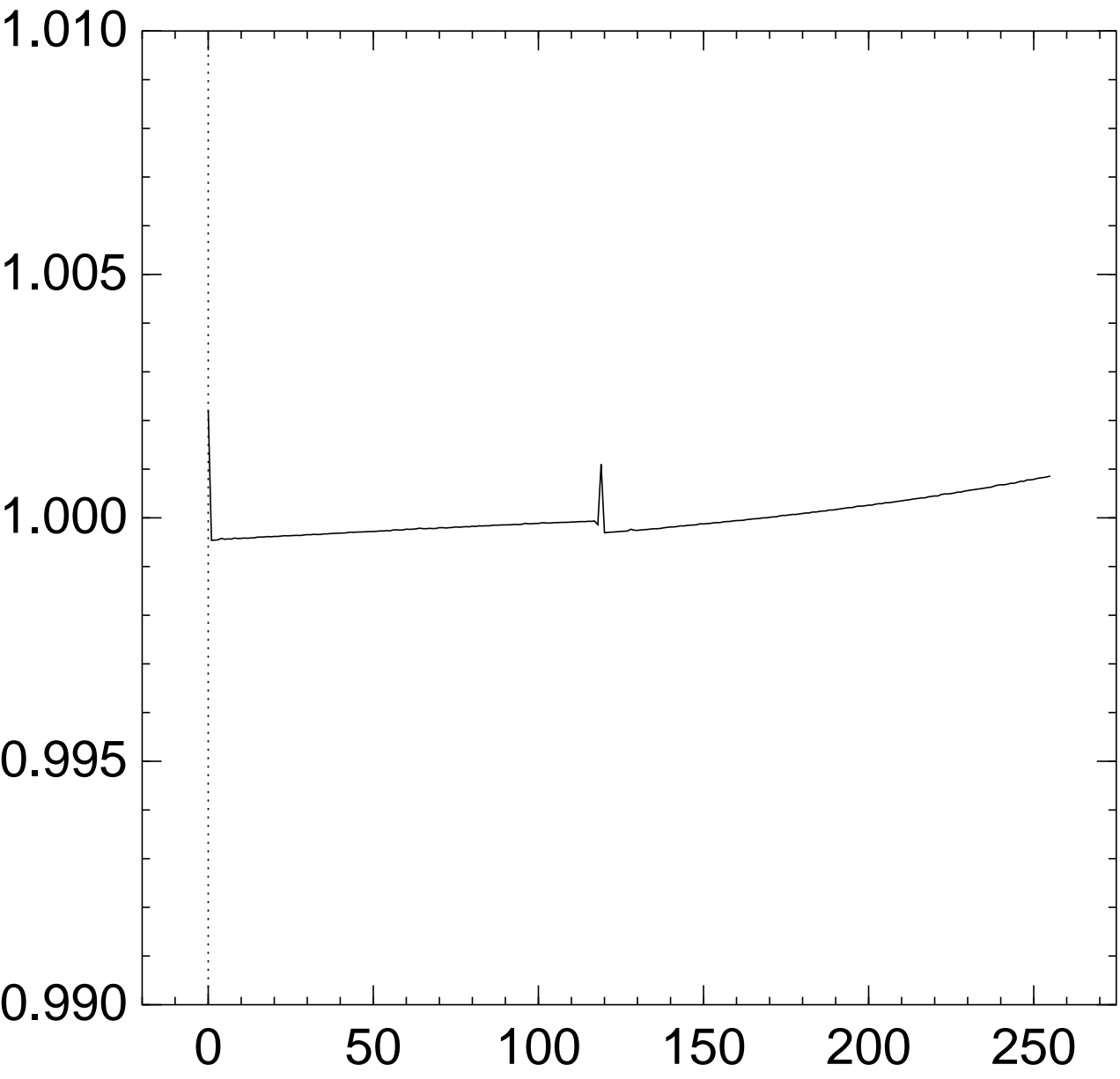
Graph of $256 \Pr[z_{117} = x]$:



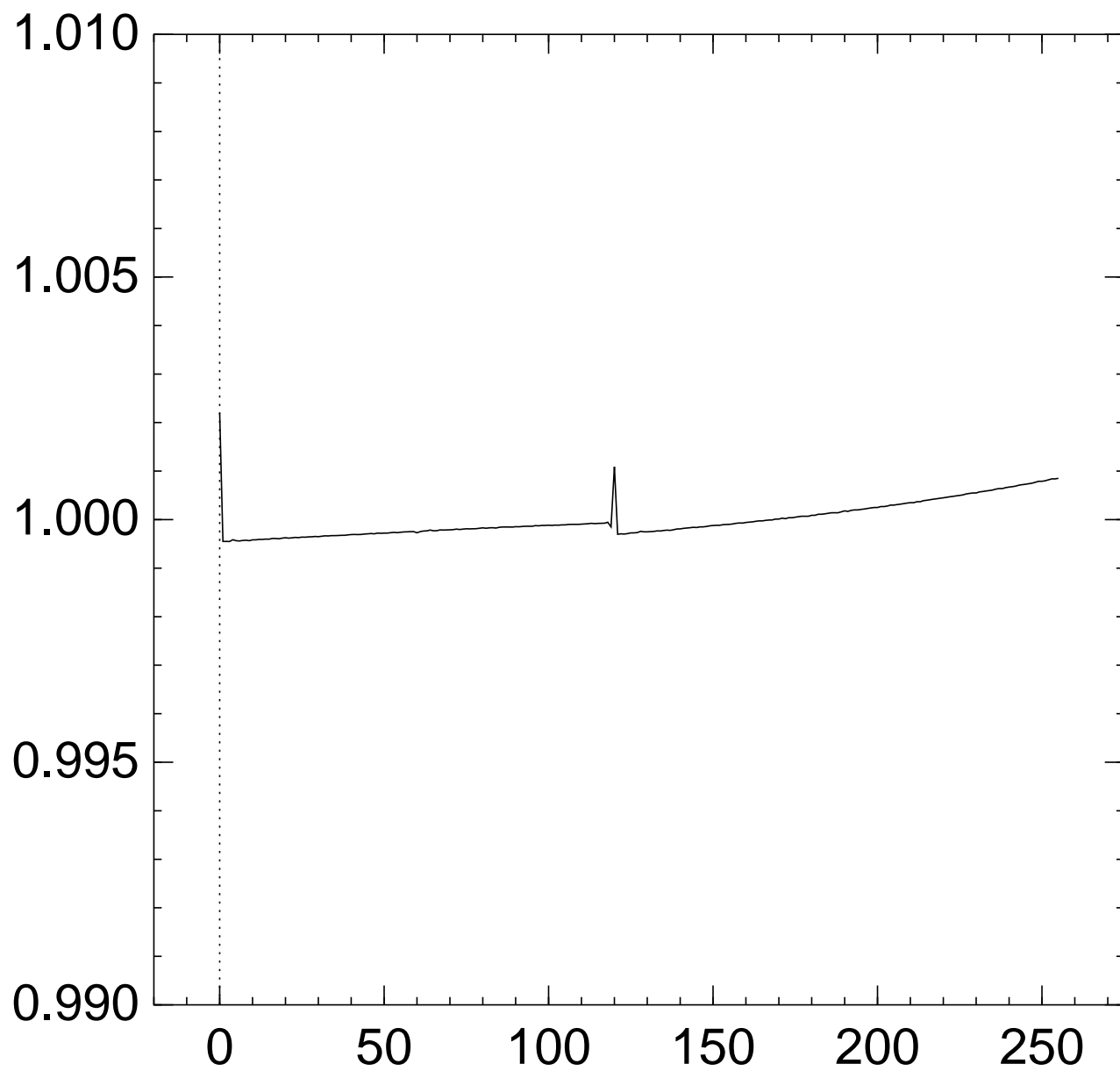
Graph of $256 \Pr[z_{118} = x]$:



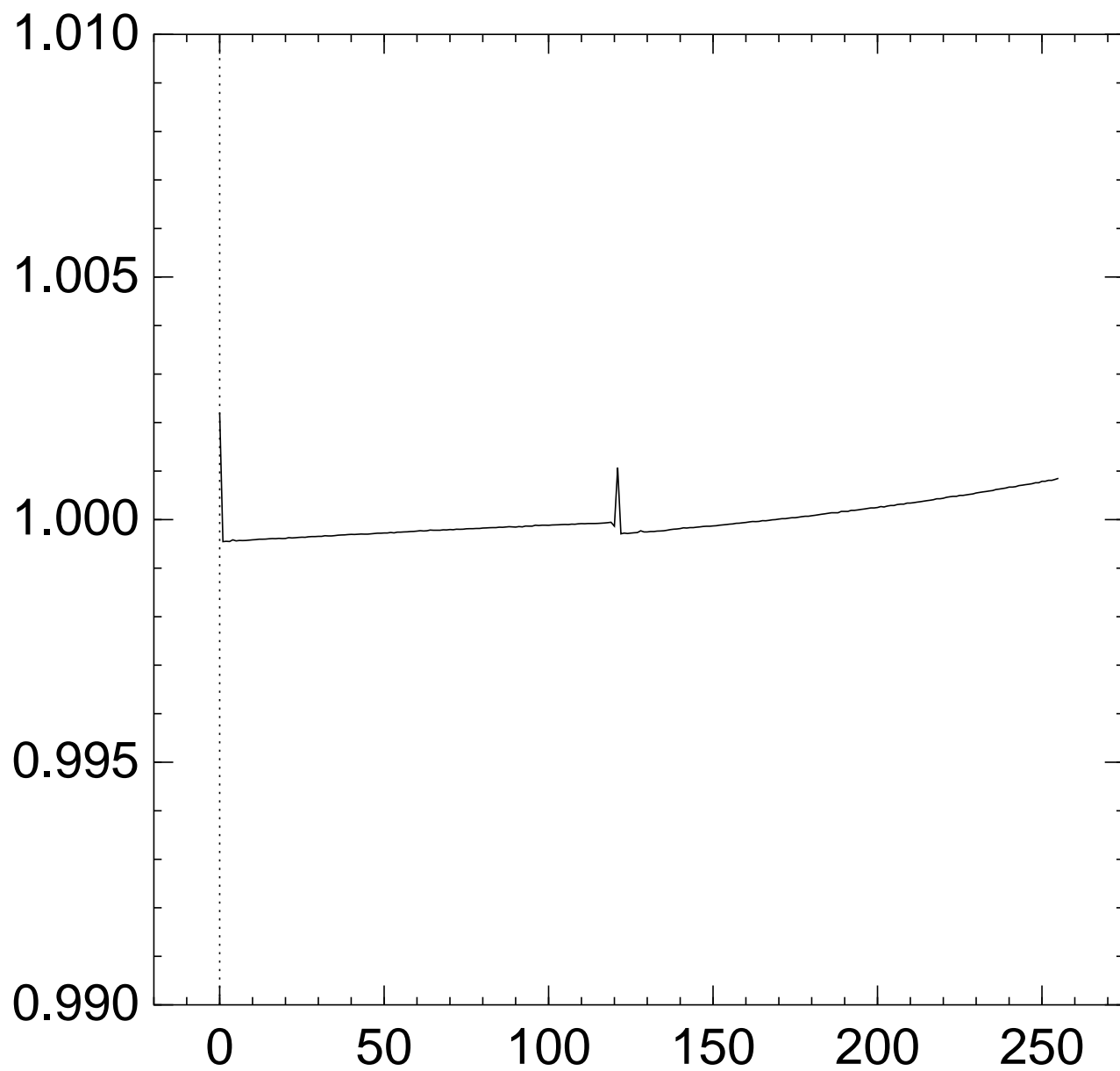
Graph of $256 \Pr[z_{119} = x]$:



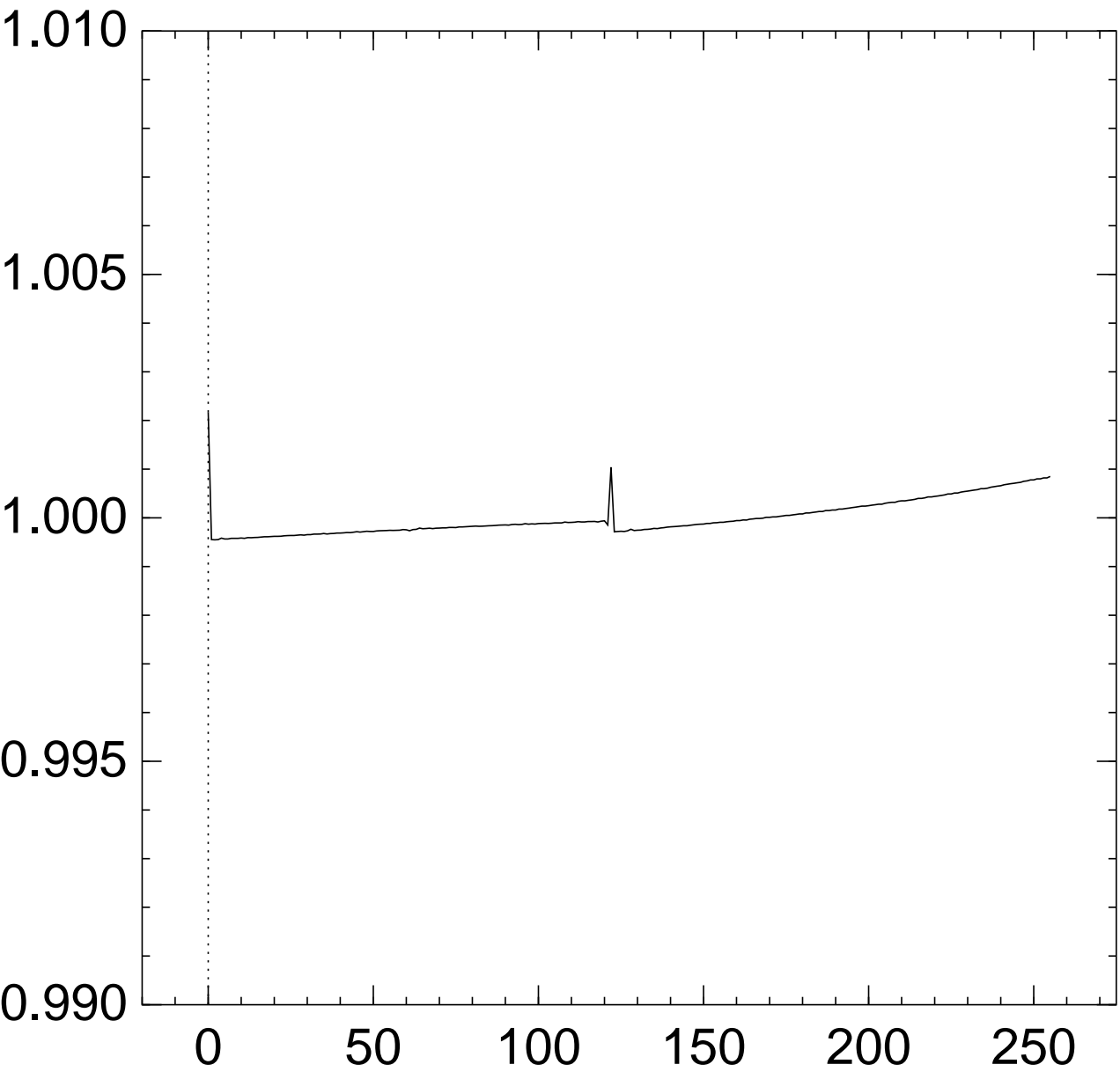
Graph of $256 \Pr[z_{120} = x]$:



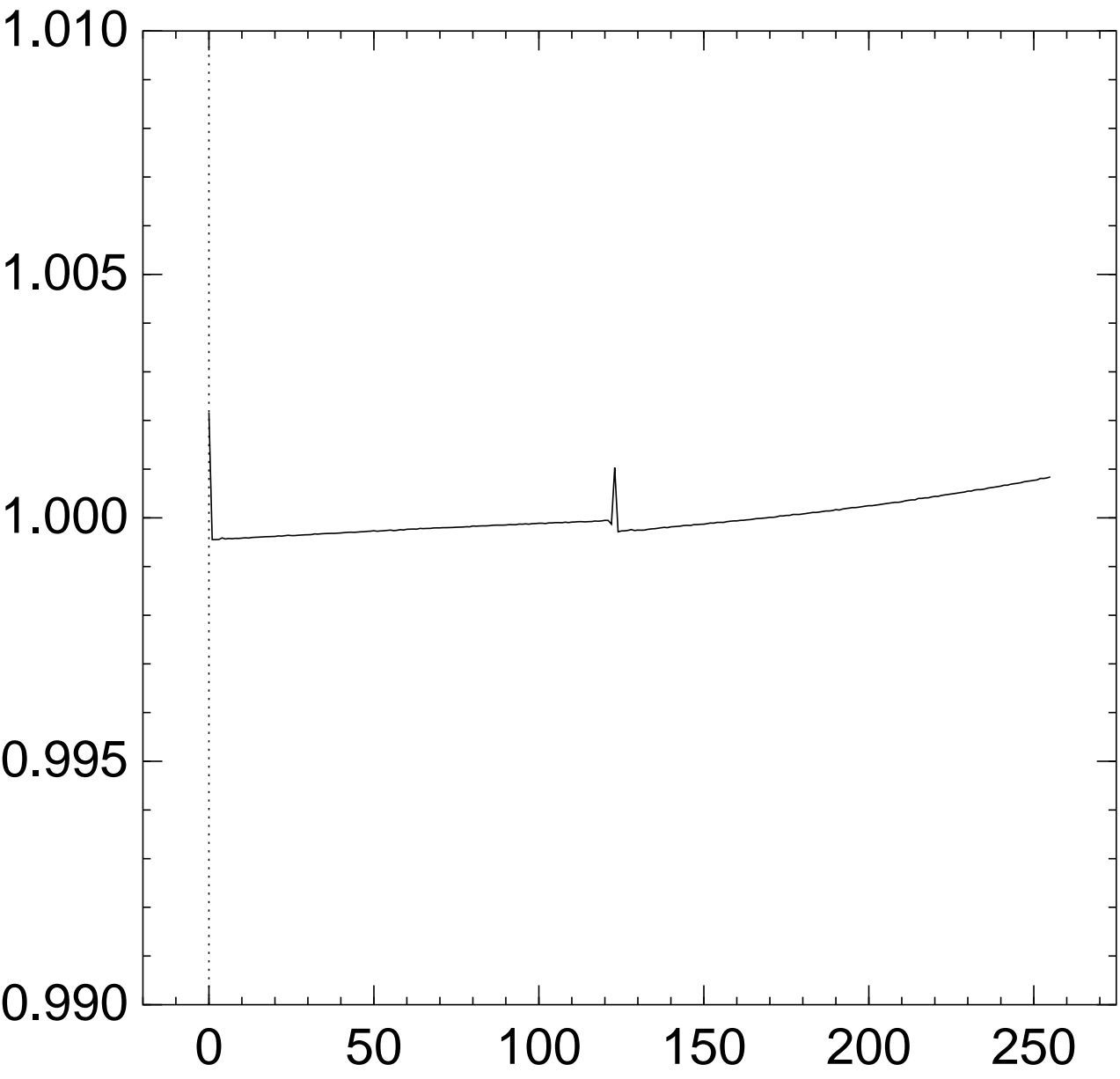
Graph of $256 \Pr[z_{121} = x]$:



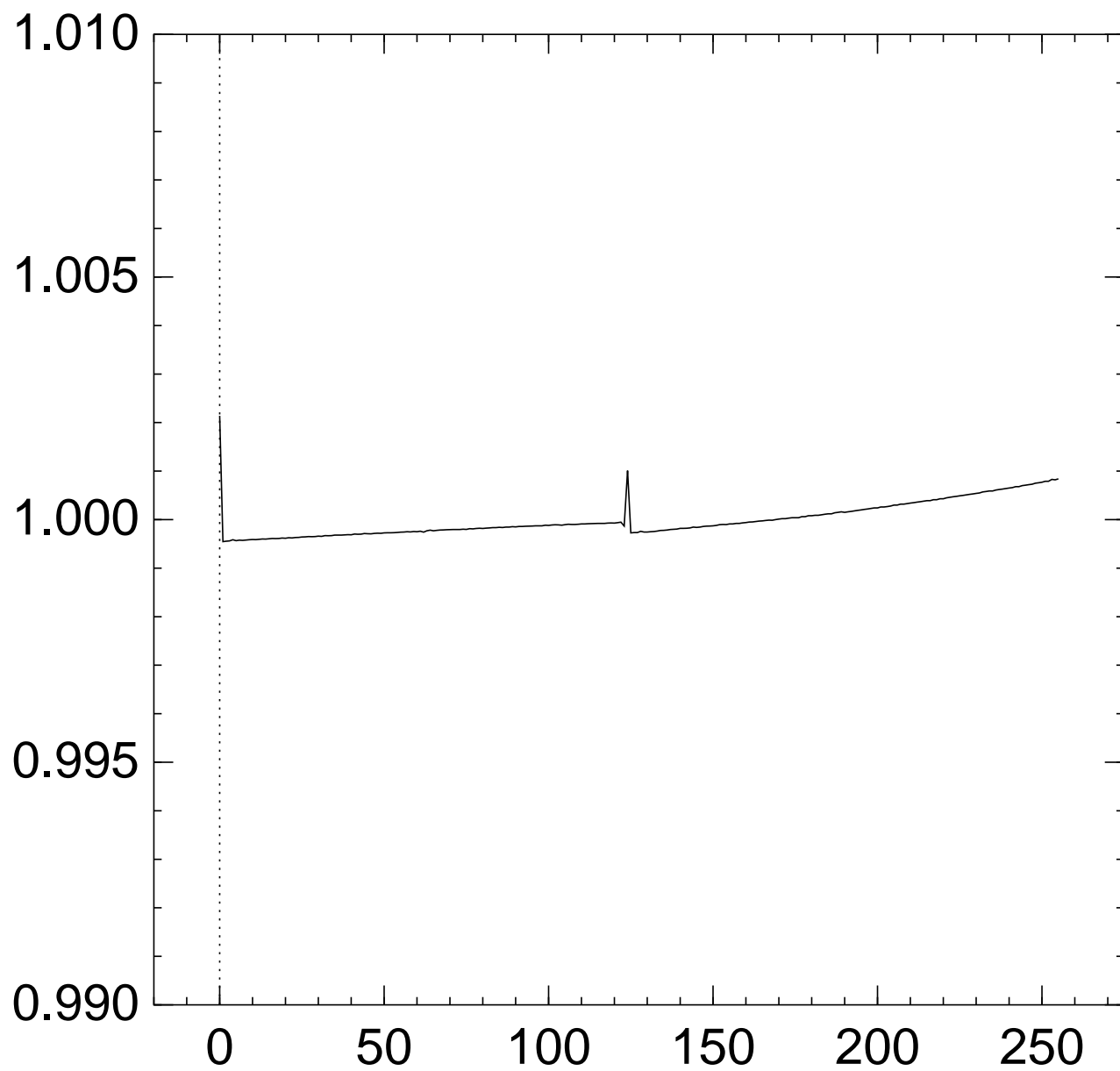
Graph of $256 \Pr[z_{122} = x]$:



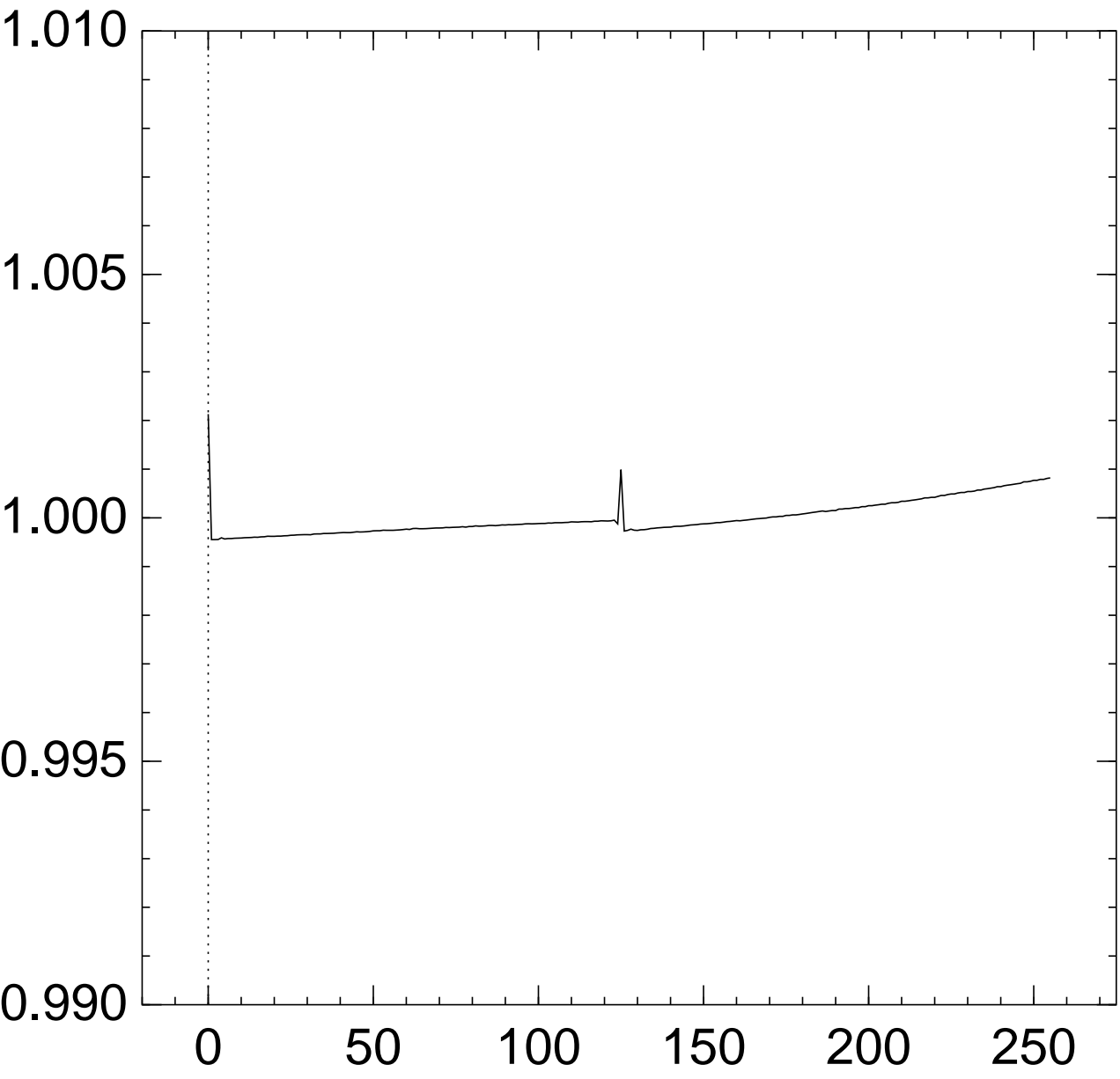
Graph of $256 \Pr[z_{123} = x]$:



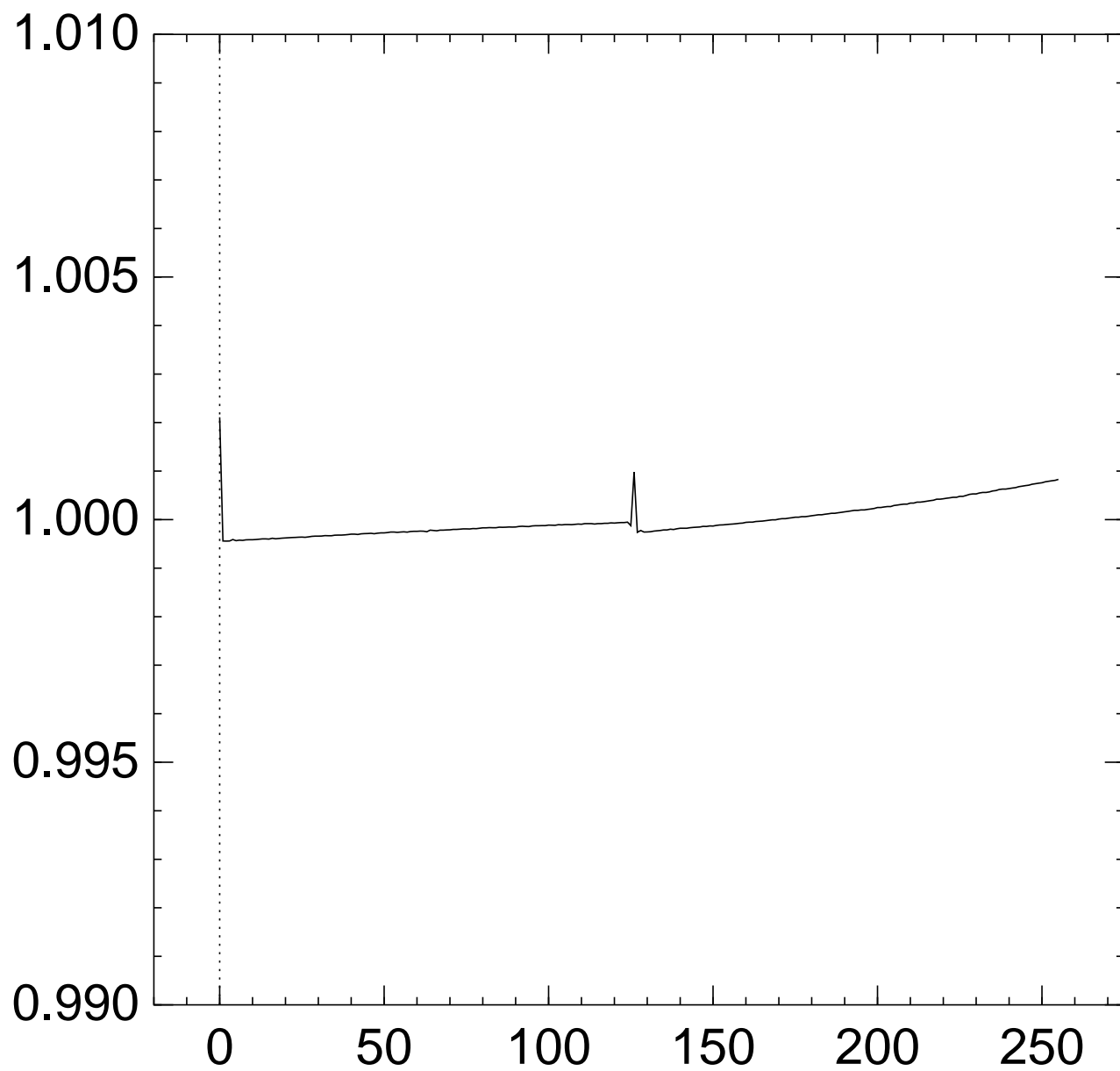
Graph of $256 \Pr[z_{124} = x]$:



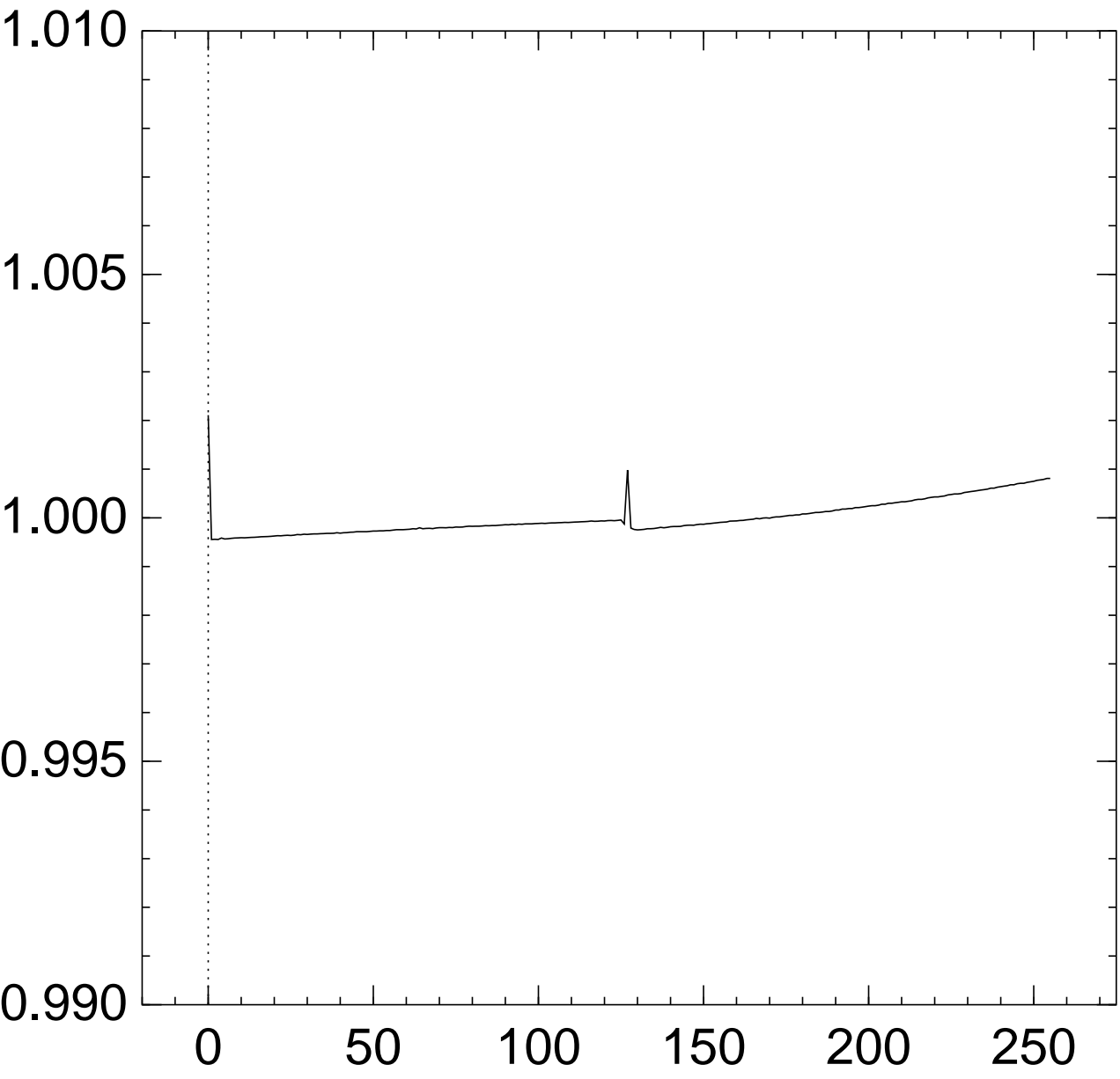
Graph of $256 \Pr[z_{125} = x]$:



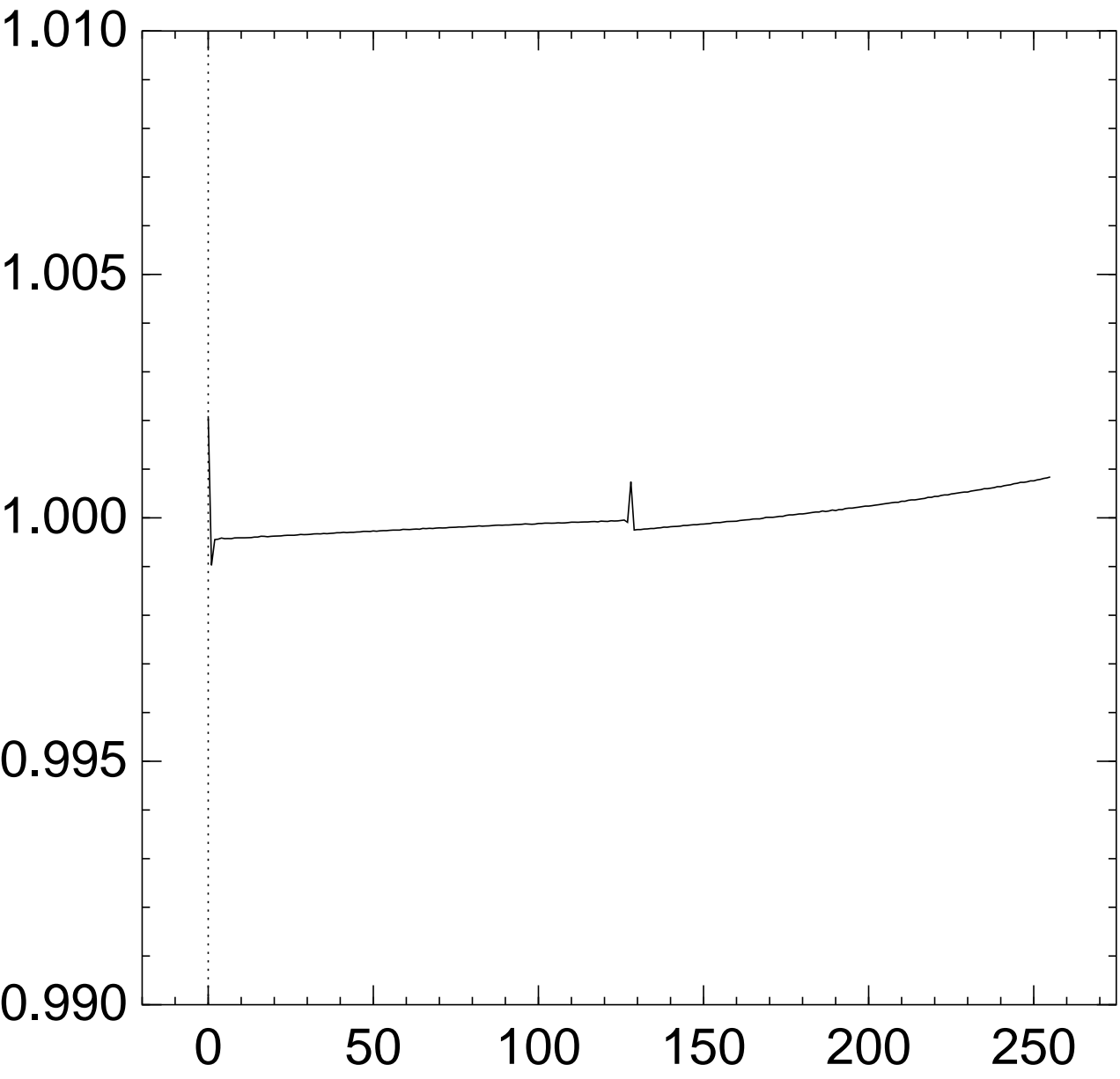
Graph of $256 \Pr[z_{126} = x]$:



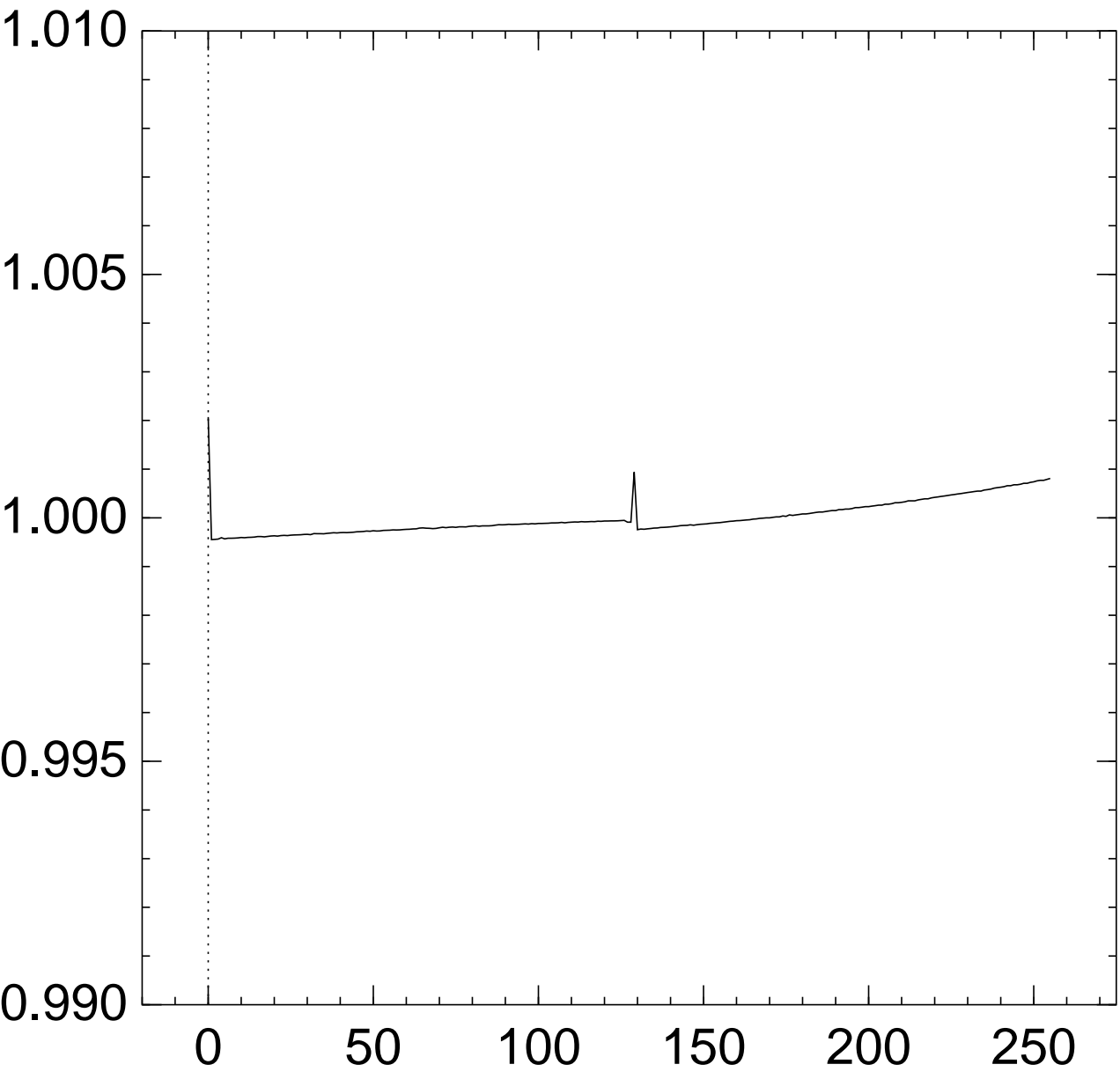
Graph of $256 \Pr[z_{127} = x]$:



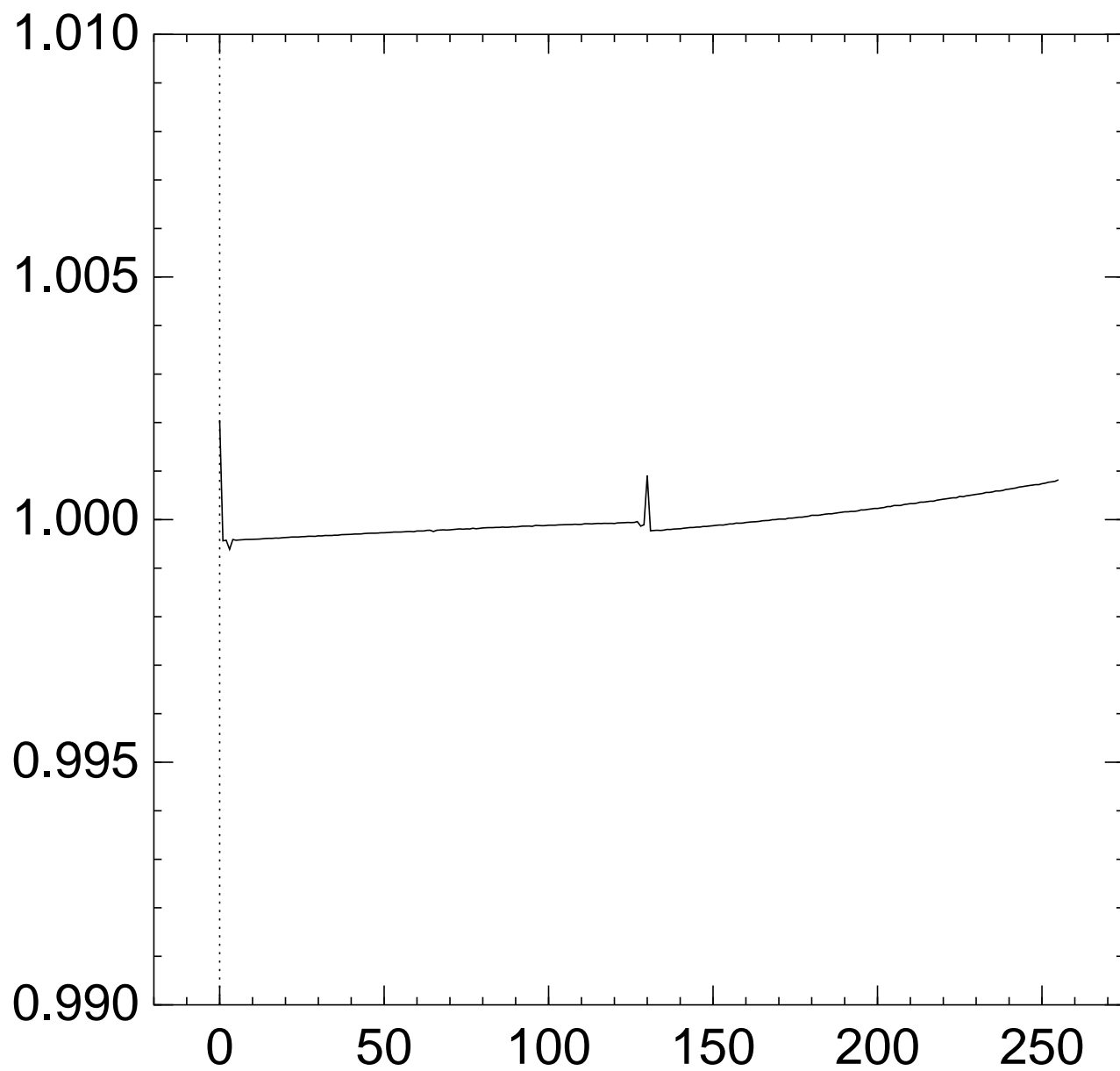
Graph of $256 \Pr[z_{128} = x]$:



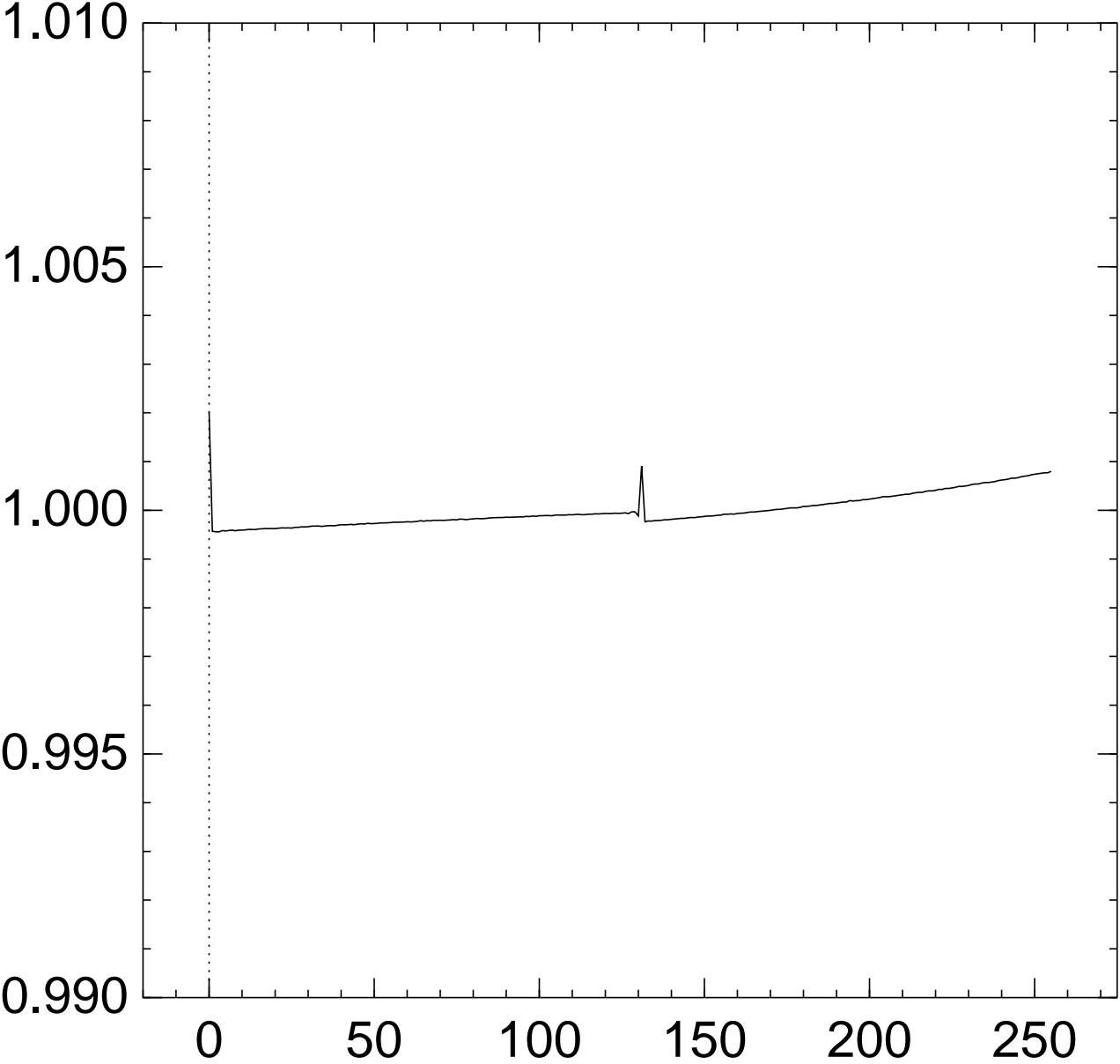
Graph of $256 \Pr[z_{129} = x]$:



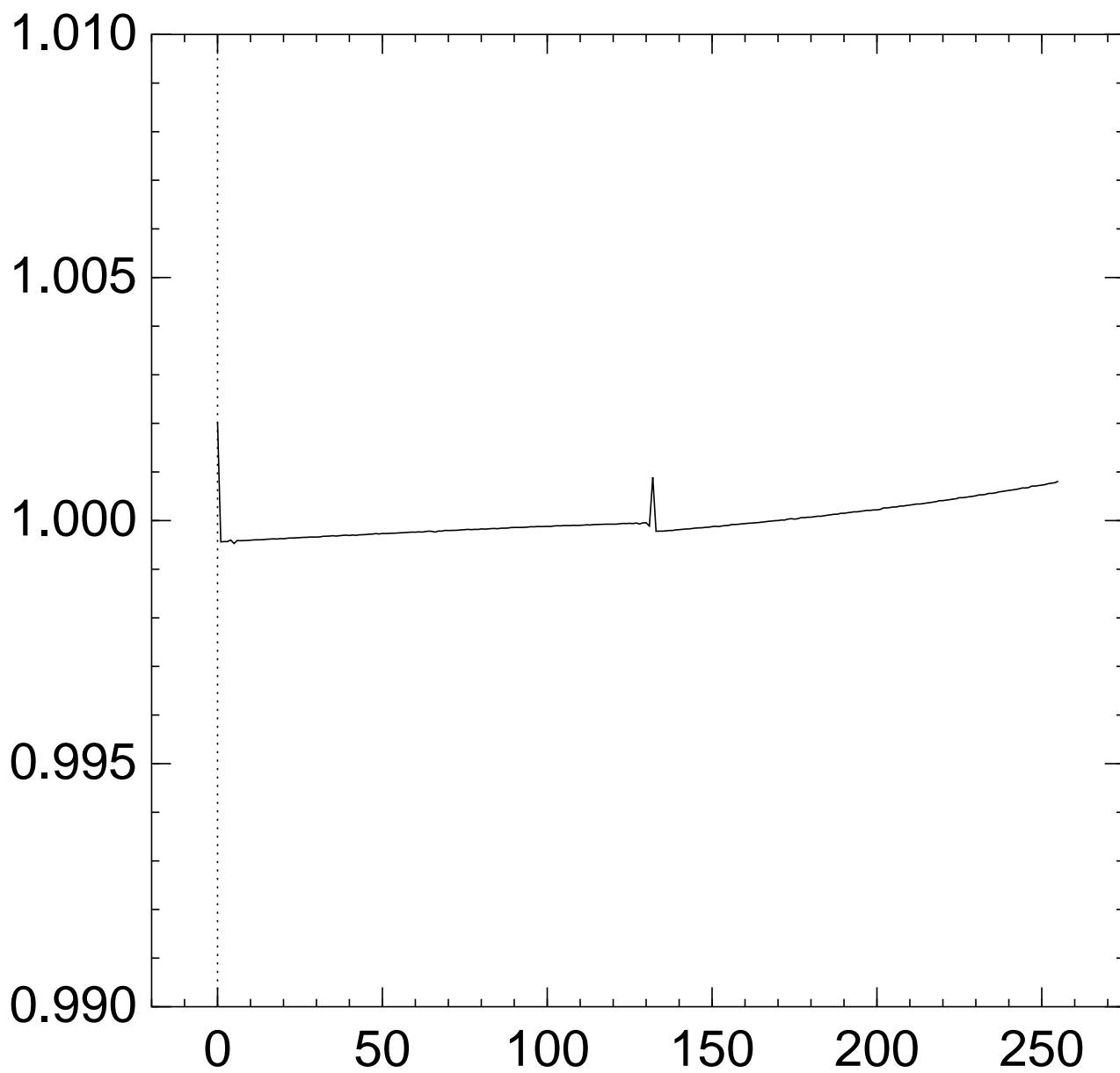
Graph of $256 \Pr[z_{130} = x]$:



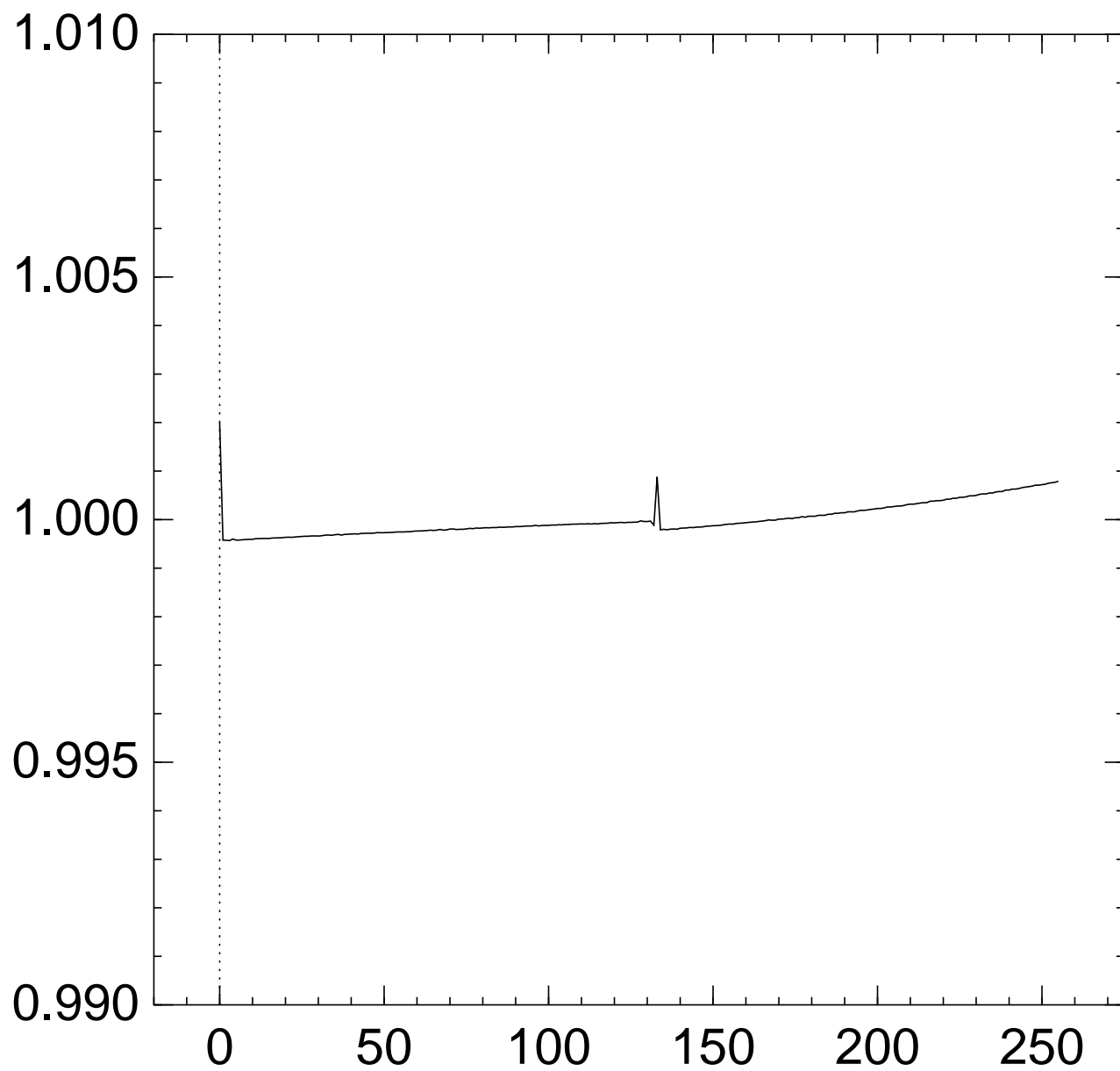
Graph of $256 \Pr[z_{131} = x]$:



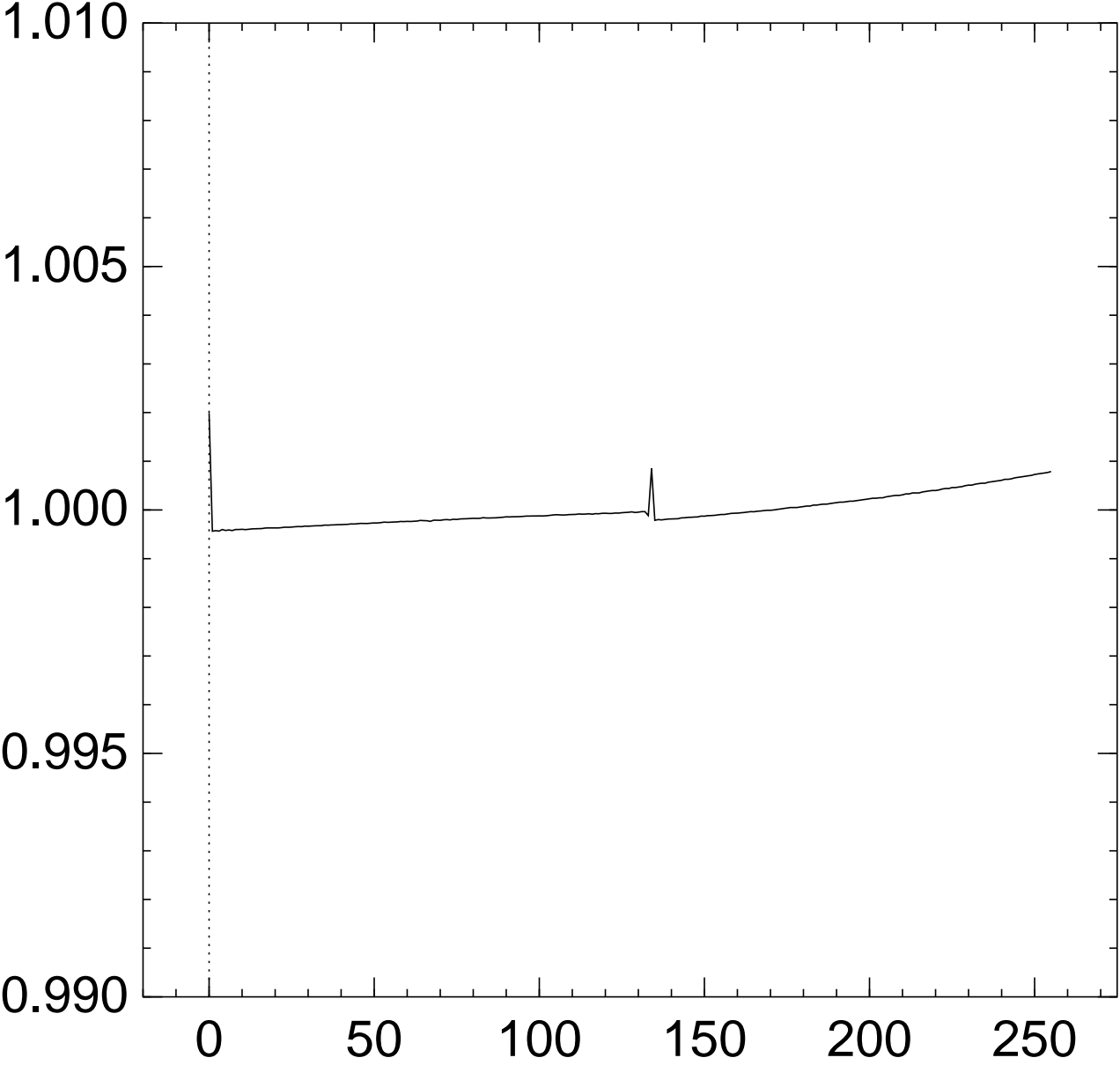
Graph of $256 \Pr[z_{132} = x]$:



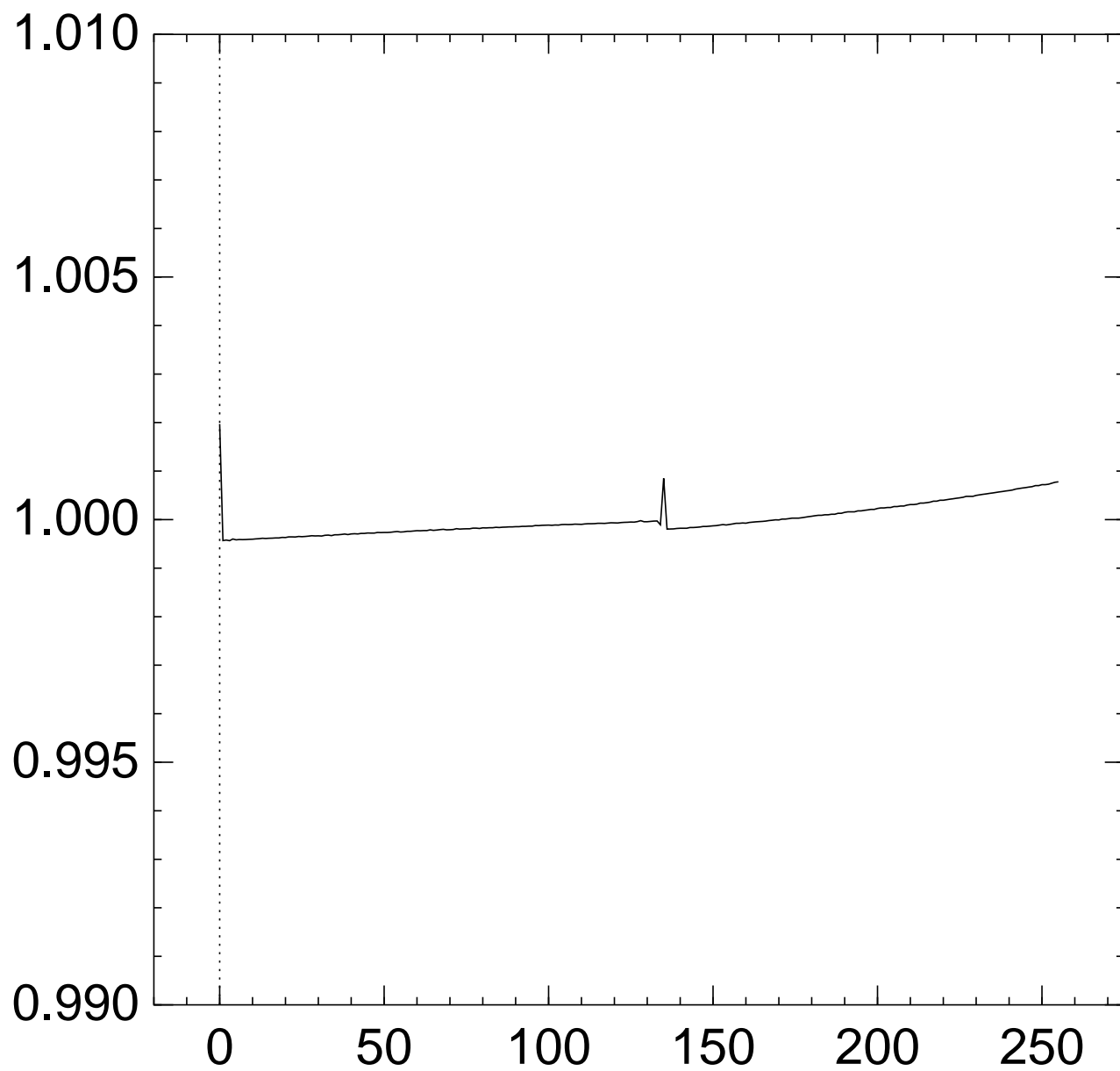
Graph of $256 \Pr[z_{133} = x]$:



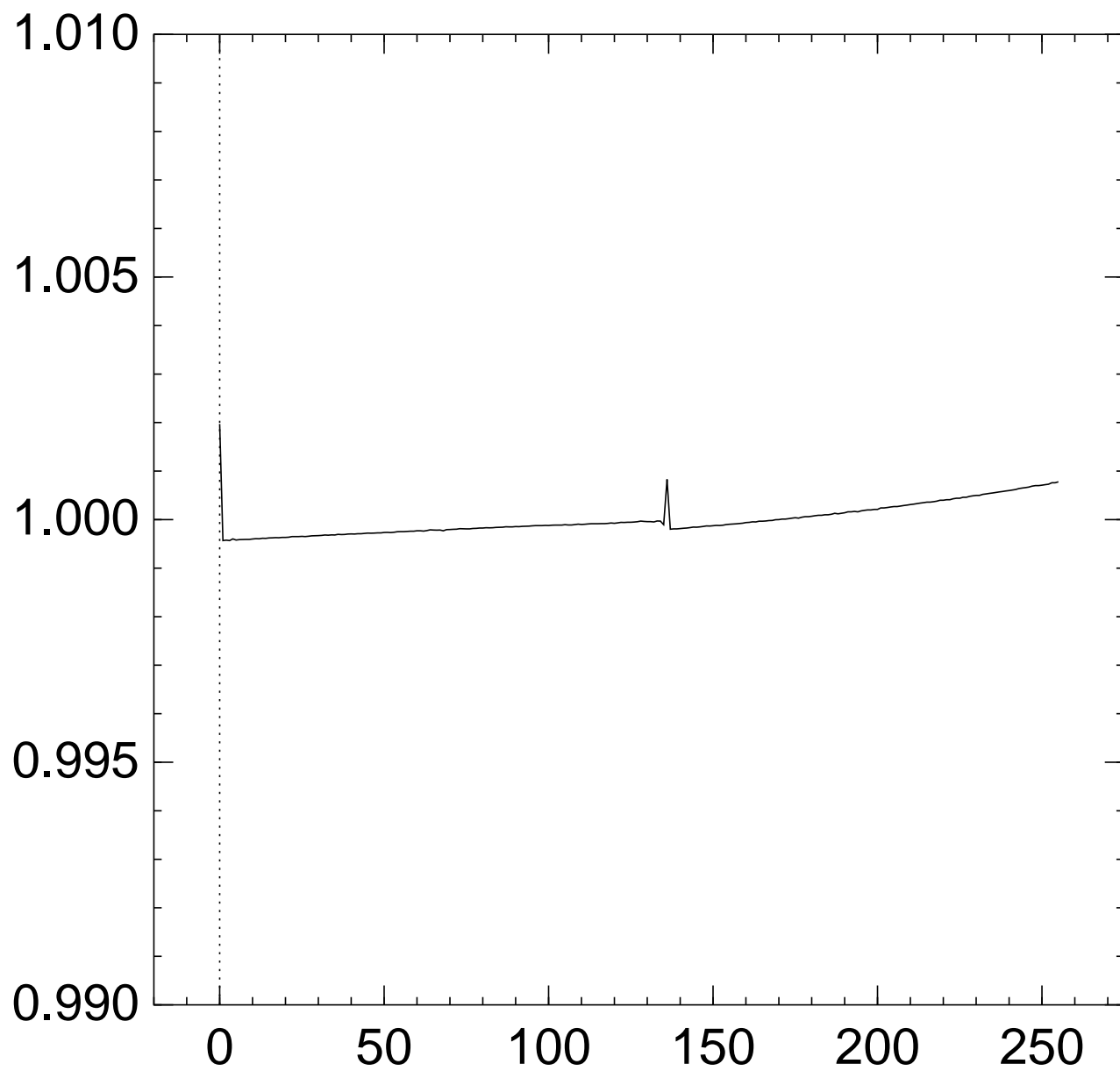
Graph of $256 \Pr[z_{134} = x]$:



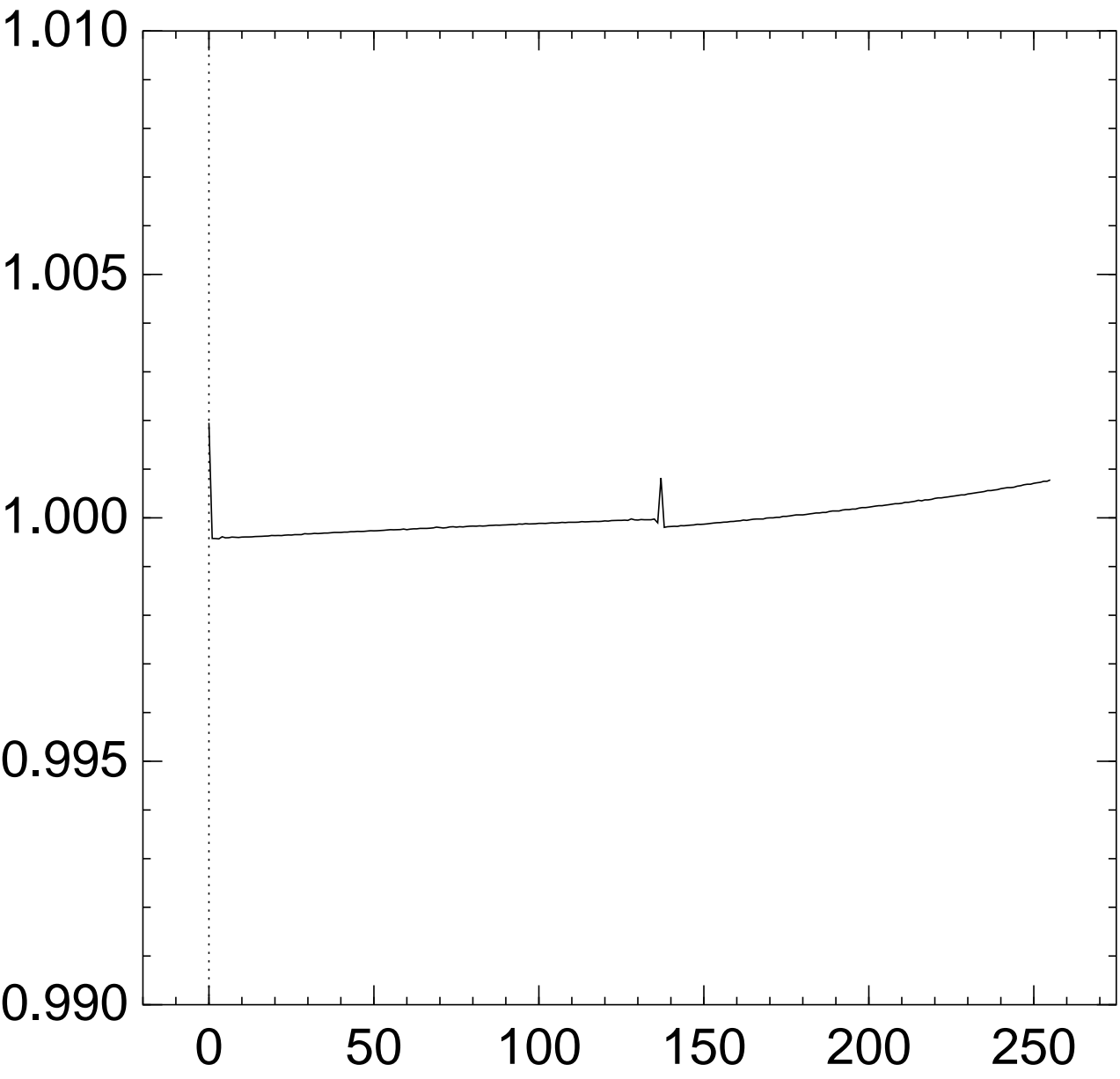
Graph of $256 \Pr[z_{135} = x]$:



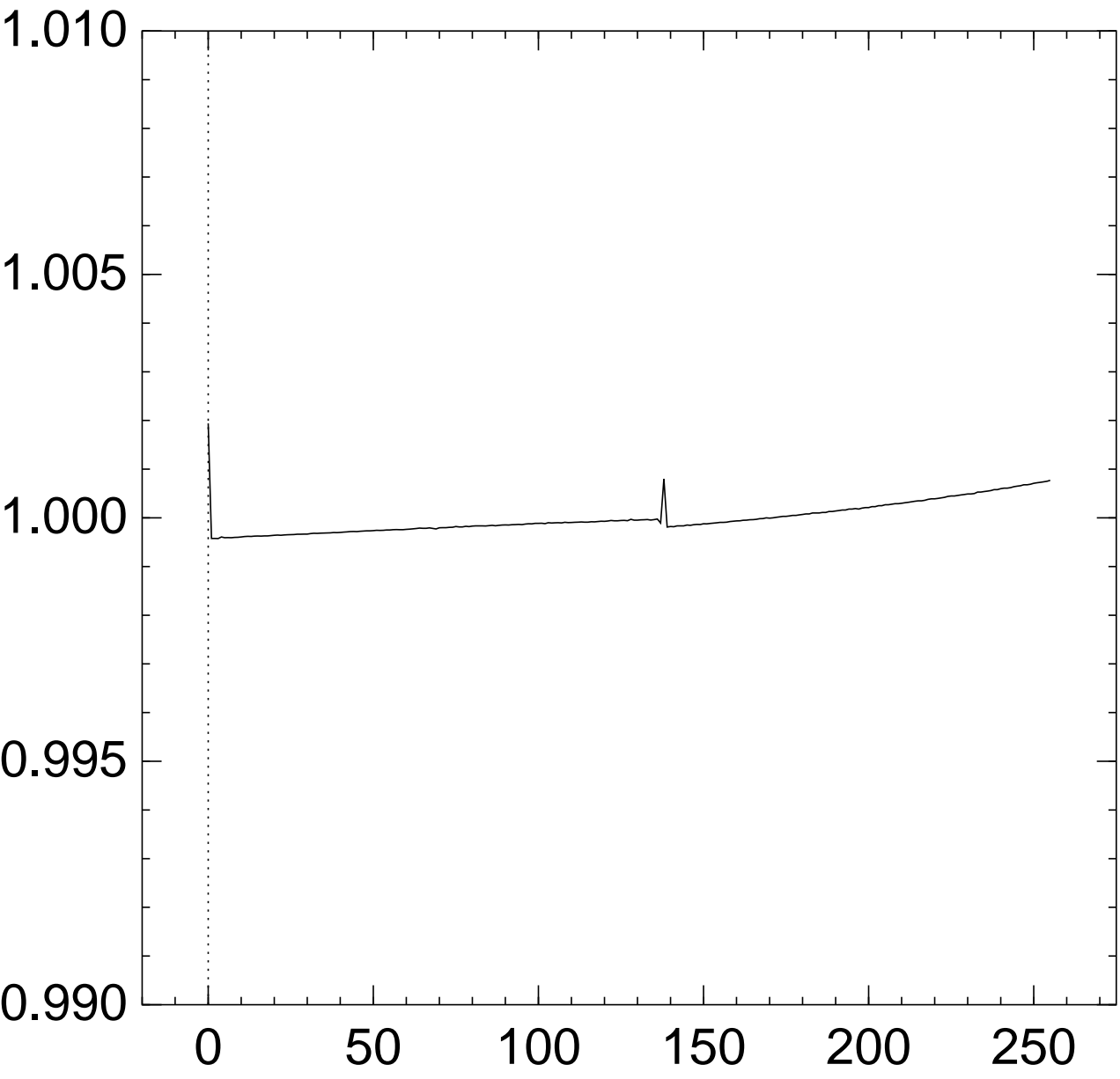
Graph of $256 \Pr[z_{136} = x]$:



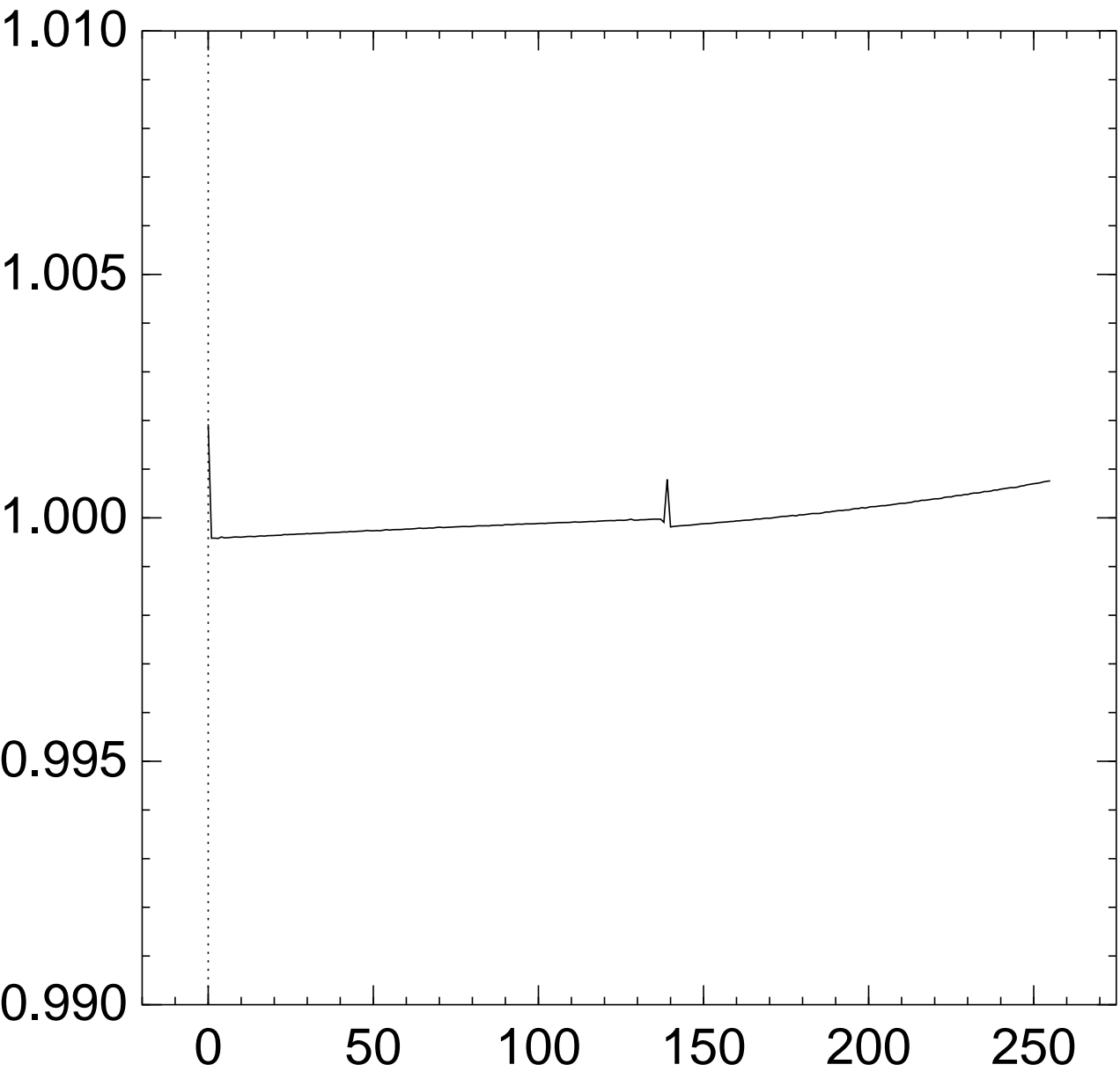
Graph of $256 \Pr[z_{137} = x]$:



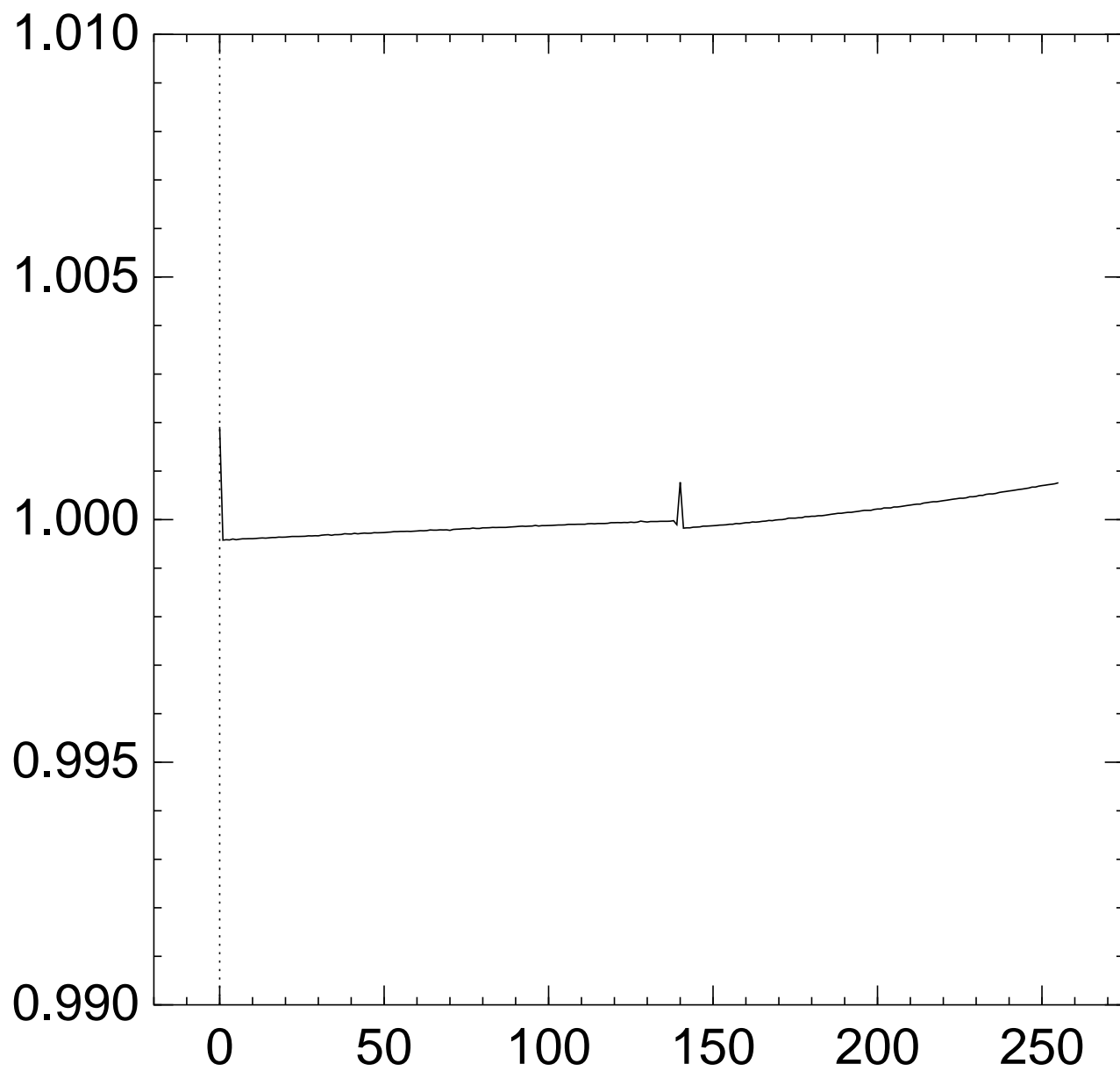
Graph of $256 \Pr[z_{138} = x]$:



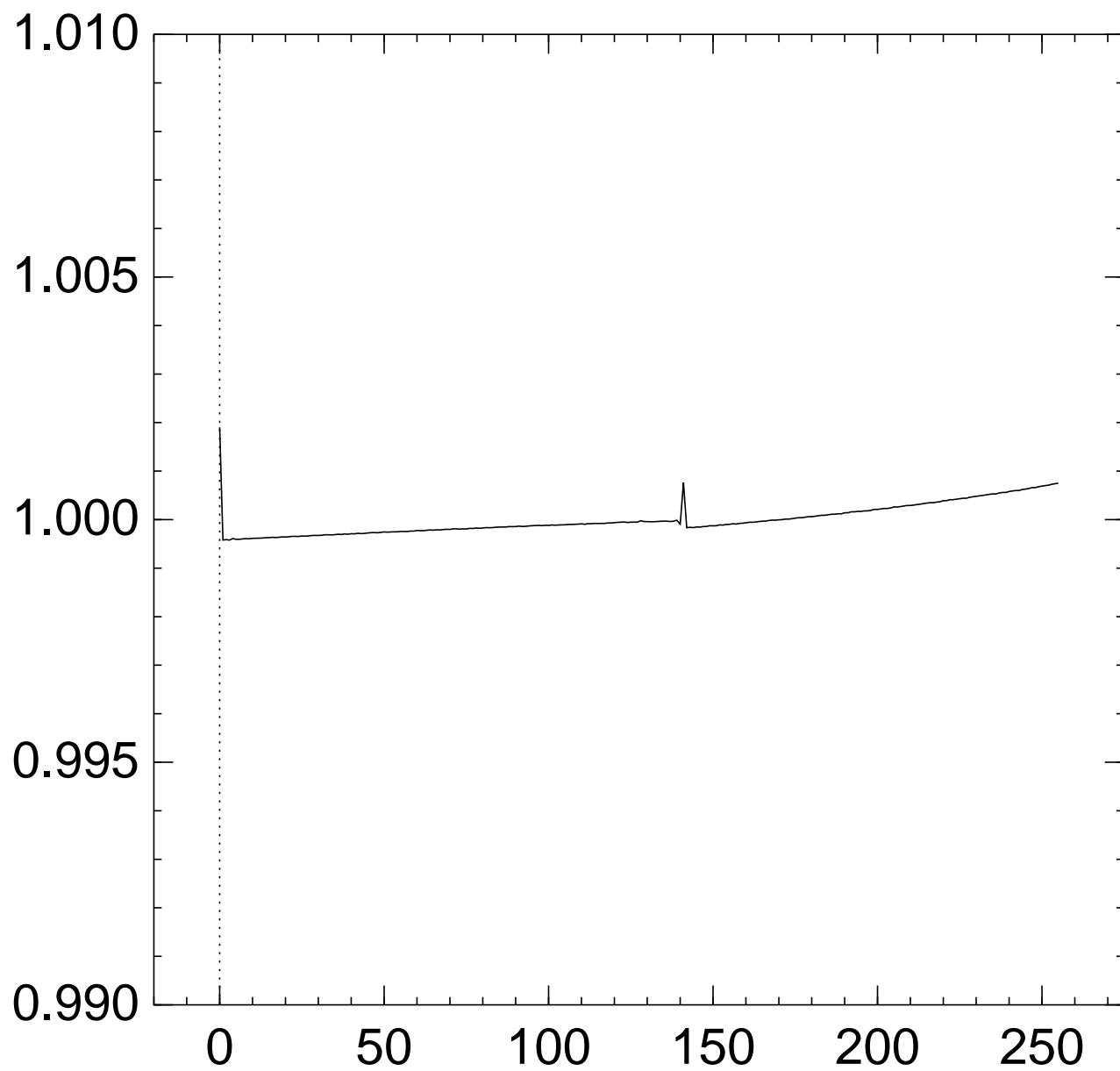
Graph of $256 \Pr[z_{139} = x]$:



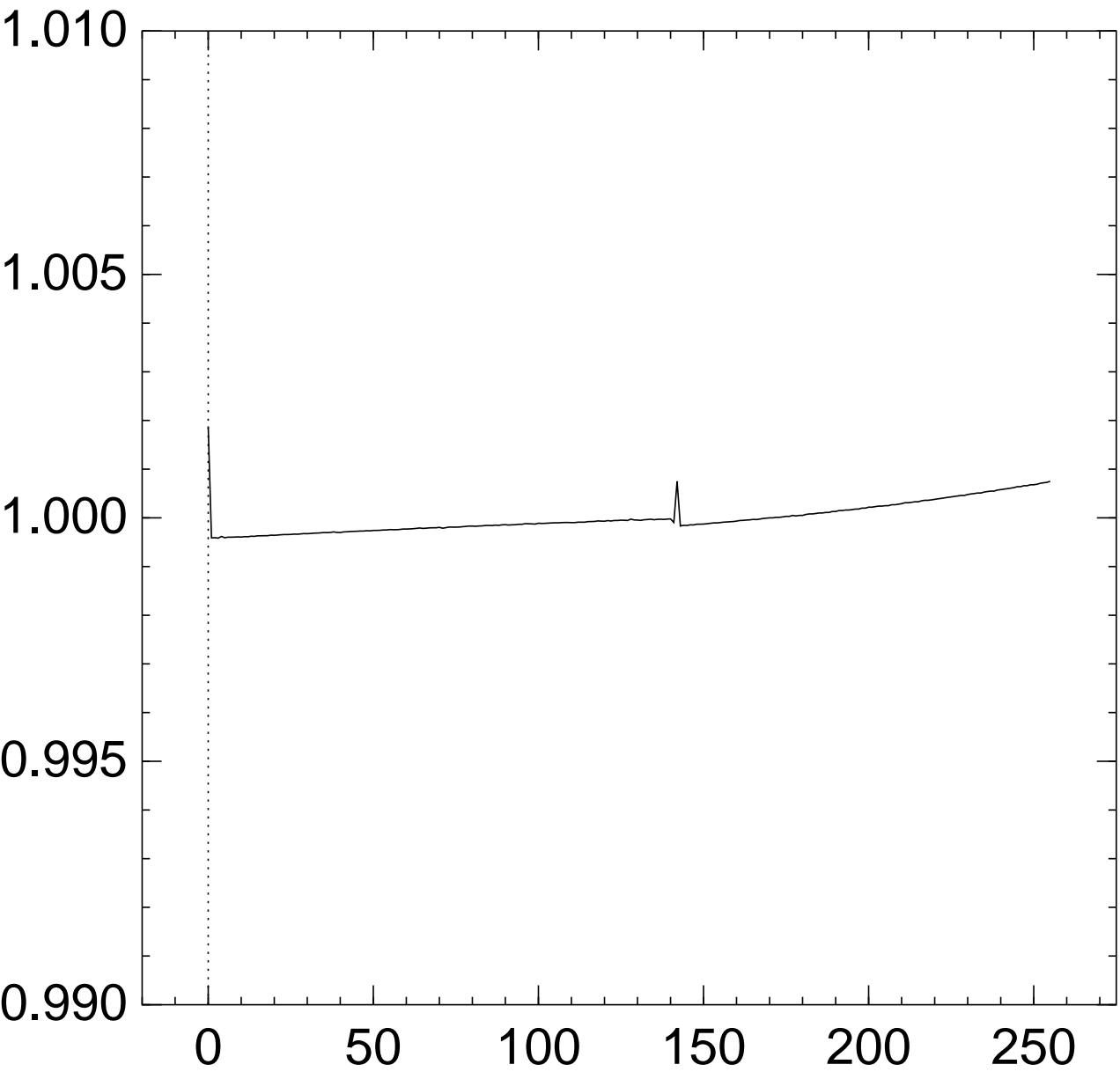
Graph of $256 \Pr[z_{140} = x]$:



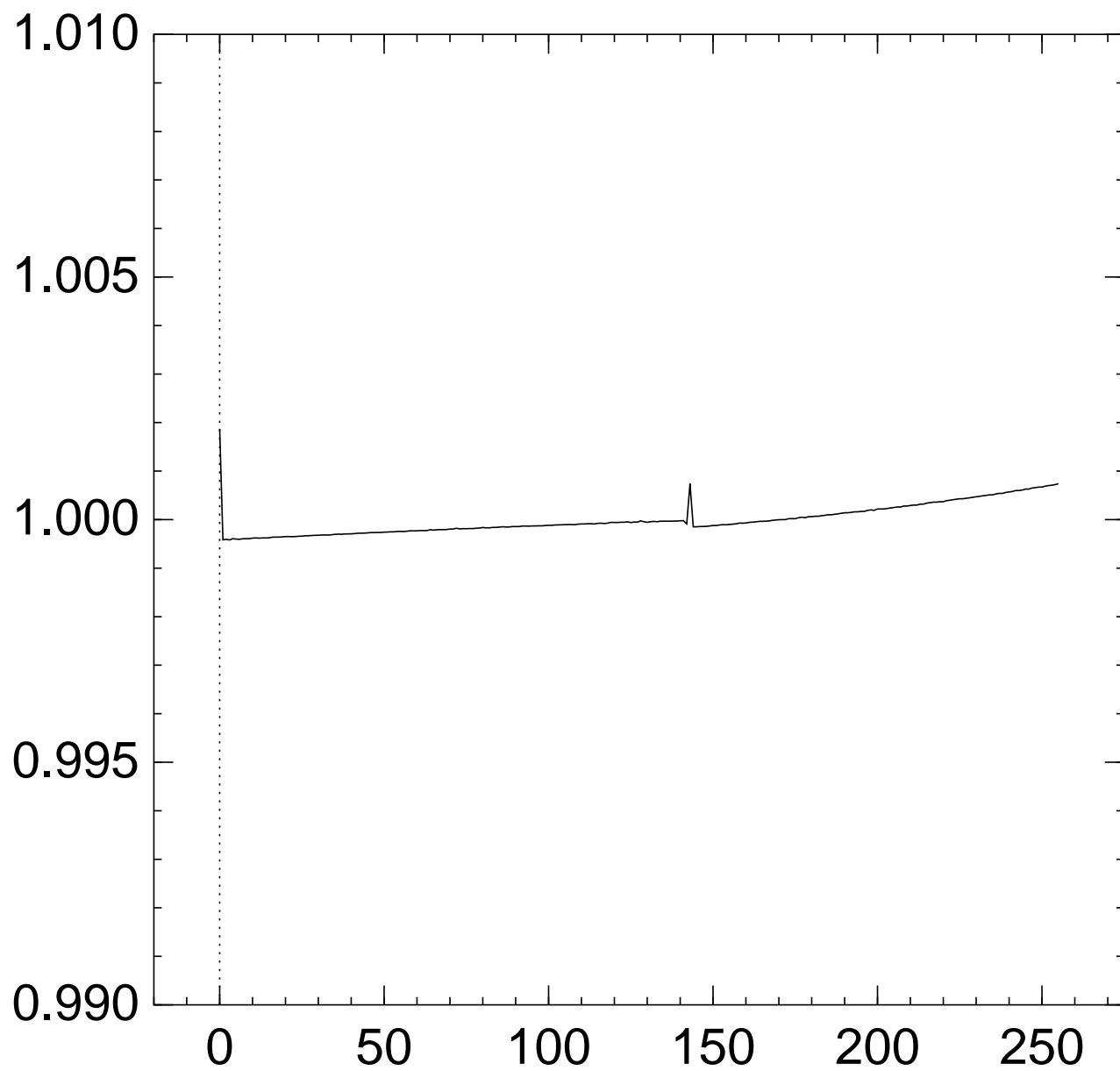
Graph of $256 \Pr[z_{141} = x]$:



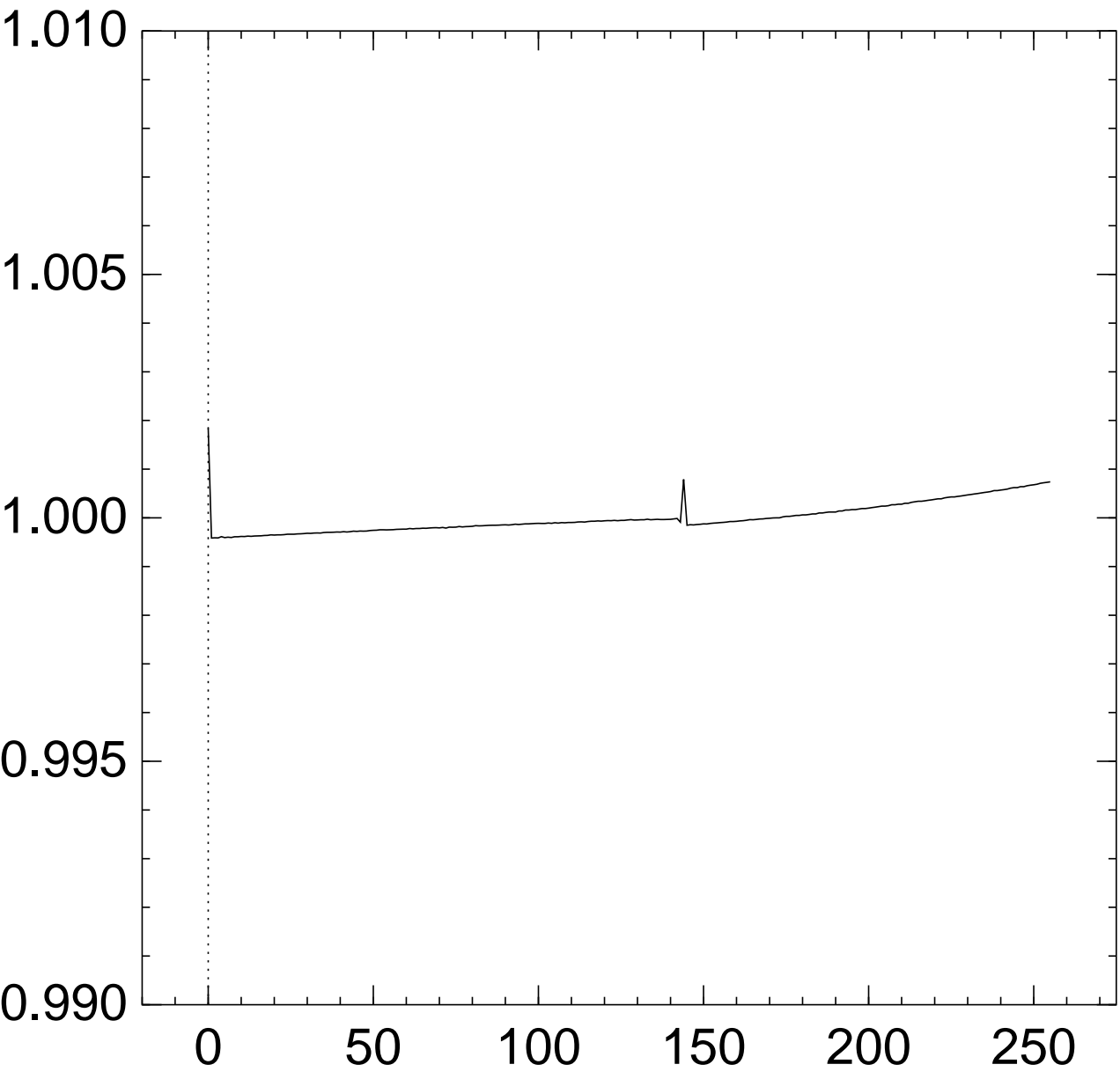
Graph of $256 \Pr[z_{142} = x]$:



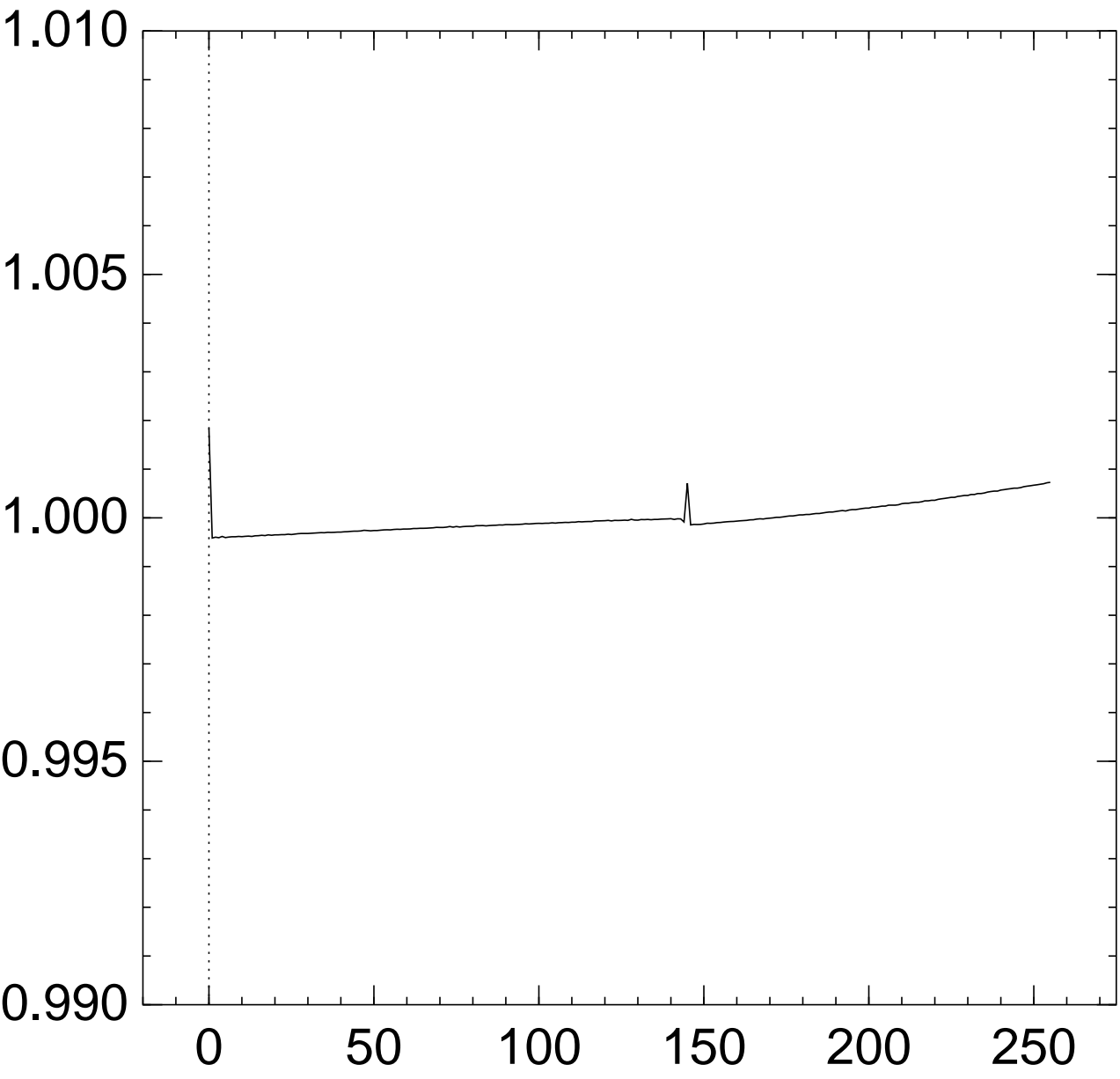
Graph of $256 \Pr[z_{143} = x]$:



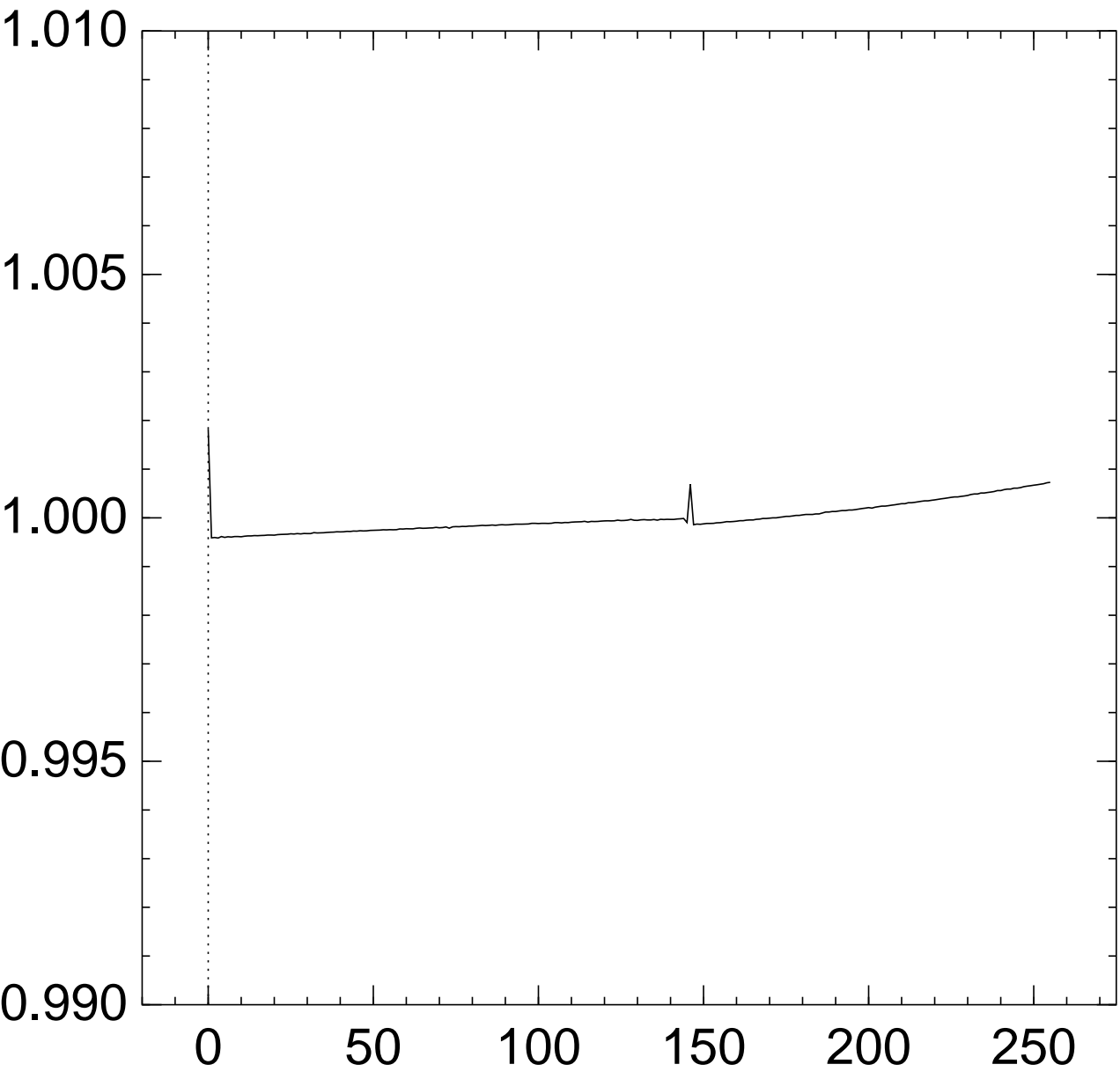
Graph of $256 \Pr[z_{144} = x]$:



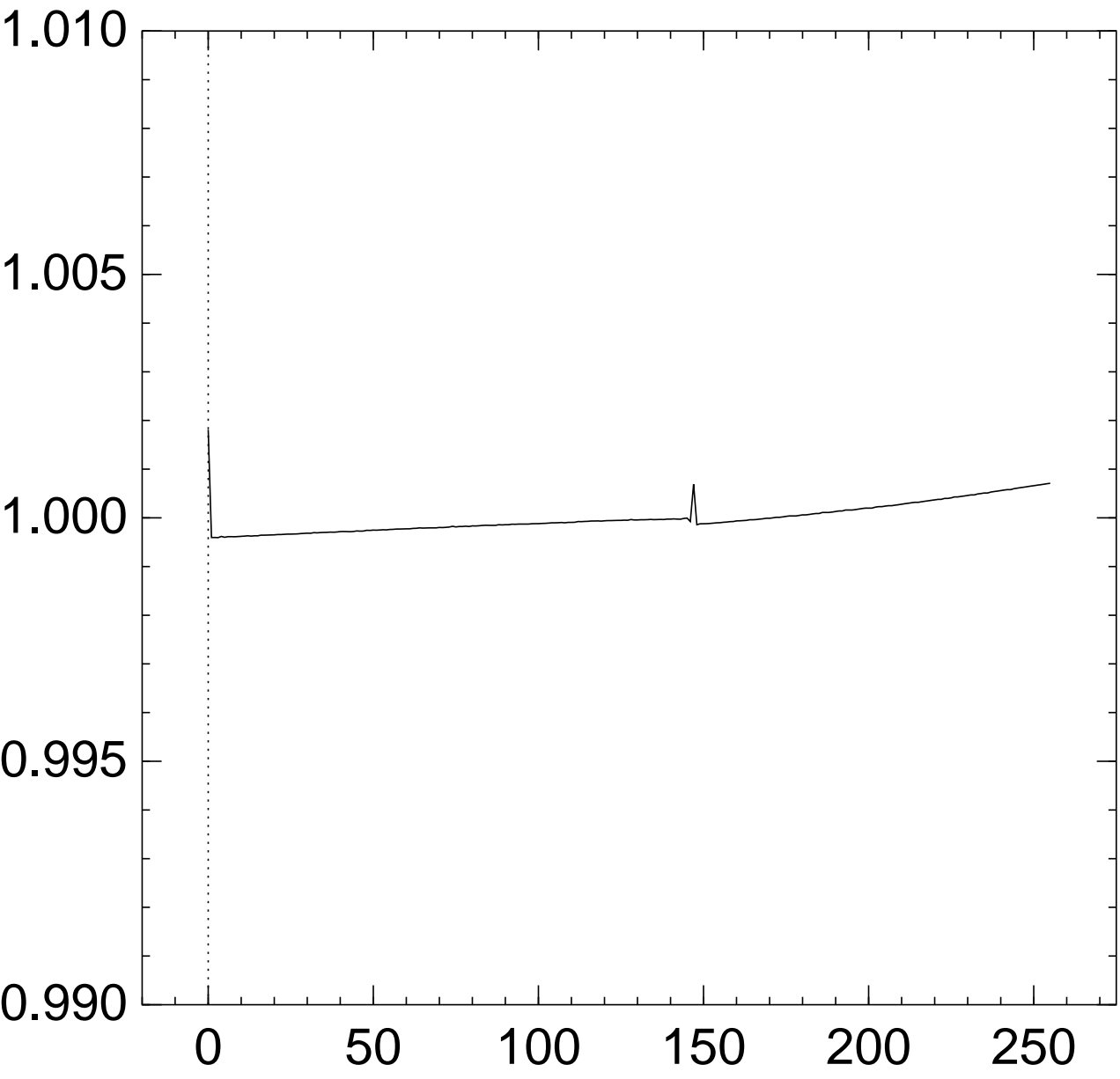
Graph of $256 \Pr[z_{145} = x]$:



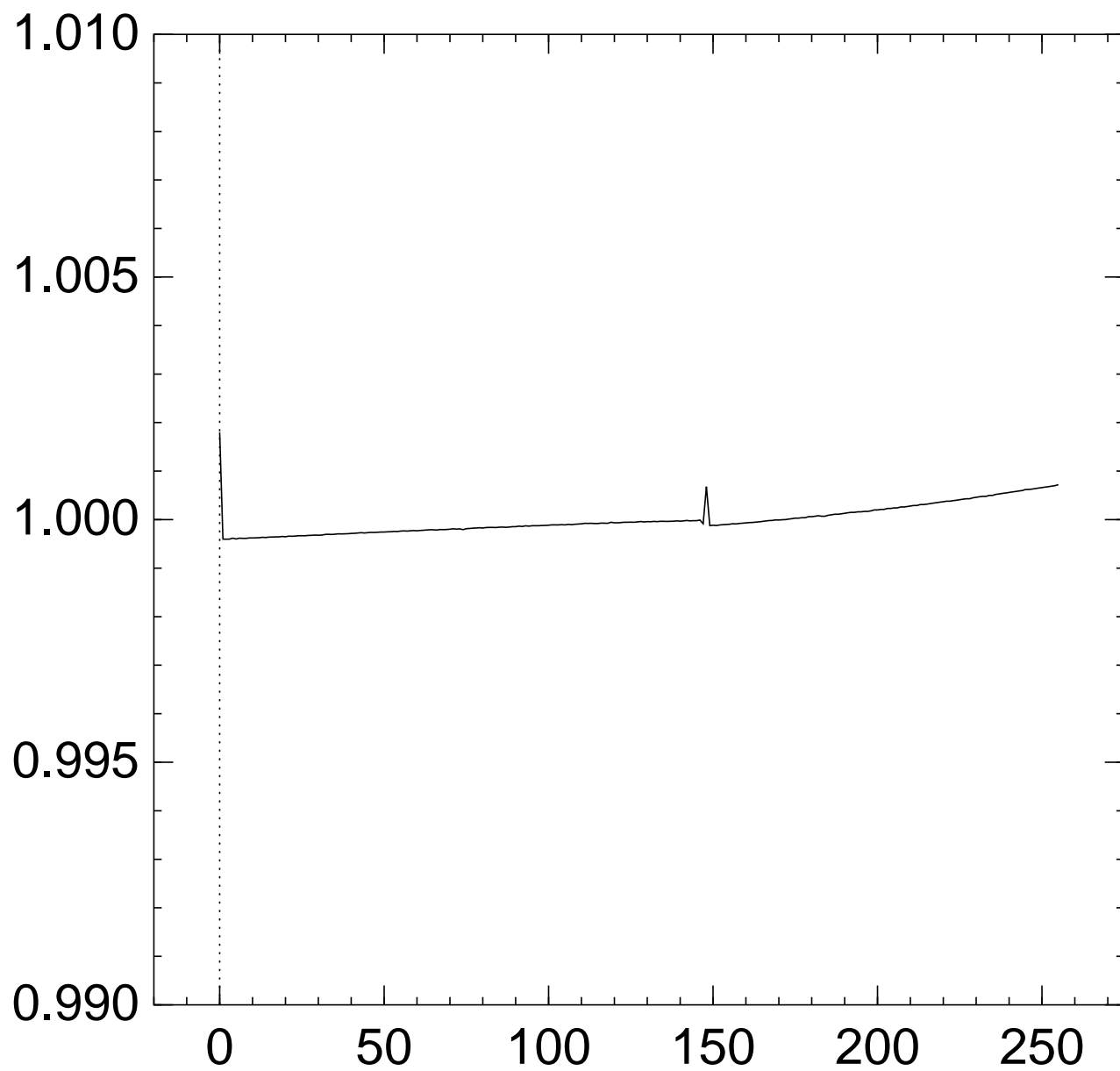
Graph of $256 \Pr[z_{146} = x]$:



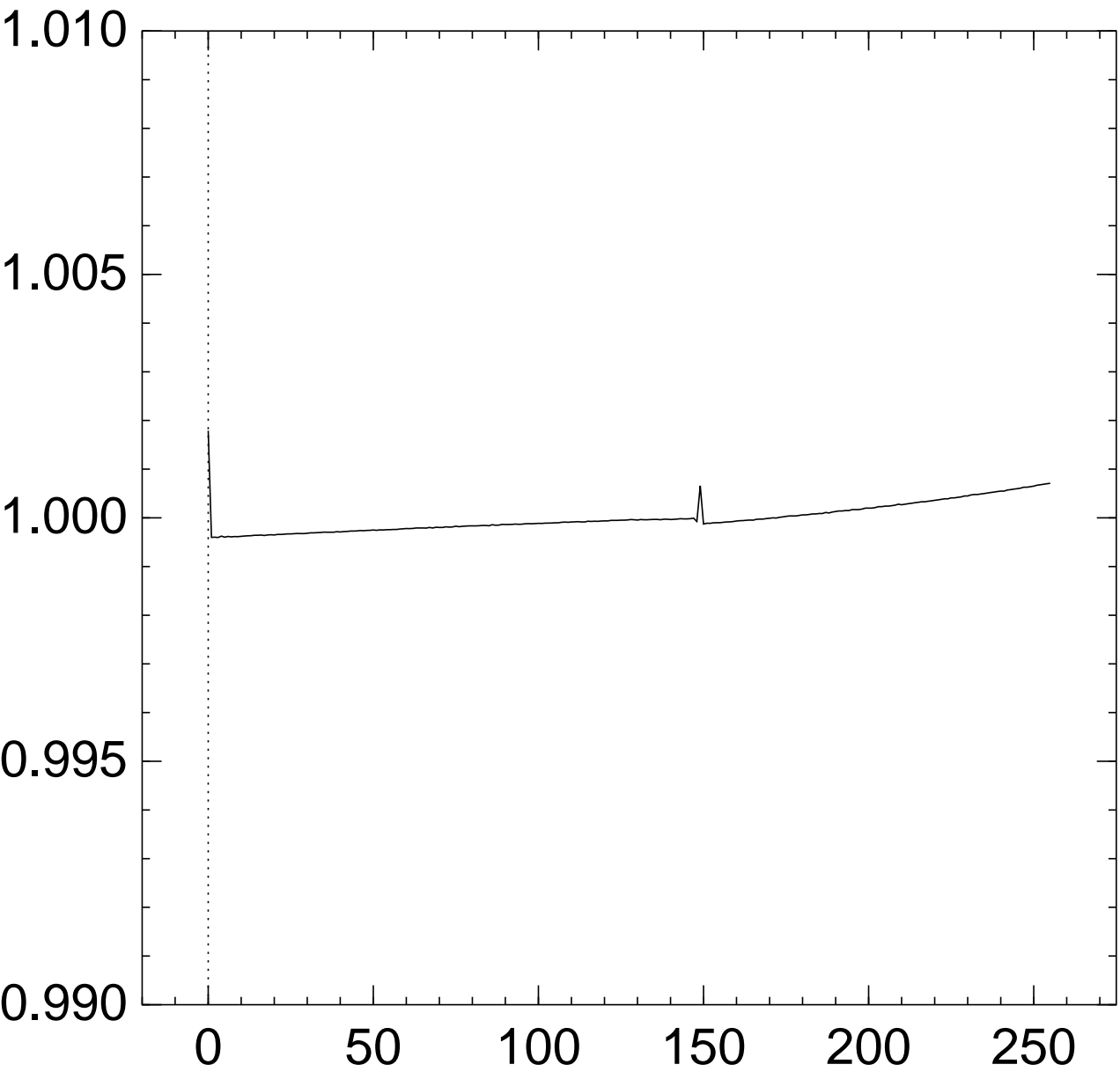
Graph of $256 \Pr[z_{147} = x]$:



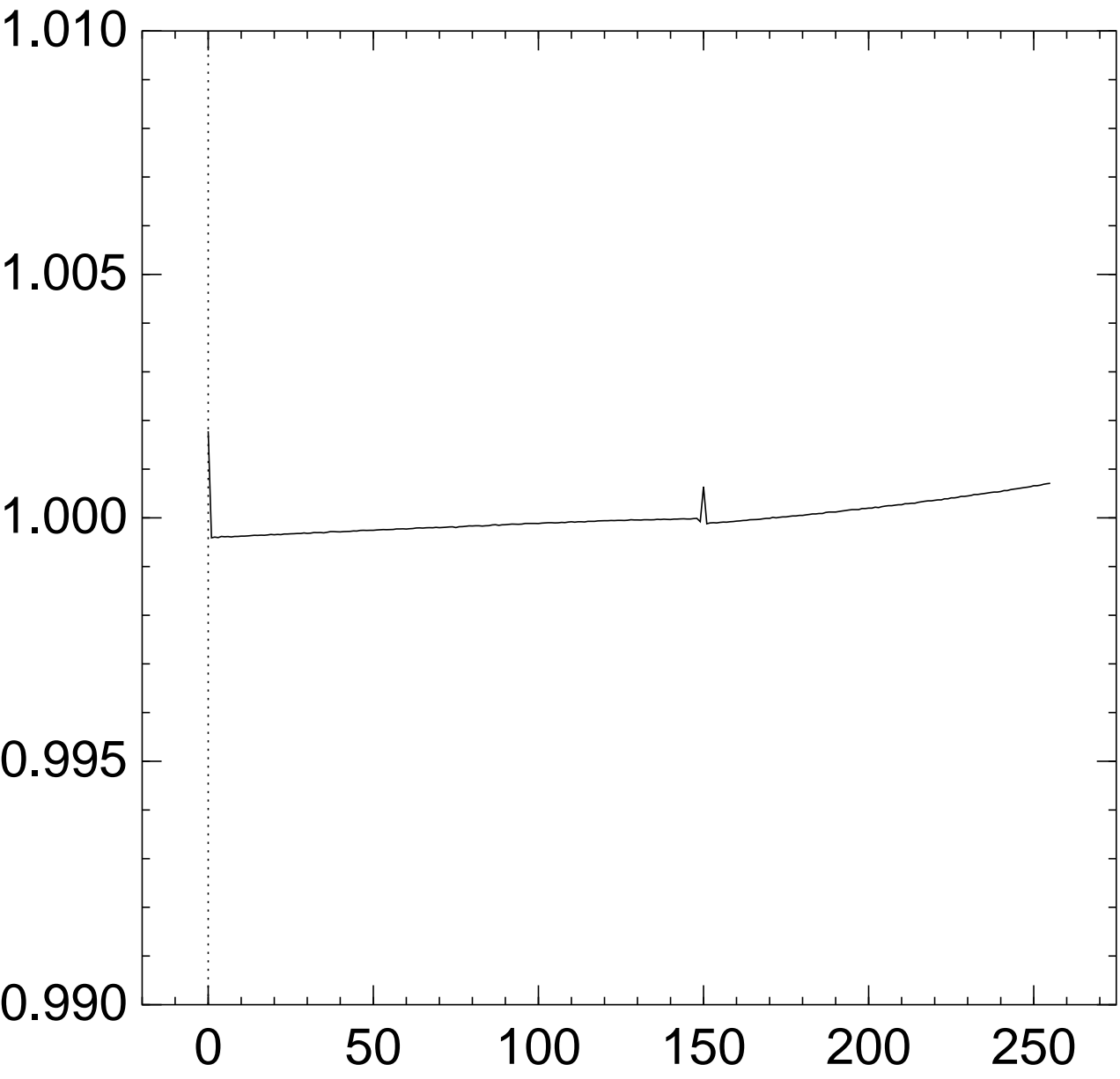
Graph of $256 \Pr[z_{148} = x]$:



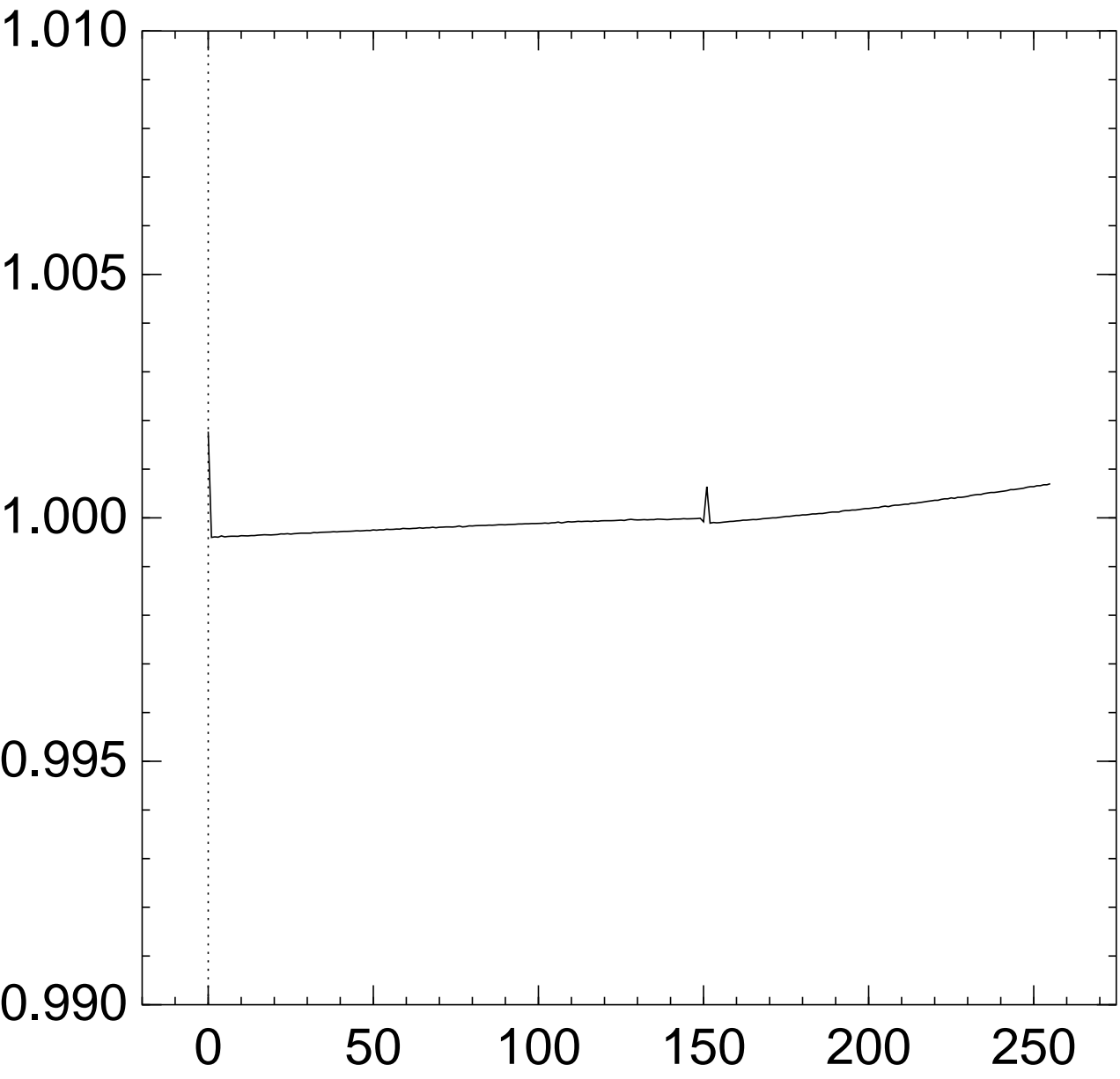
Graph of $256 \Pr[z_{149} = x]$:



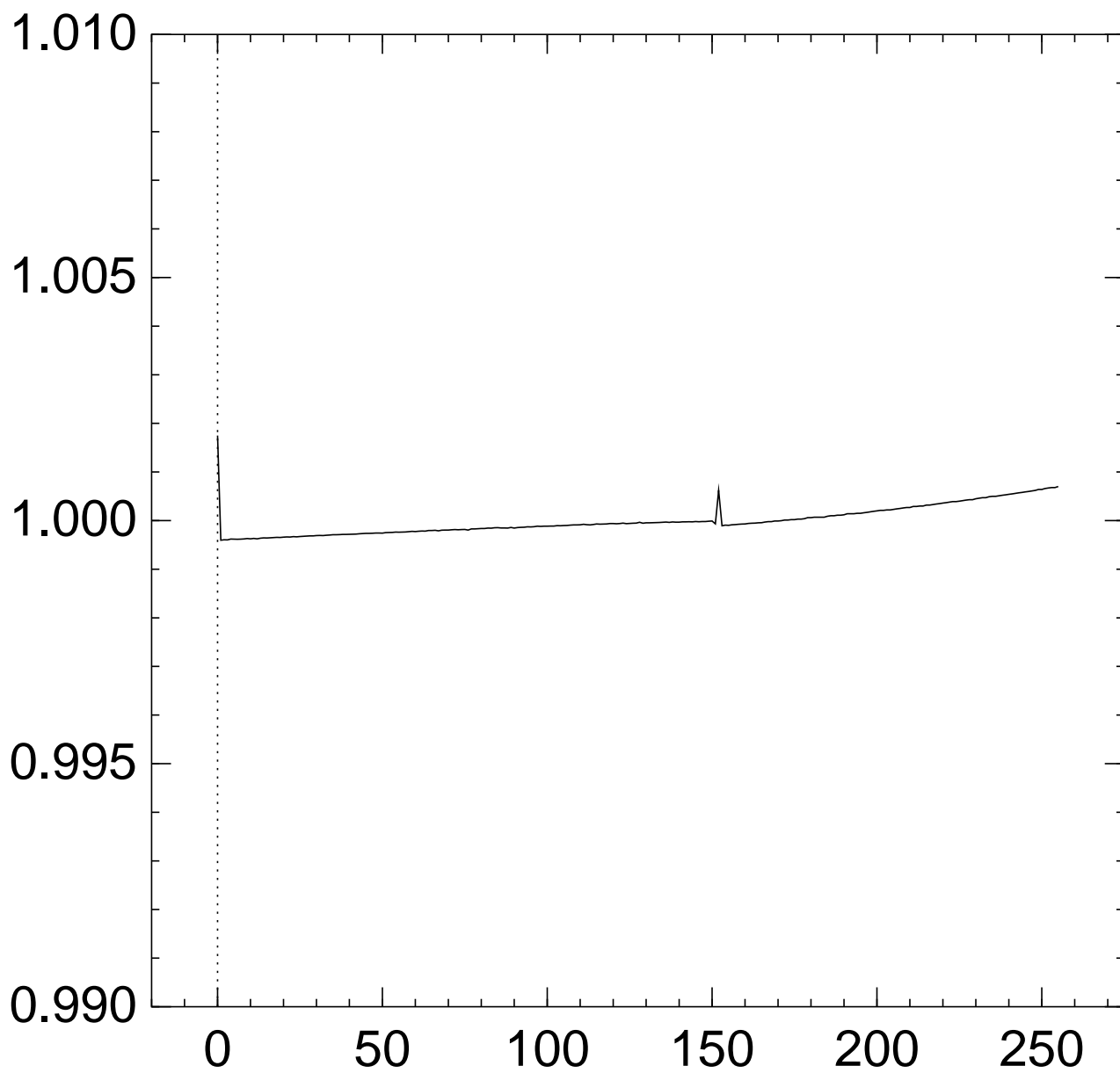
Graph of $256 \Pr[z_{150} = x]$:



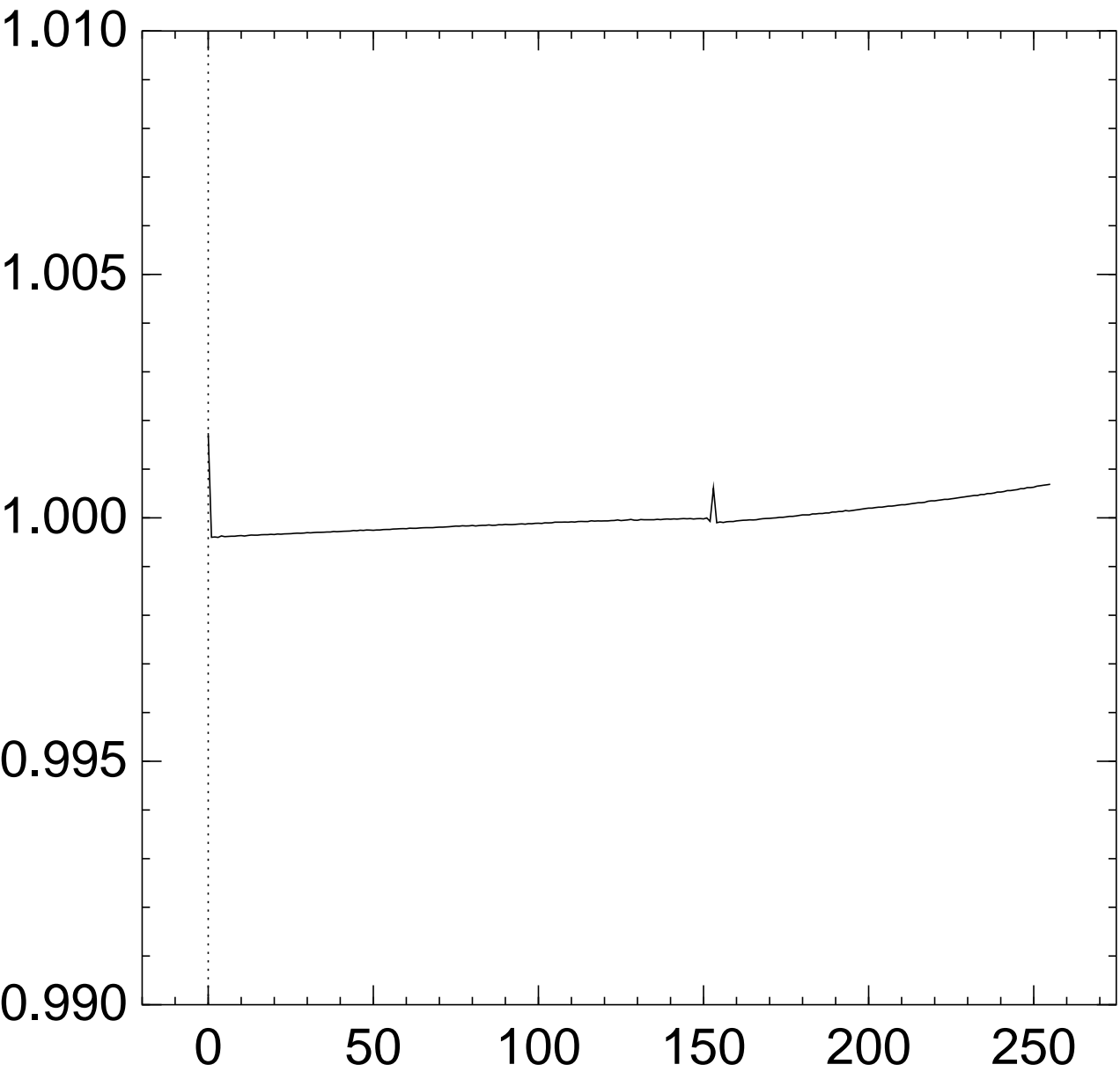
Graph of $256 \Pr[z_{151} = x]$:



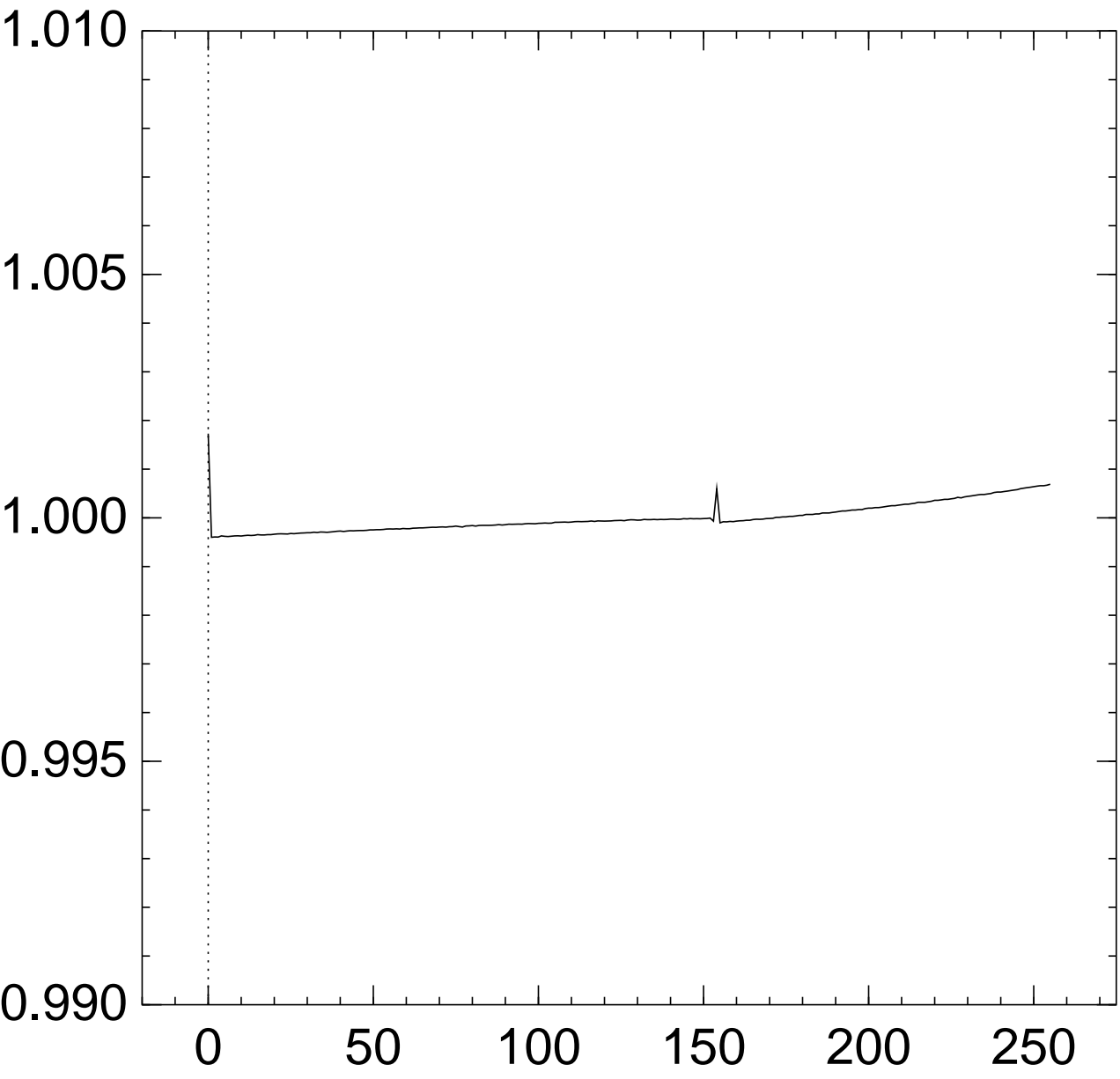
Graph of $256 \Pr[z_{152} = x]$:



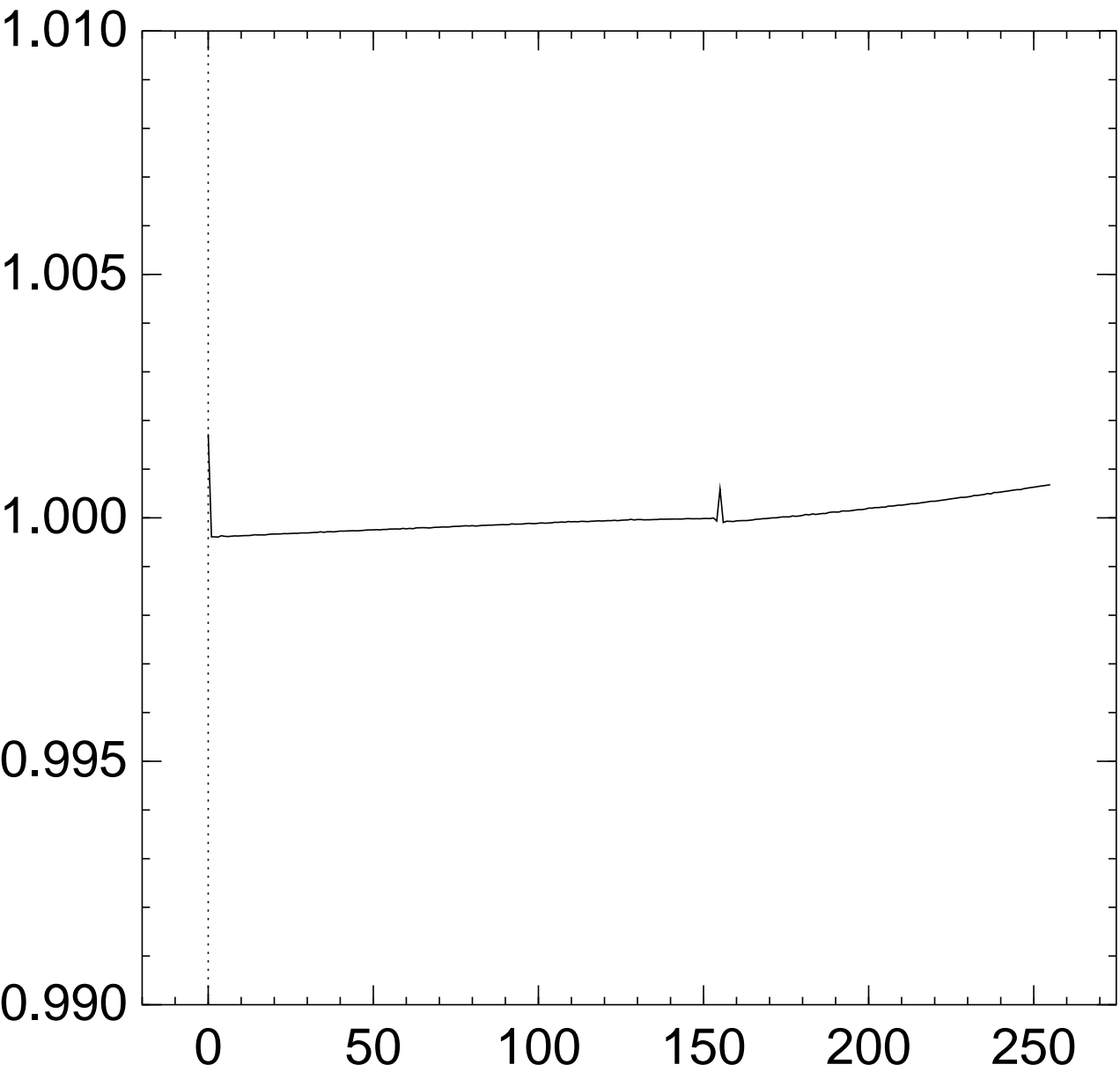
Graph of $256 \Pr[z_{153} = x]$:



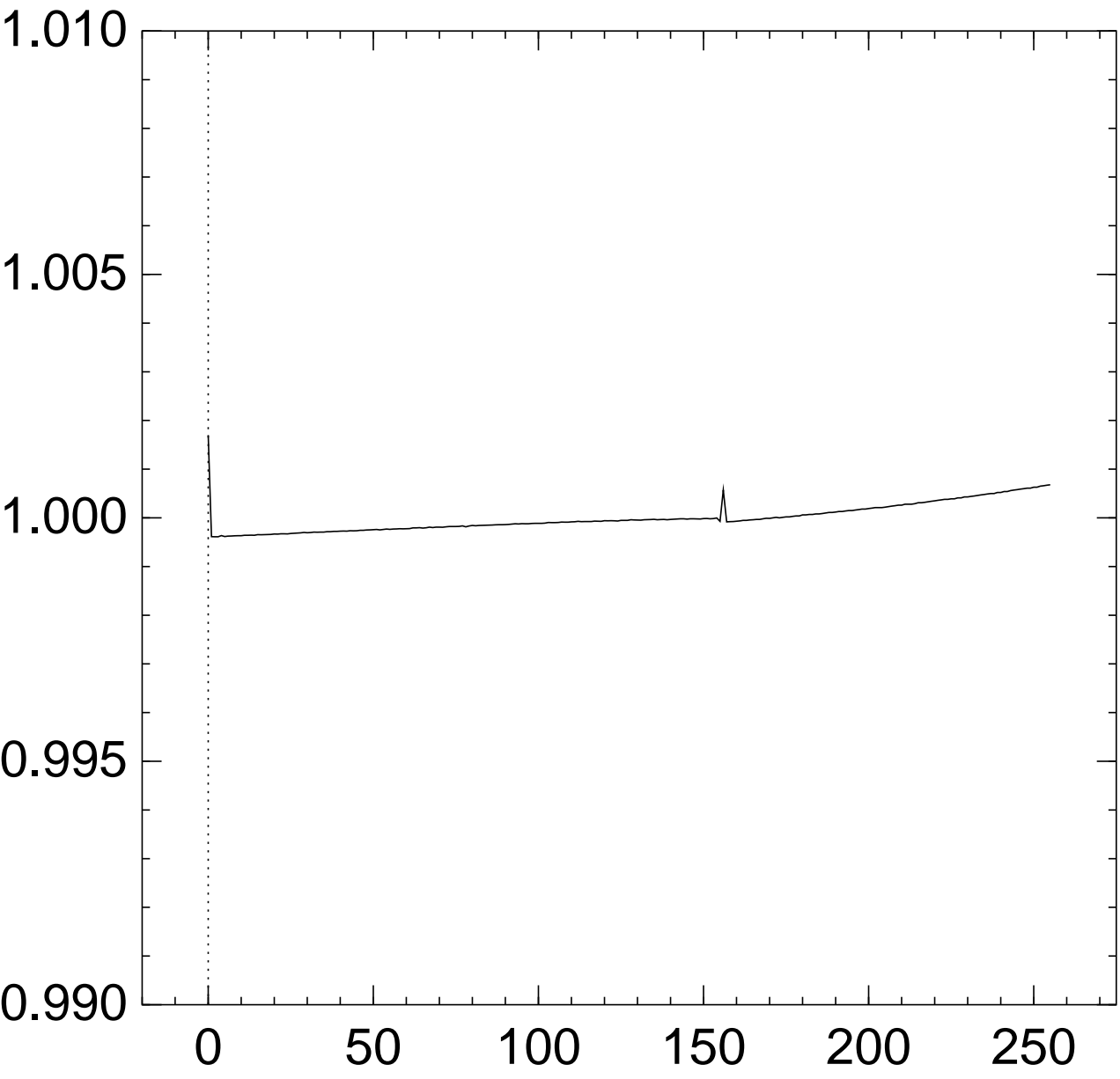
Graph of $256 \Pr[z_{154} = x]$:



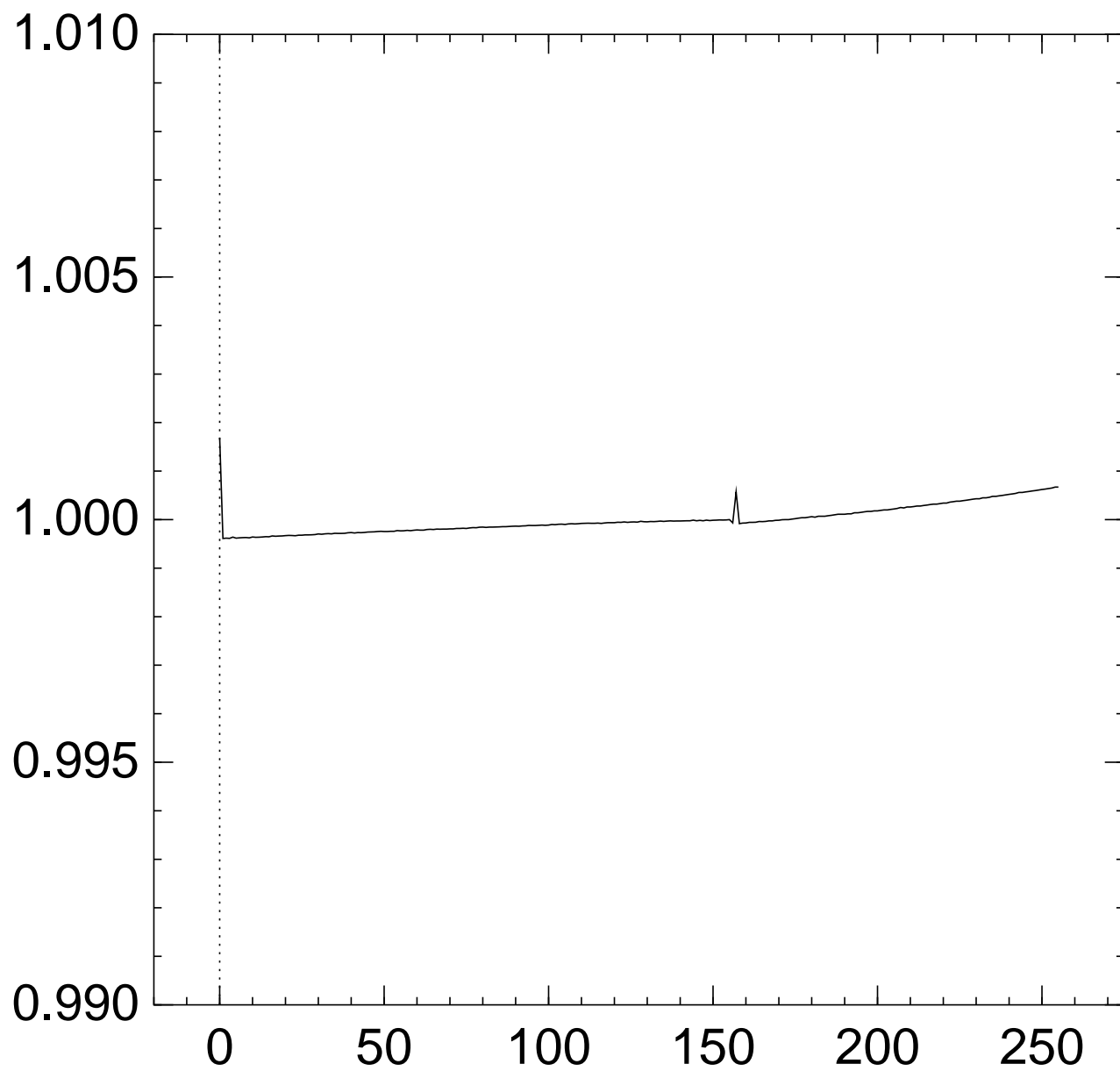
Graph of $256 \Pr[z_{155} = x]$:



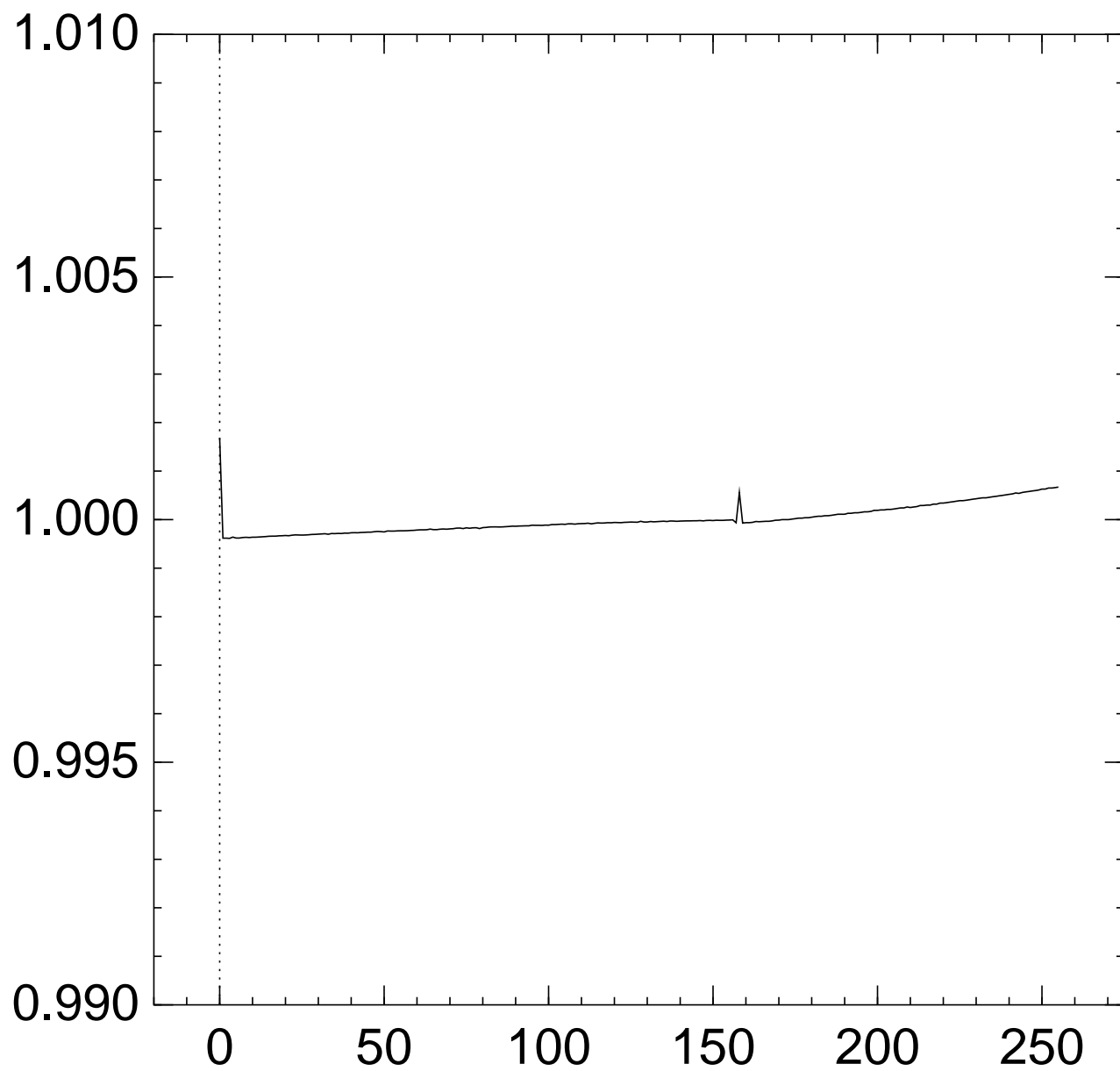
Graph of $256 \Pr[z_{156} = x]$:



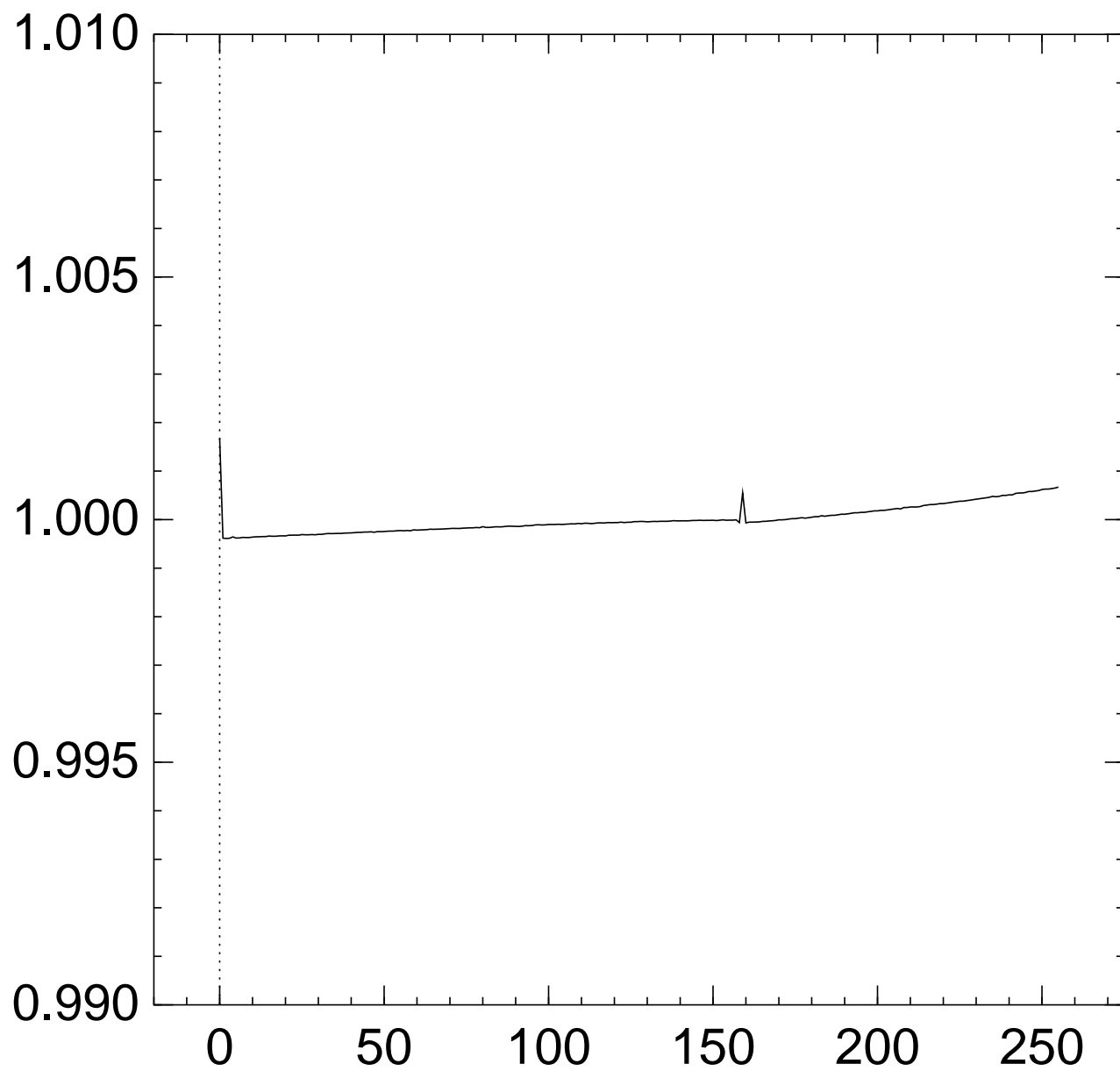
Graph of $256 \Pr[z_{157} = x]$:



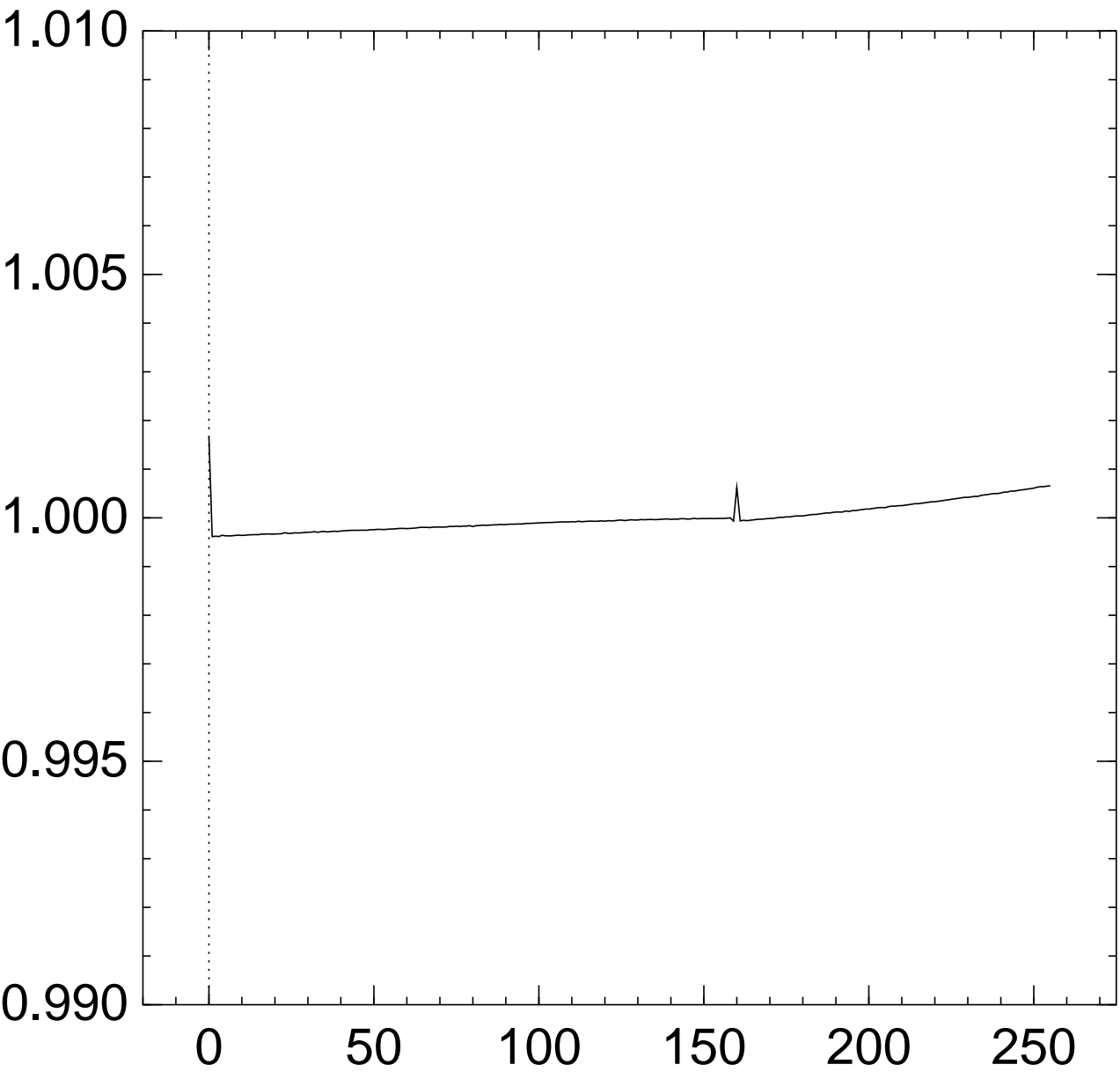
Graph of $256 \Pr[z_{158} = x]$:



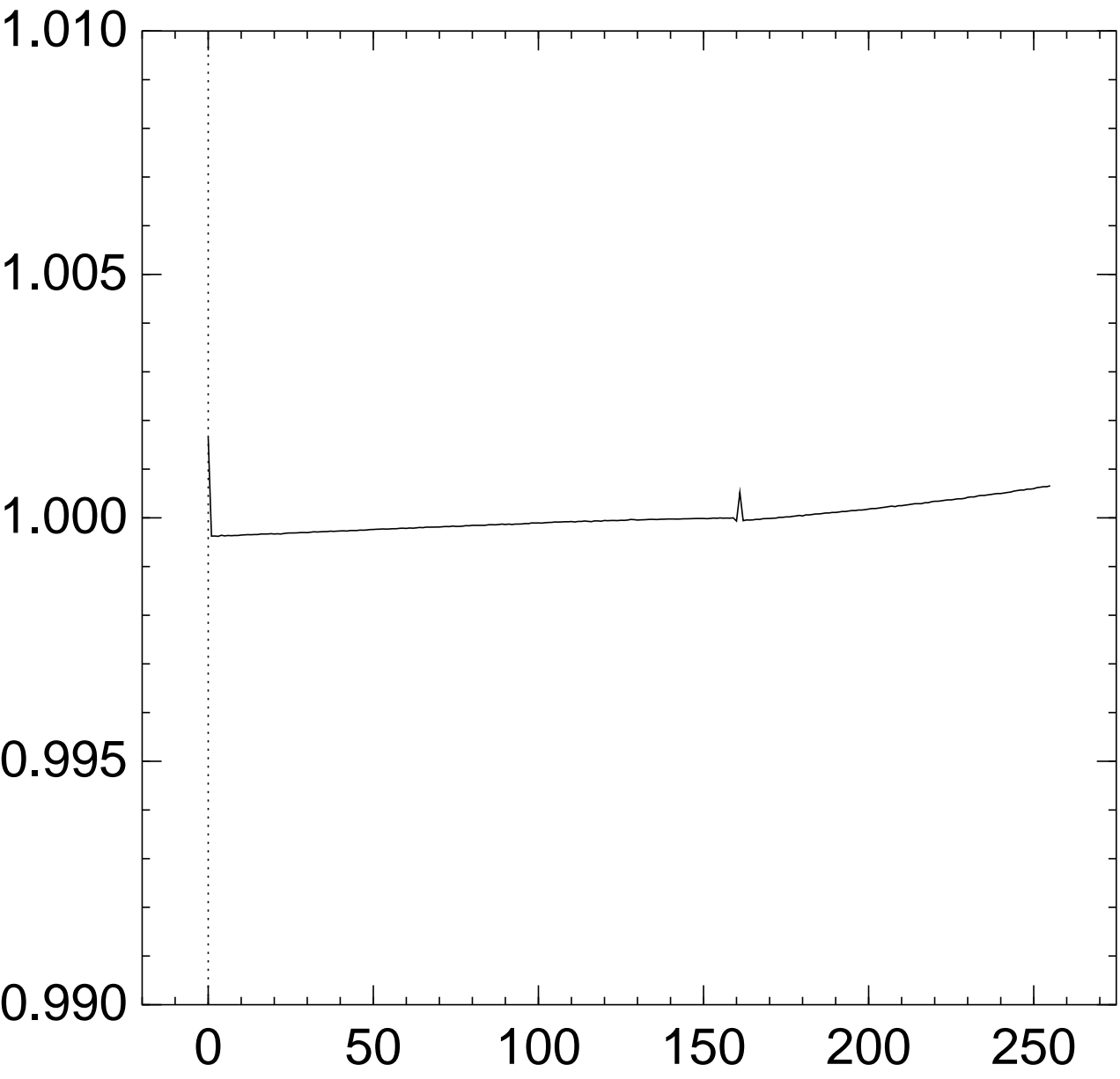
Graph of $256 \Pr[z_{159} = x]$:



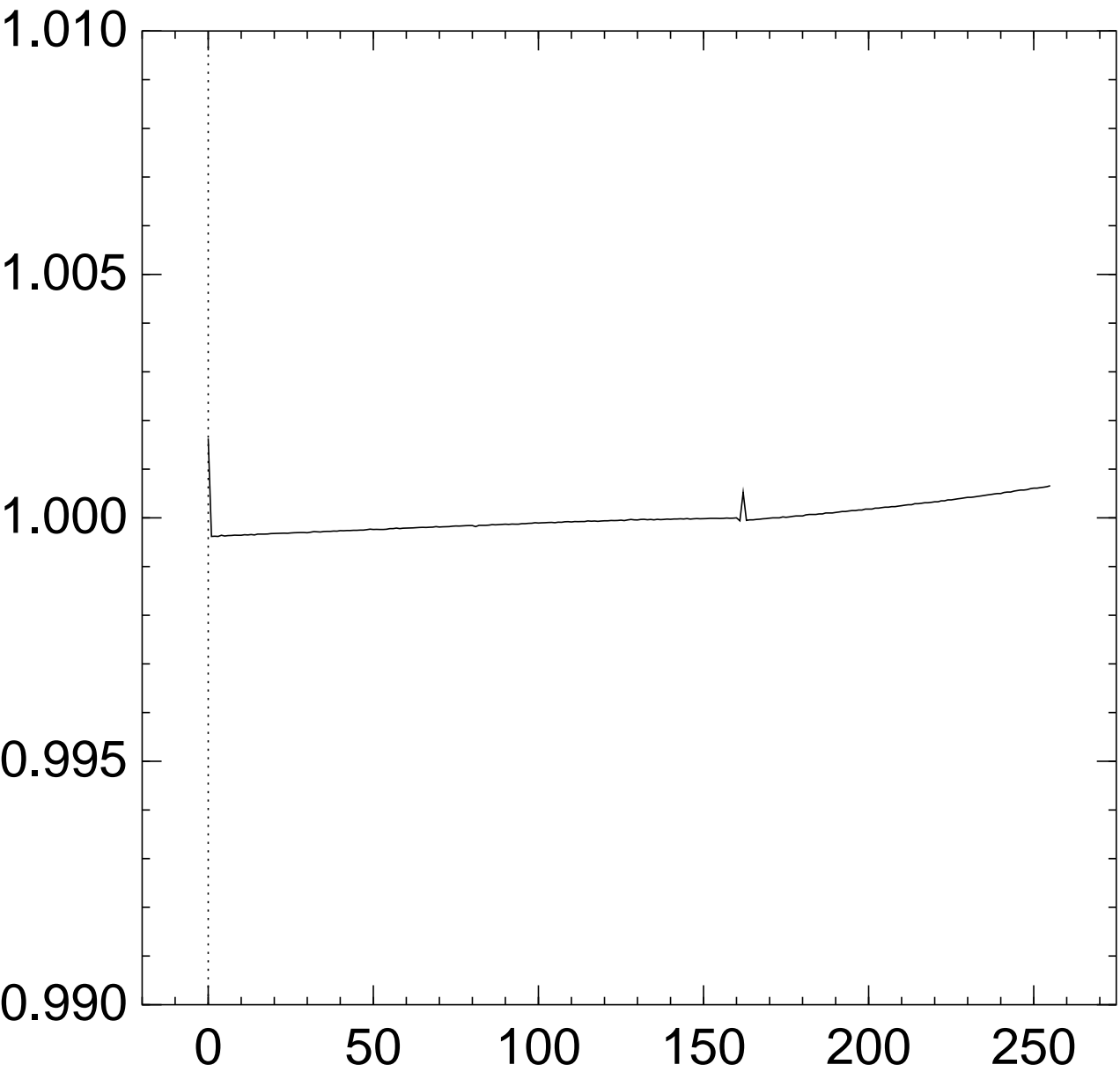
Graph of $256 \Pr[z_{160} = x]$:



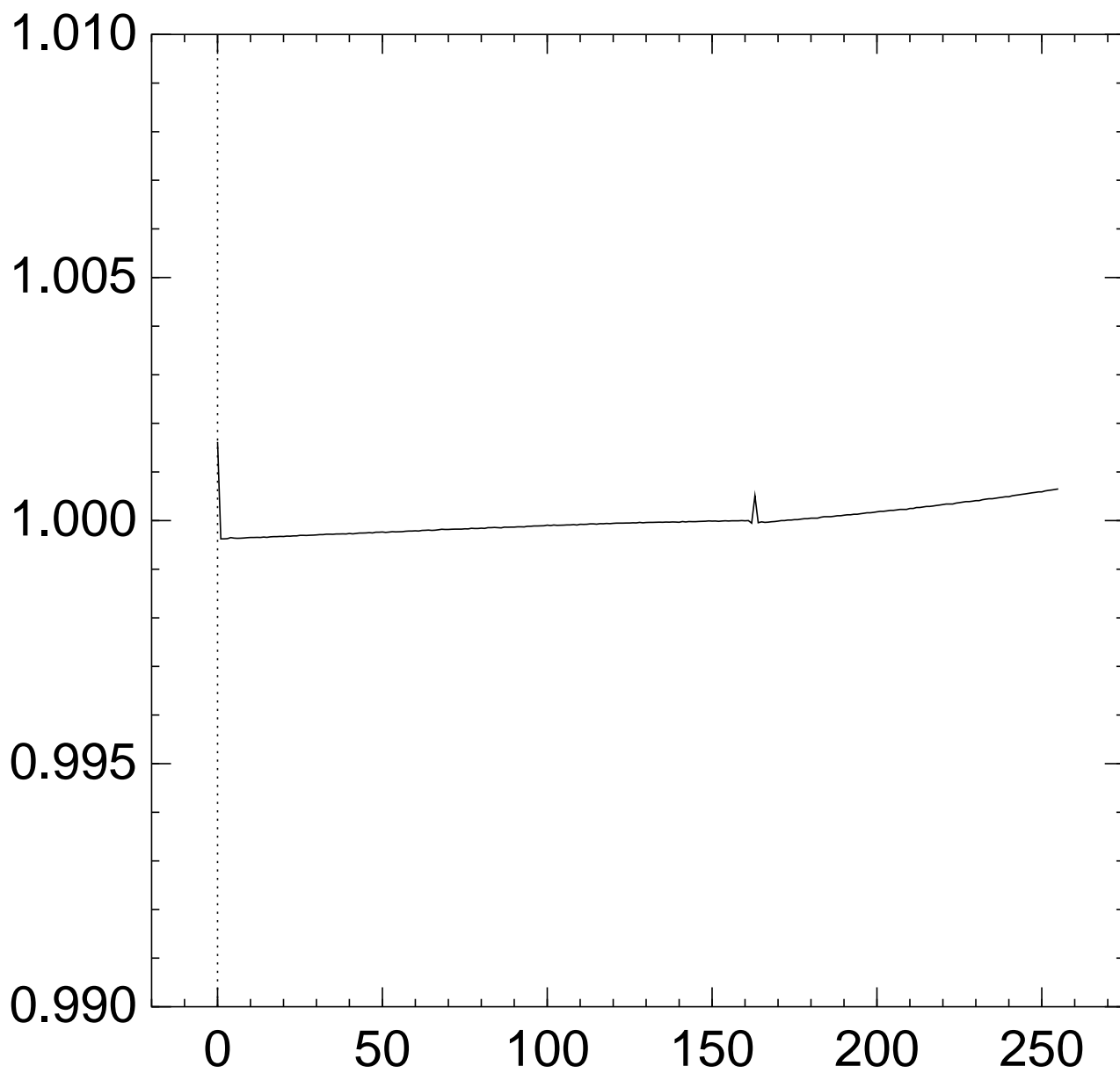
Graph of $256 \Pr[z_{161} = x]$:



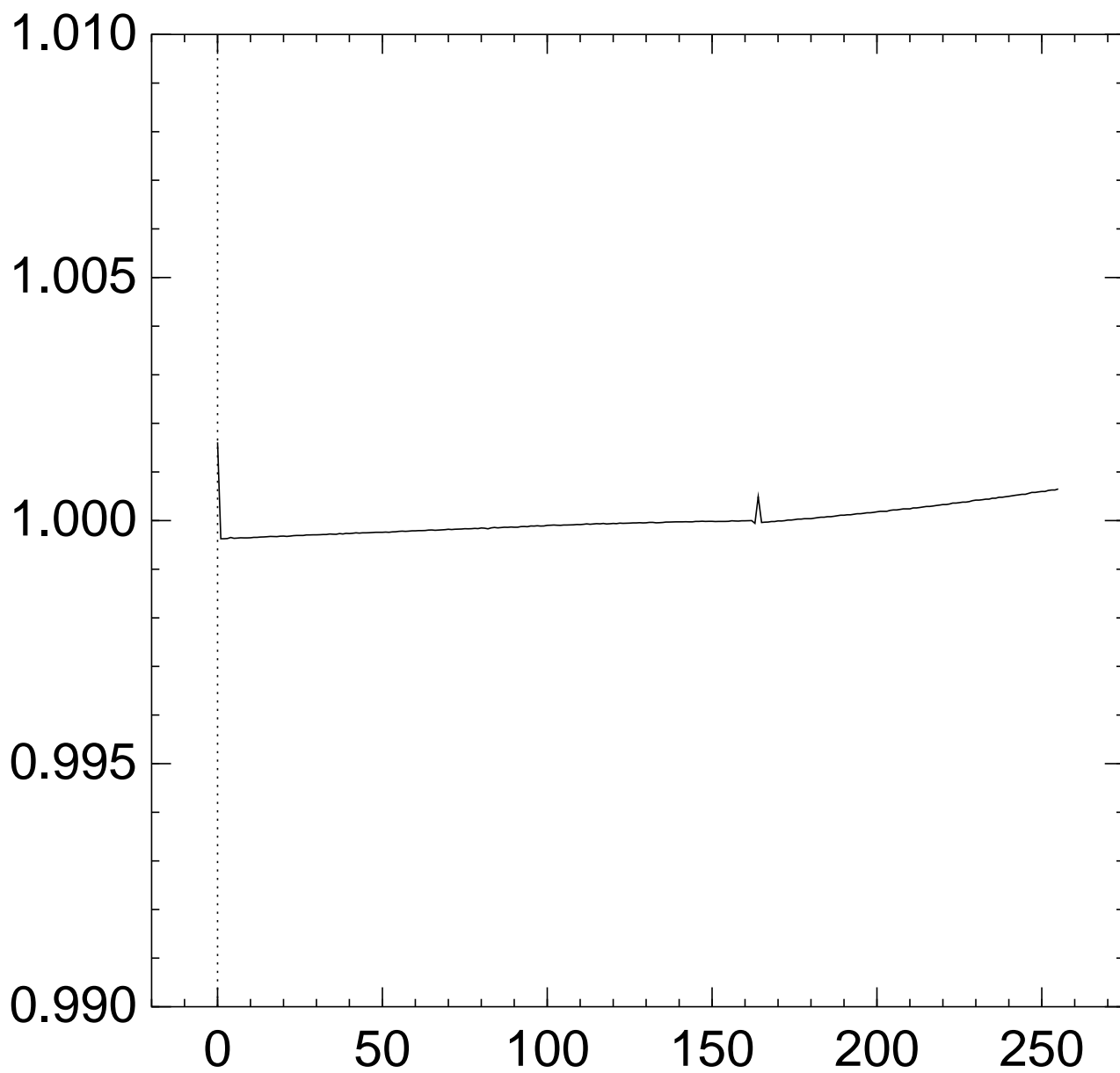
Graph of $256 \Pr[z_{162} = x]$:



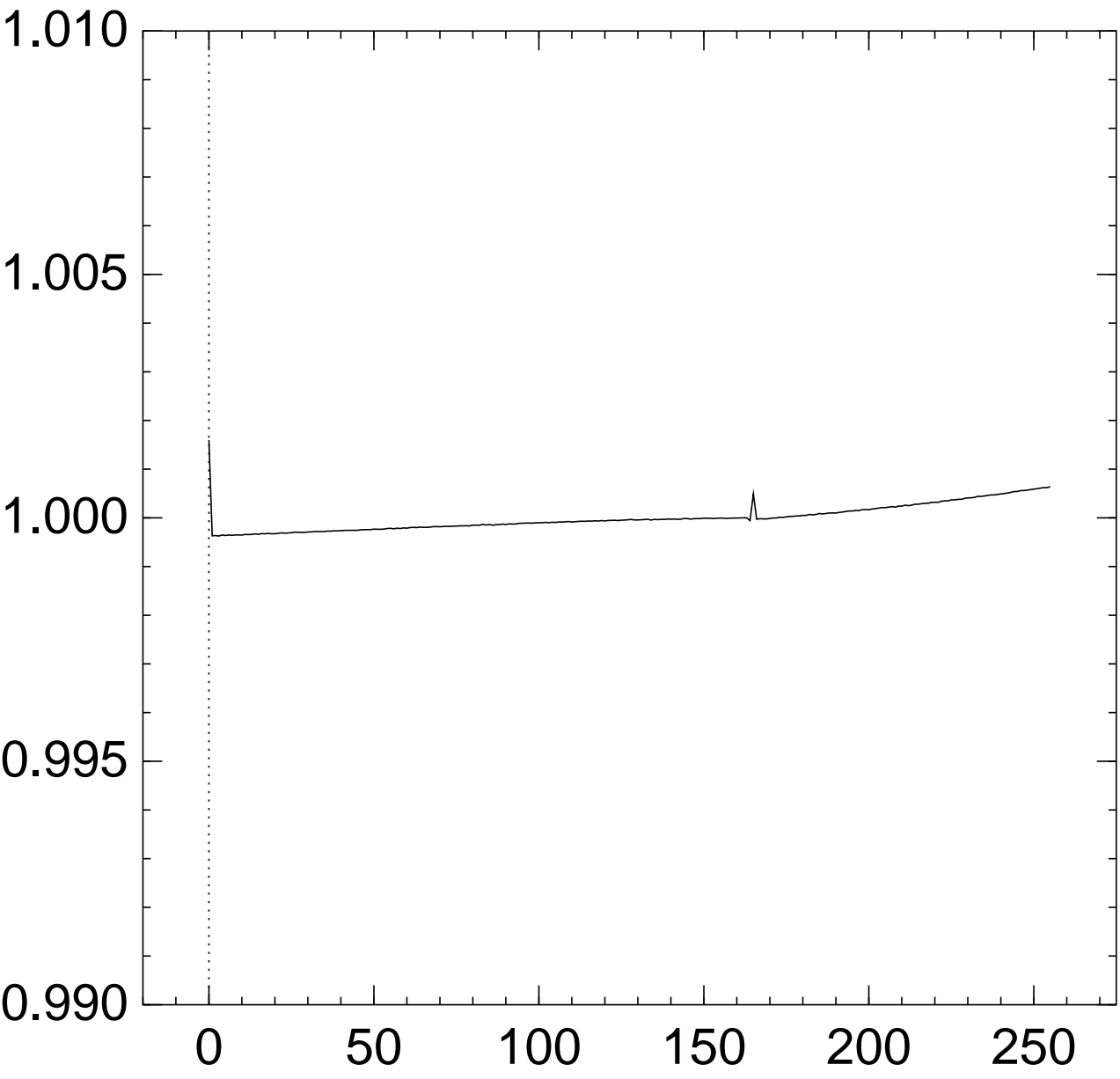
Graph of $256 \Pr[z_{163} = x]$:



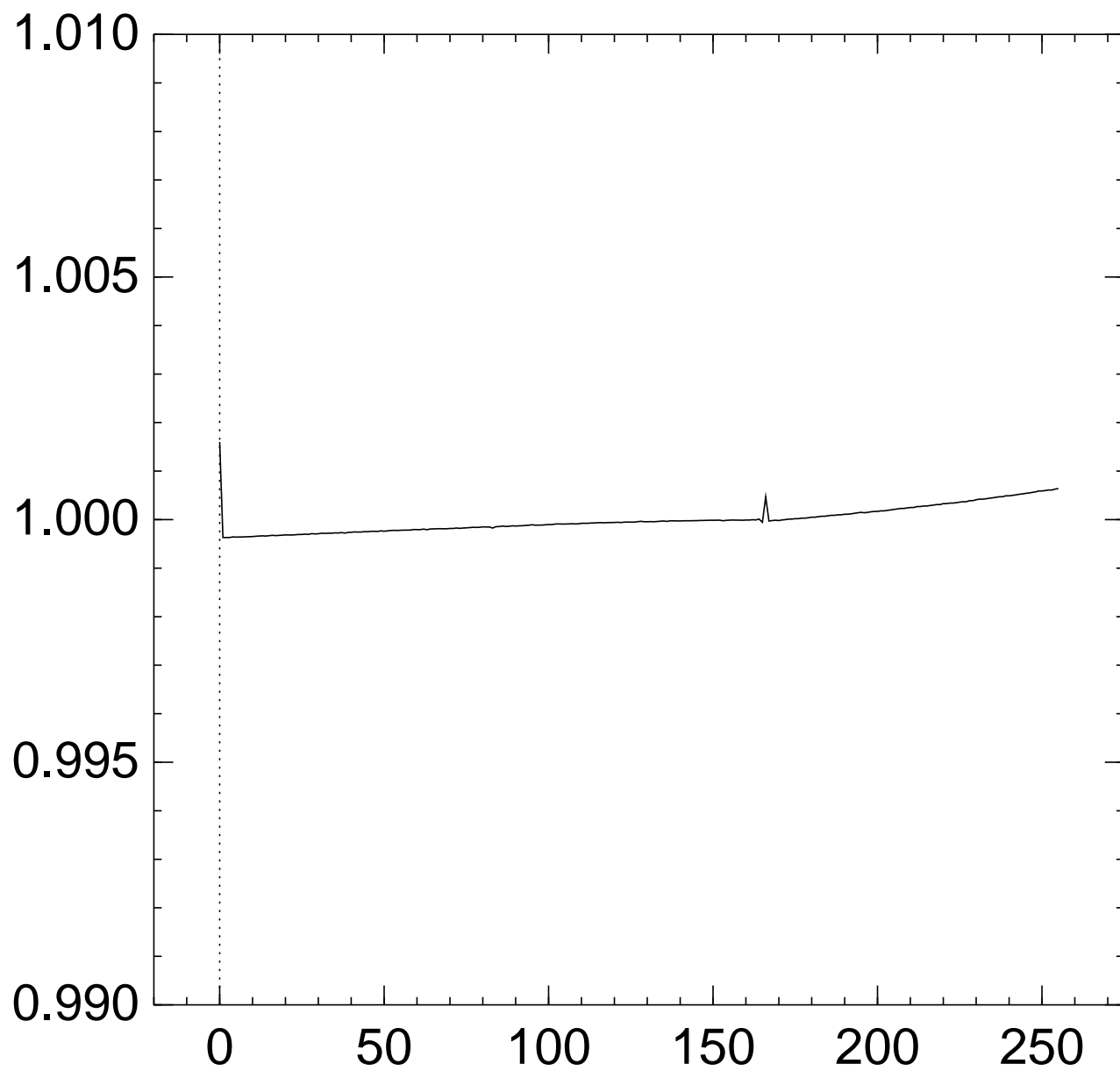
Graph of $256 \Pr[z_{164} = x]$:



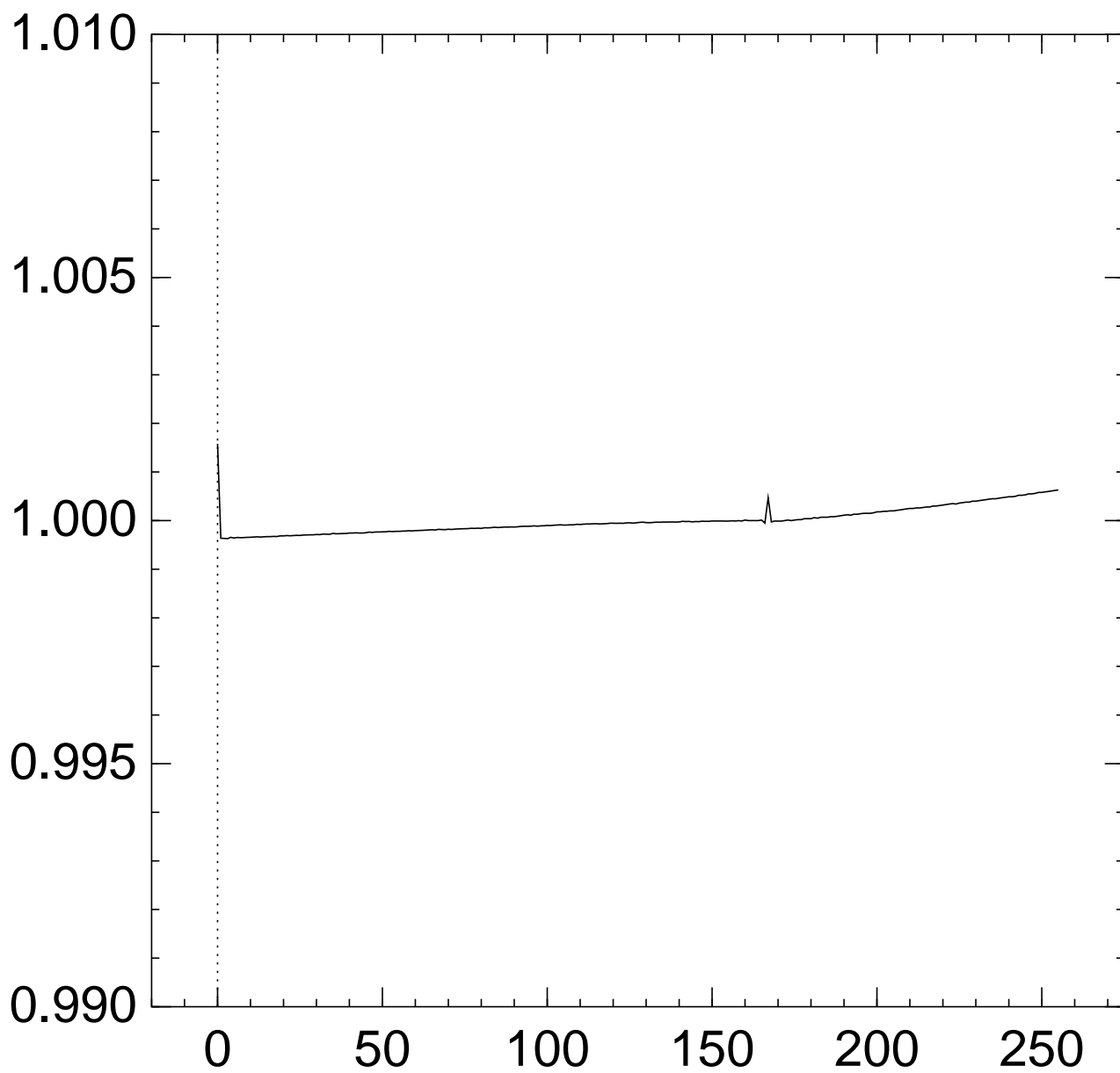
Graph of $256 \Pr[z_{165} = x]$:



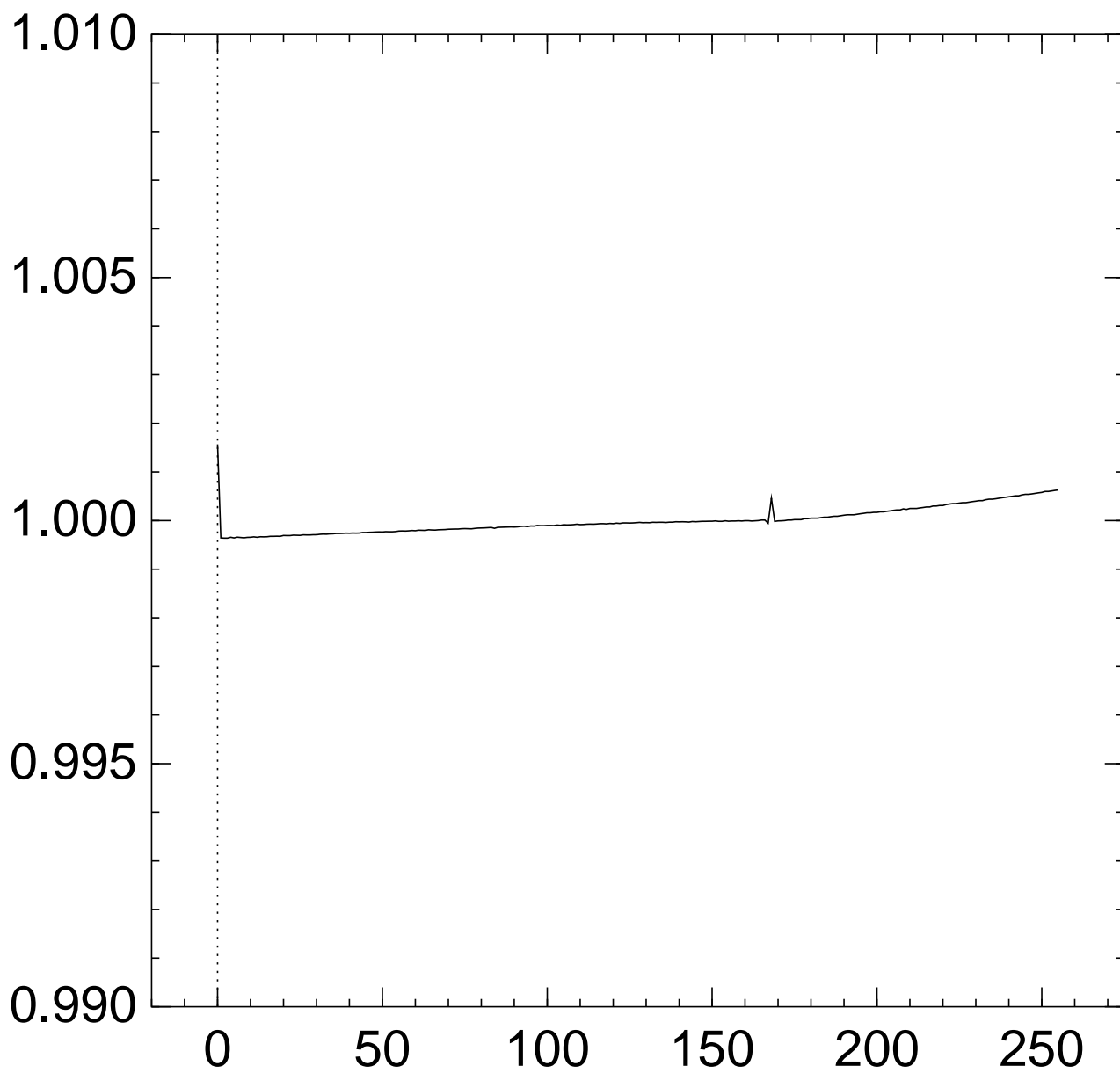
Graph of $256 \Pr[z_{166} = x]$:



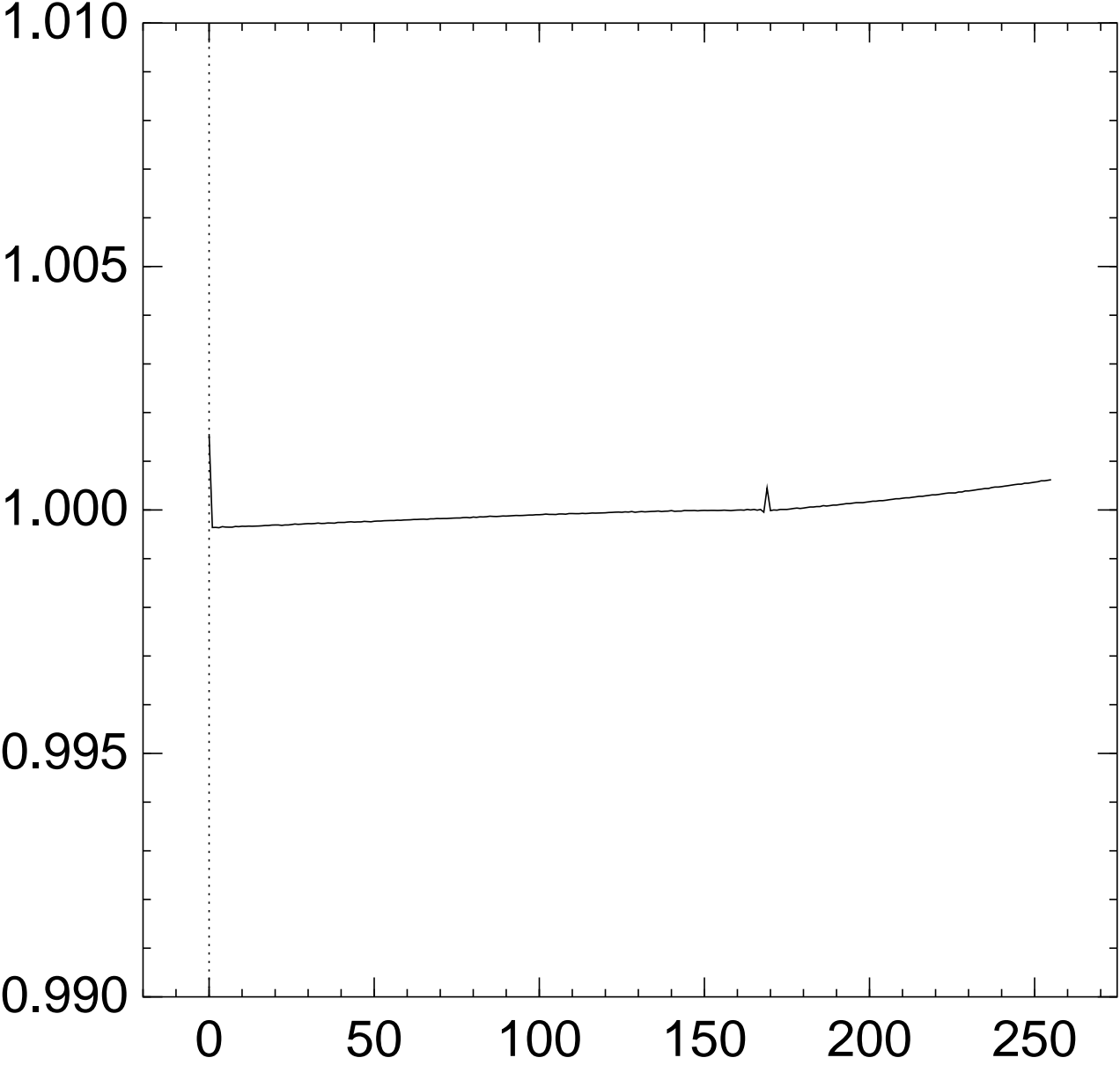
Graph of $256 \Pr[z_{167} = x]$:



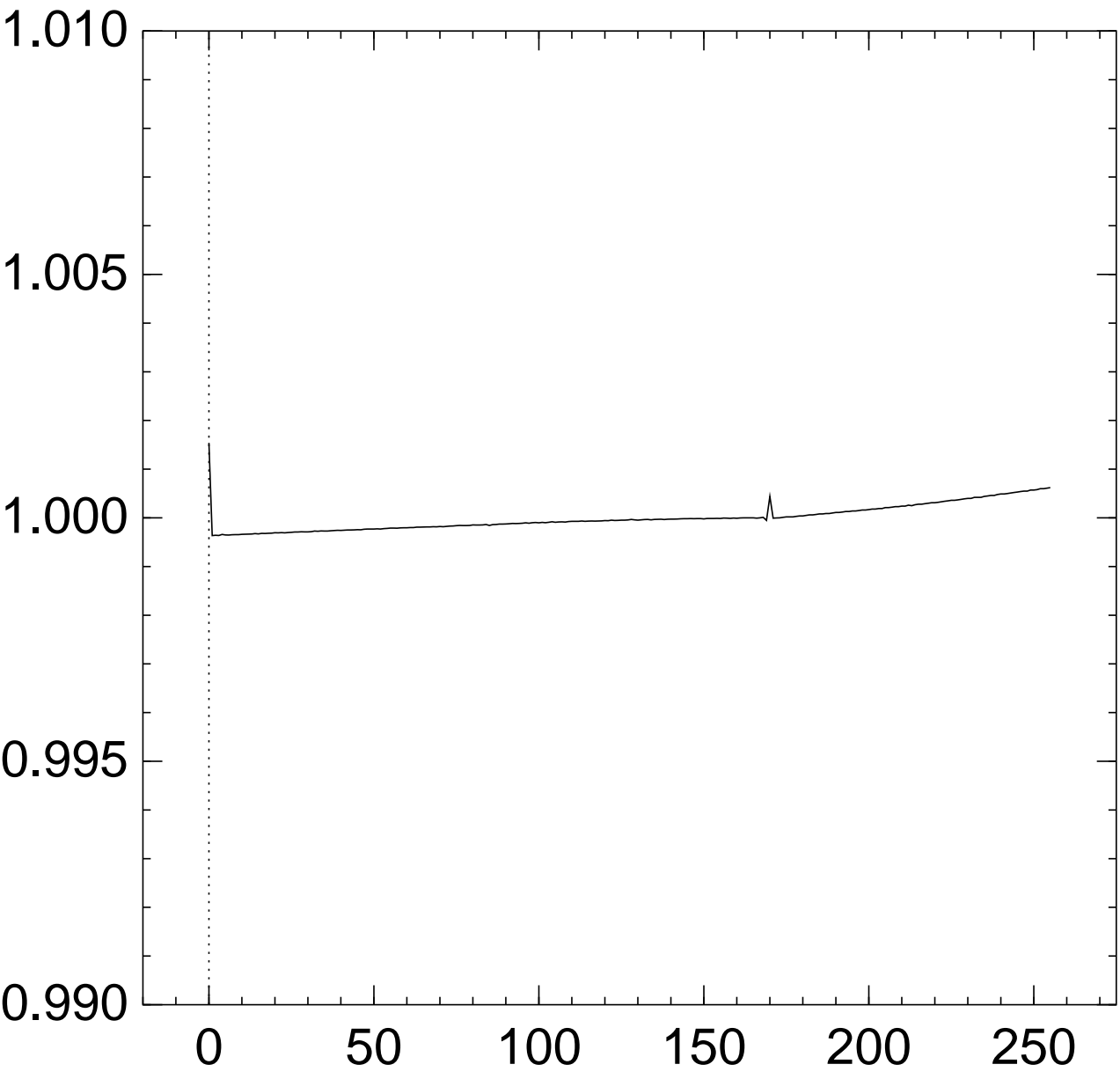
Graph of $256 \Pr[z_{168} = x]$:



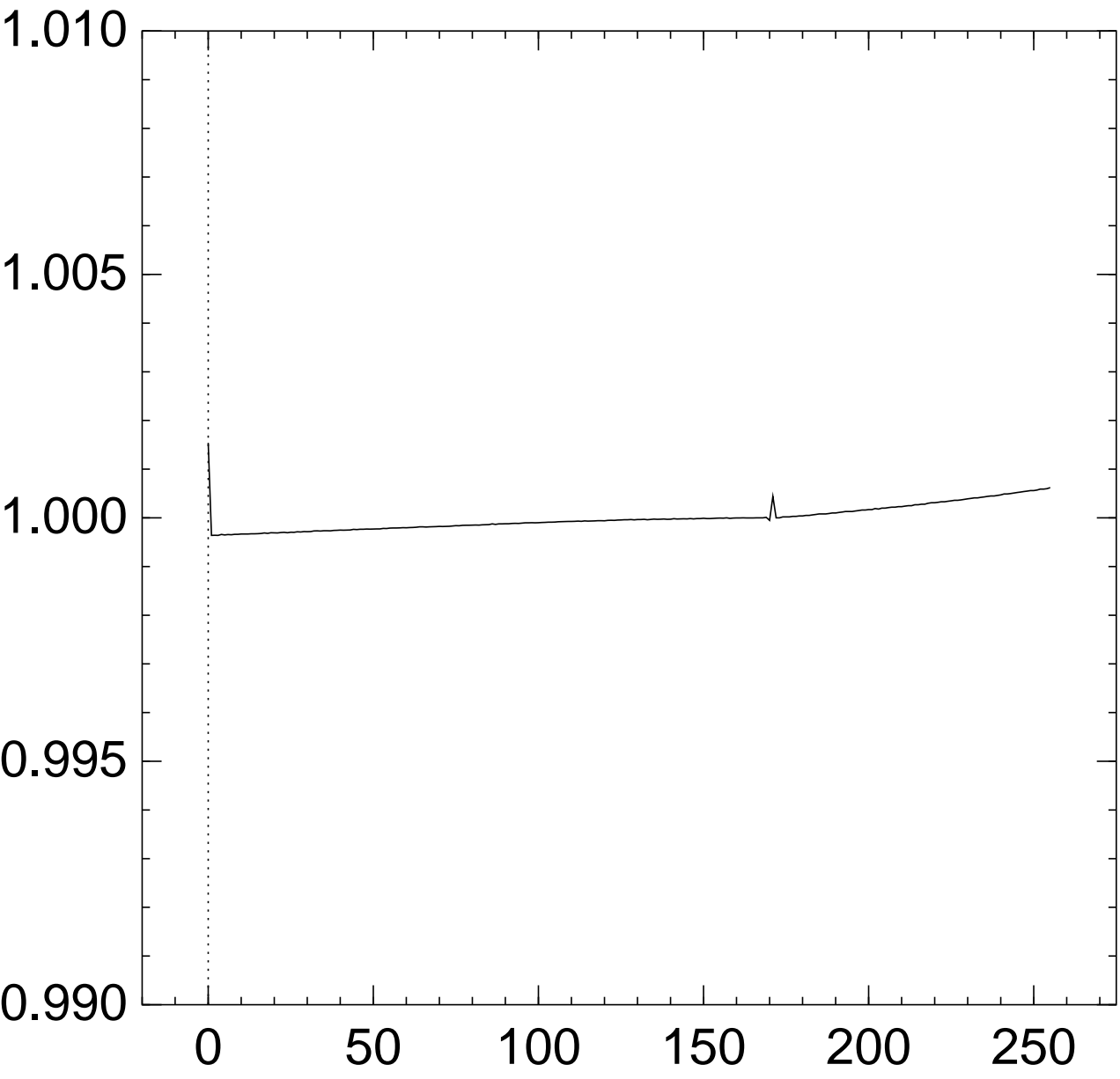
Graph of $256 \Pr[z_{169} = x]$:



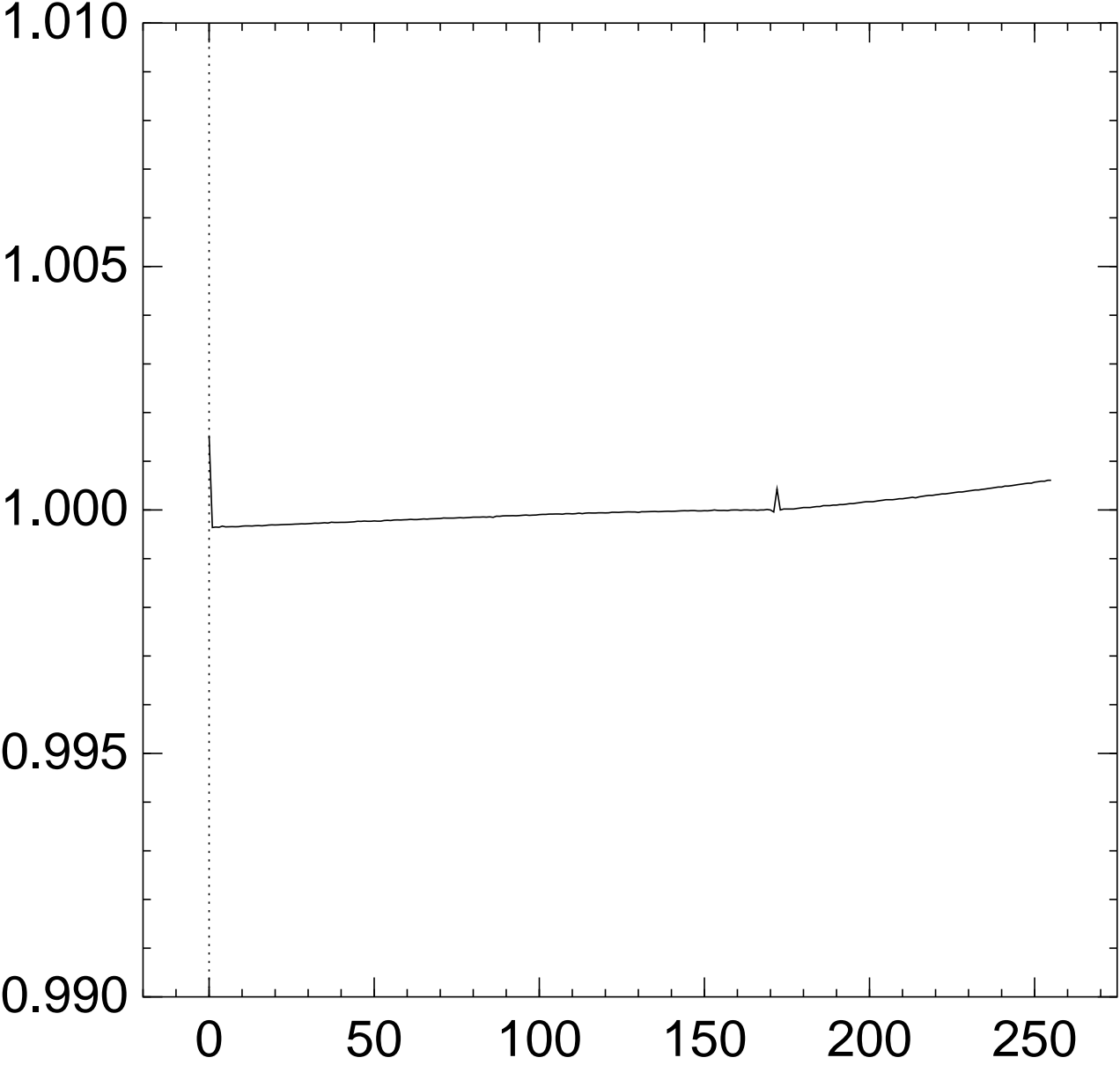
Graph of $256 \Pr[z_{170} = x]$:



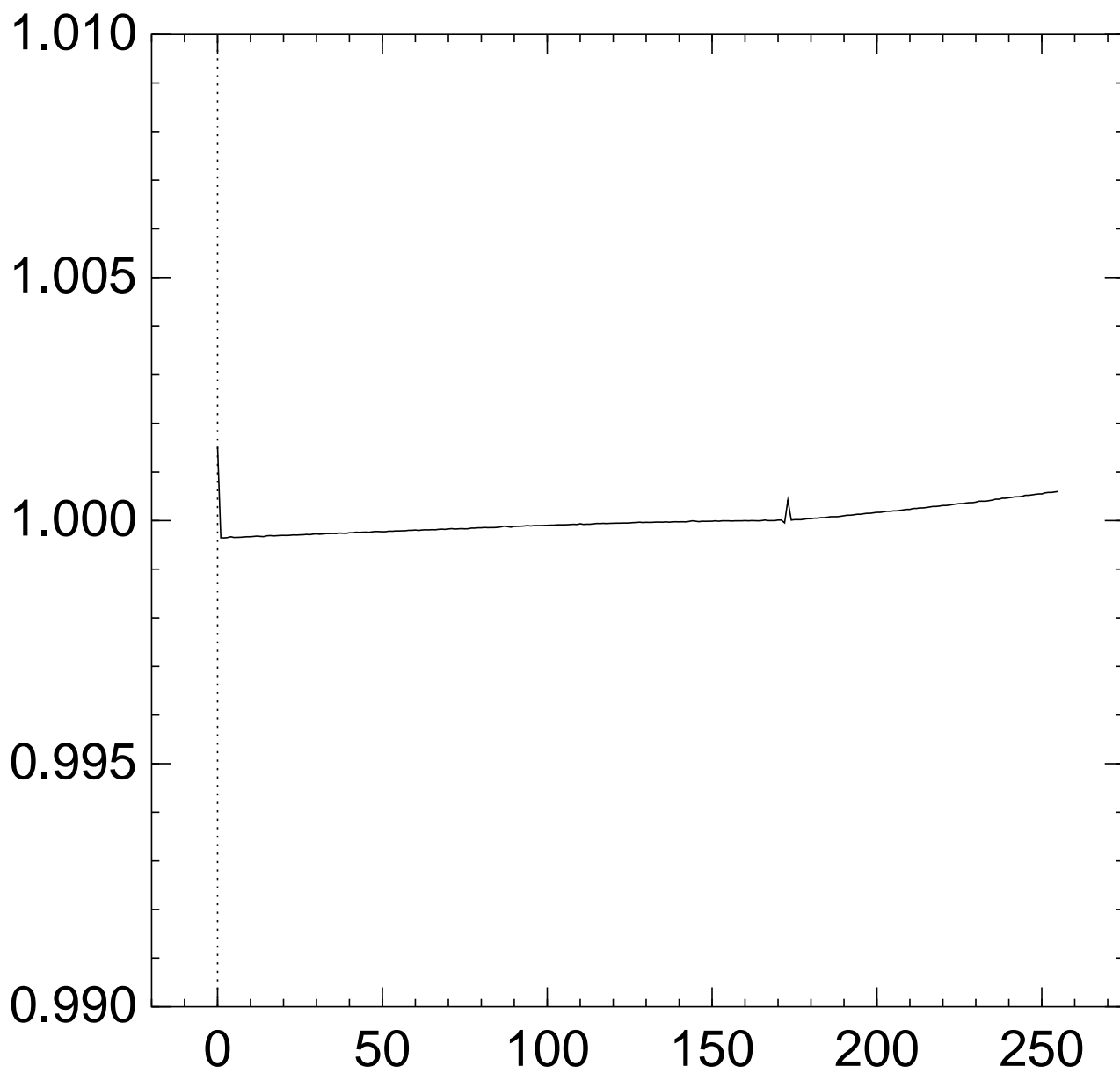
Graph of $256 \Pr[z_{171} = x]$:



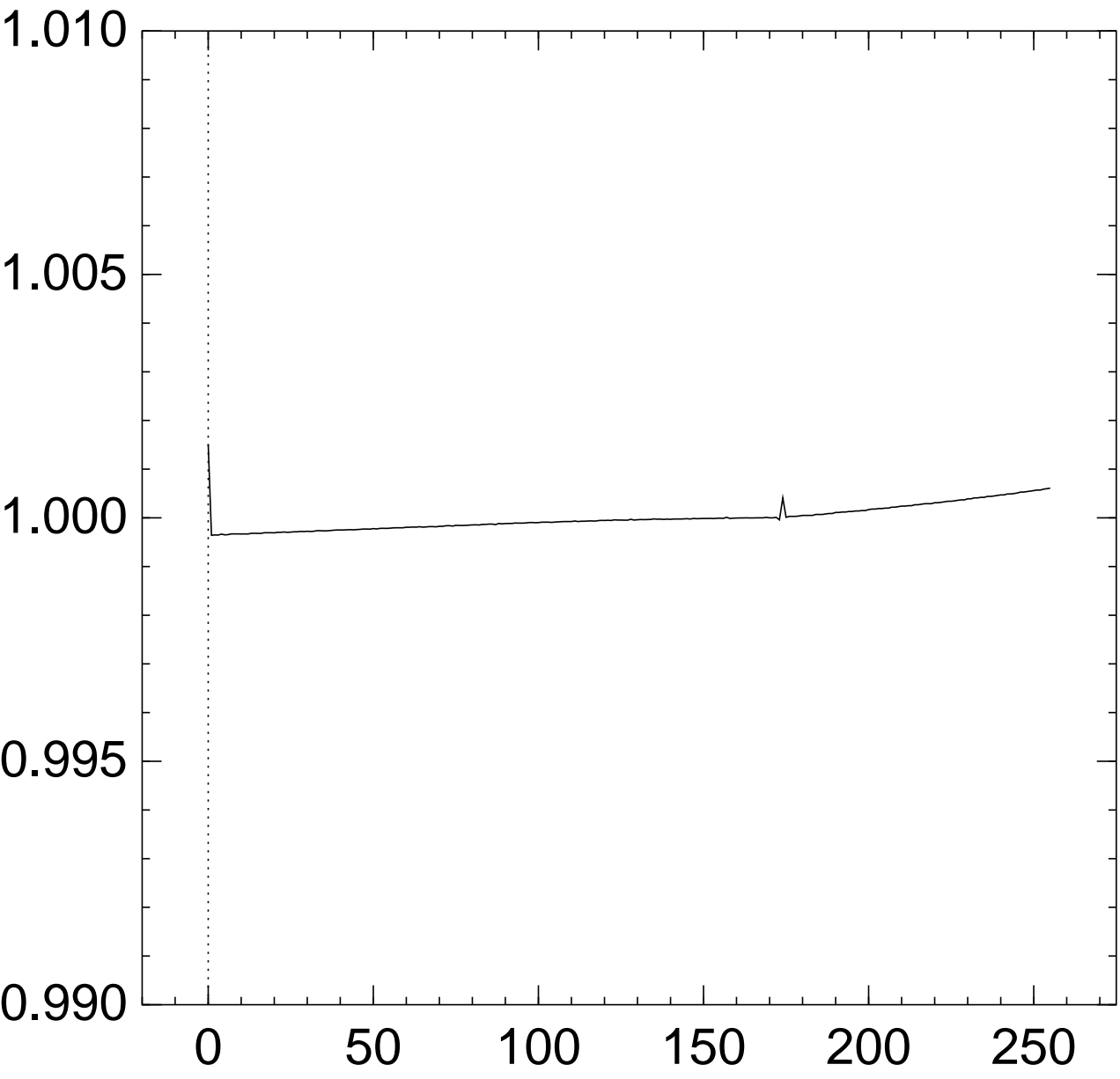
Graph of $256 \Pr[z_{172} = x]$:



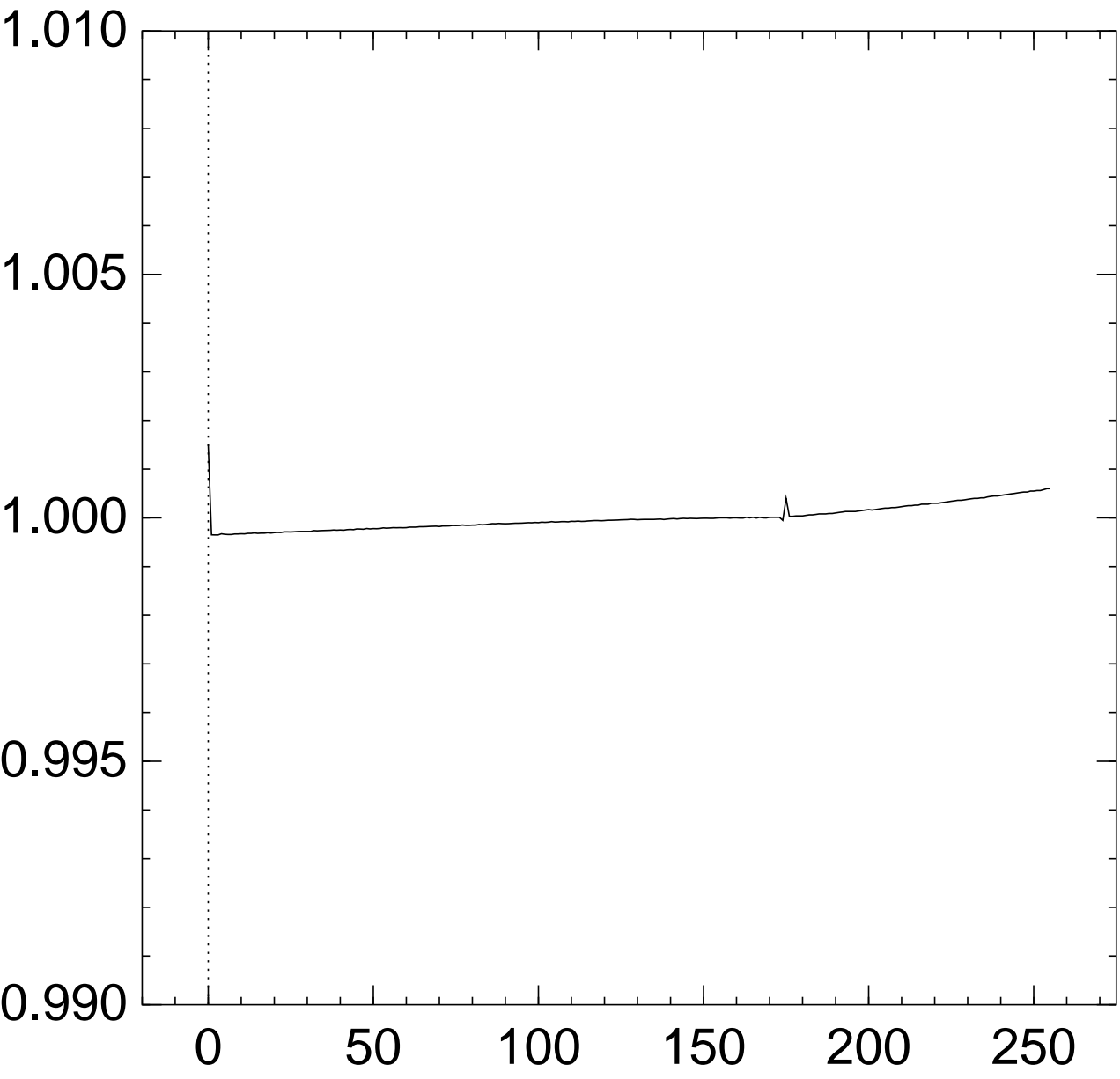
Graph of $256 \Pr[z_{173} = x]$:



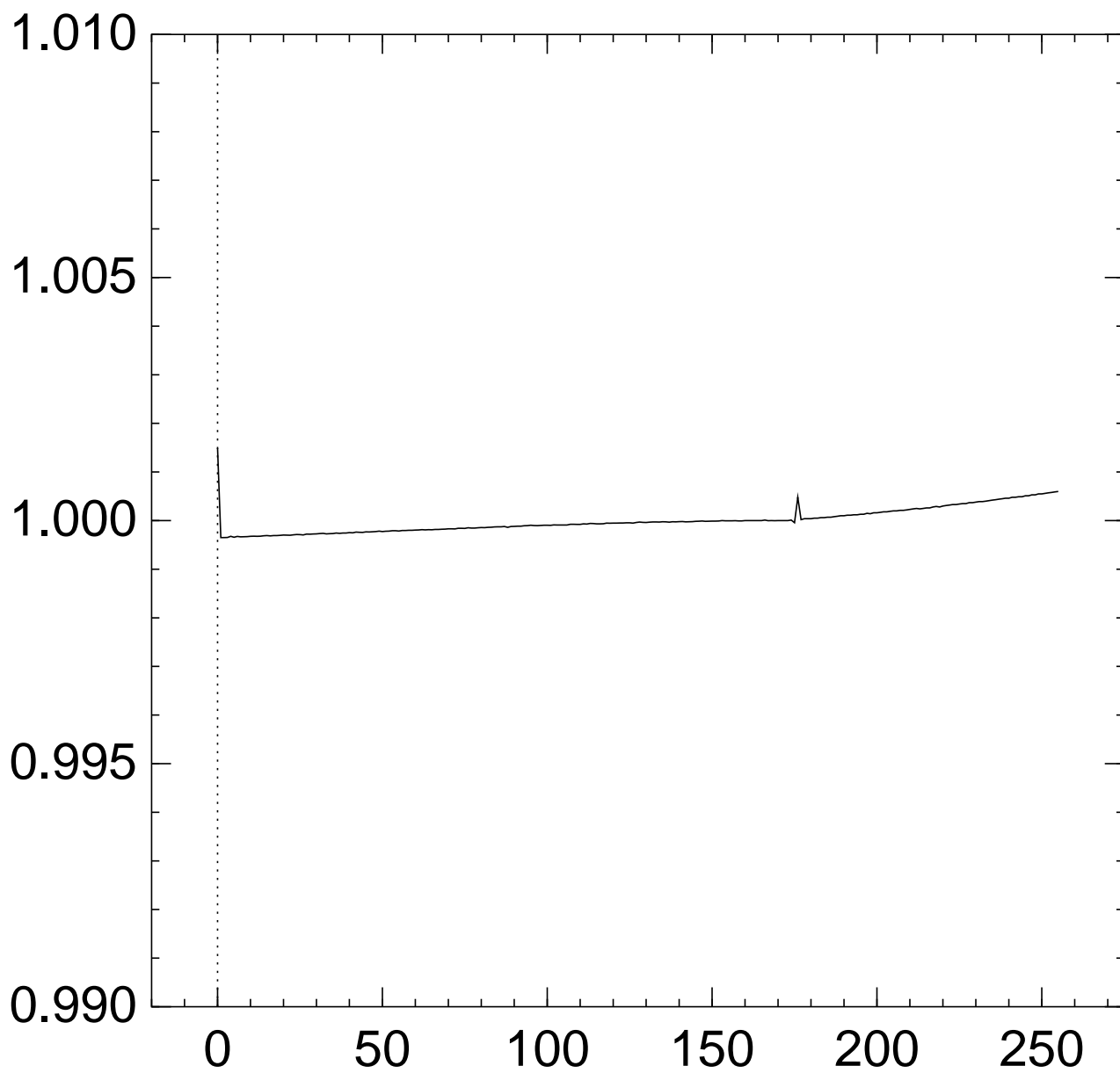
Graph of $256 \Pr[z_{174} = x]$:



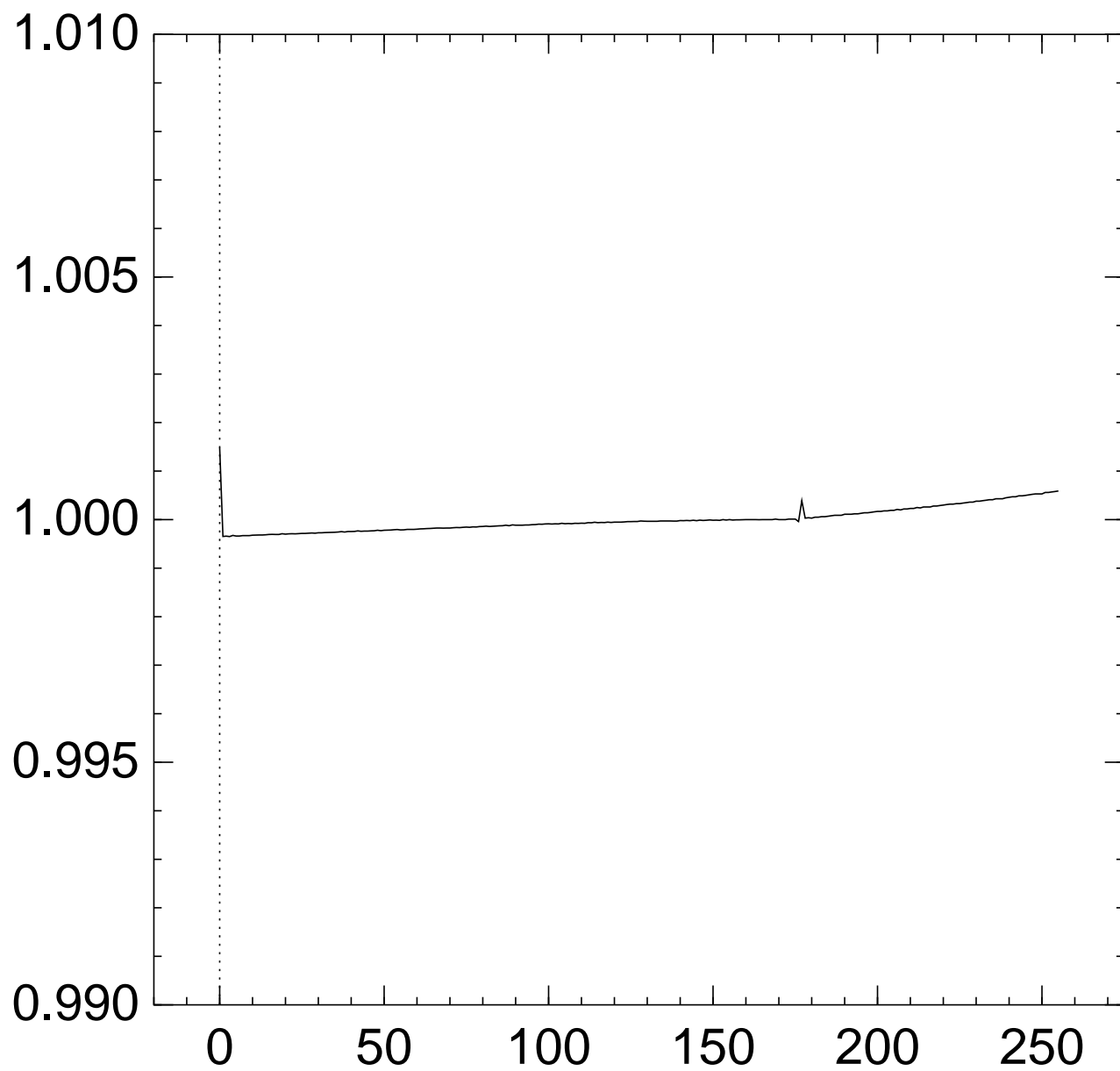
Graph of $256 \Pr[z_{175} = x]$:



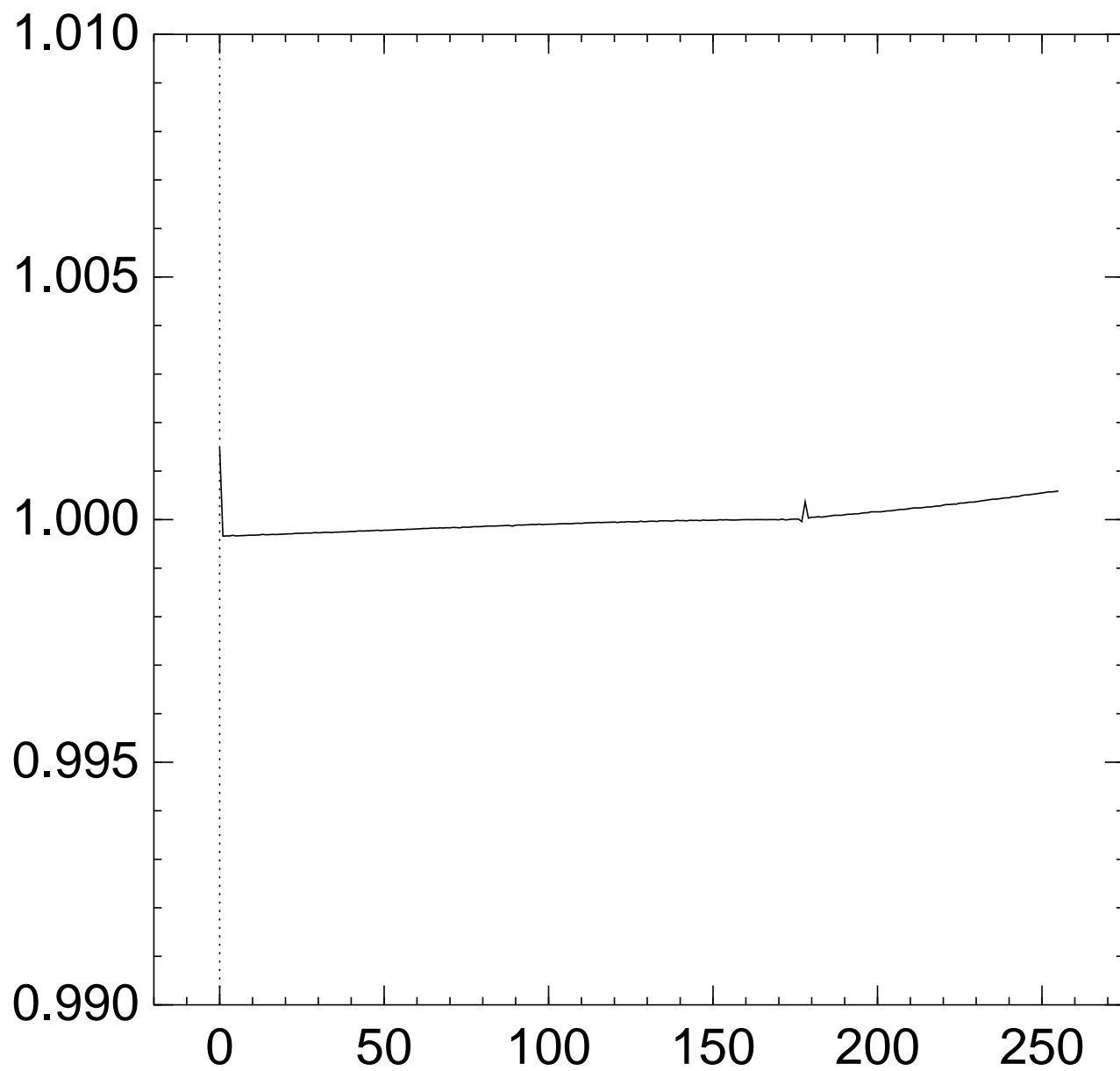
Graph of $256 \Pr[z_{176} = x]$:



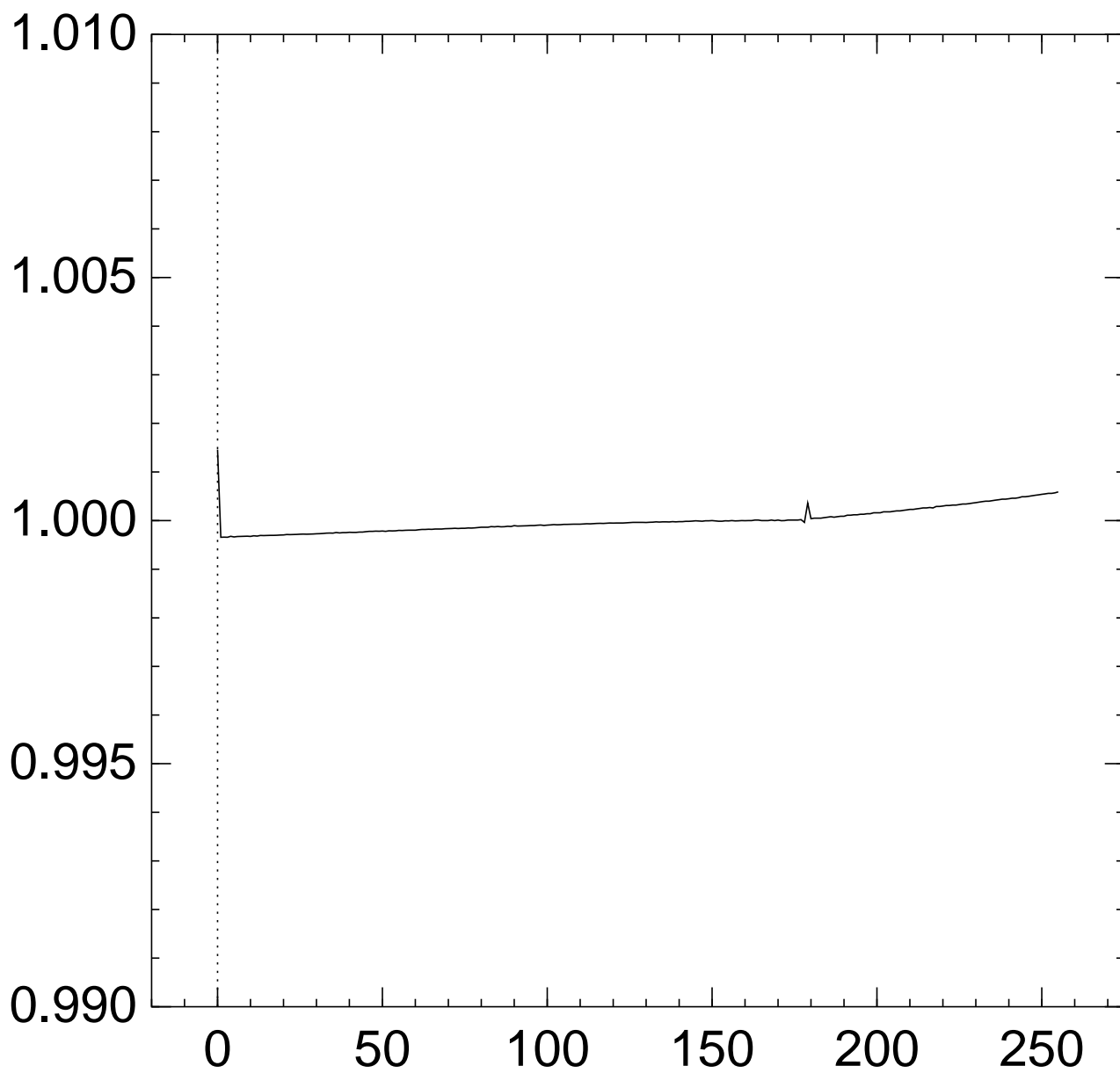
Graph of $256 \Pr[z_{177} = x]$:



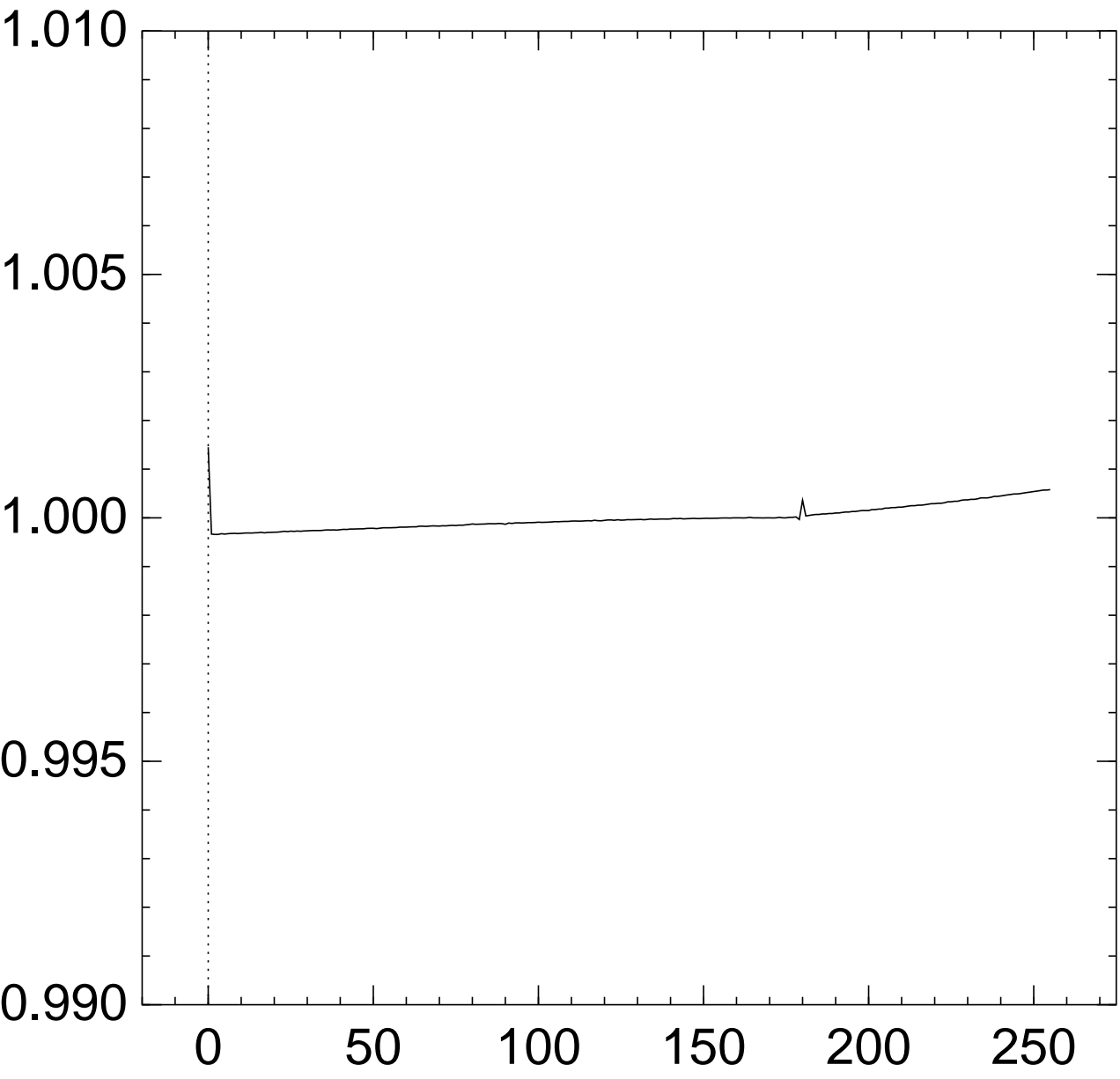
Graph of $256 \Pr[z_{178} = x]$:



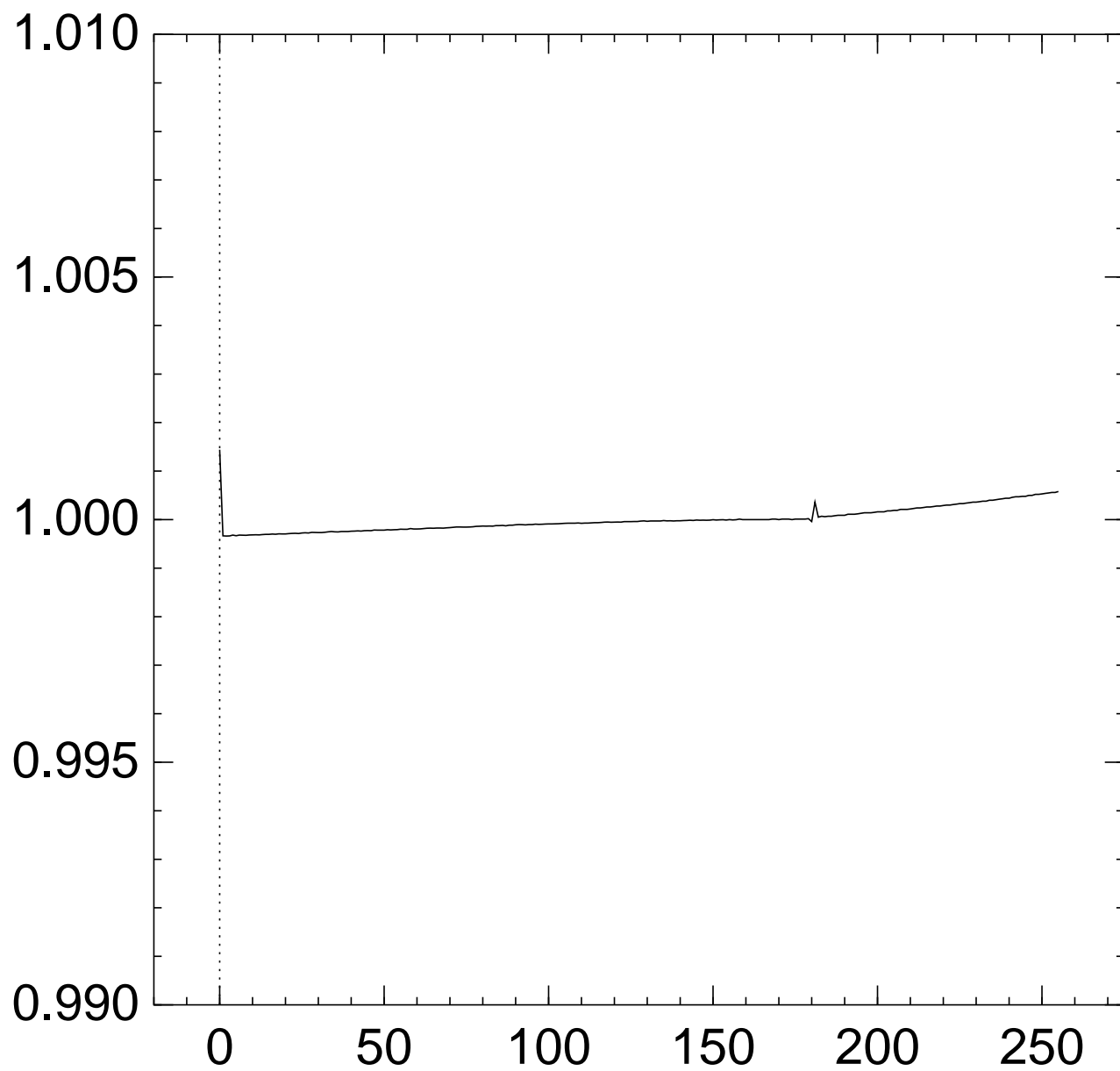
Graph of $256 \Pr[z_{179} = x]$:



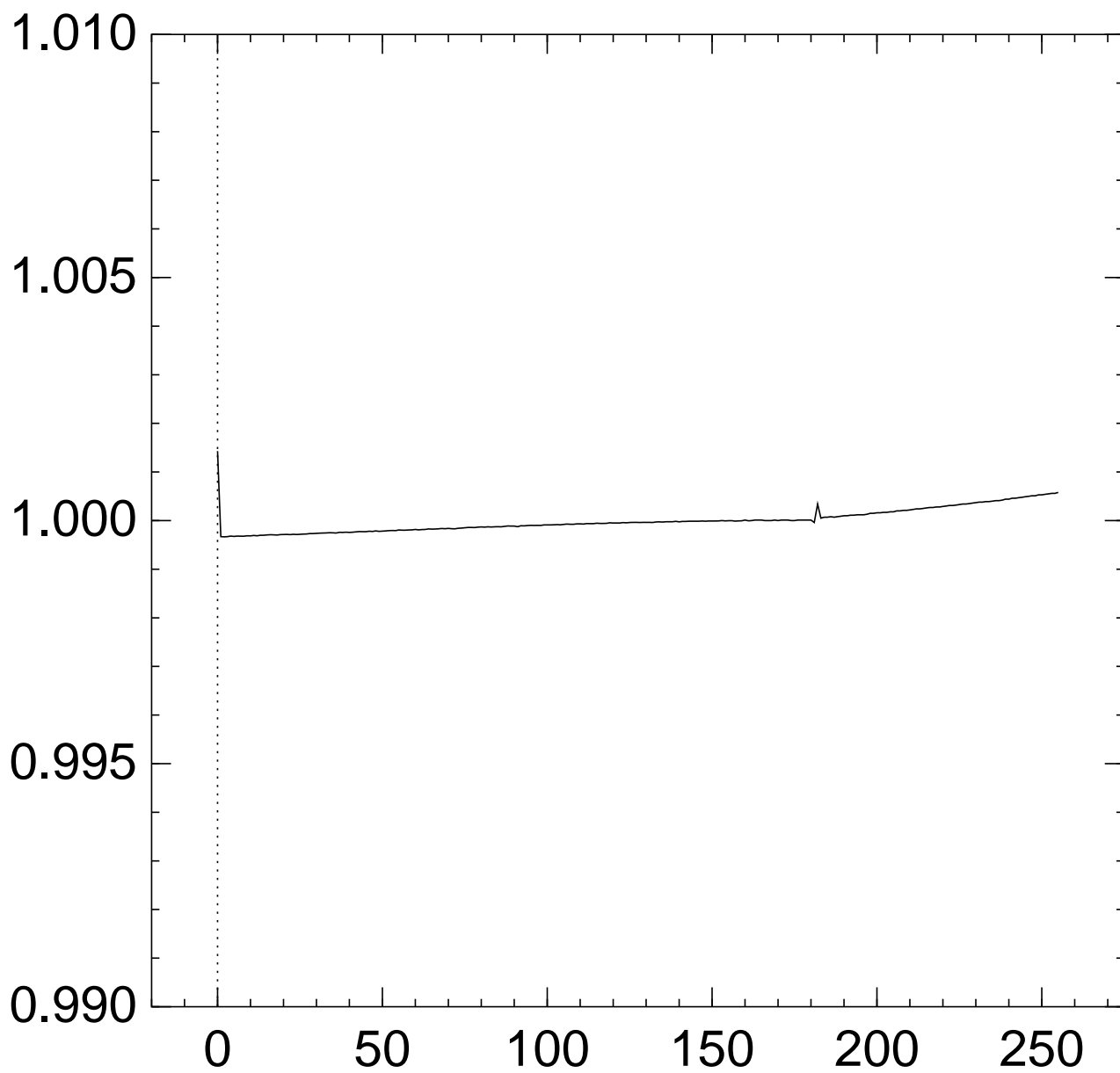
Graph of $256 \Pr[z_{180} = x]$:



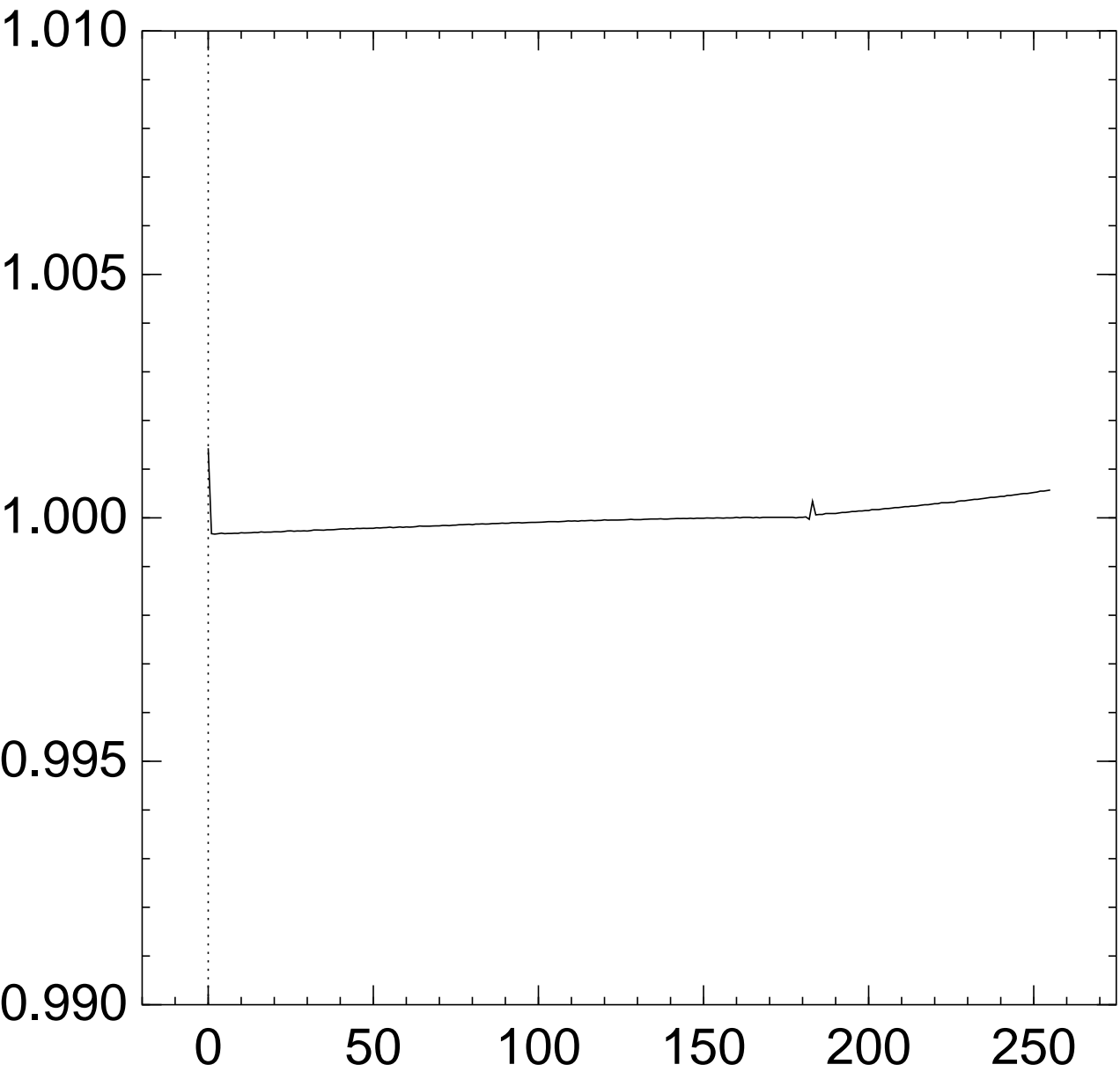
Graph of $256 \Pr[z_{181} = x]$:



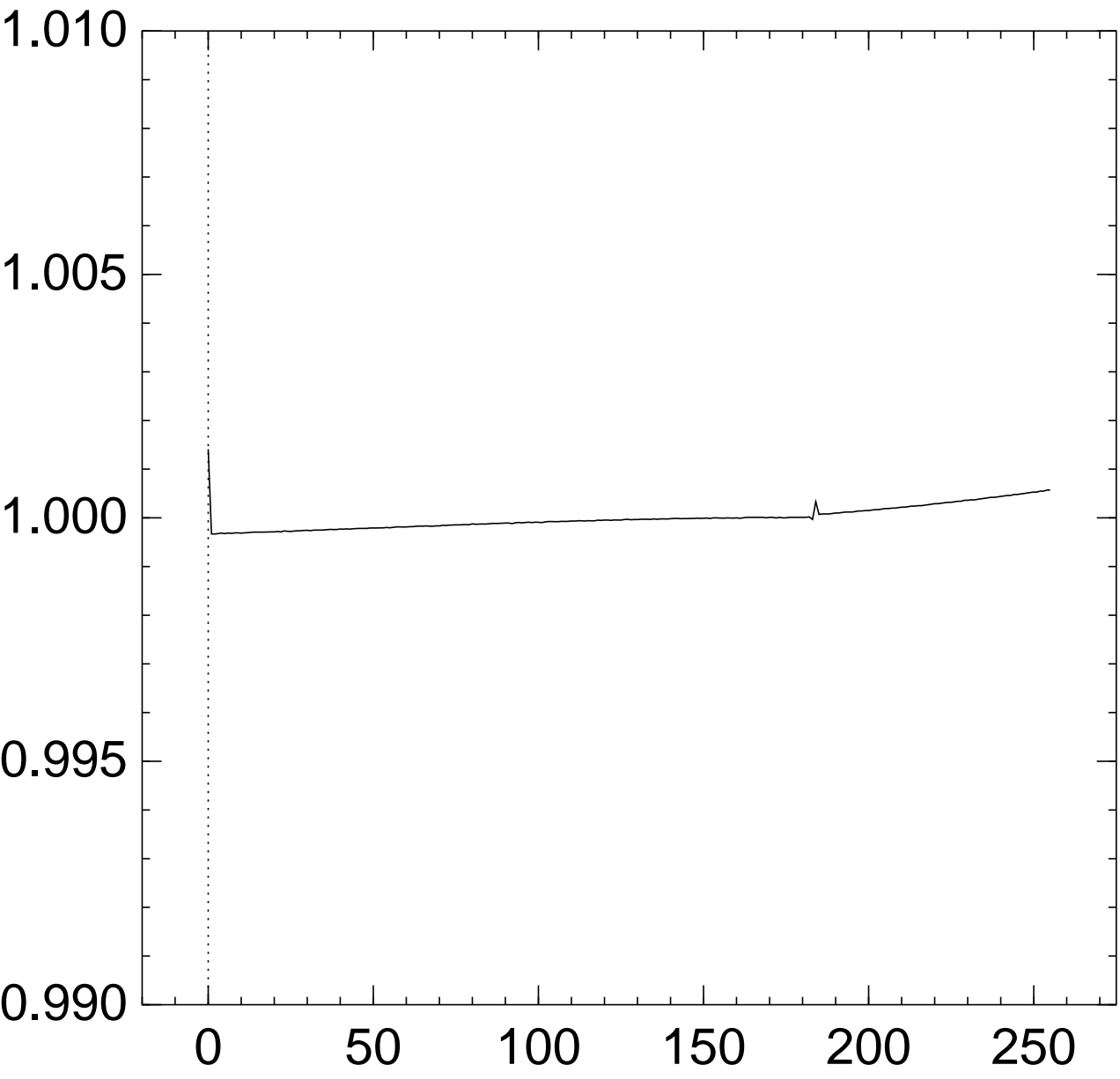
Graph of $256 \Pr[z_{182} = x]$:



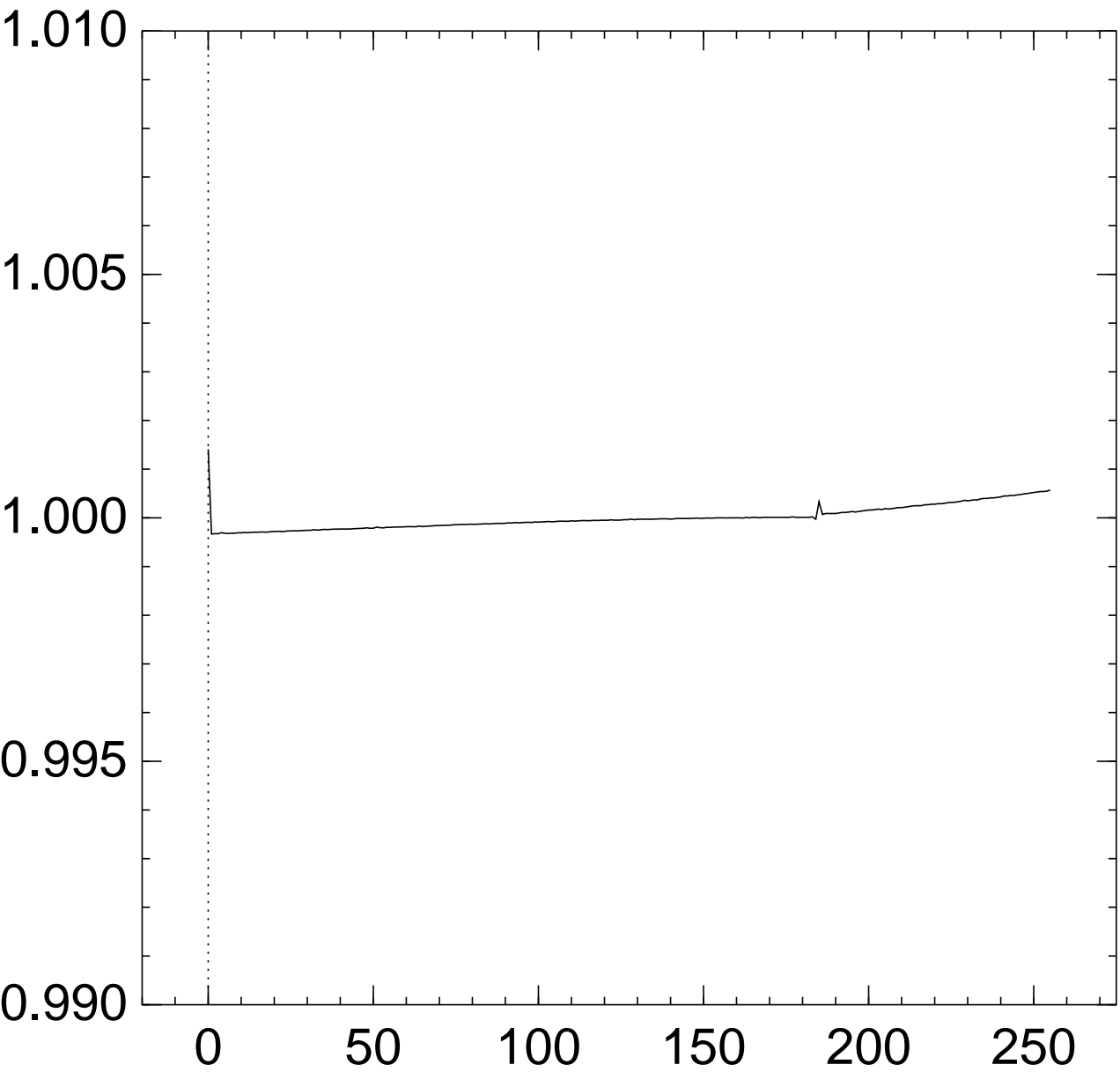
Graph of $256 \Pr[z_{183} = x]$:



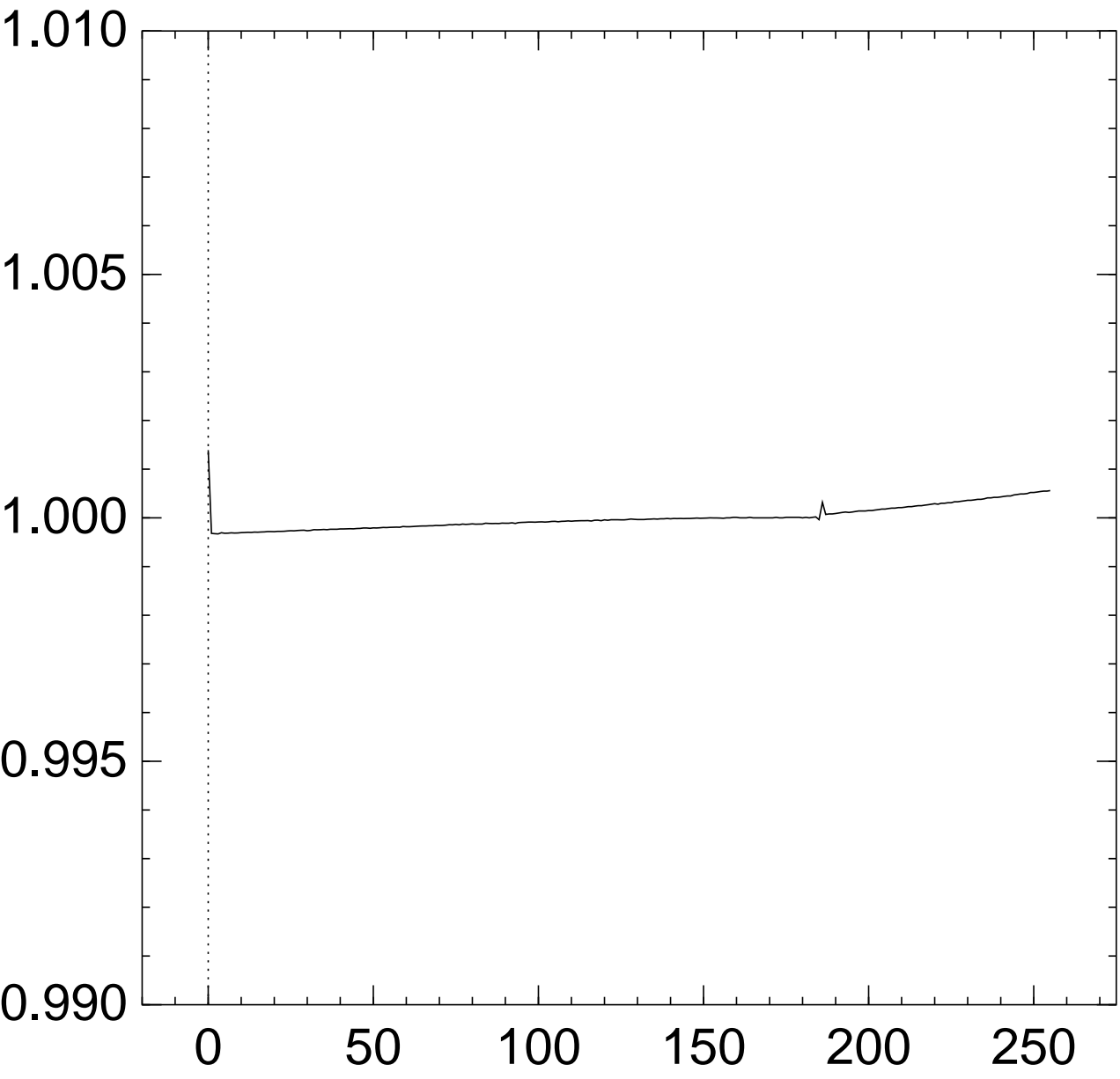
Graph of $256 \Pr[z_{184} = x]$:



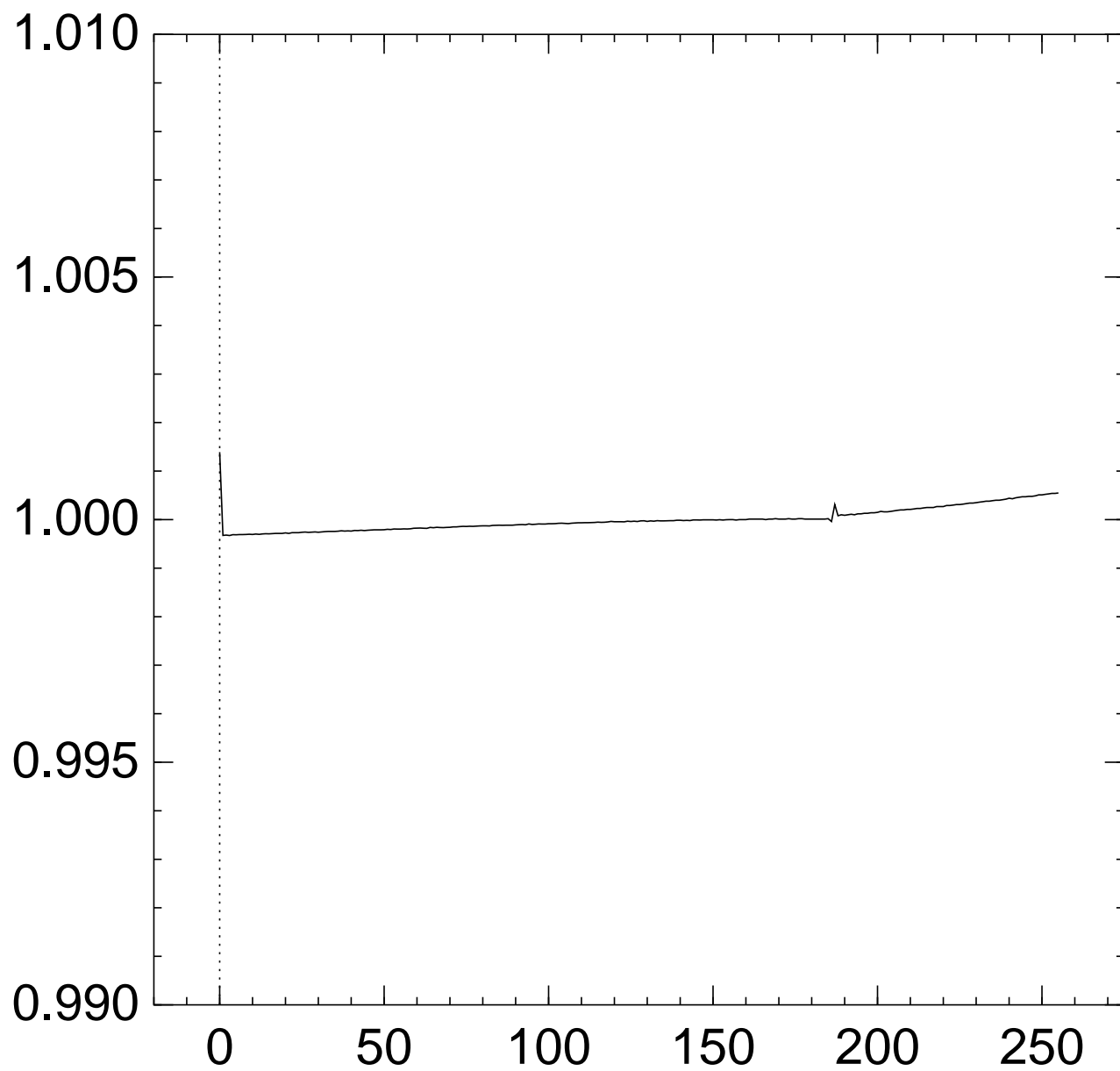
Graph of $256 \Pr[z_{185} = x]$:



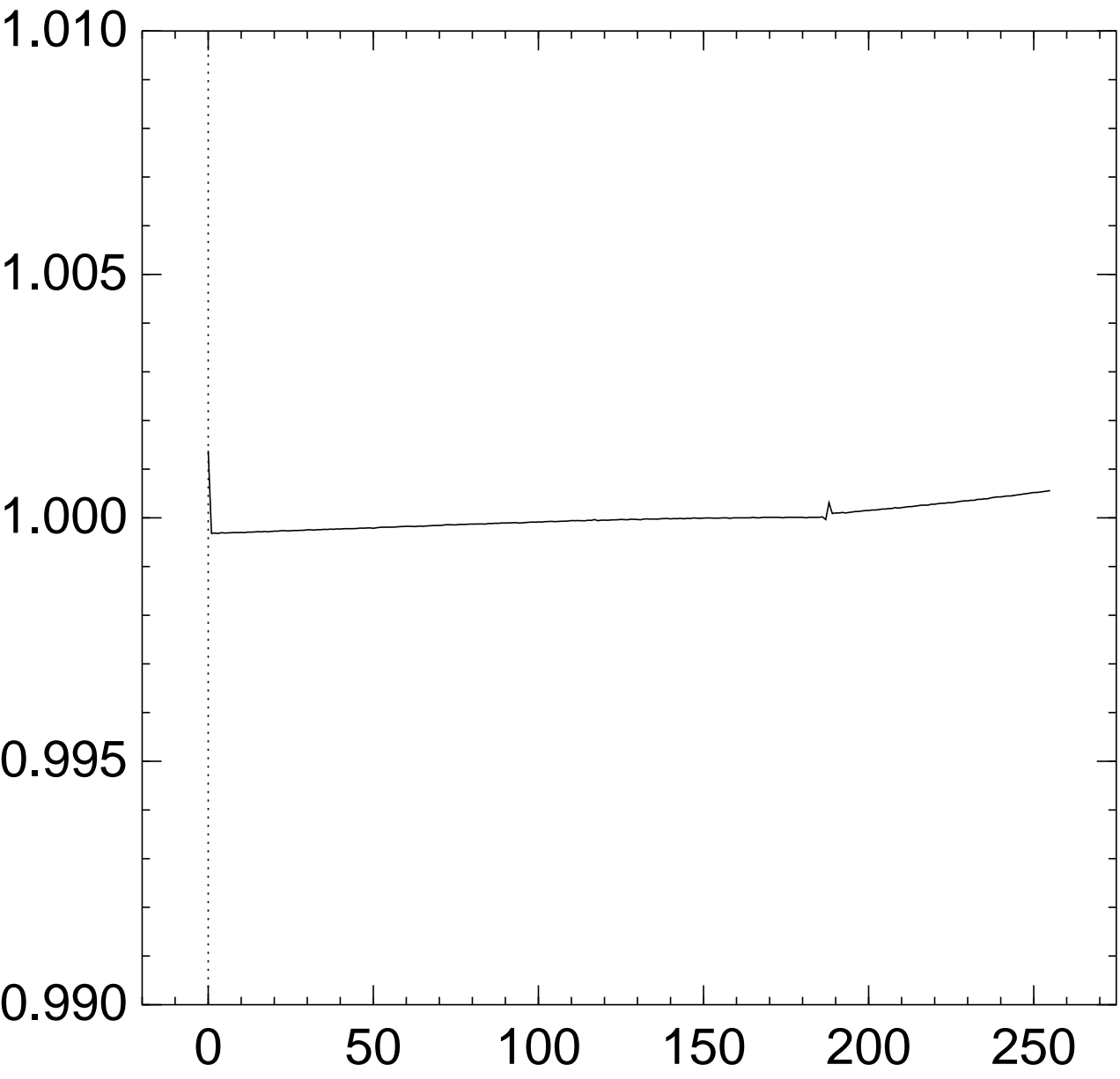
Graph of $256 \Pr[z_{186} = x]$:



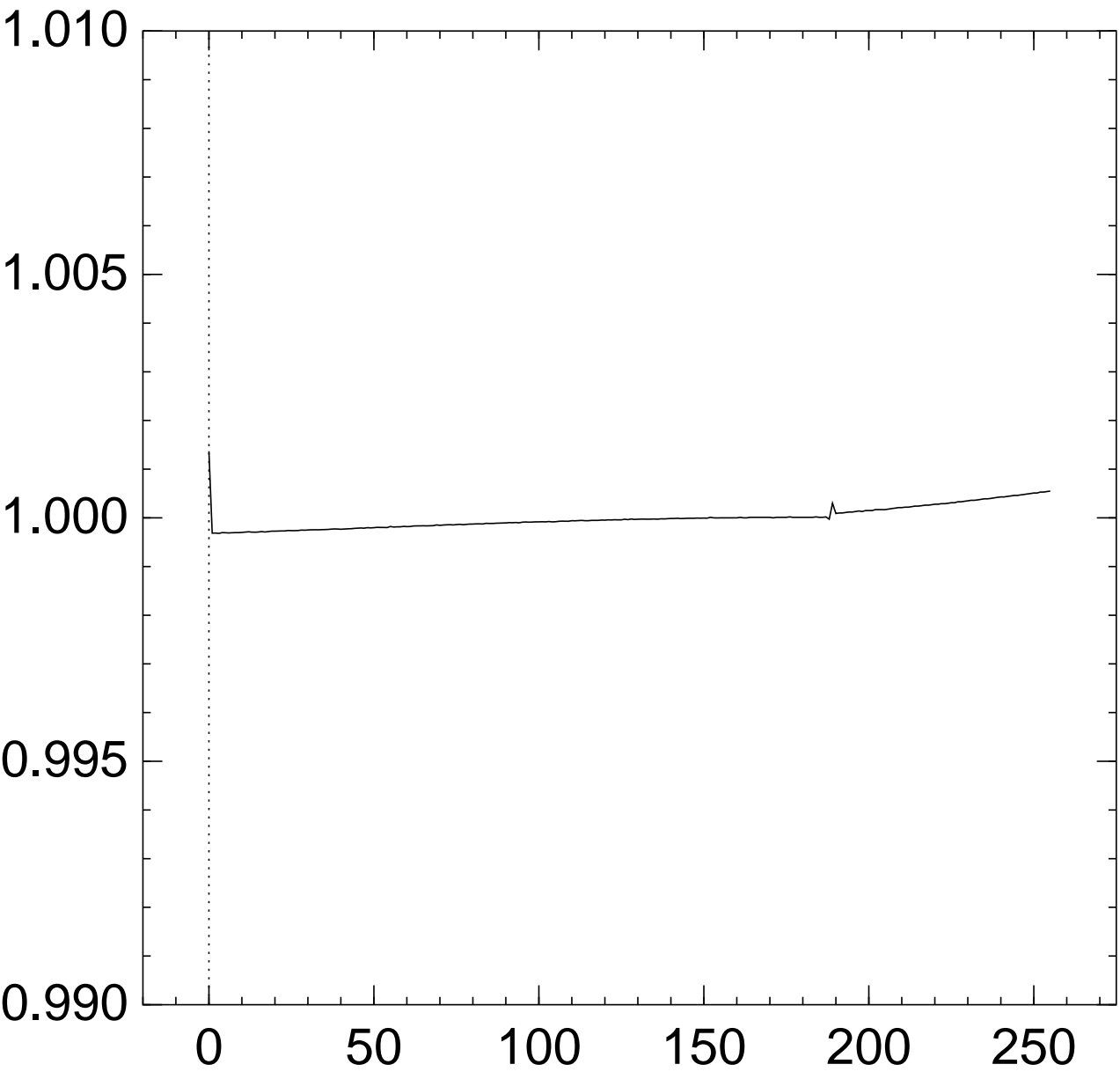
Graph of $256 \Pr[z_{187} = x]$:



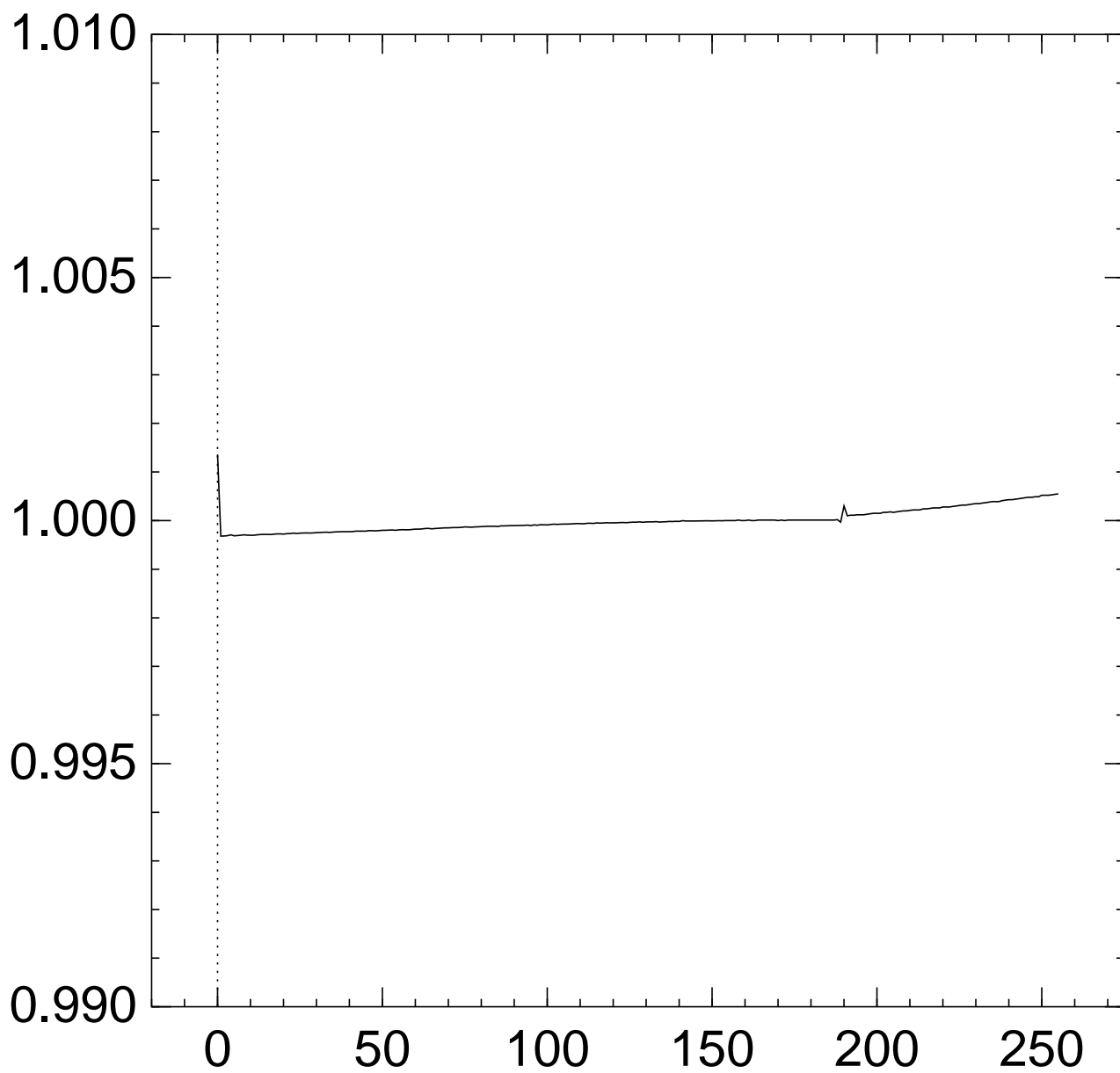
Graph of $256 \Pr[z_{188} = x]$:



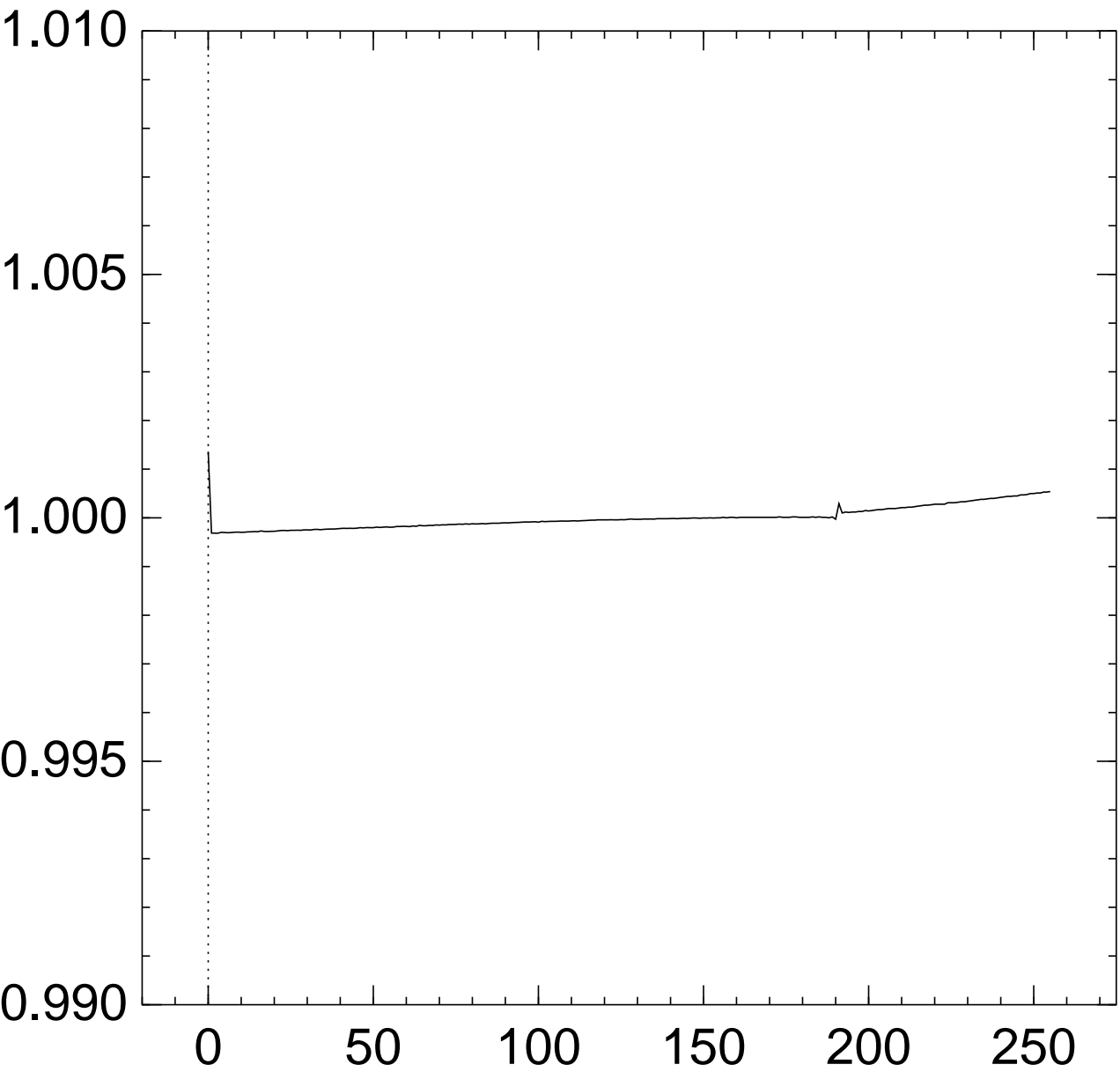
Graph of $256 \Pr[z_{189} = x]$:



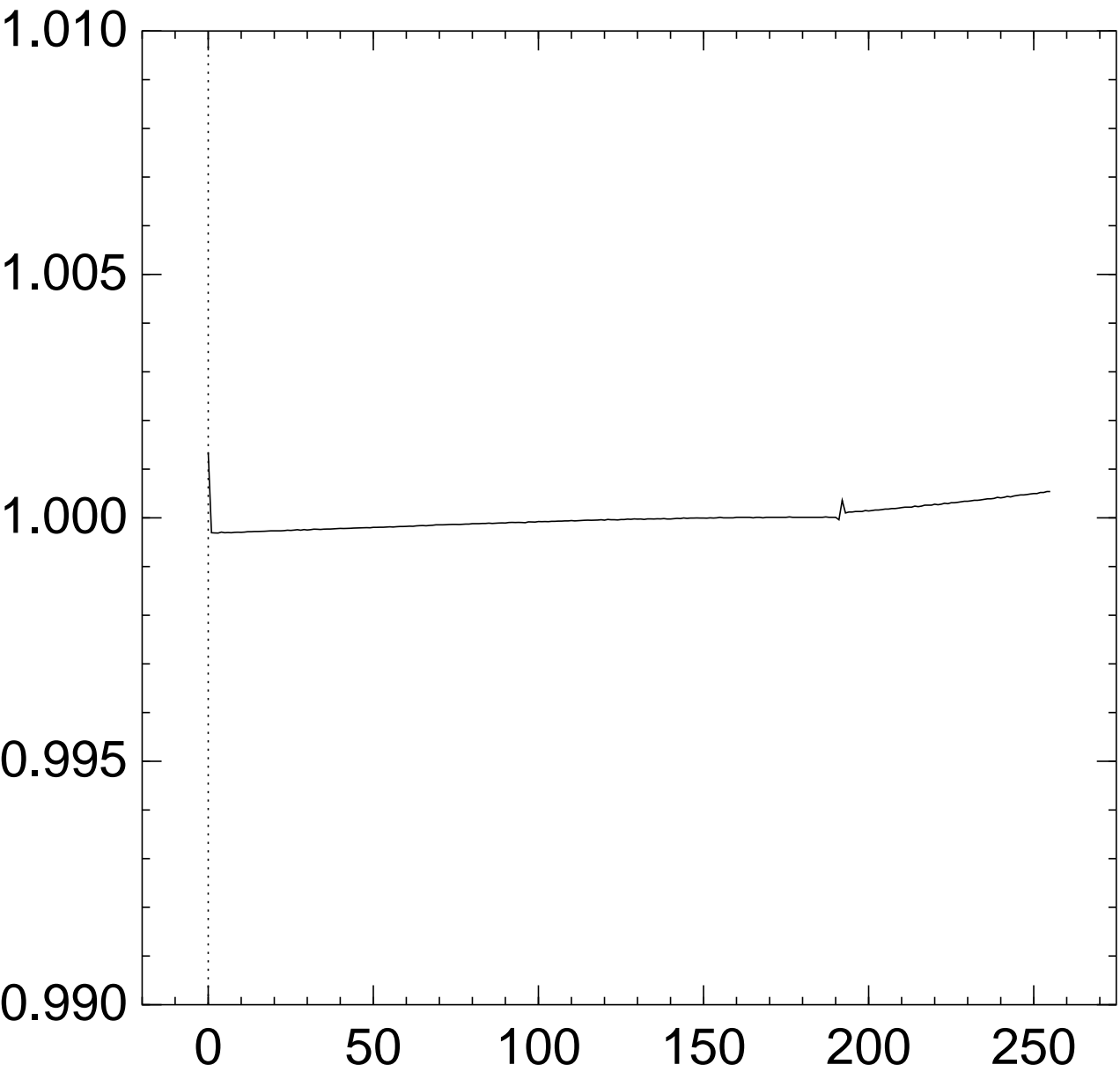
Graph of $256 \Pr[z_{190} = x]$:



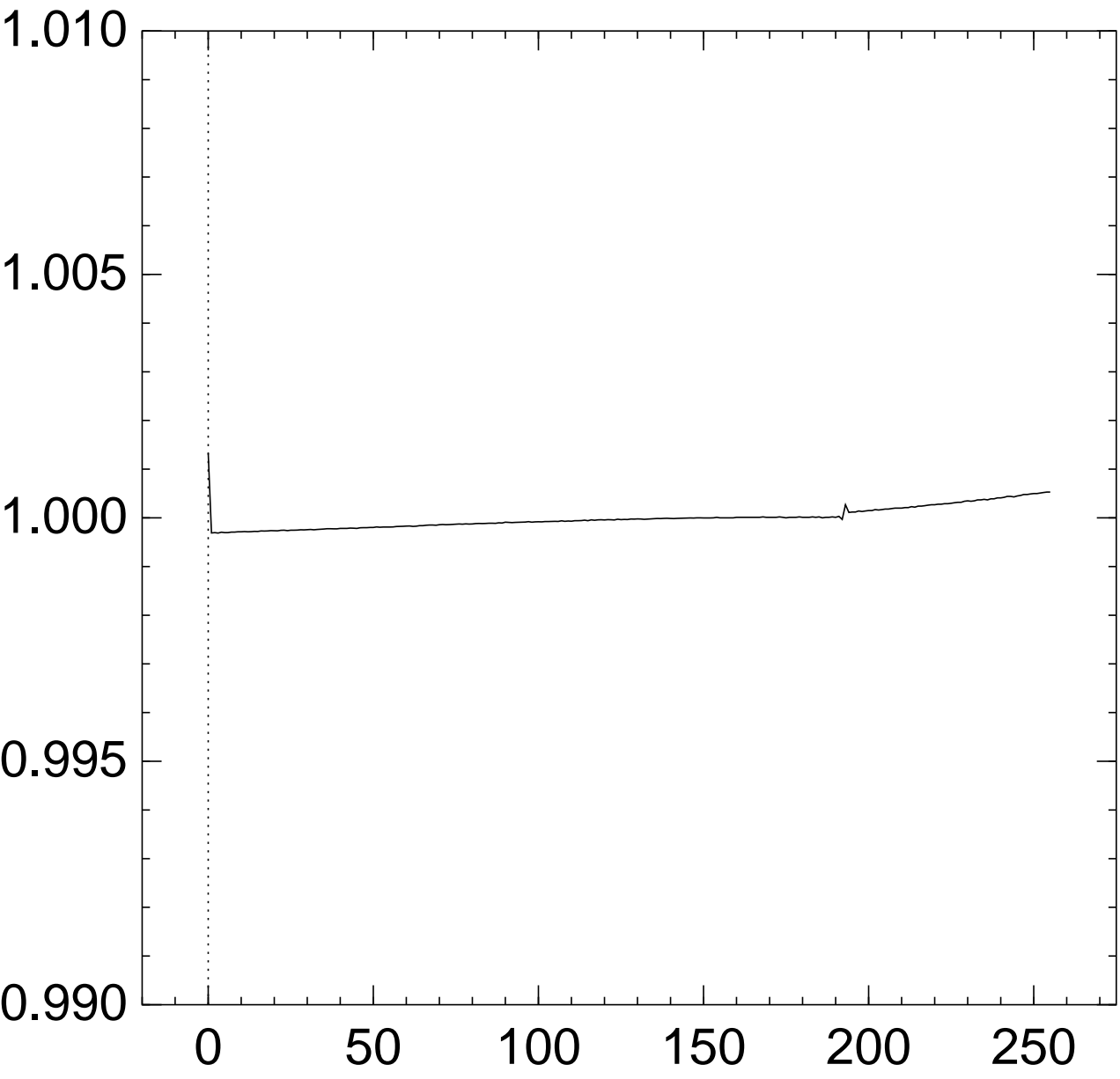
Graph of $256 \Pr[z_{191} = x]$:



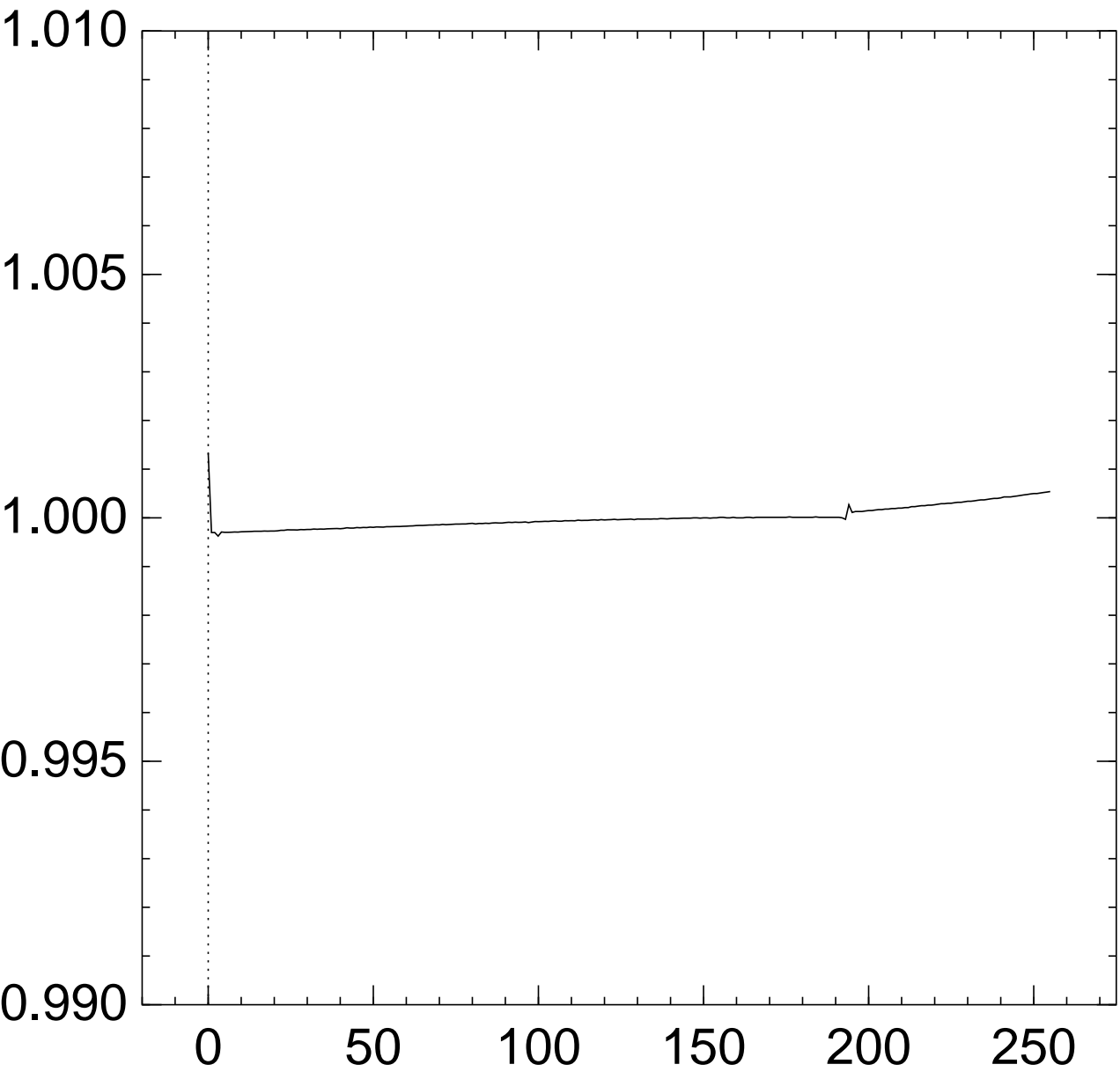
Graph of $256 \Pr[z_{192} = x]$:



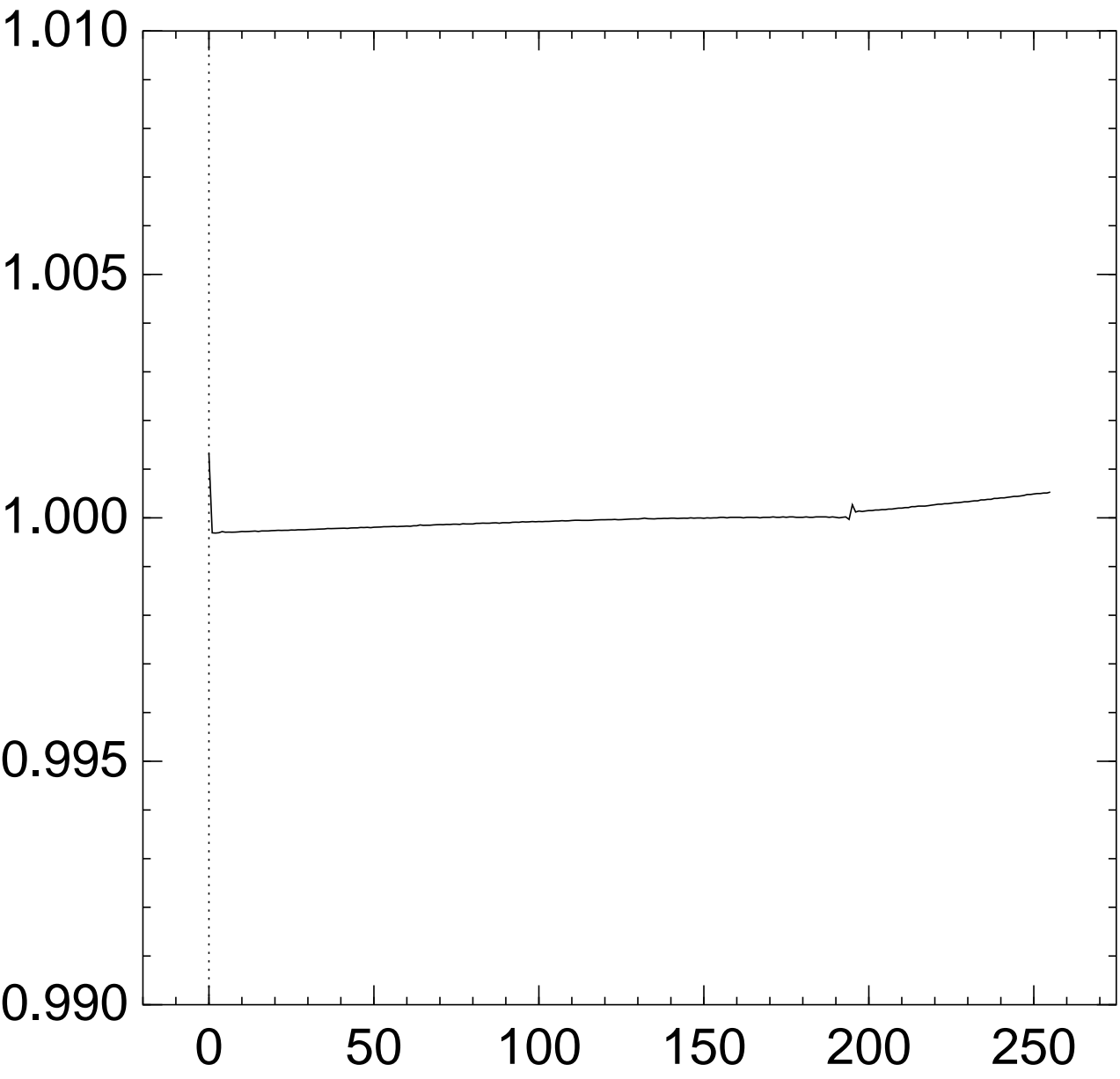
Graph of $256 \Pr[z_{193} = x]$:



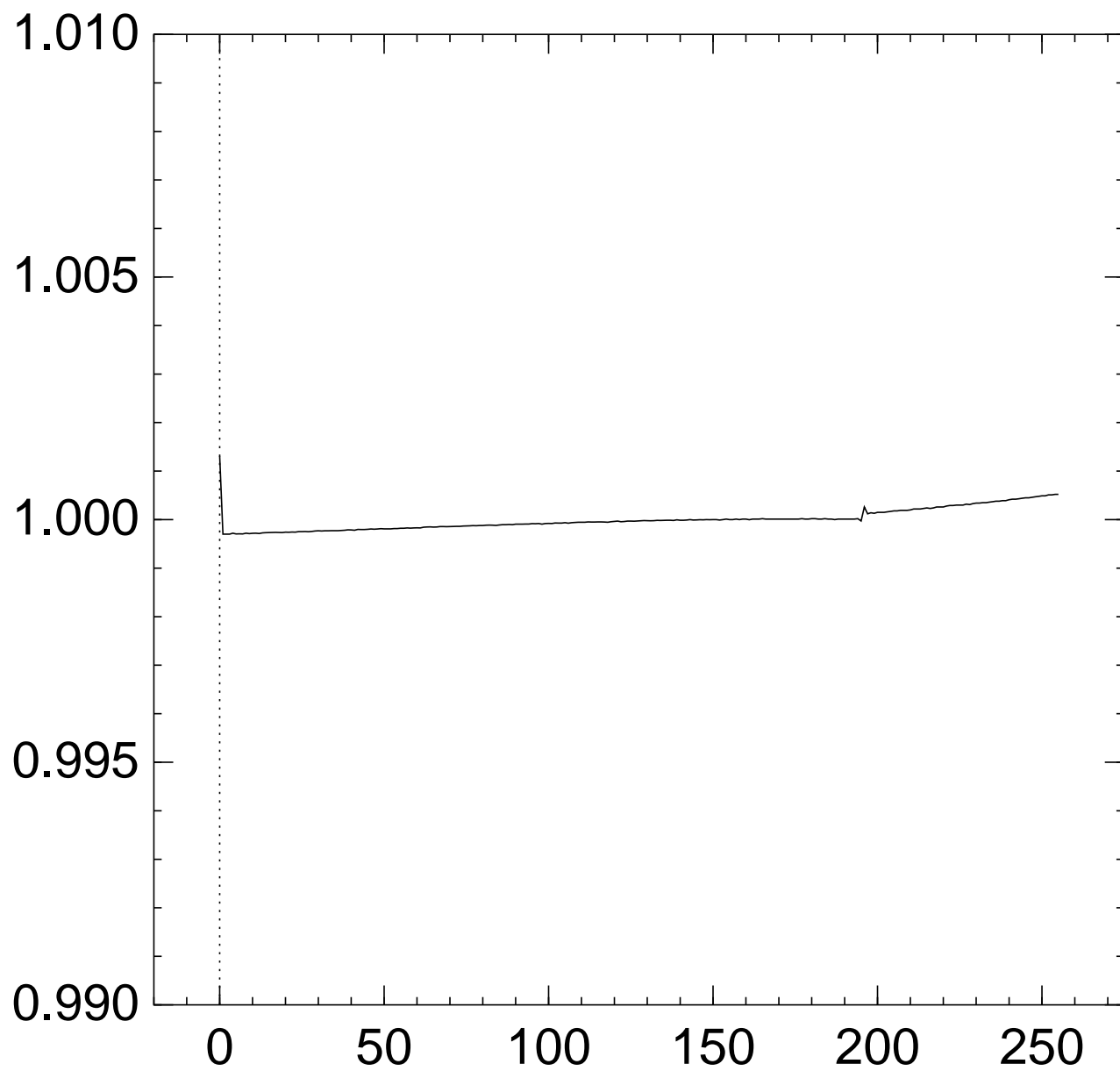
Graph of $256 \Pr[z_{194} = x]$:



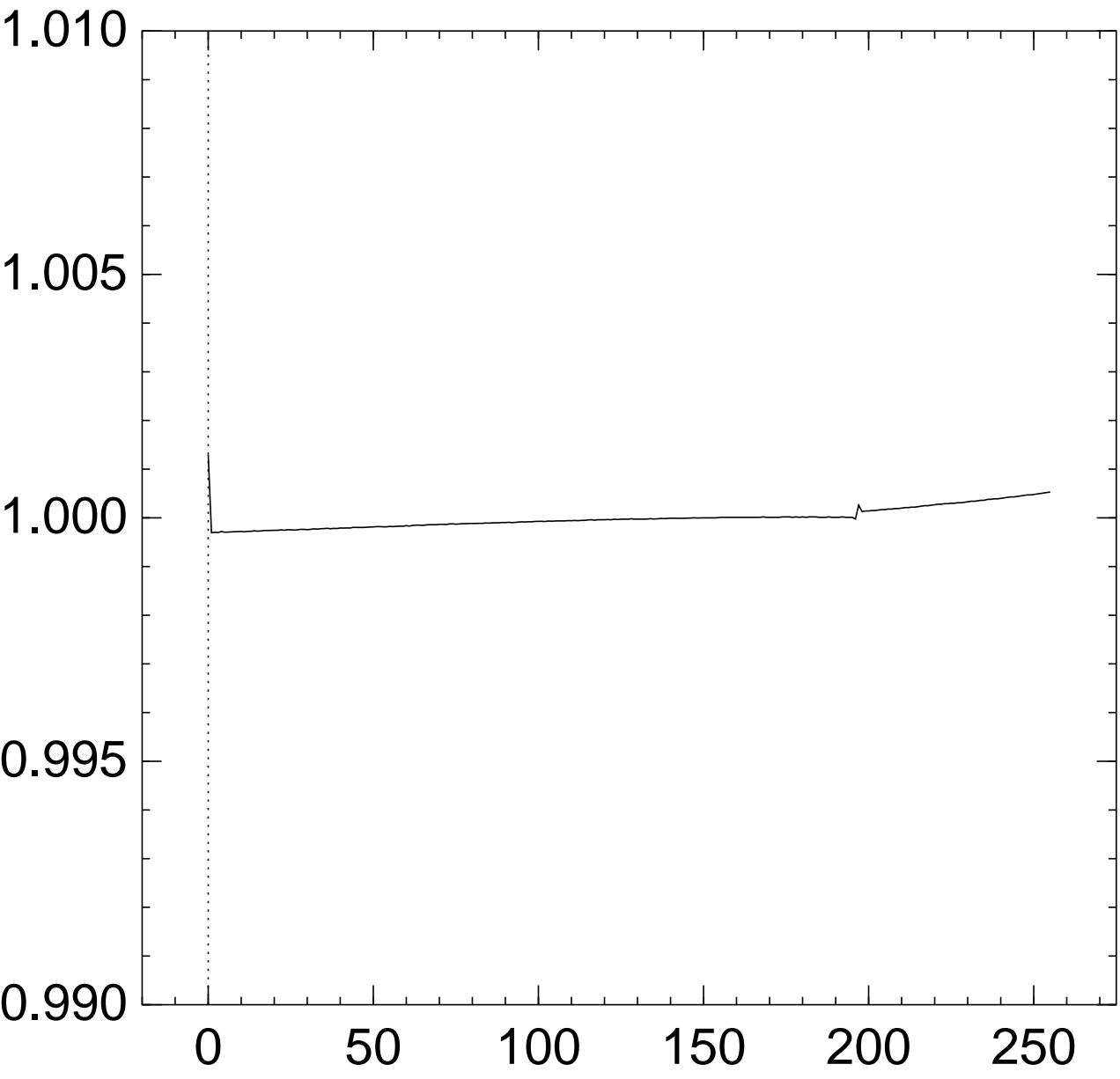
Graph of $256 \Pr[z_{195} = x]$:



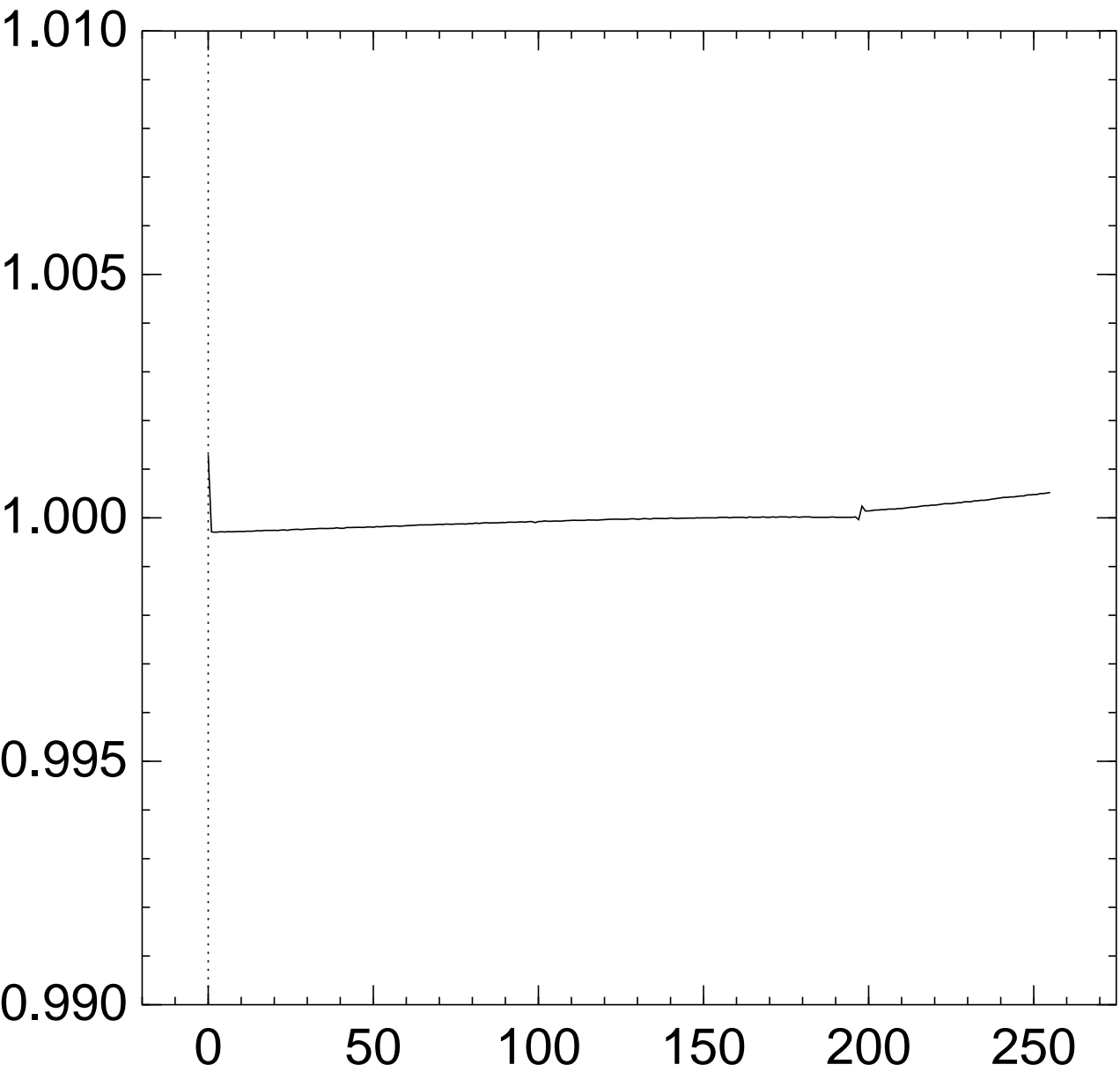
Graph of $256 \Pr[z_{196} = x]$:



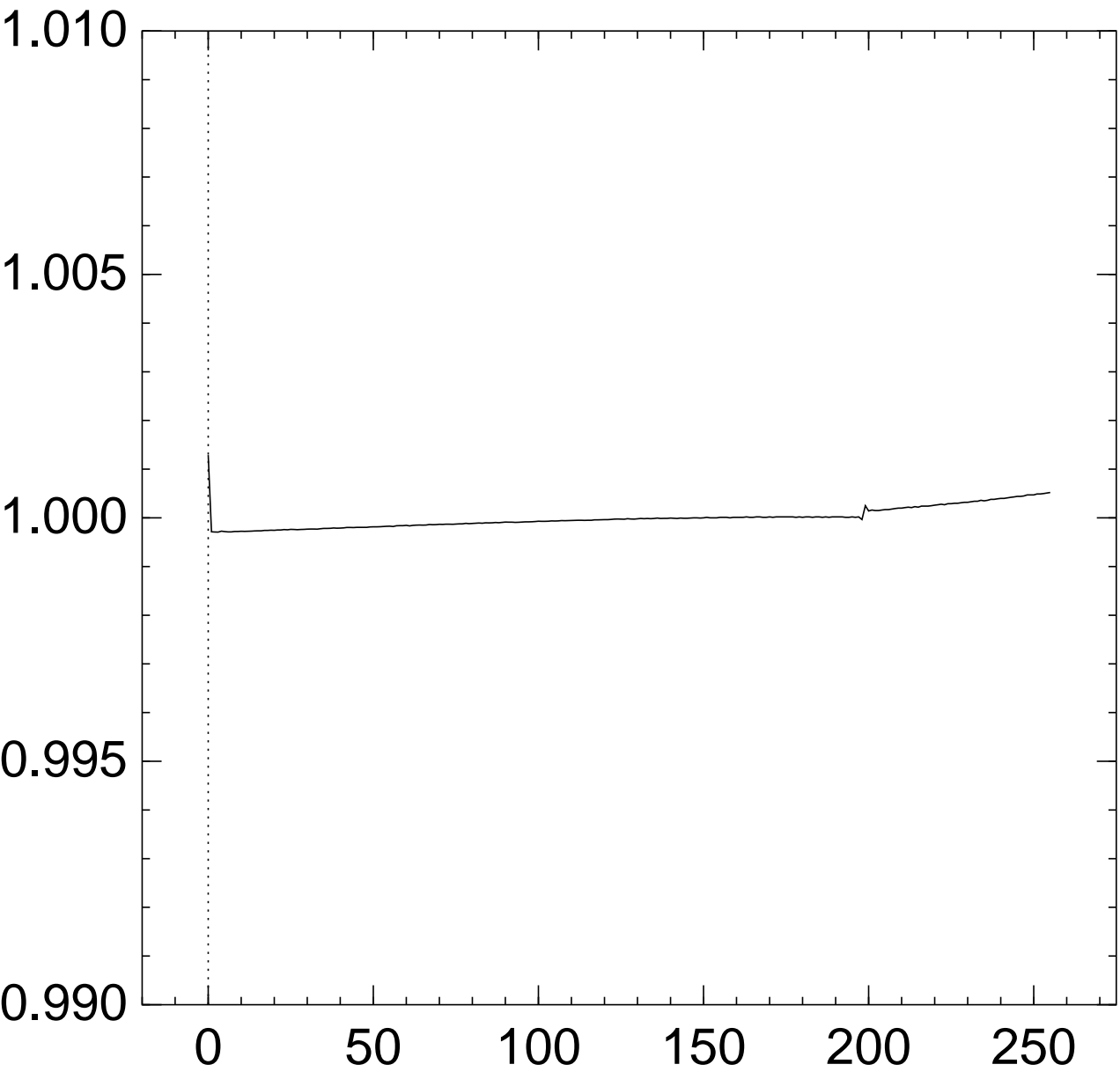
Graph of $256 \Pr[z_{197} = x]$:



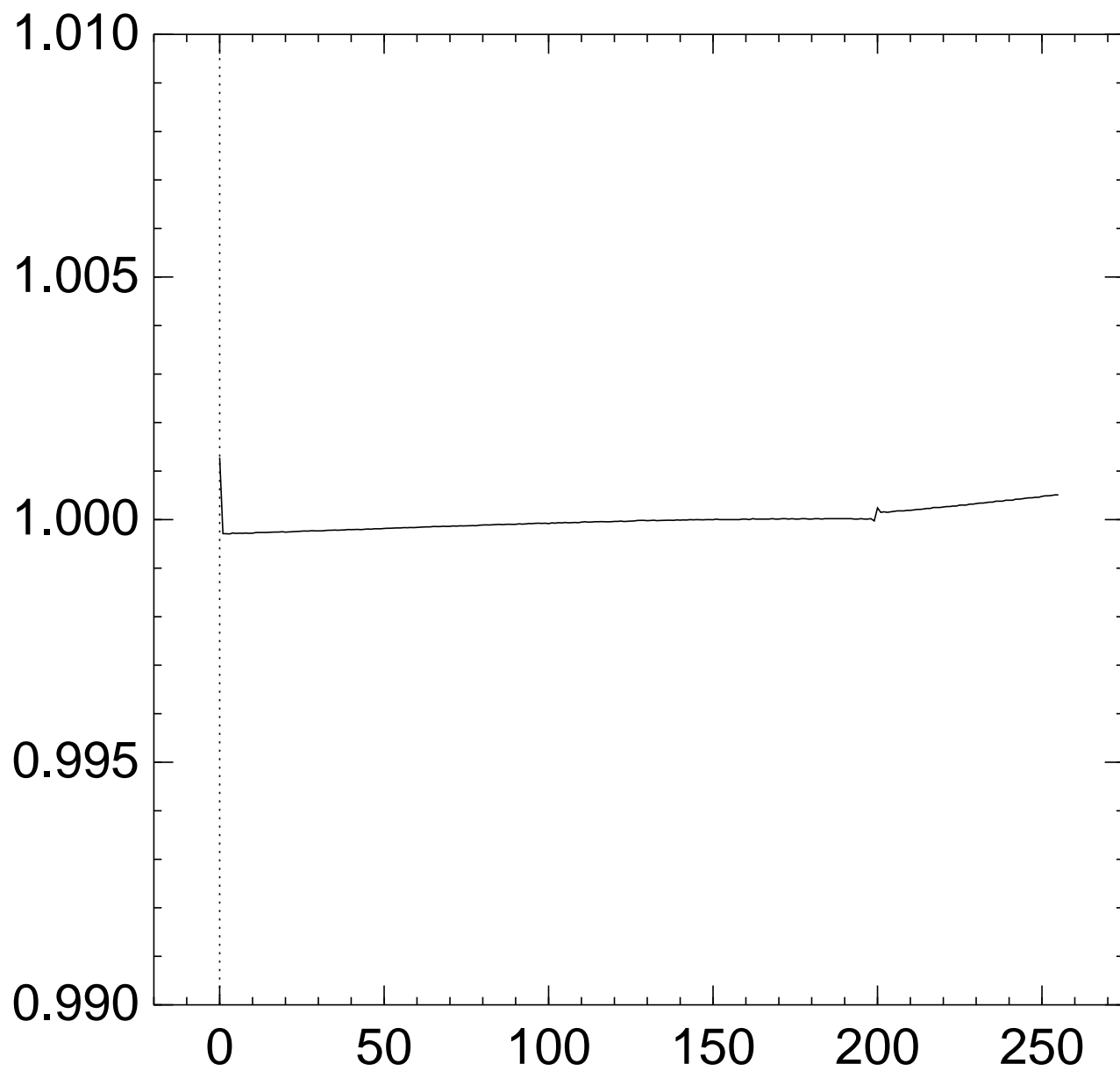
Graph of $256 \Pr[z_{198} = x]$:



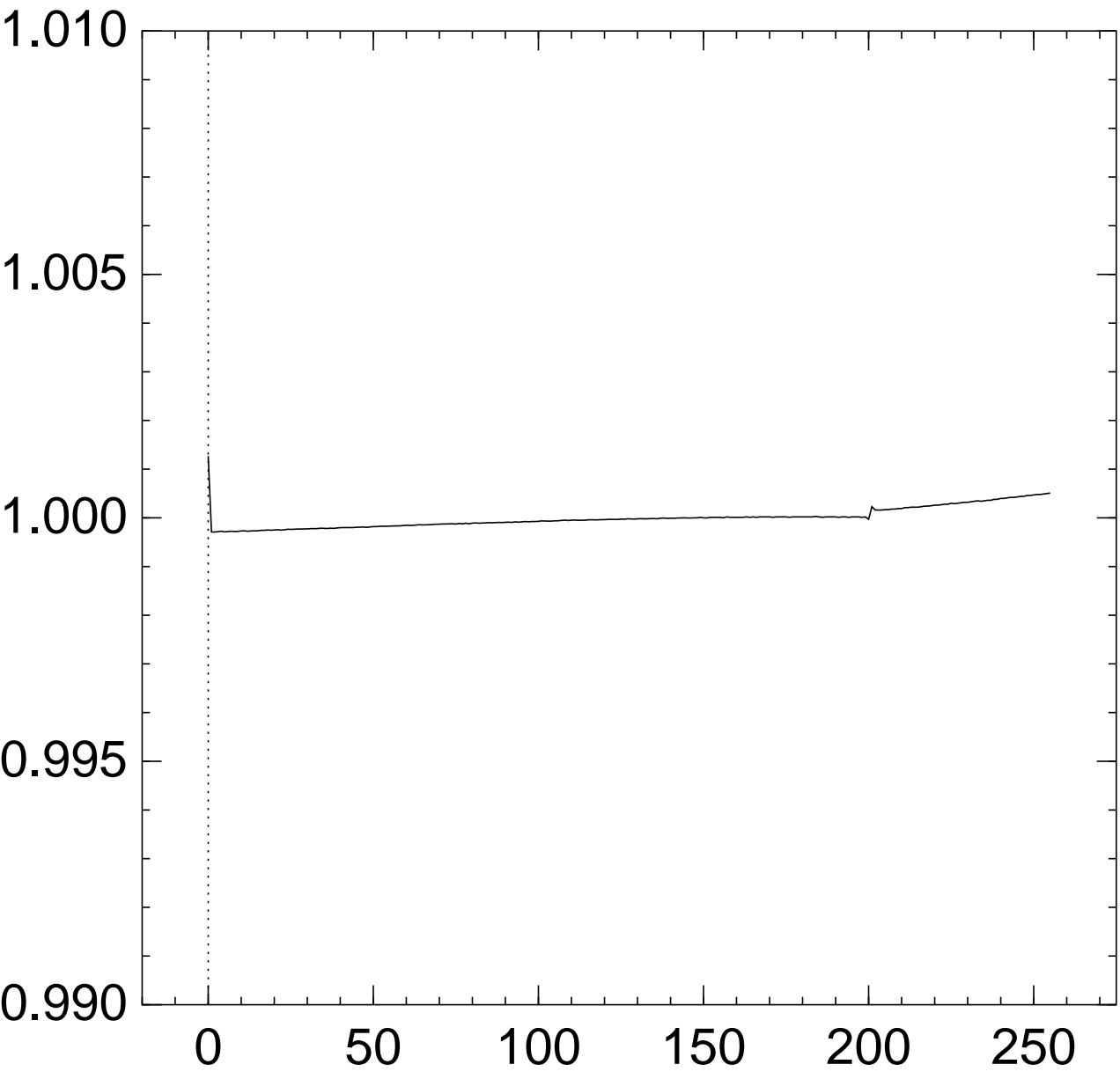
Graph of $256 \Pr[z_{199} = x]$:



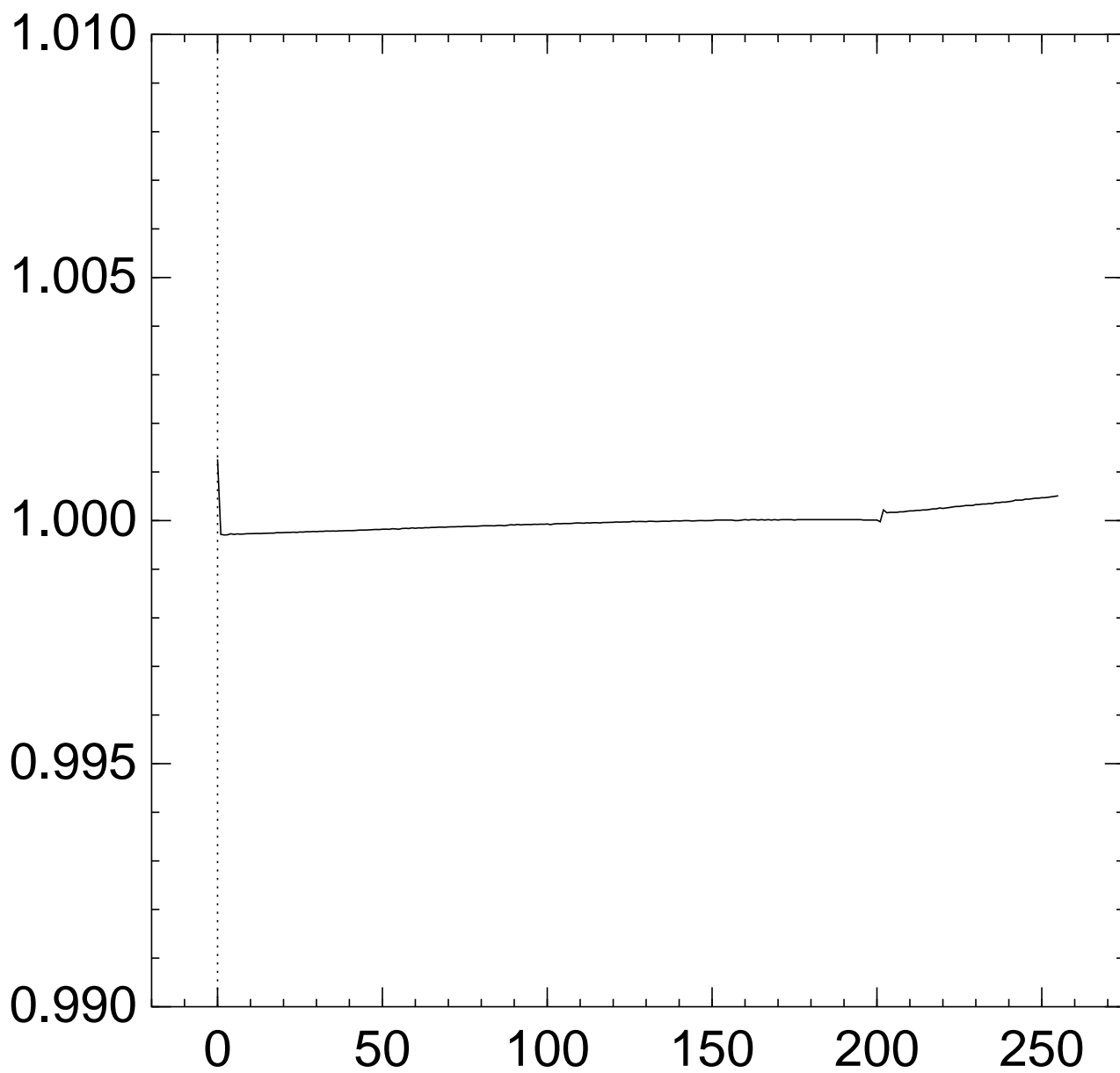
Graph of $256 \Pr[z_{200} = x]$:



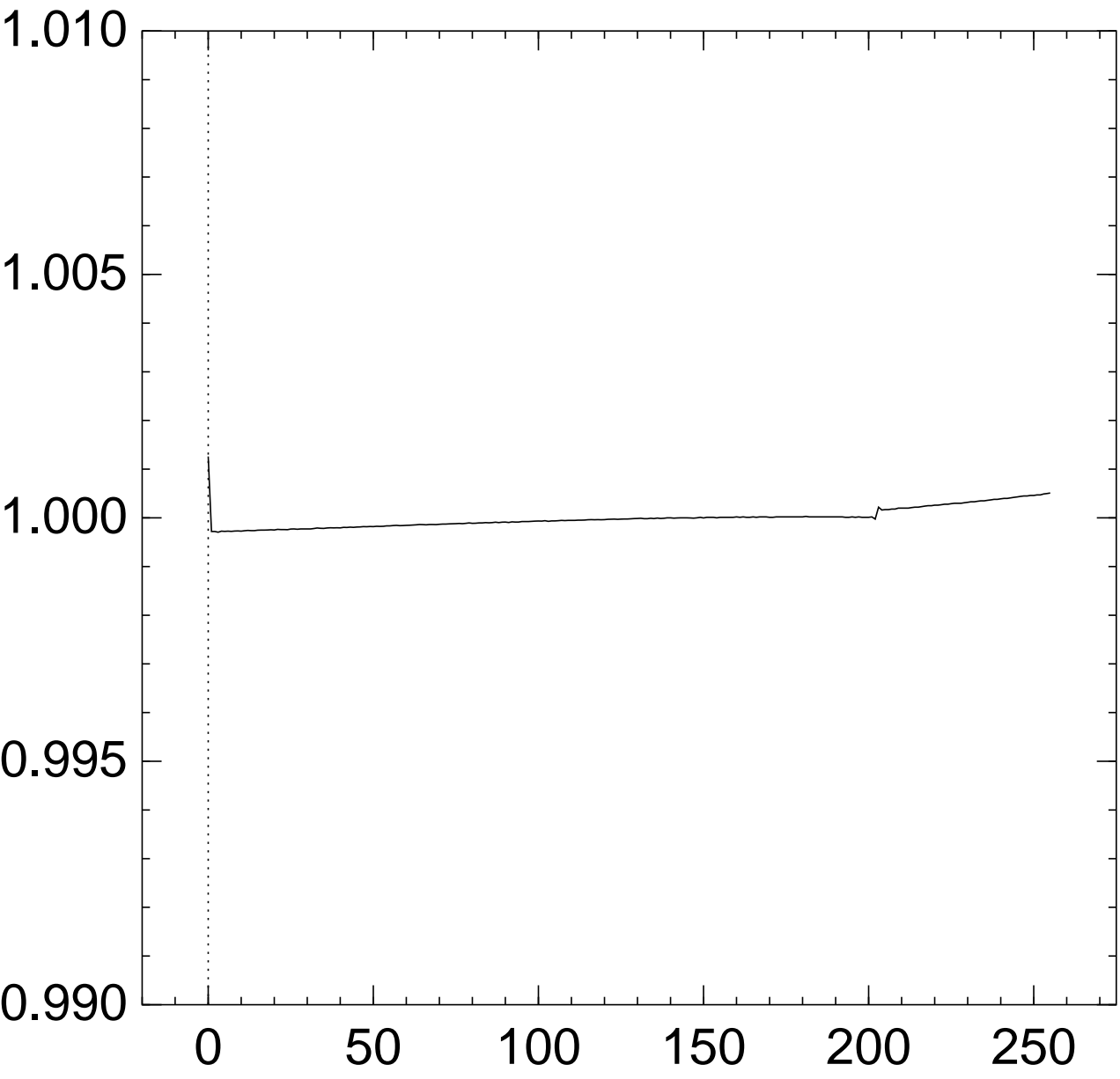
Graph of $256 \Pr[z_{201} = x]$:



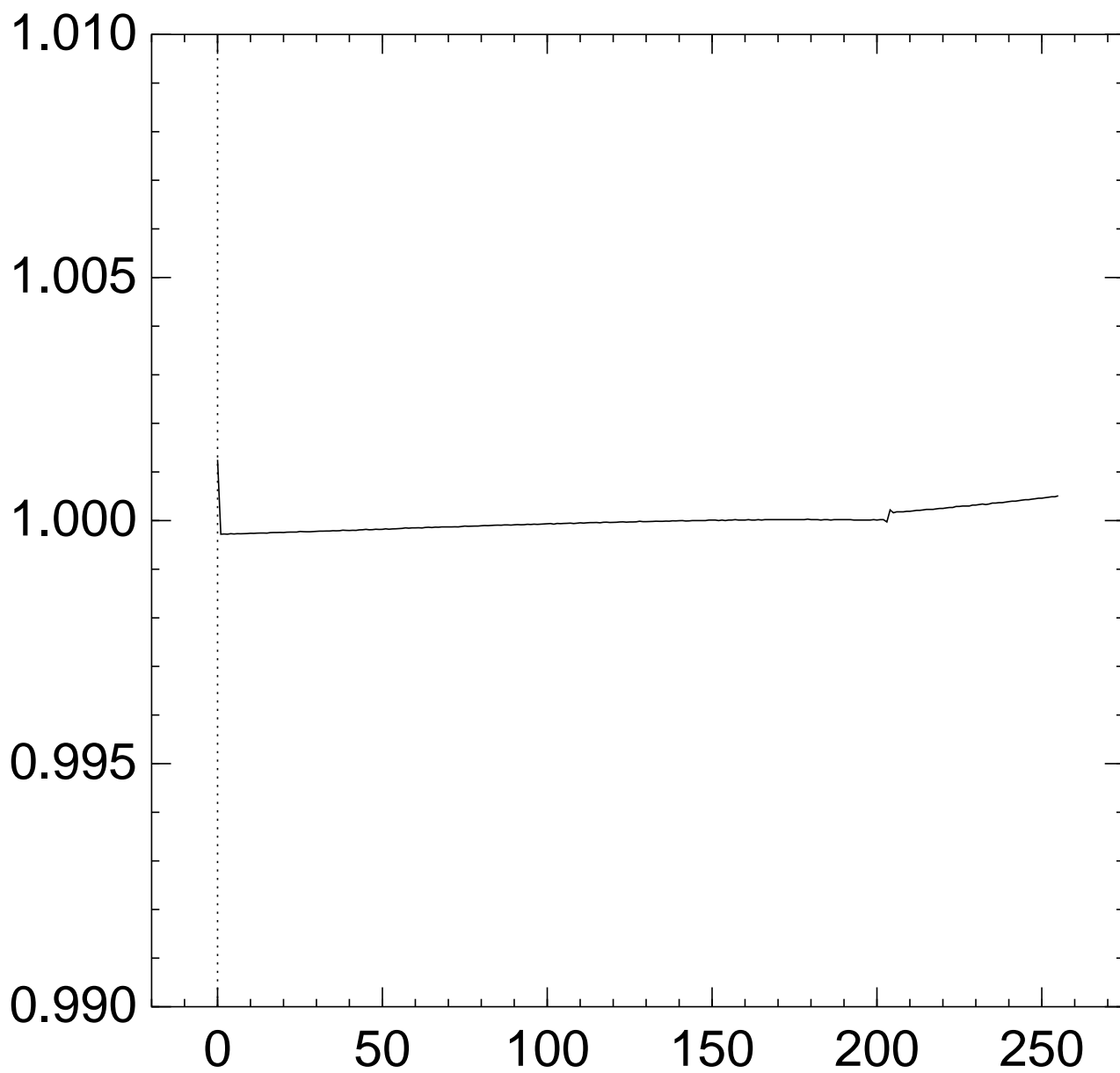
Graph of $256 \Pr[z_{202} = x]$:



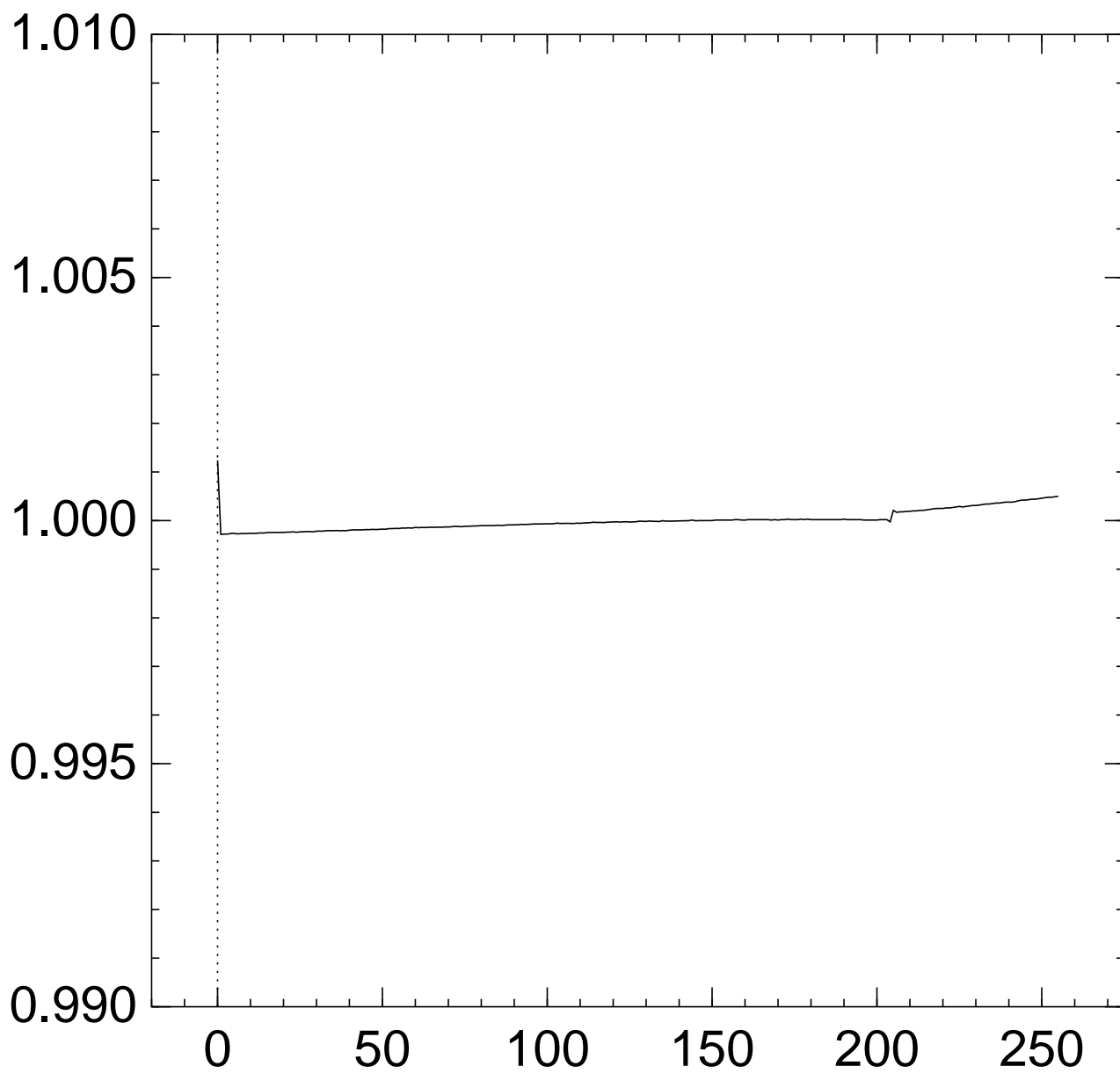
Graph of $256 \Pr[z_{203} = x]$:



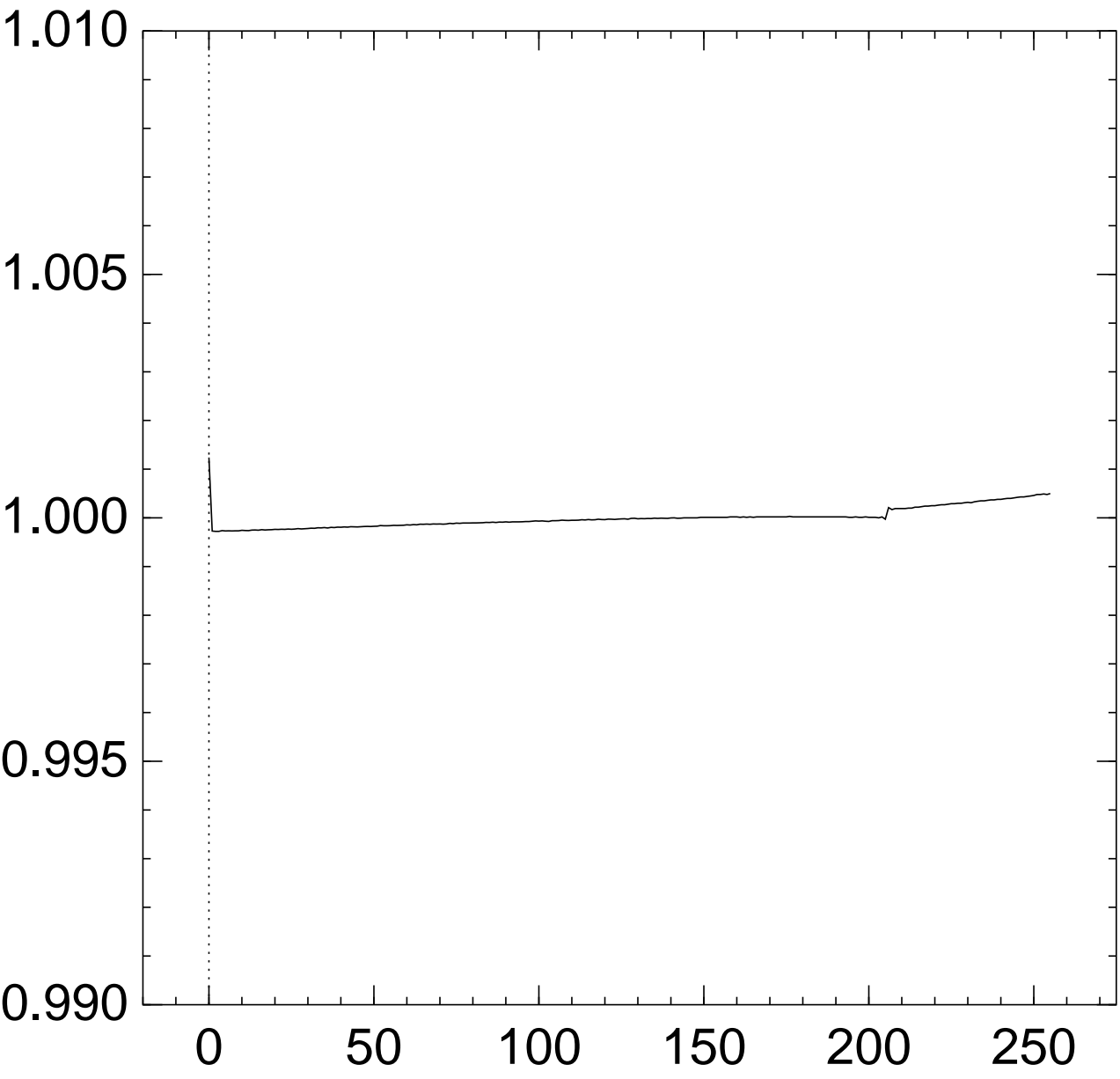
Graph of $256 \Pr[z_{204} = x]$:



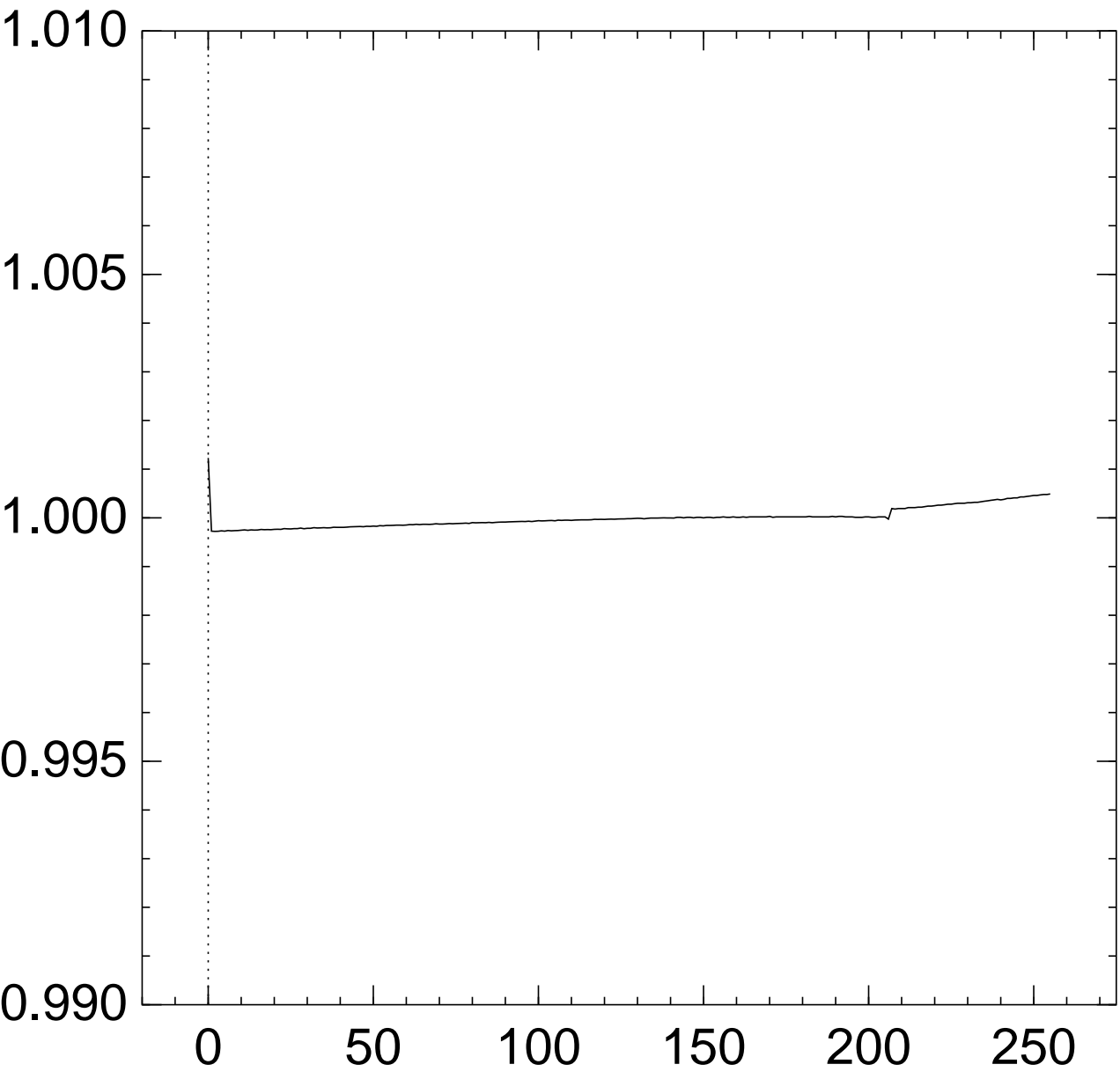
Graph of $256 \Pr[z_{205} = x]$:



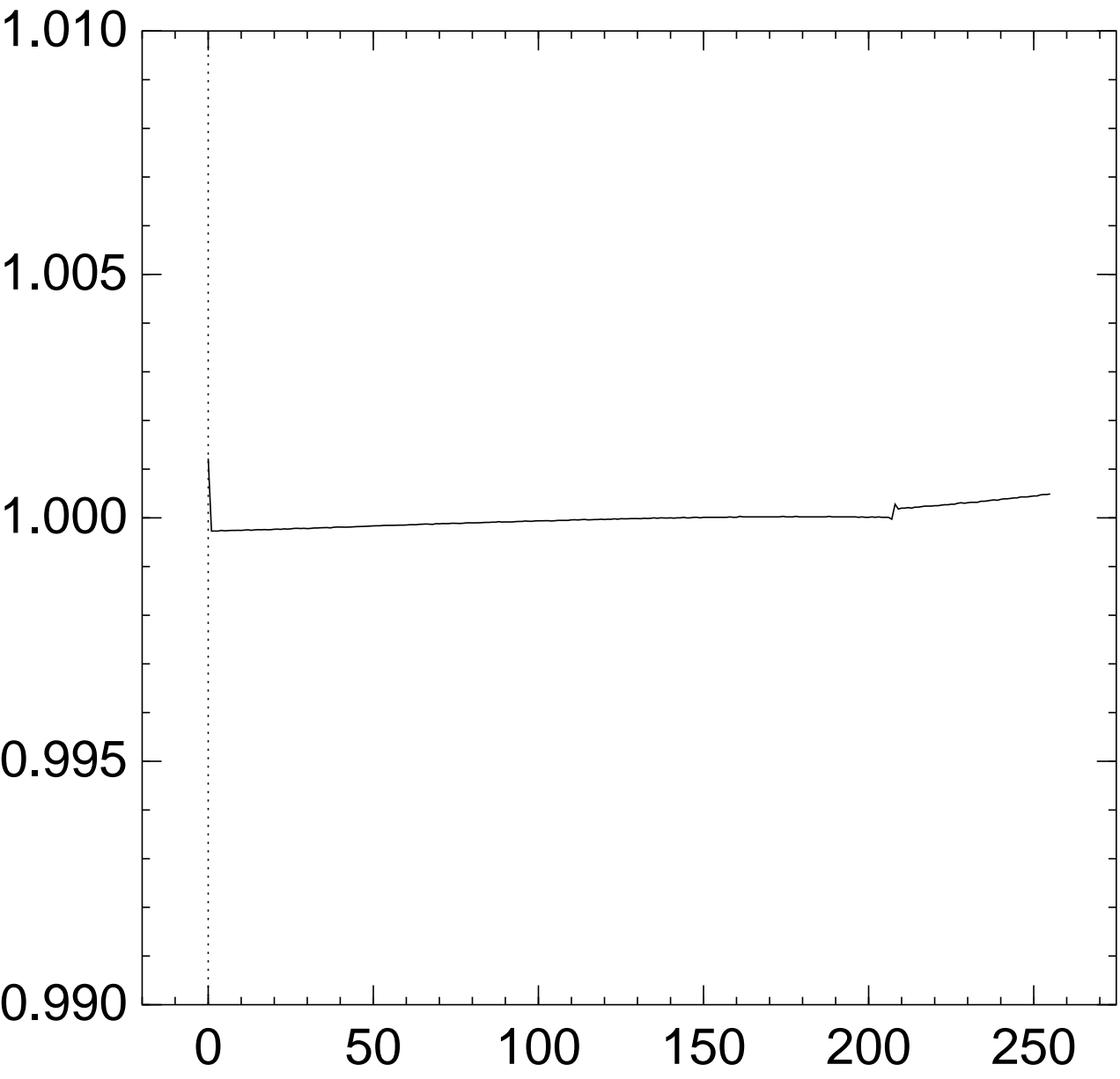
Graph of $256 \Pr[z_{206} = x]$:



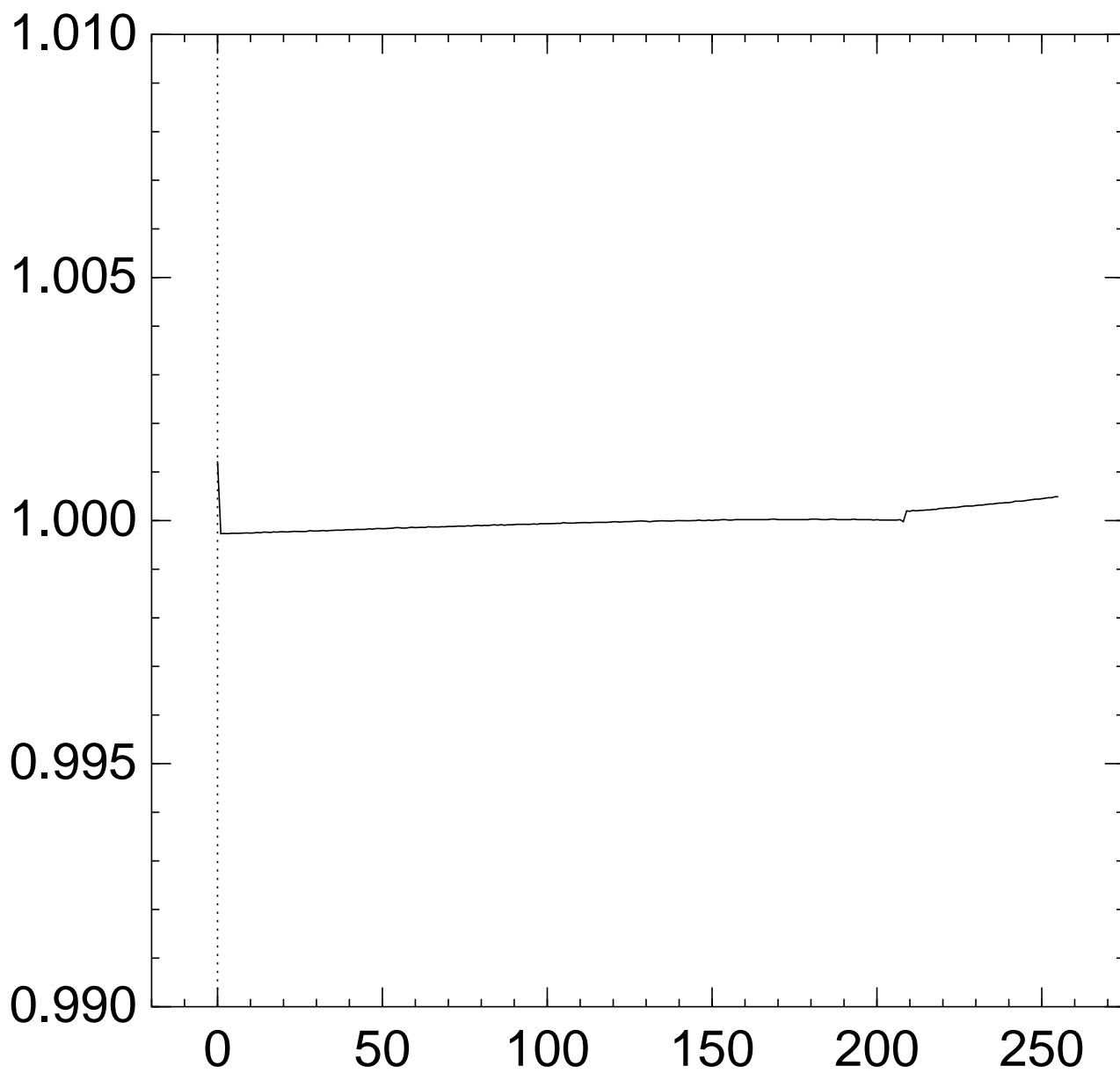
Graph of $256 \Pr[z_{207} = x]$:



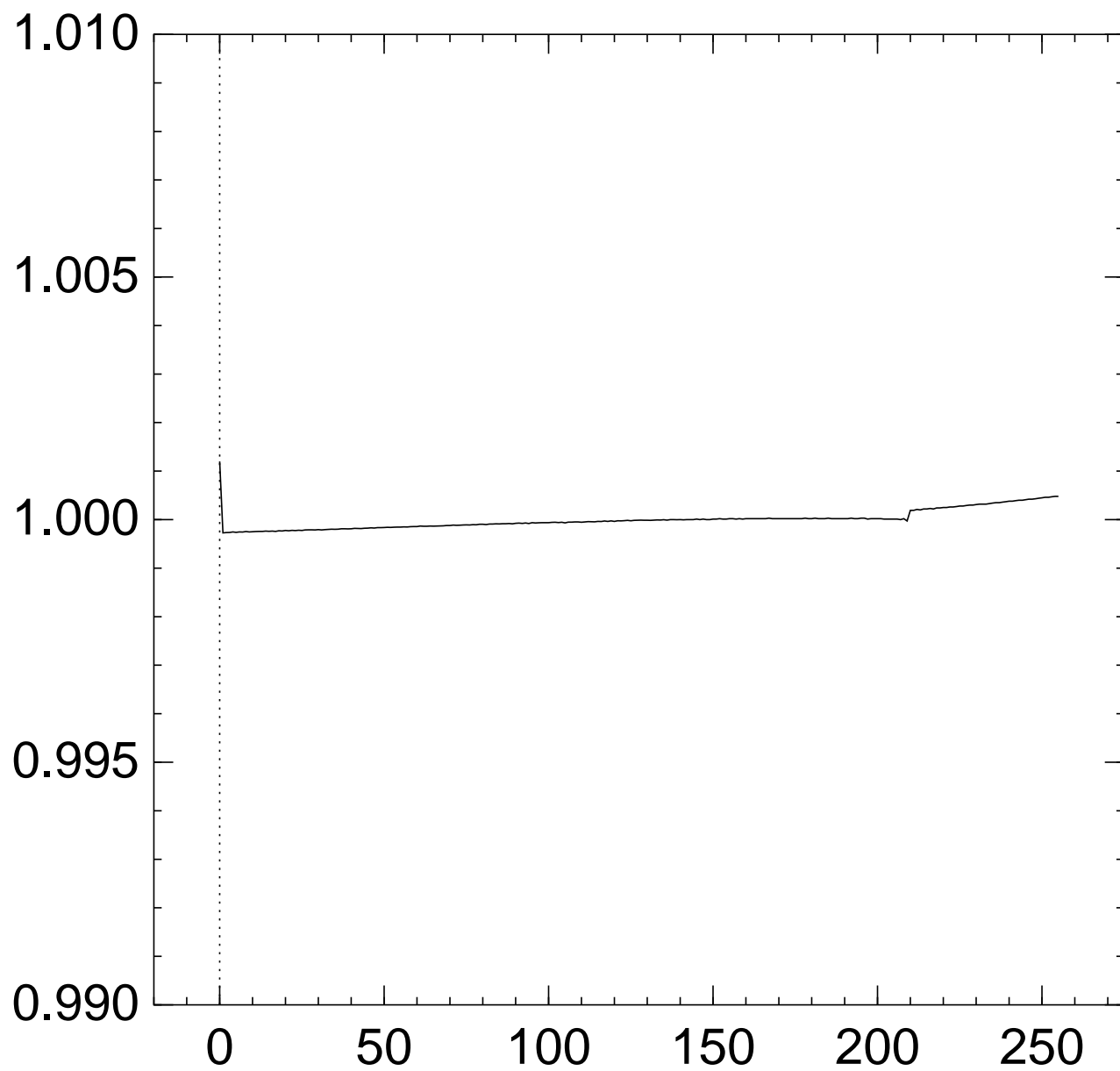
Graph of $256 \Pr[z_{208} = x]$:



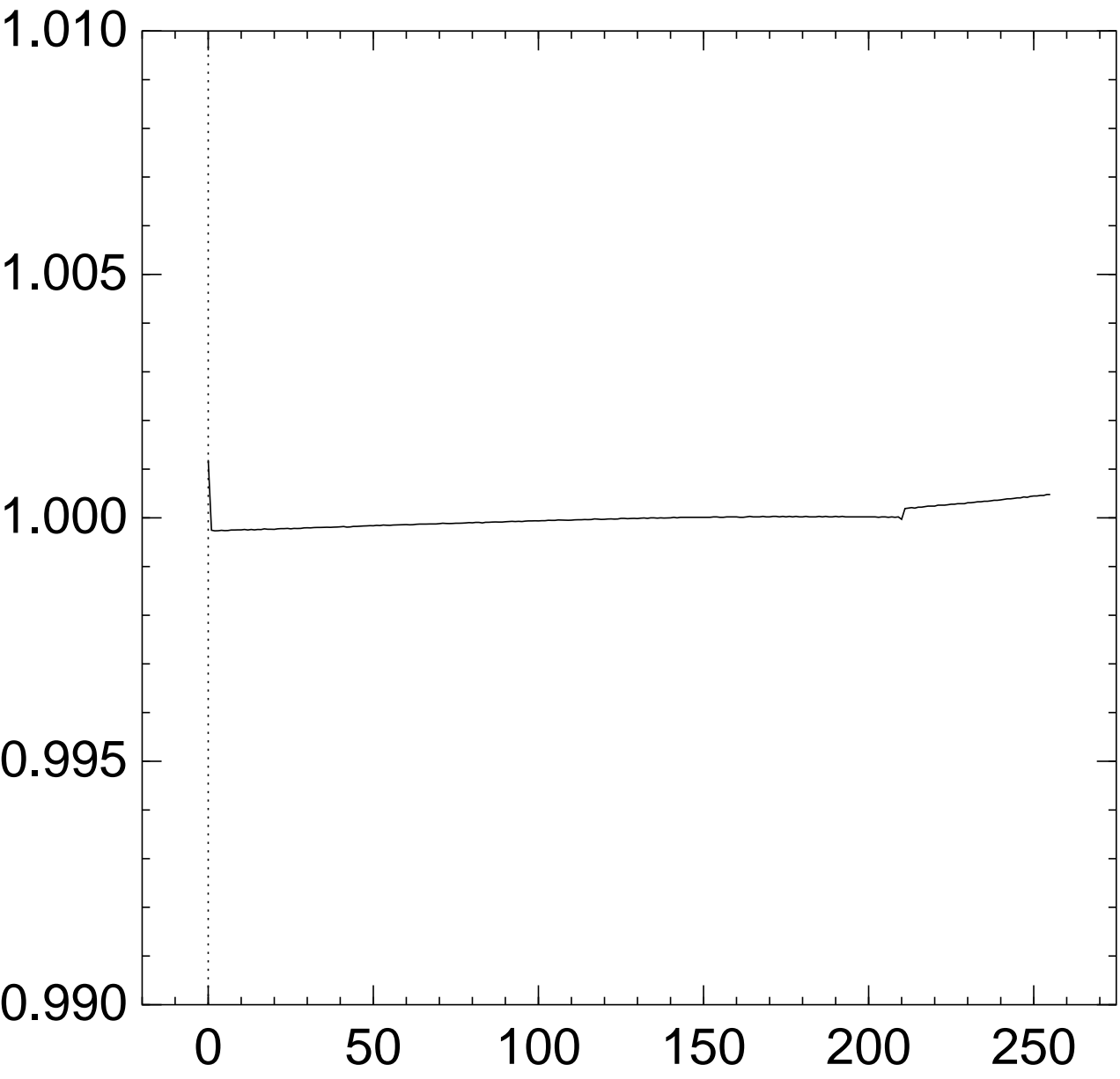
Graph of $256 \Pr[z_{209} = x]$:



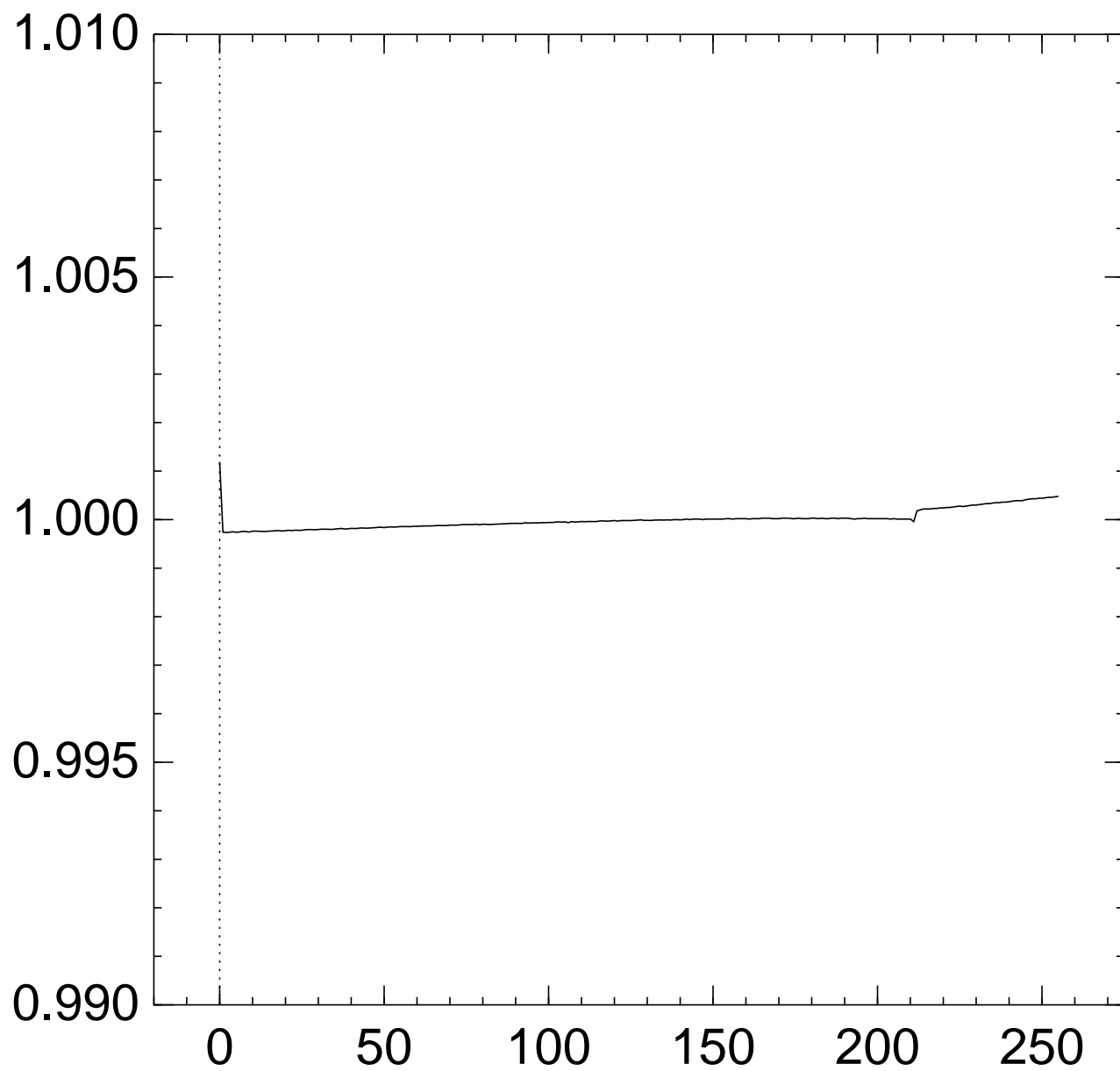
Graph of $256 \Pr[z_{210} = x]$:



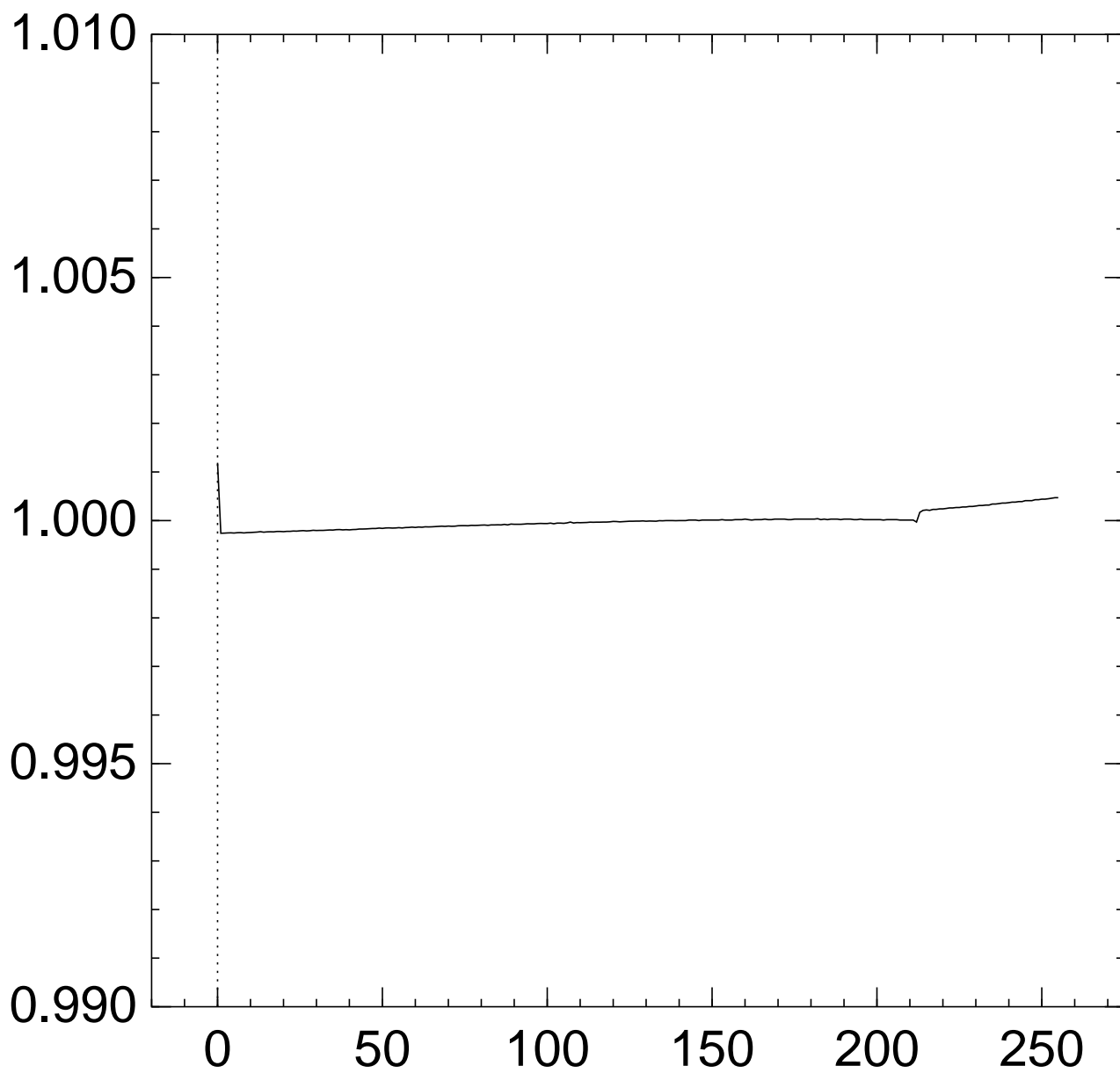
Graph of $256 \Pr[z_{211} = x]$:



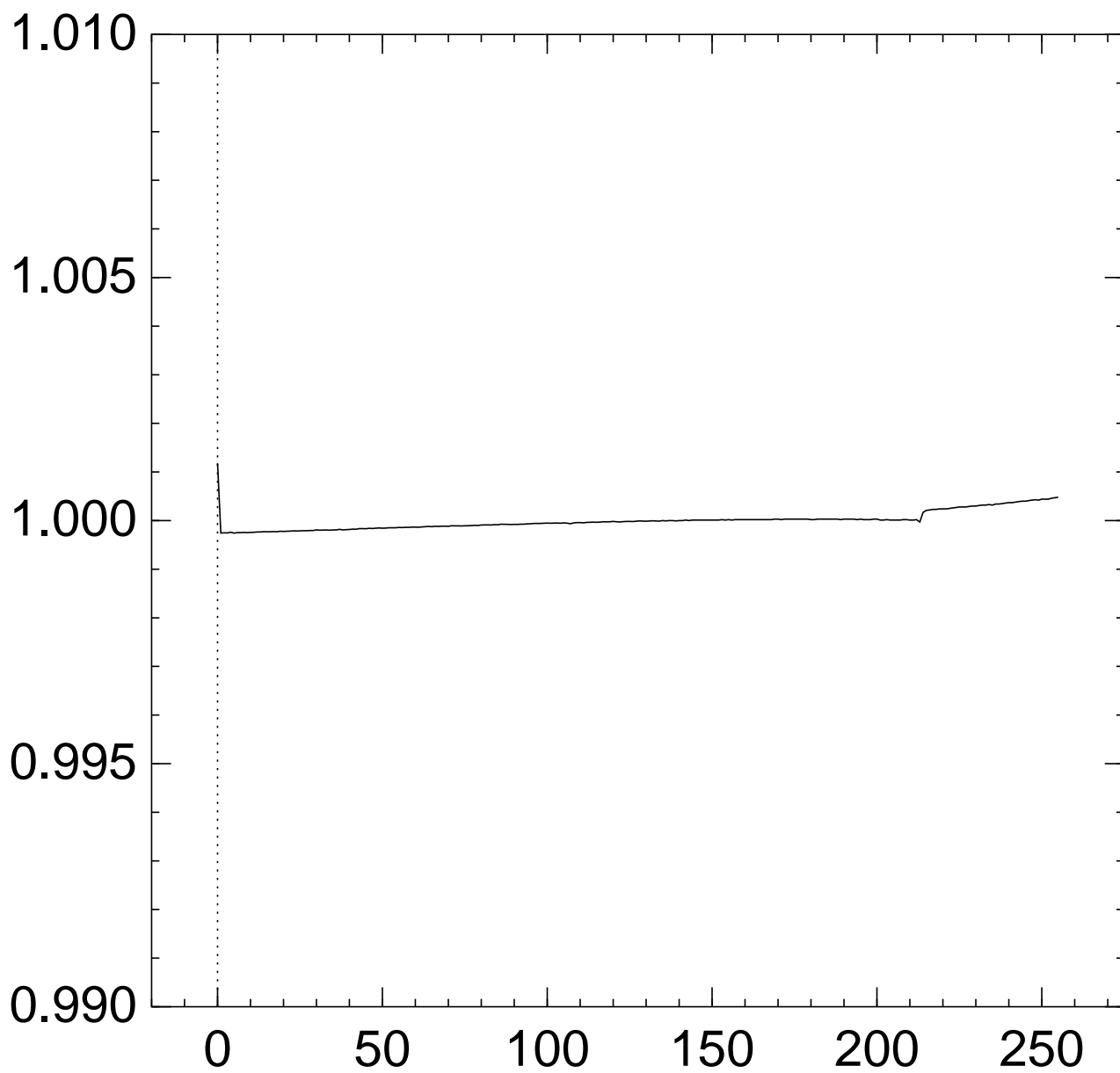
Graph of $256 \Pr[z_{212} = x]$:



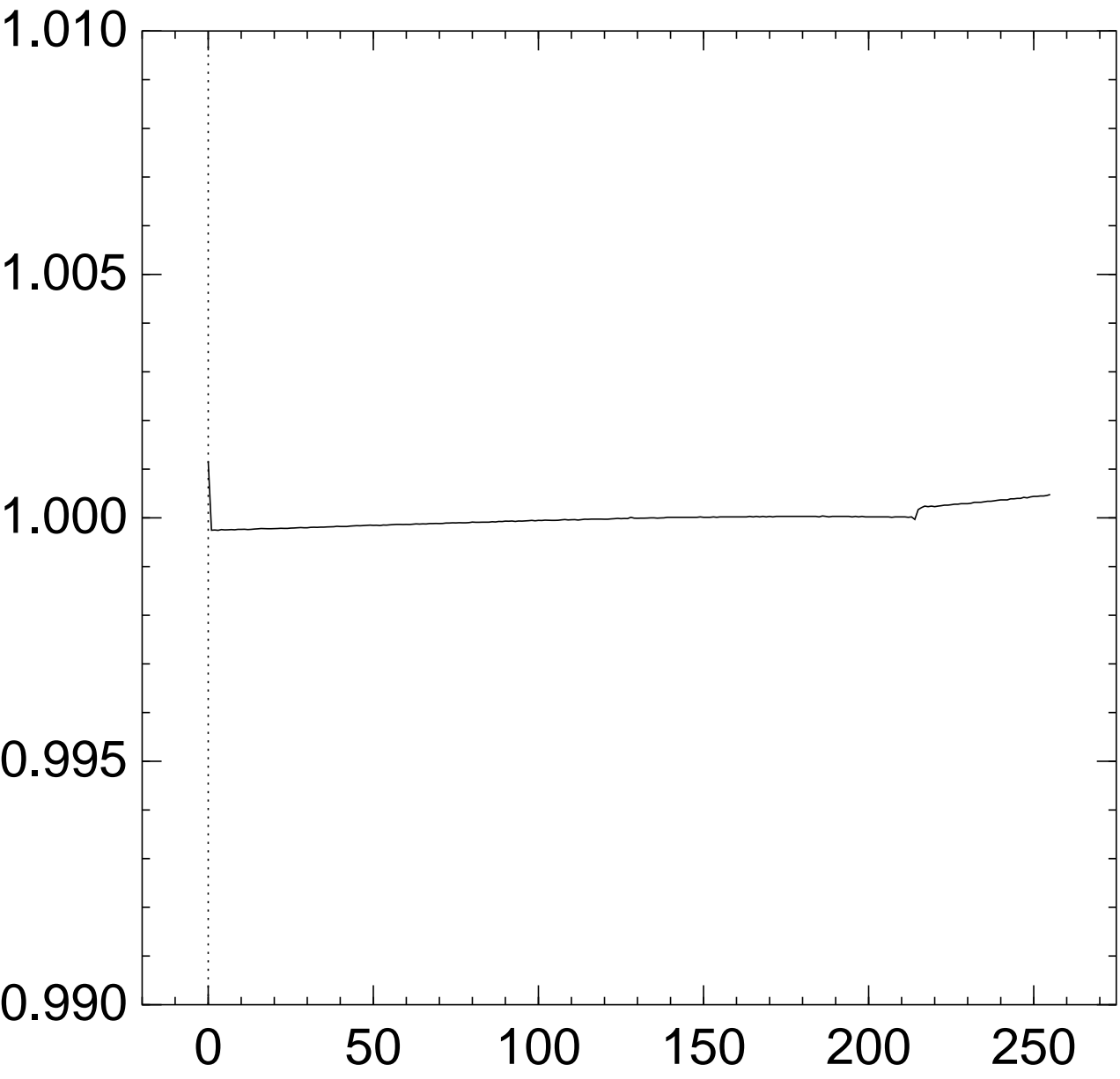
Graph of $256 \Pr[z_{213} = x]$:



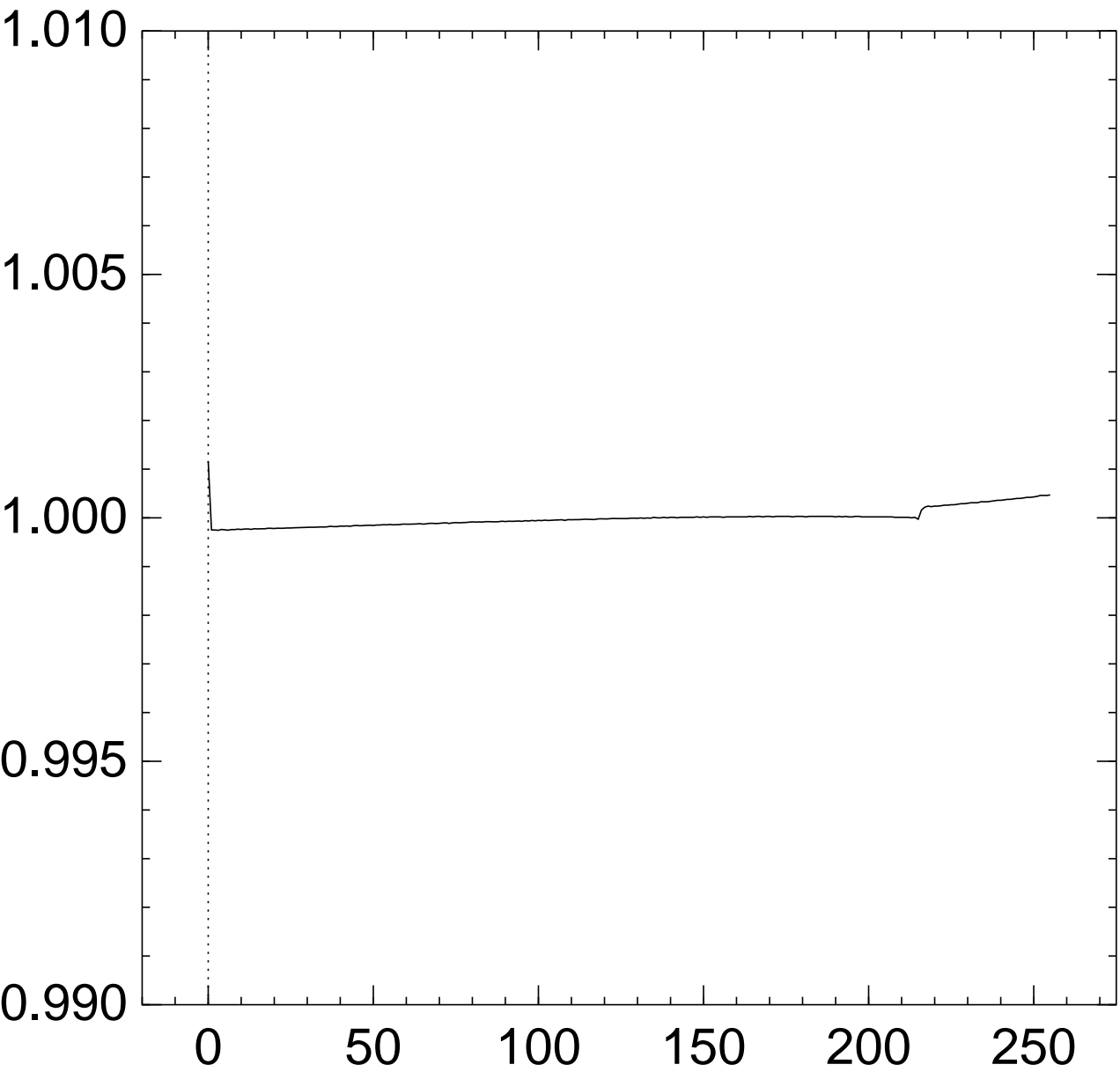
Graph of $256 \Pr[z_{214} = x]$:



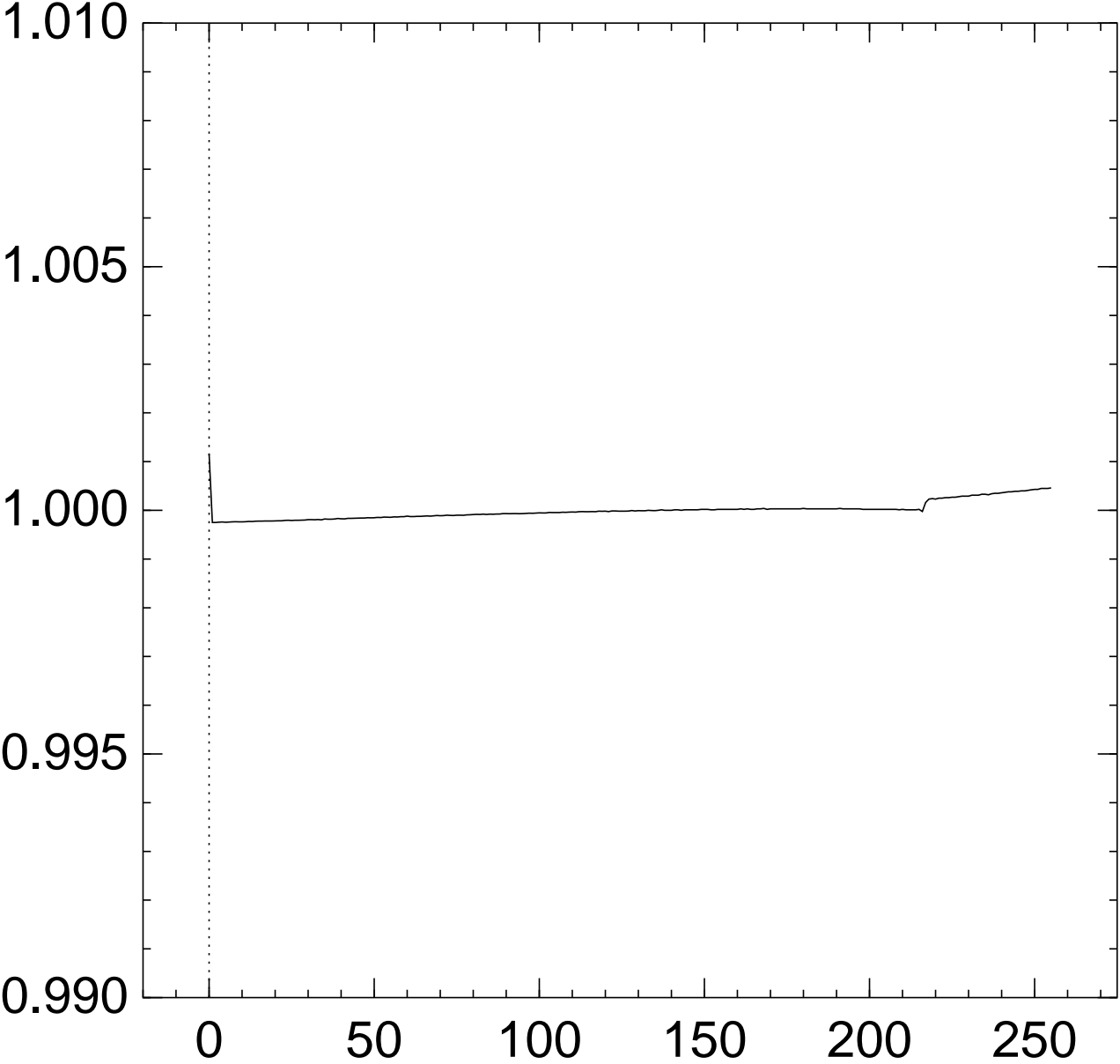
Graph of $256 \Pr[z_{215} = x]$:



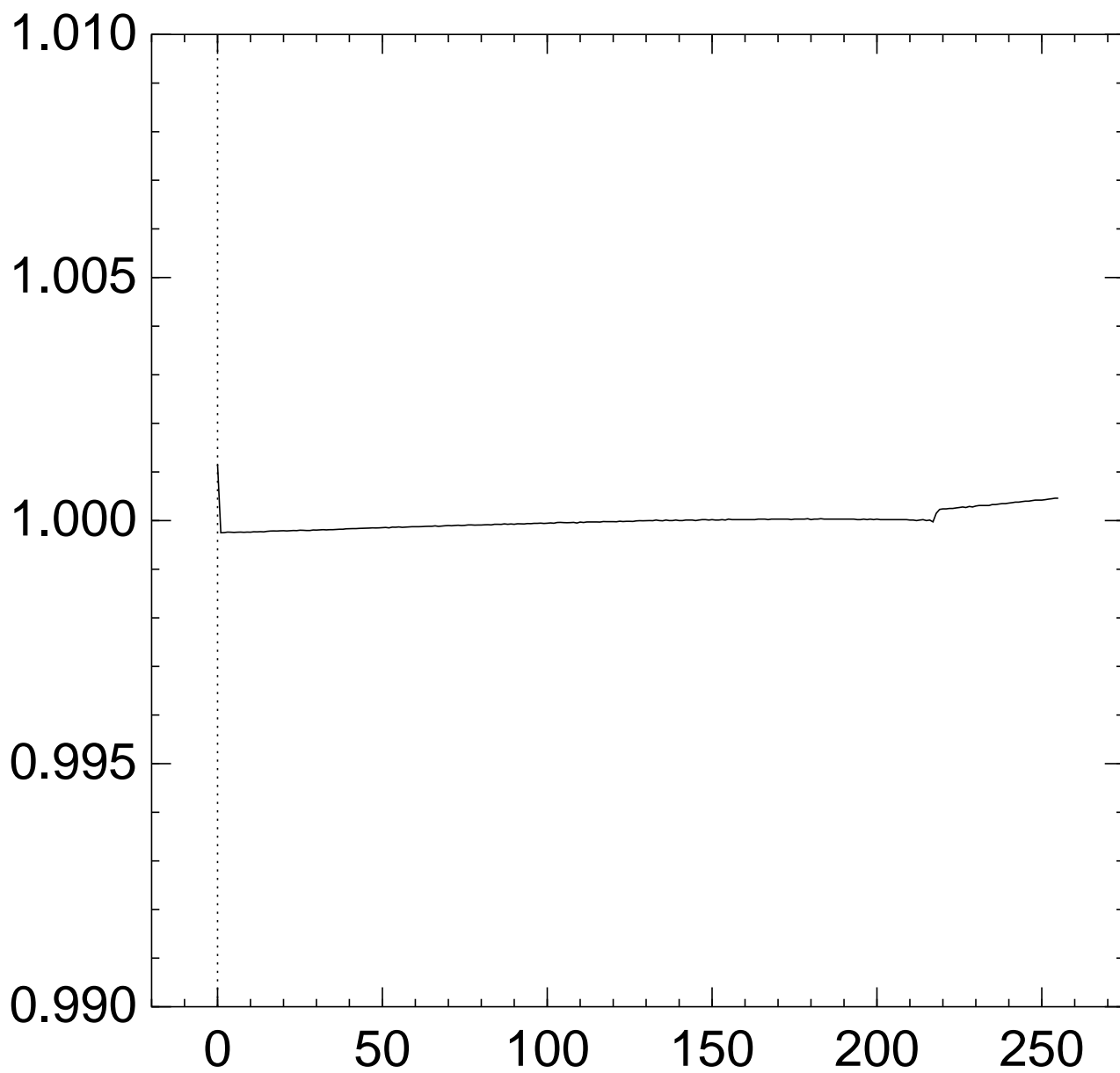
Graph of $256 \Pr[z_{216} = x]$:



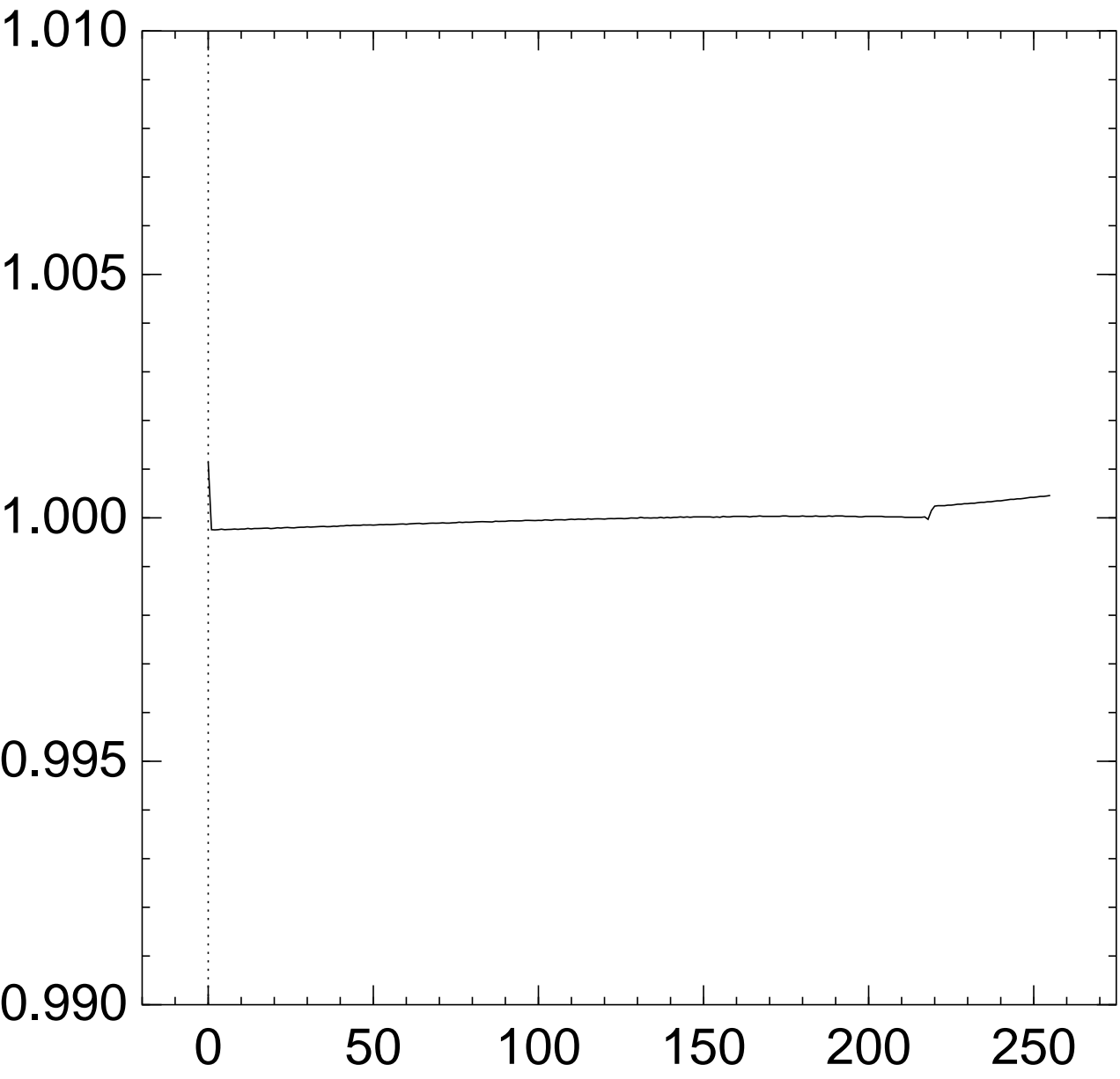
Graph of $256 \Pr[z_{217} = x]$:



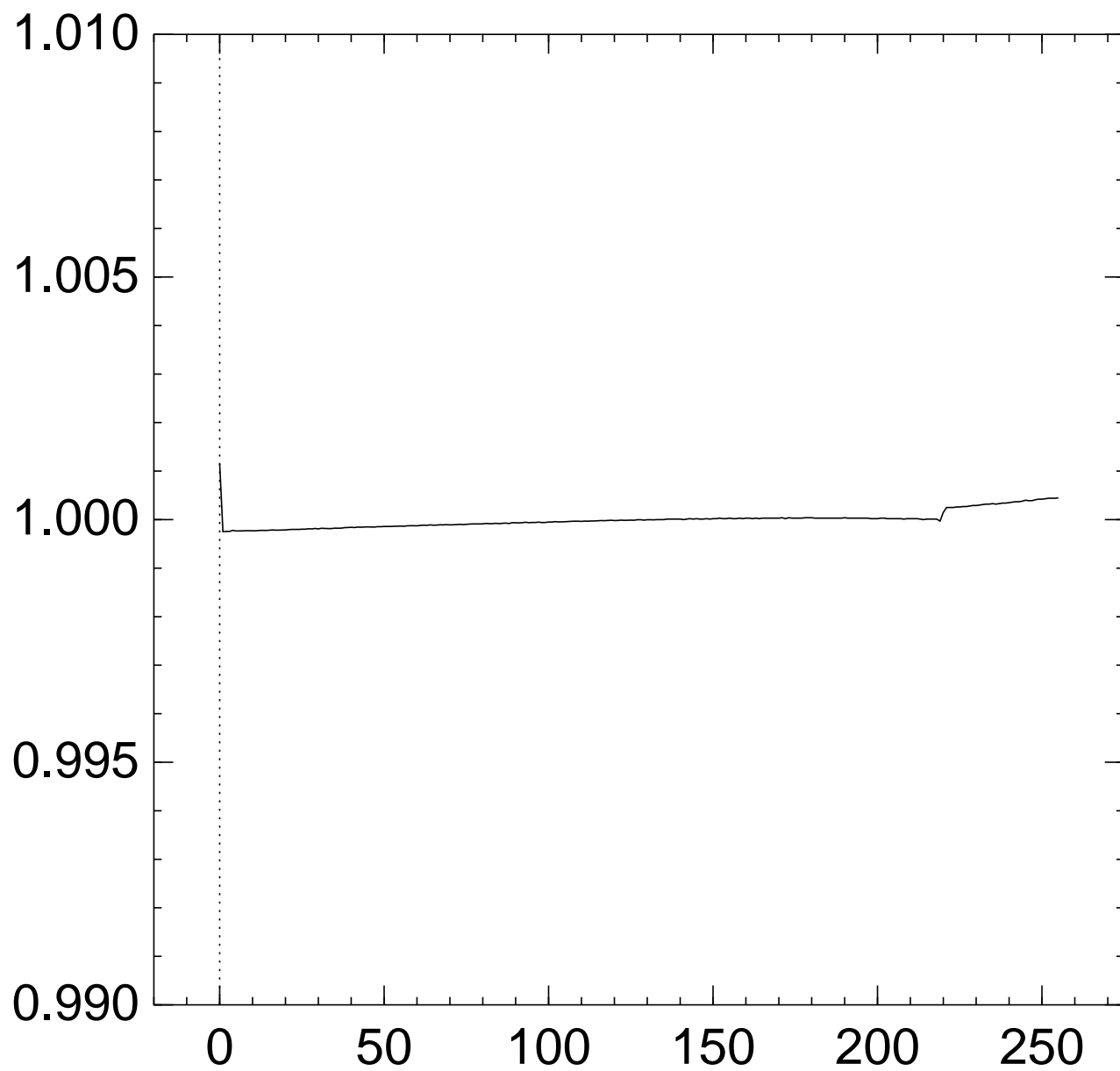
Graph of $256 \Pr[z_{218} = x]$:



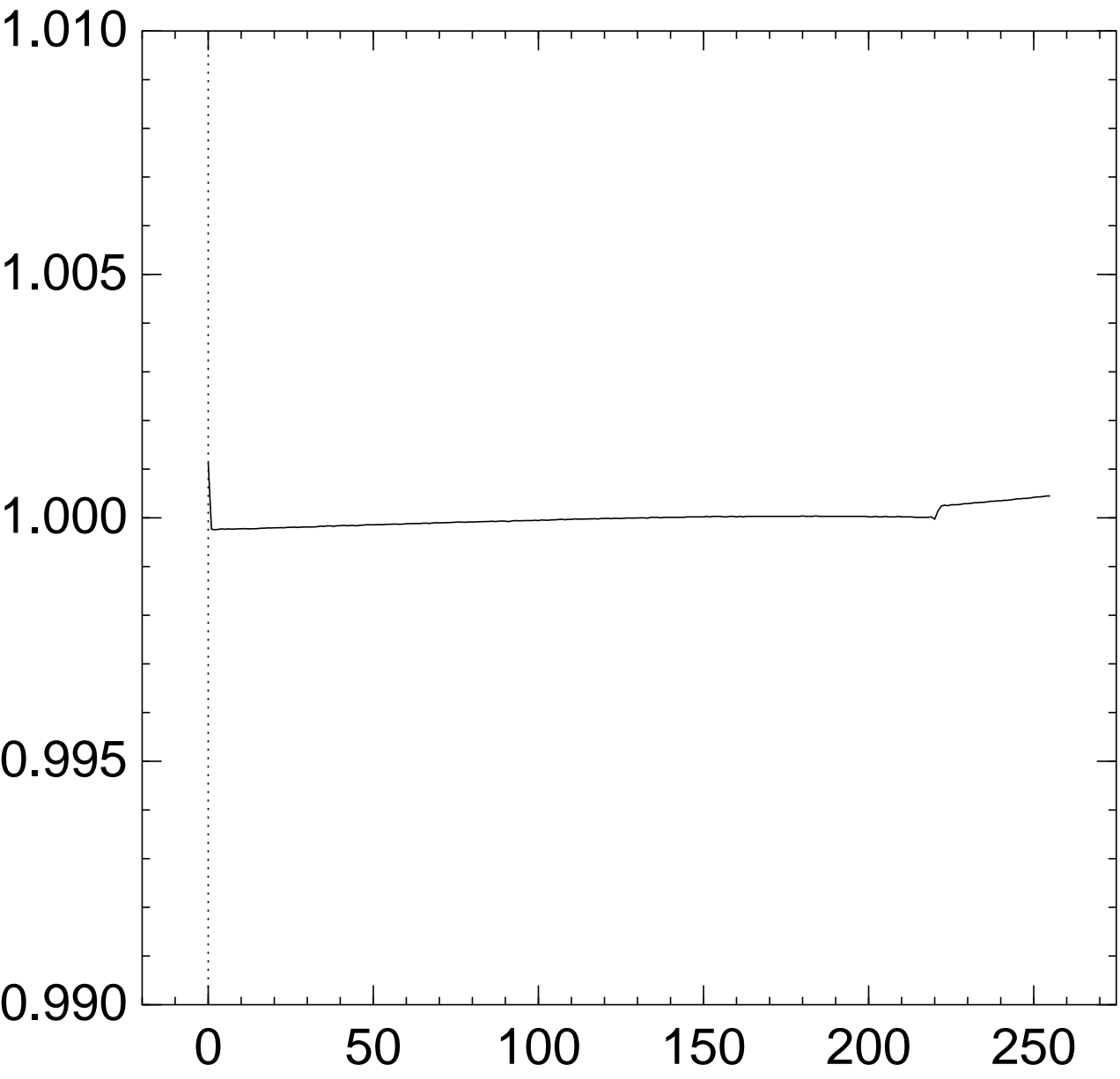
Graph of $256 \Pr[z_{219} = x]$:



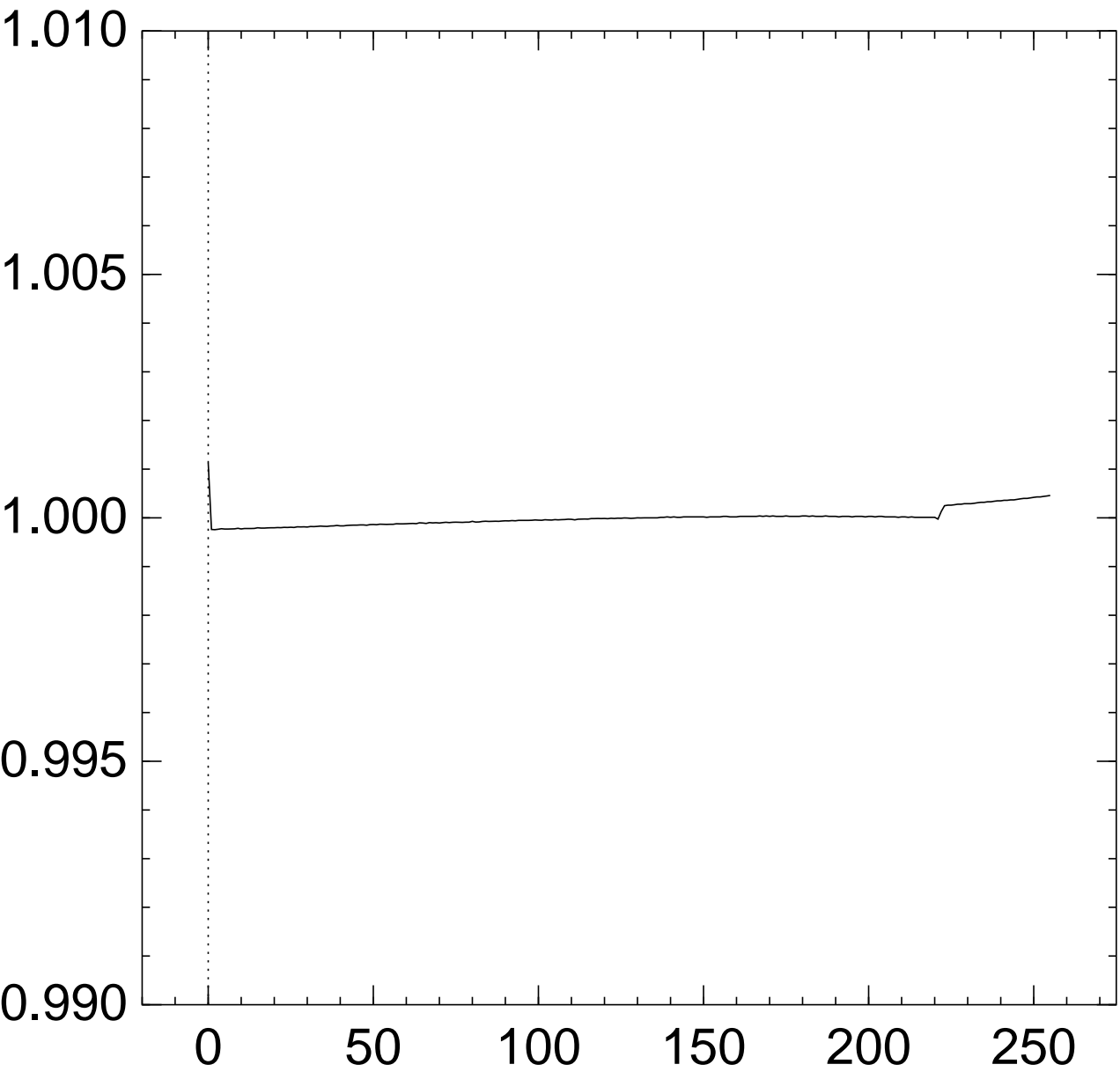
Graph of $256 \Pr[z_{220} = x]$:



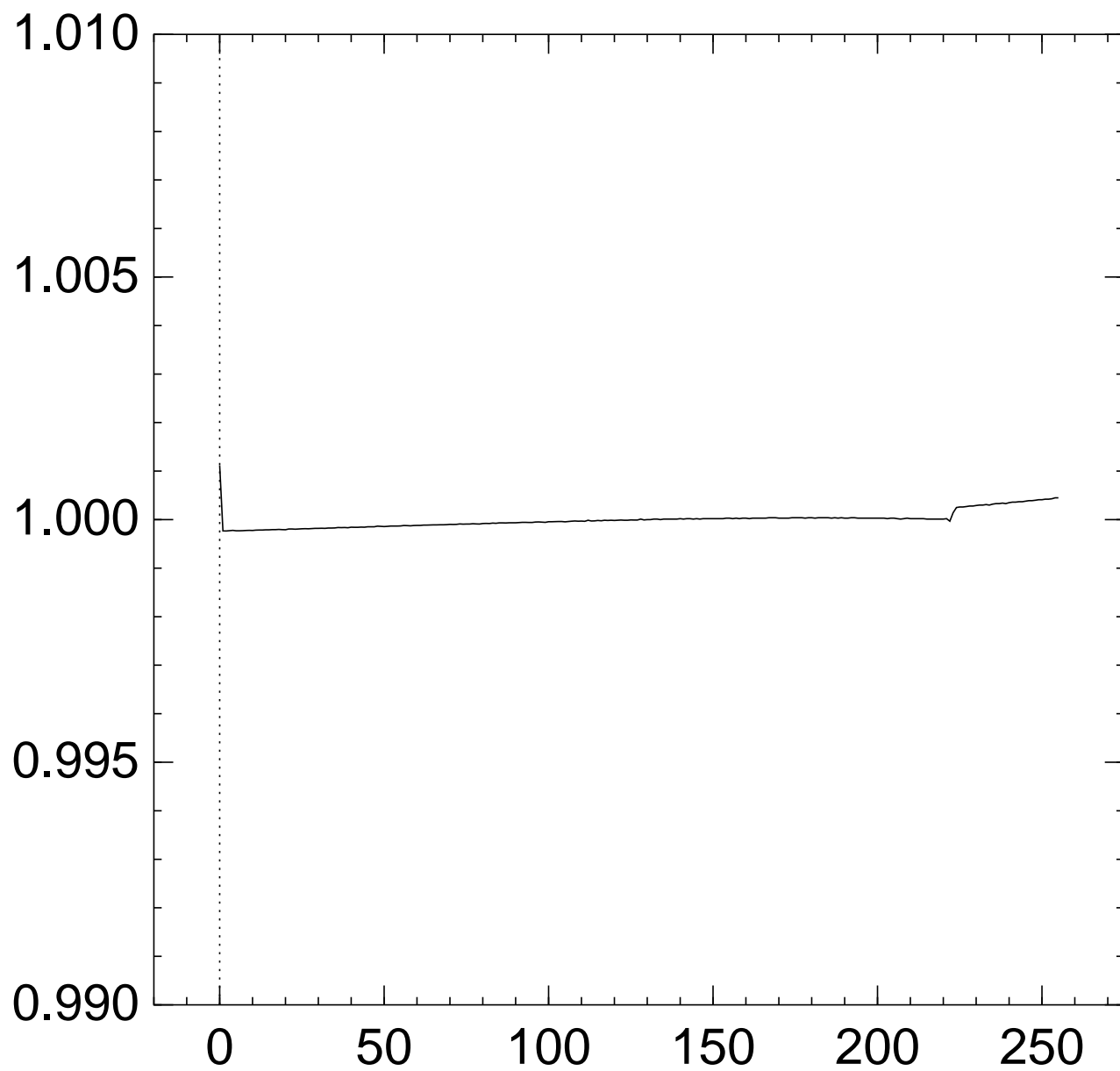
Graph of $256 \Pr[z_{221} = x]$:



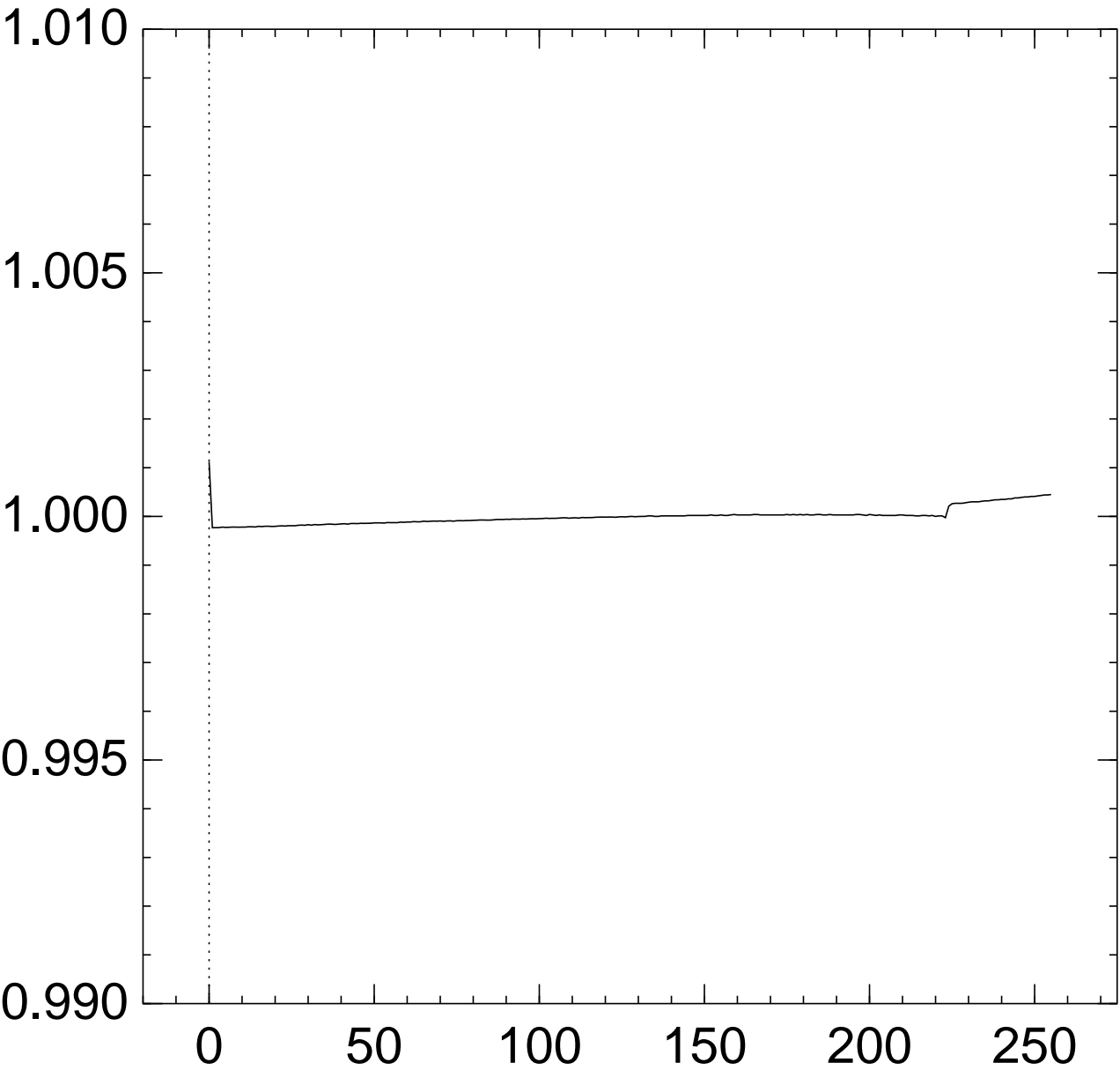
Graph of $256 \Pr[z_{222} = x]$:



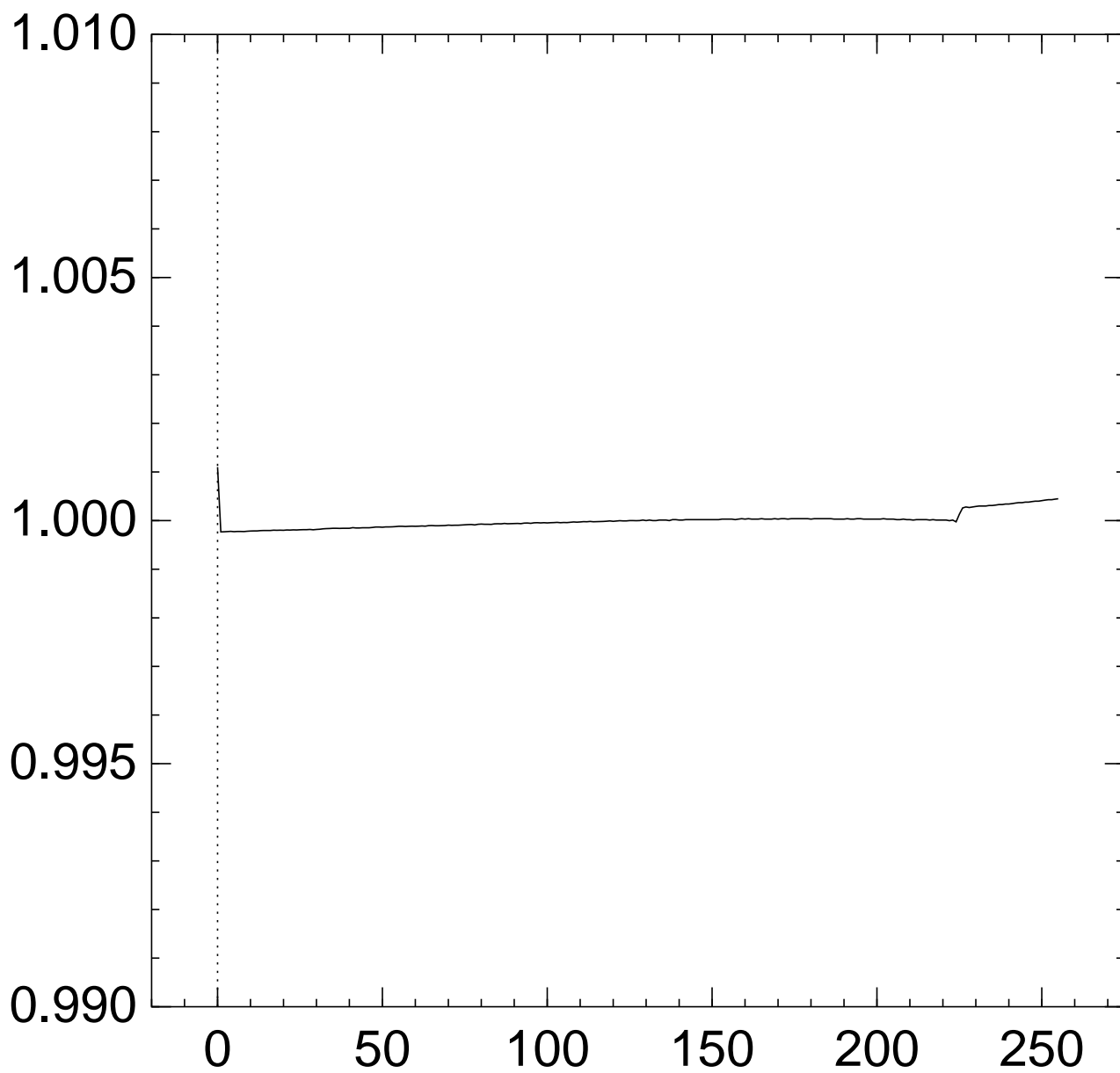
Graph of $256 \Pr[z_{223} = x]$:



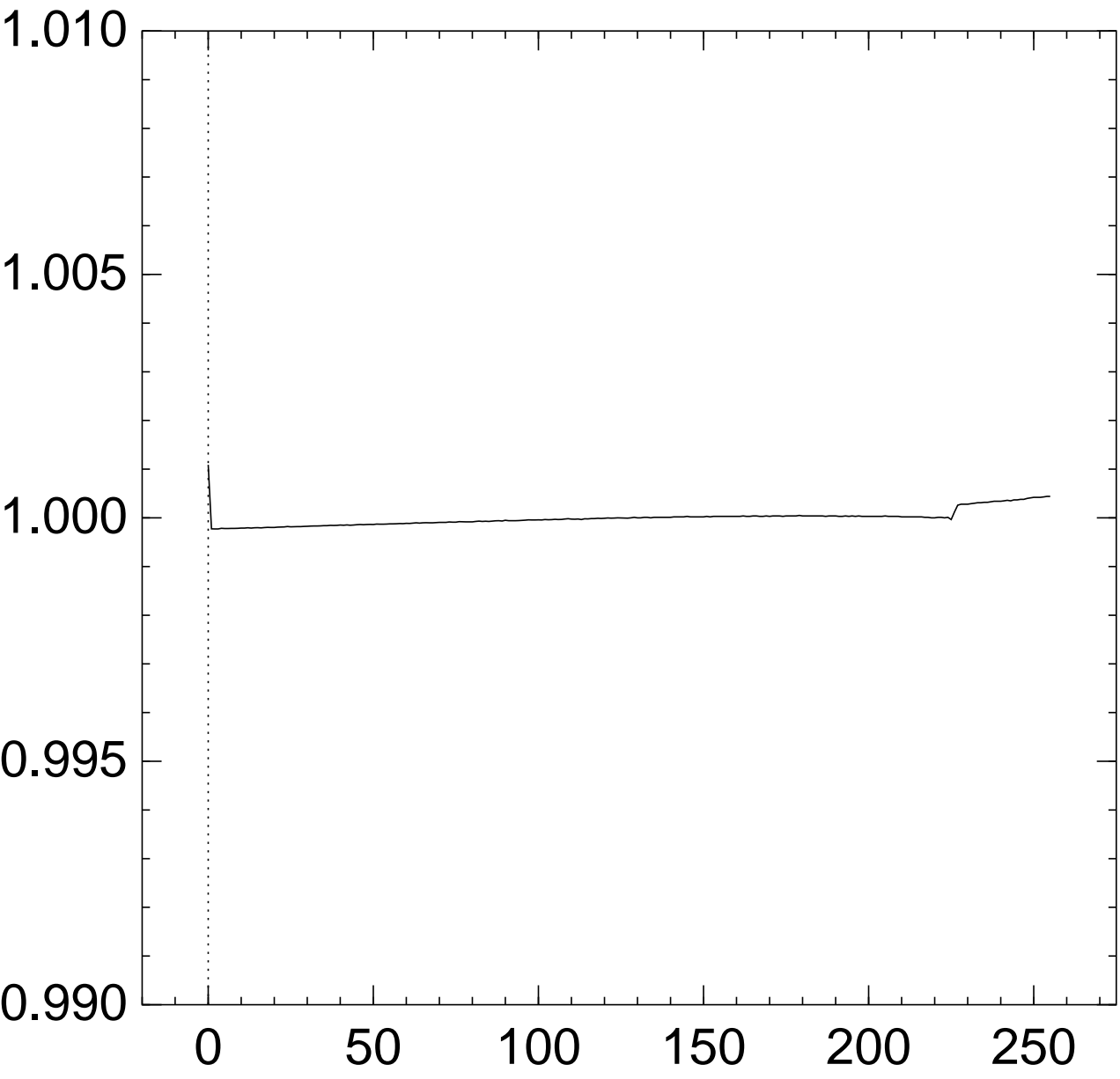
Graph of $256 \Pr[z_{224} = x]$:



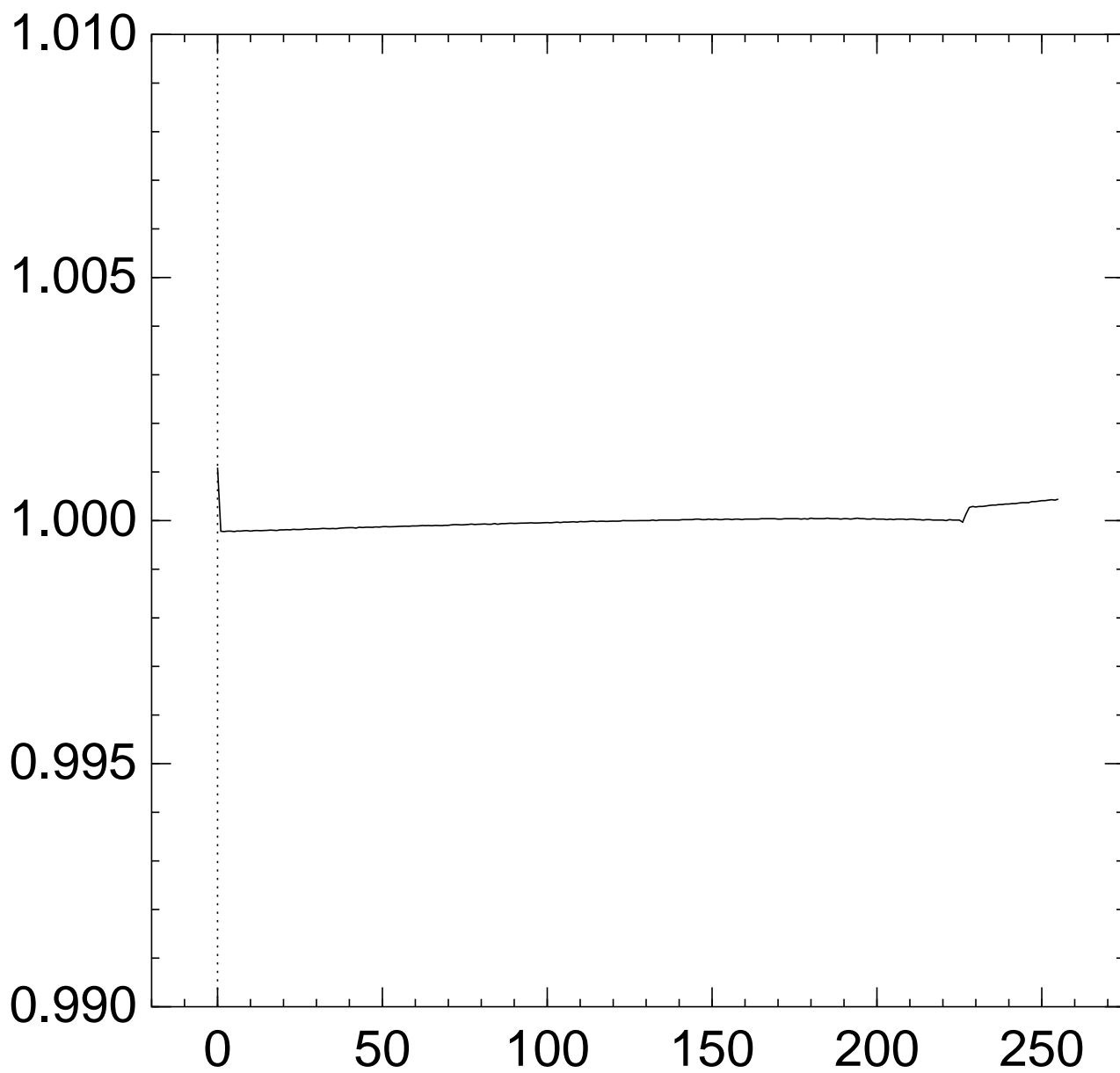
Graph of $256 \Pr[z_{225} = x]$:



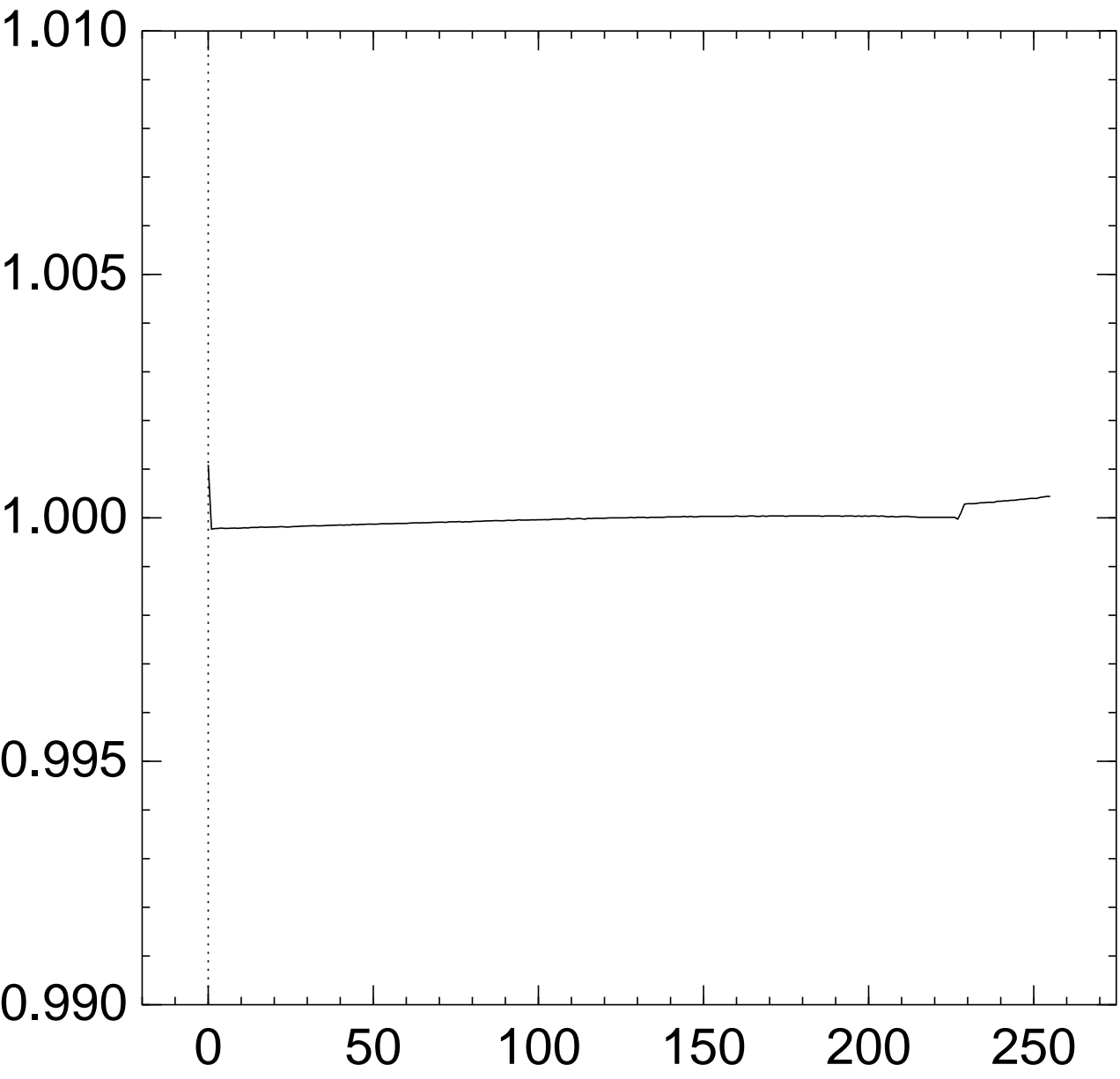
Graph of $256 \Pr[z_{226} = x]$:



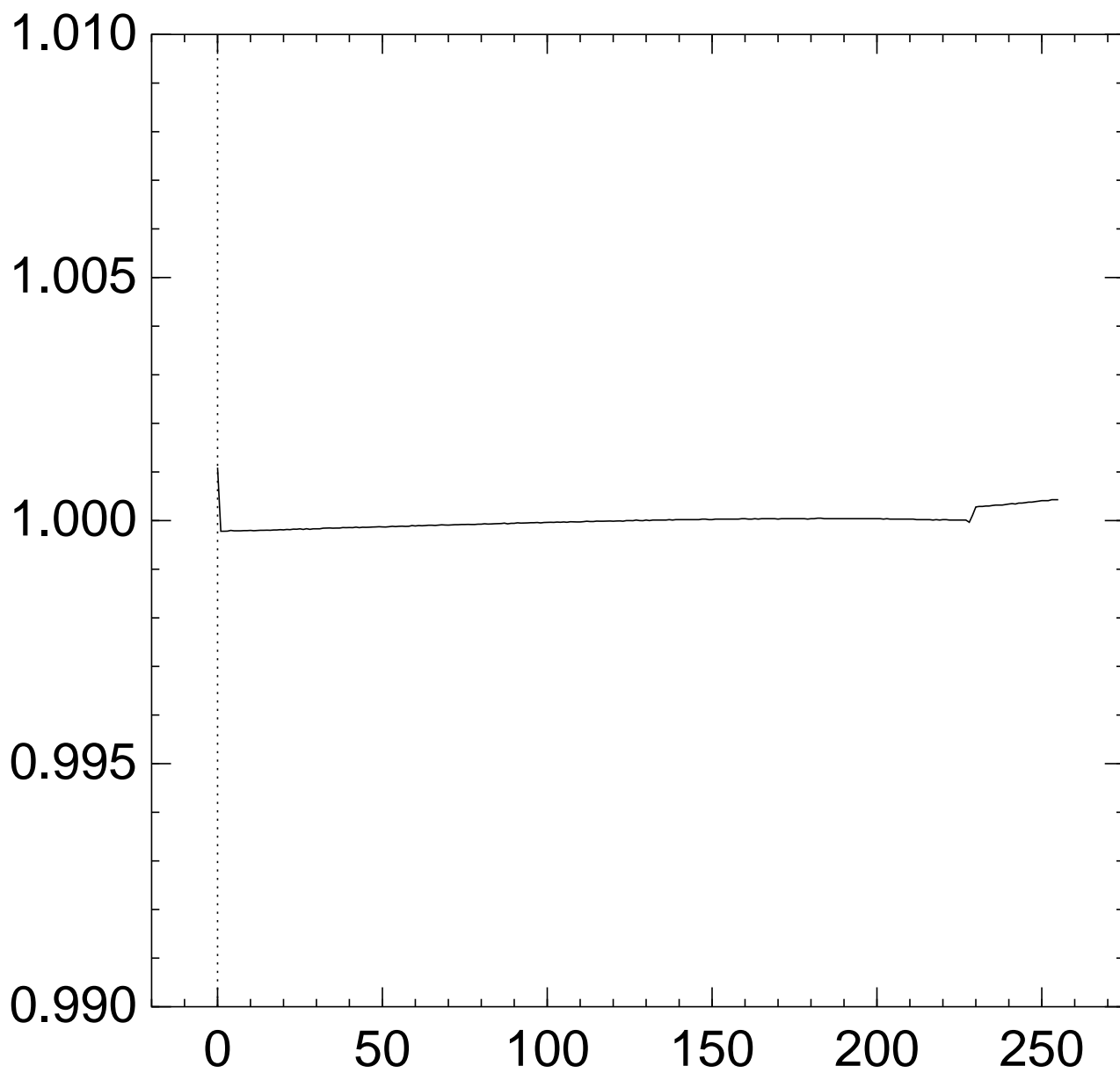
Graph of $256 \Pr[z_{227} = x]$:



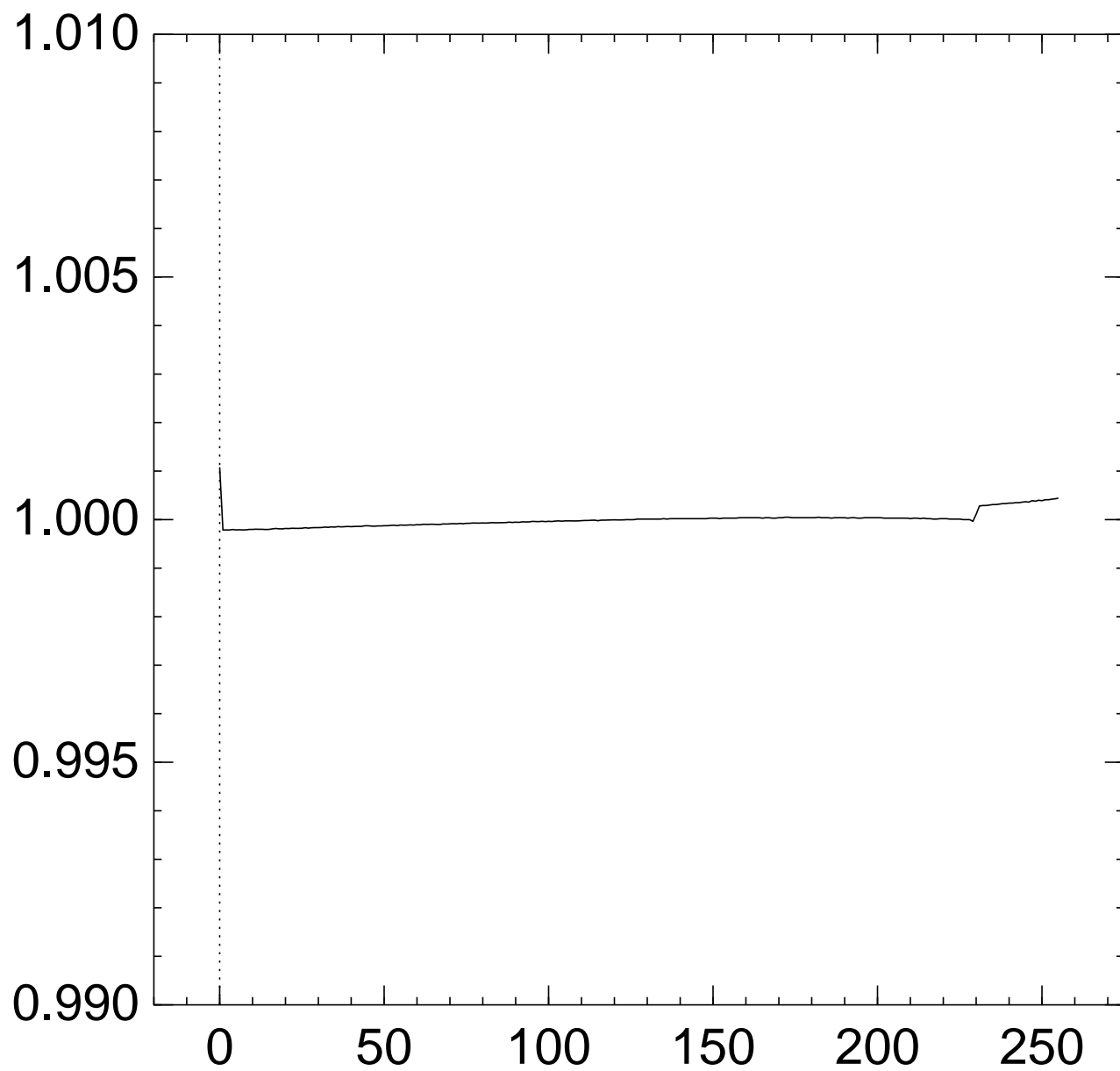
Graph of $256 \Pr[z_{228} = x]$:



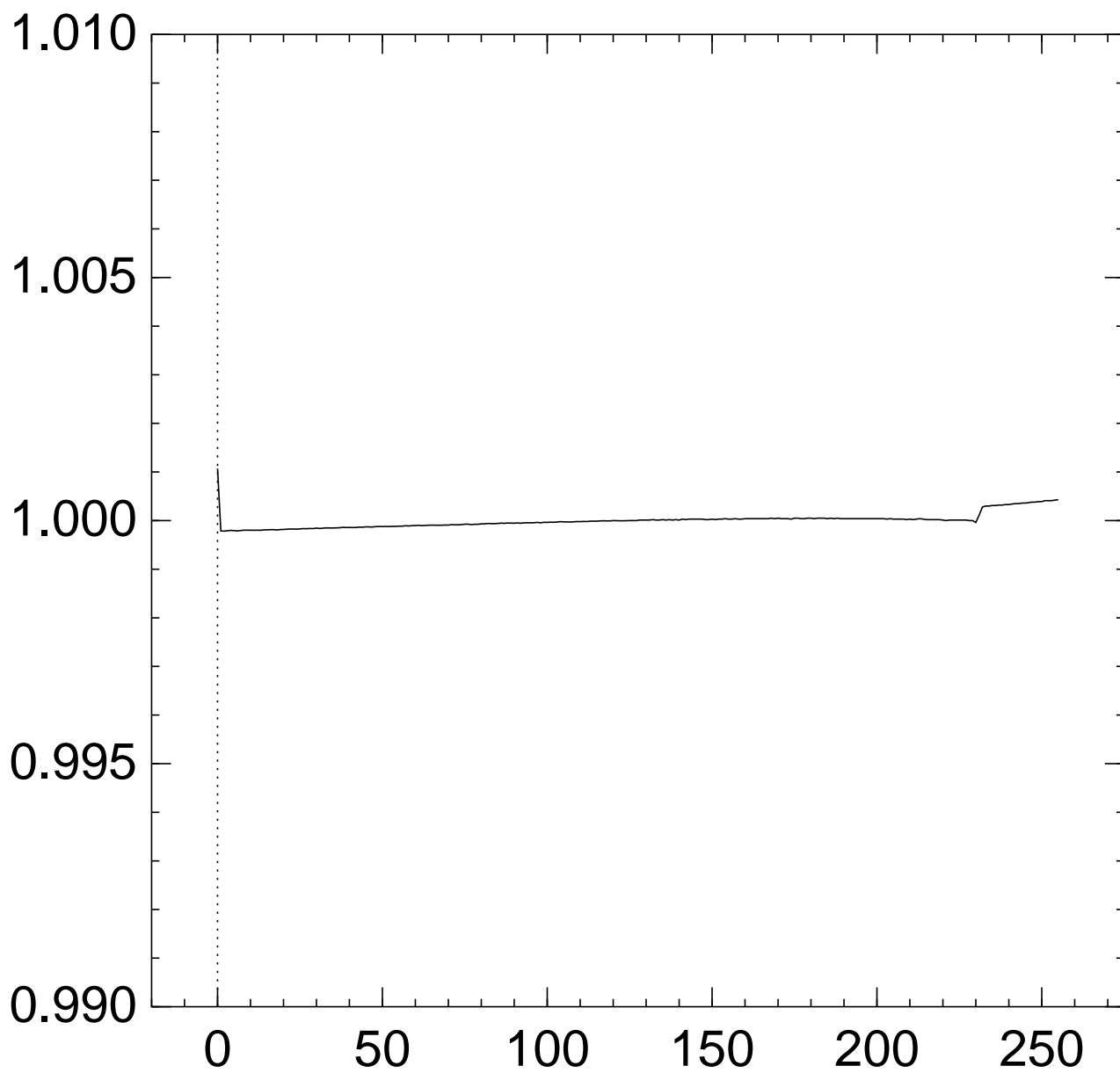
Graph of $256 \Pr[z_{229} = x]$:



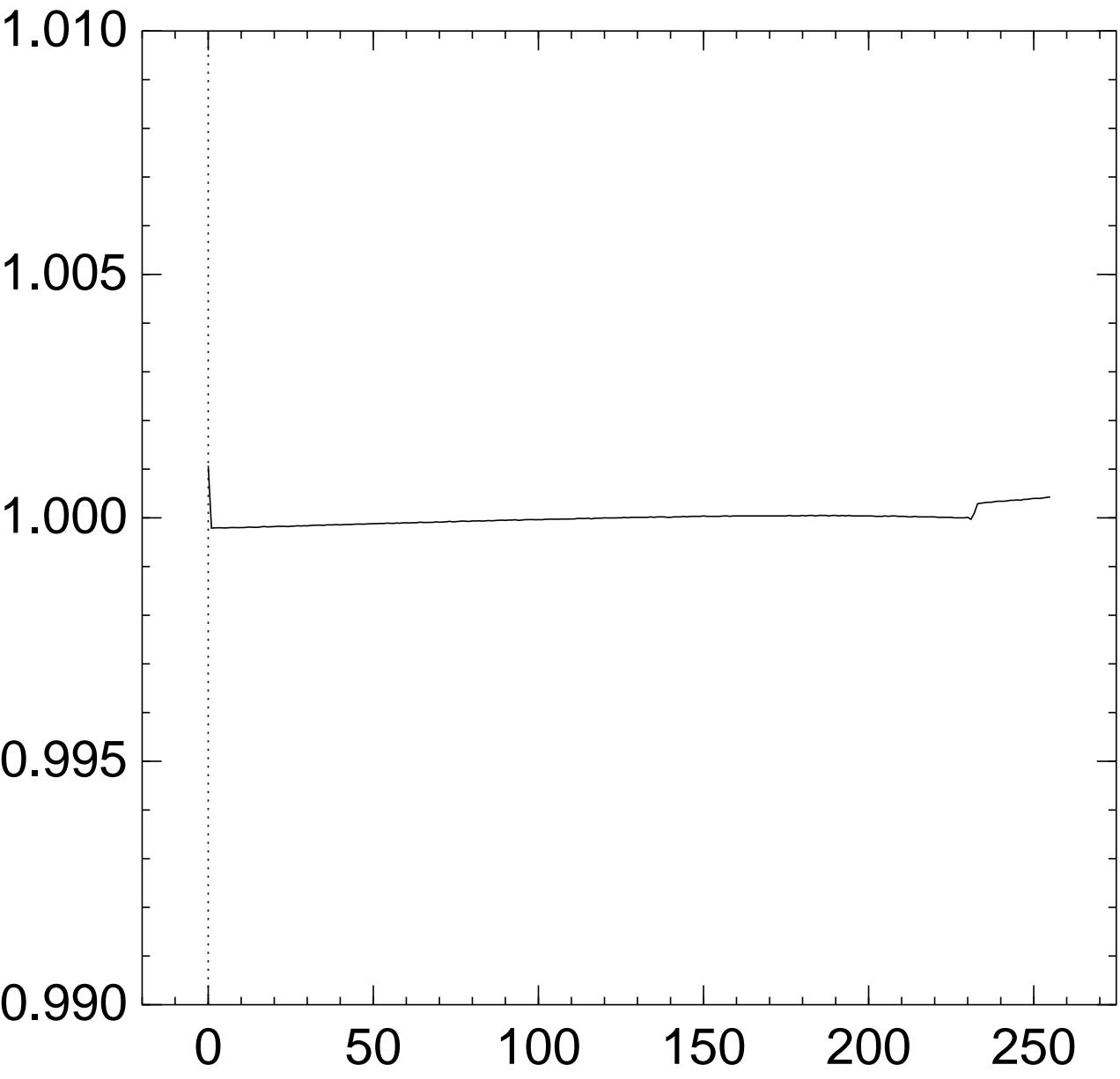
Graph of $256 \Pr[z_{230} = x]$:



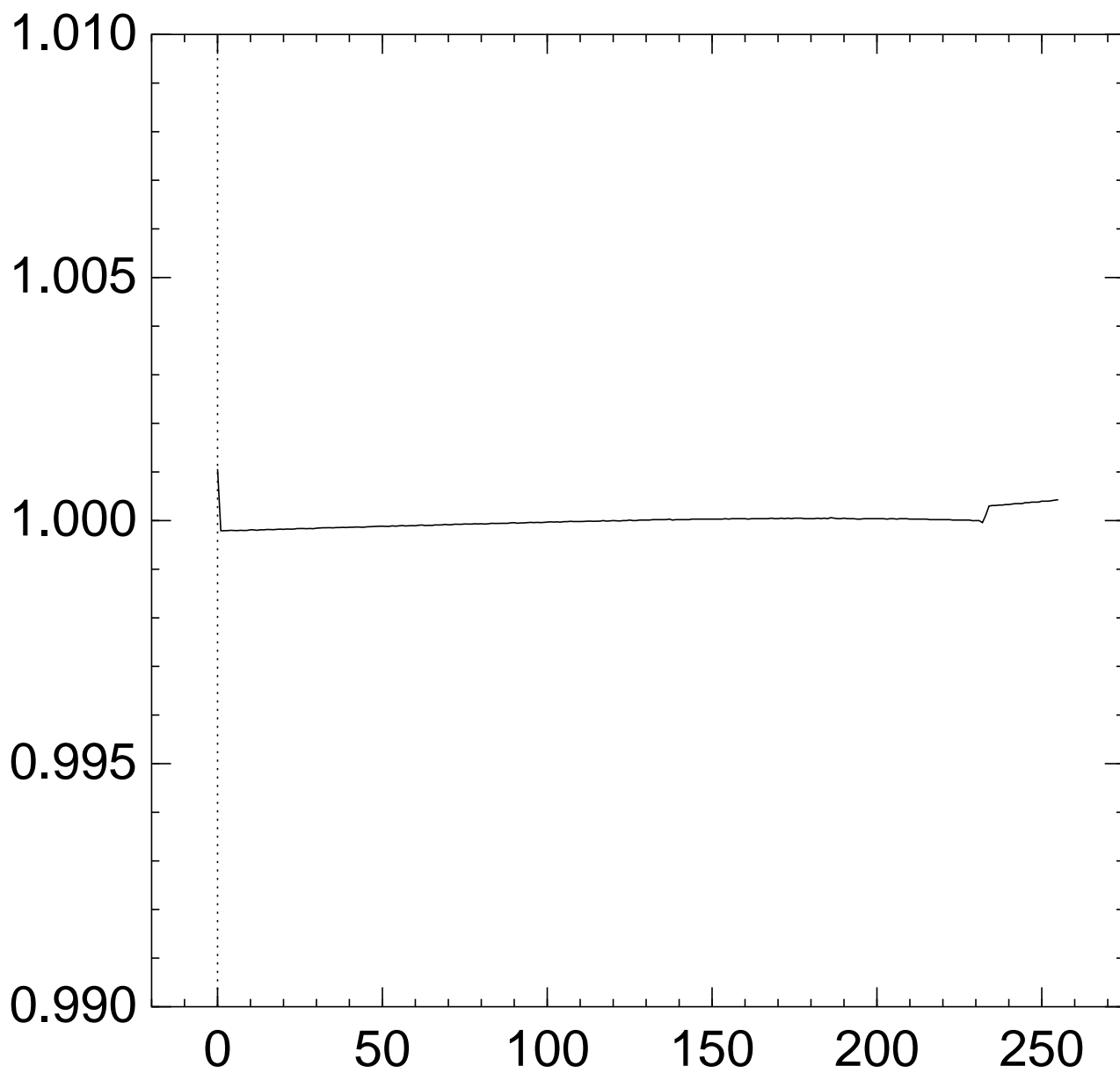
Graph of $256 \Pr[z_{231} = x]$:



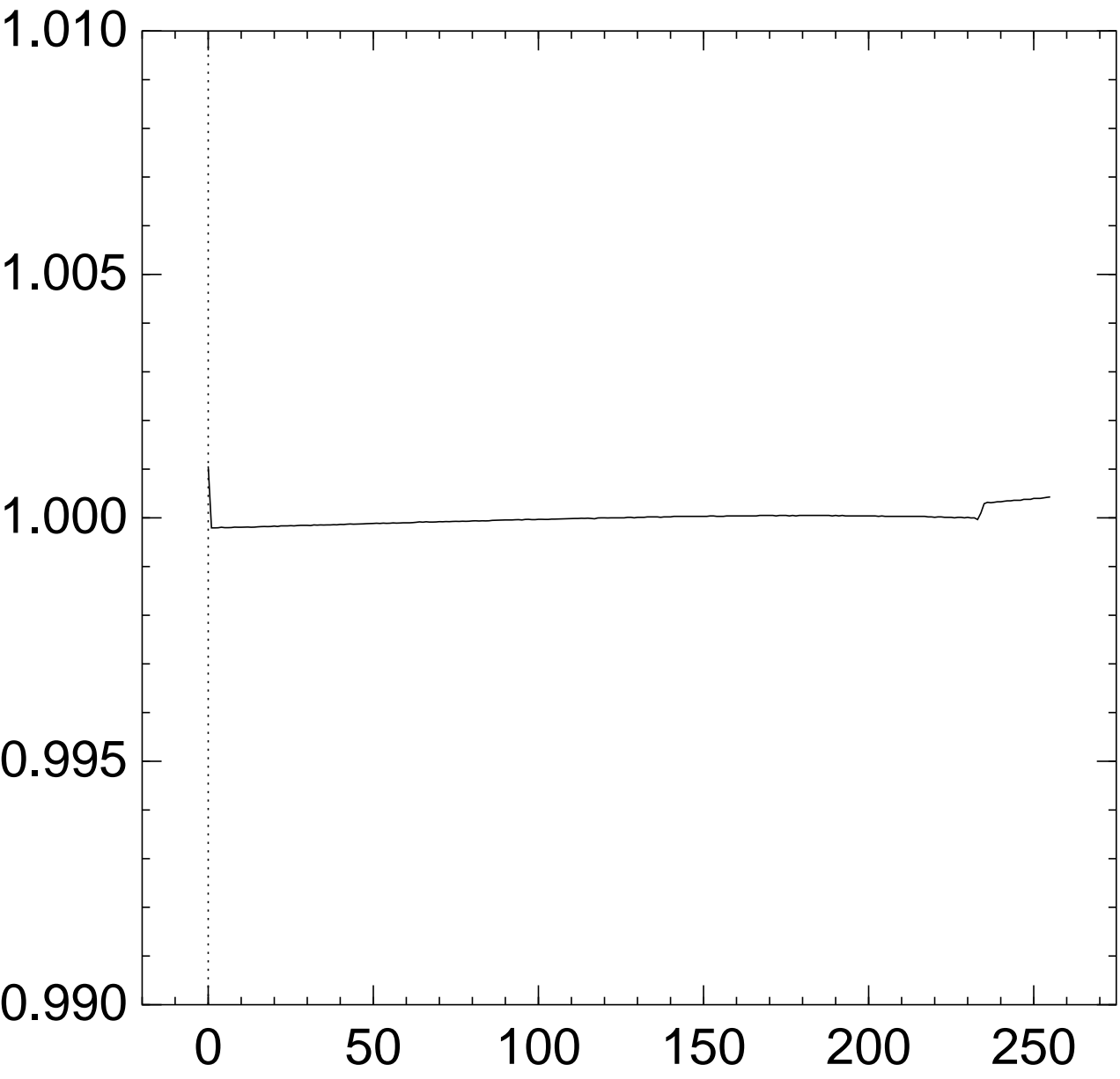
Graph of $256 \Pr[z_{232} = x]$:



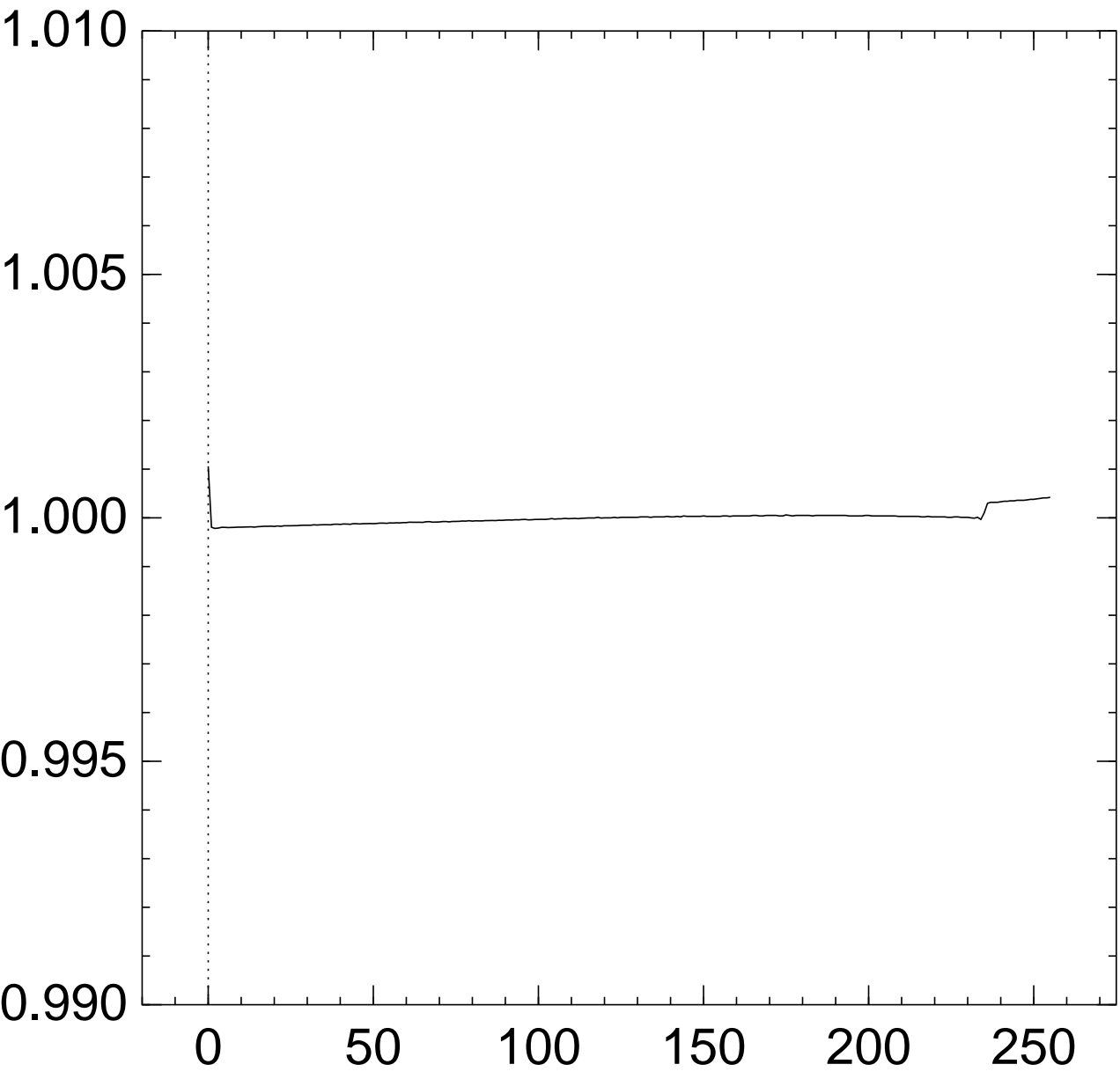
Graph of $256 \Pr[z_{233} = x]$:



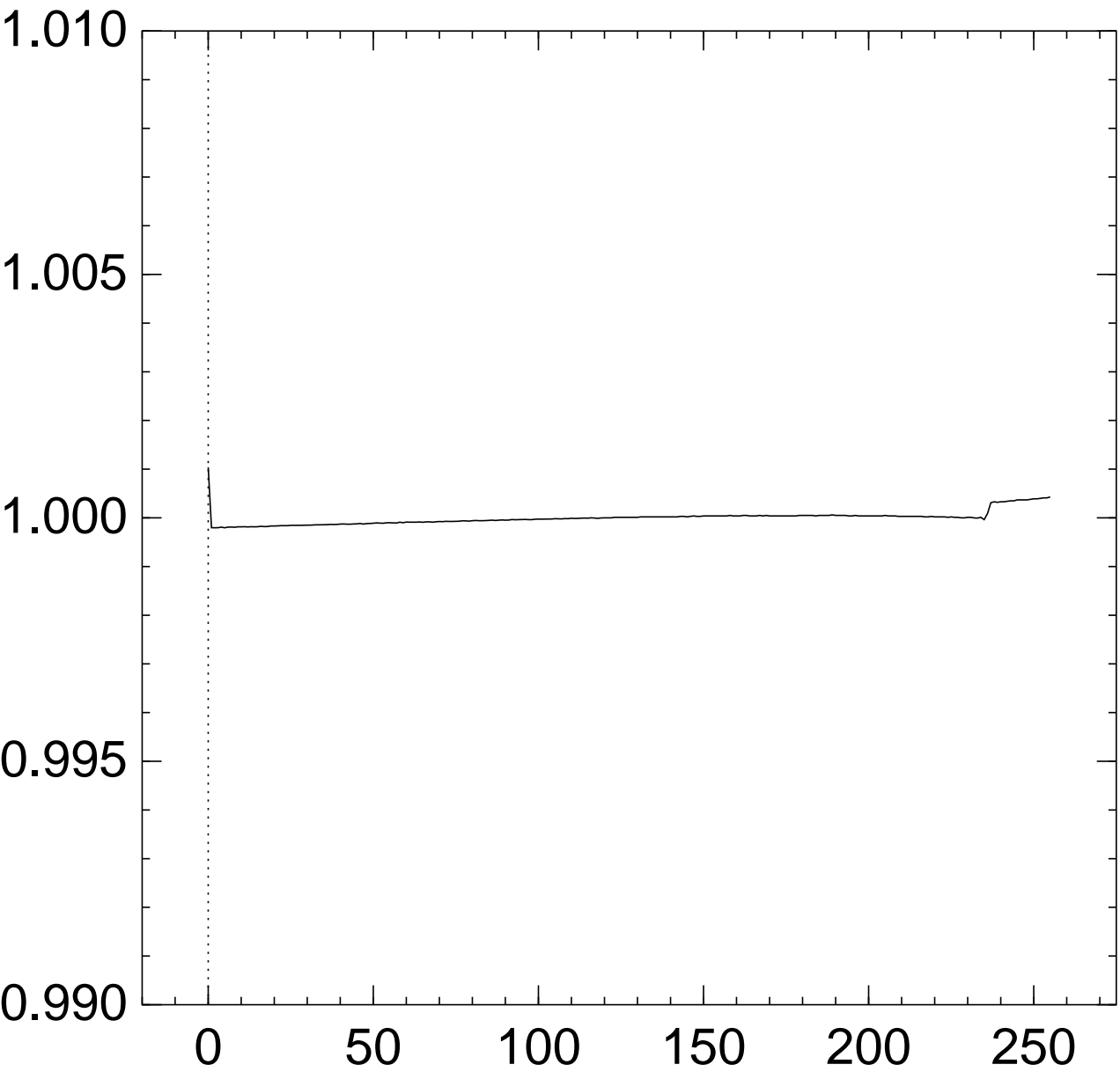
Graph of $256 \Pr[z_{234} = x]$:



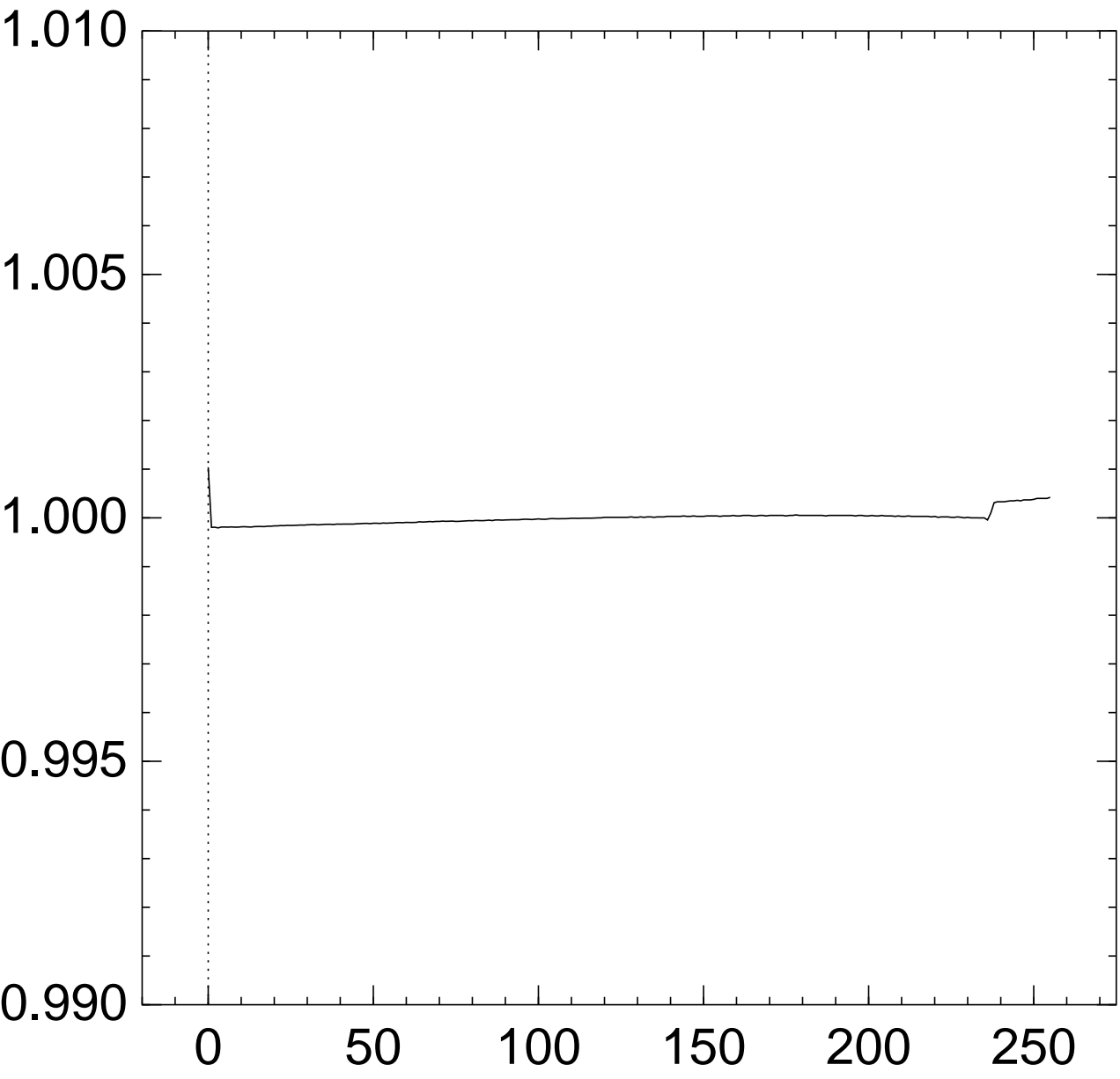
Graph of $256 \Pr[z_{235} = x]$:



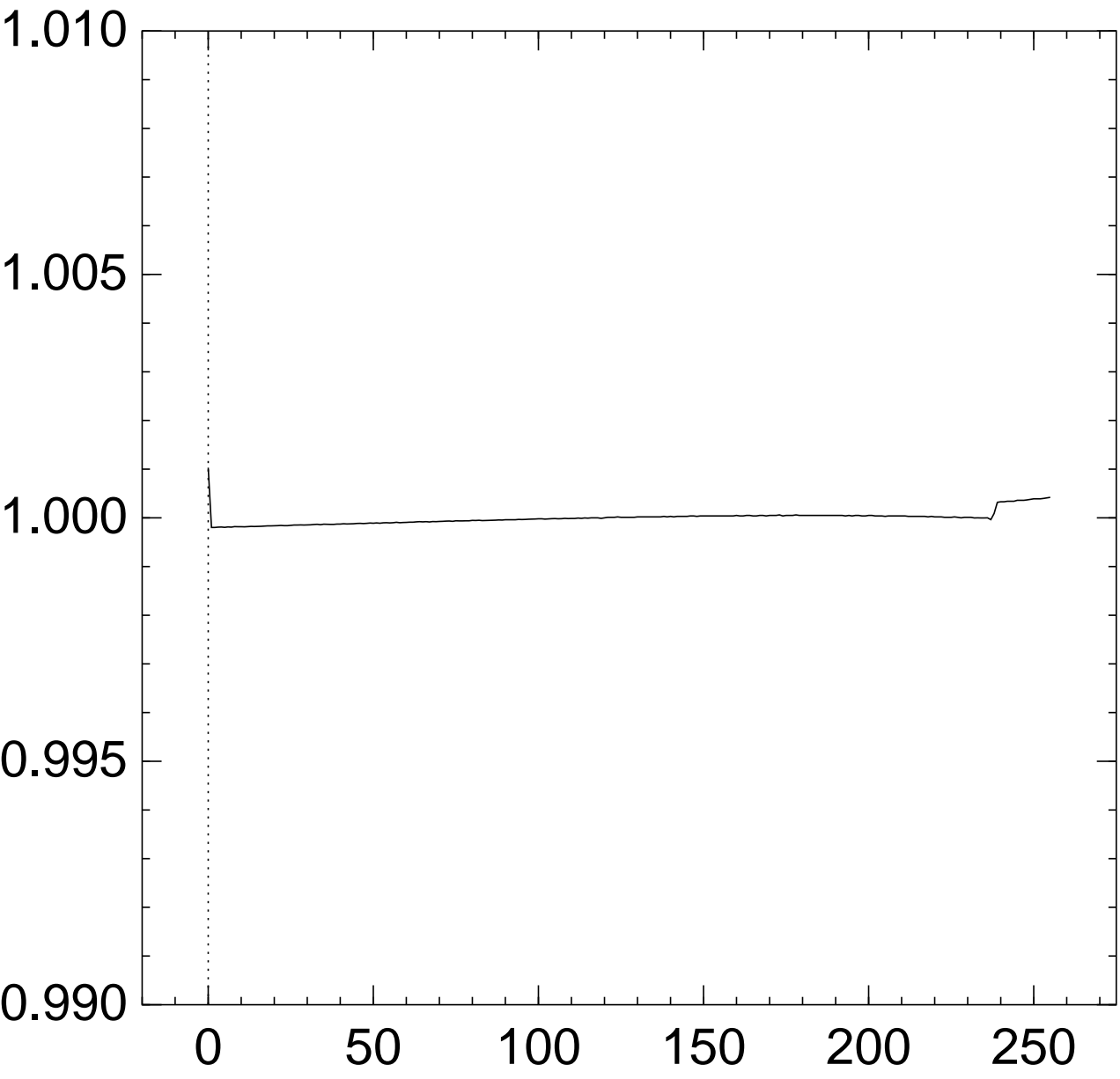
Graph of $256 \Pr[z_{236} = x]$:



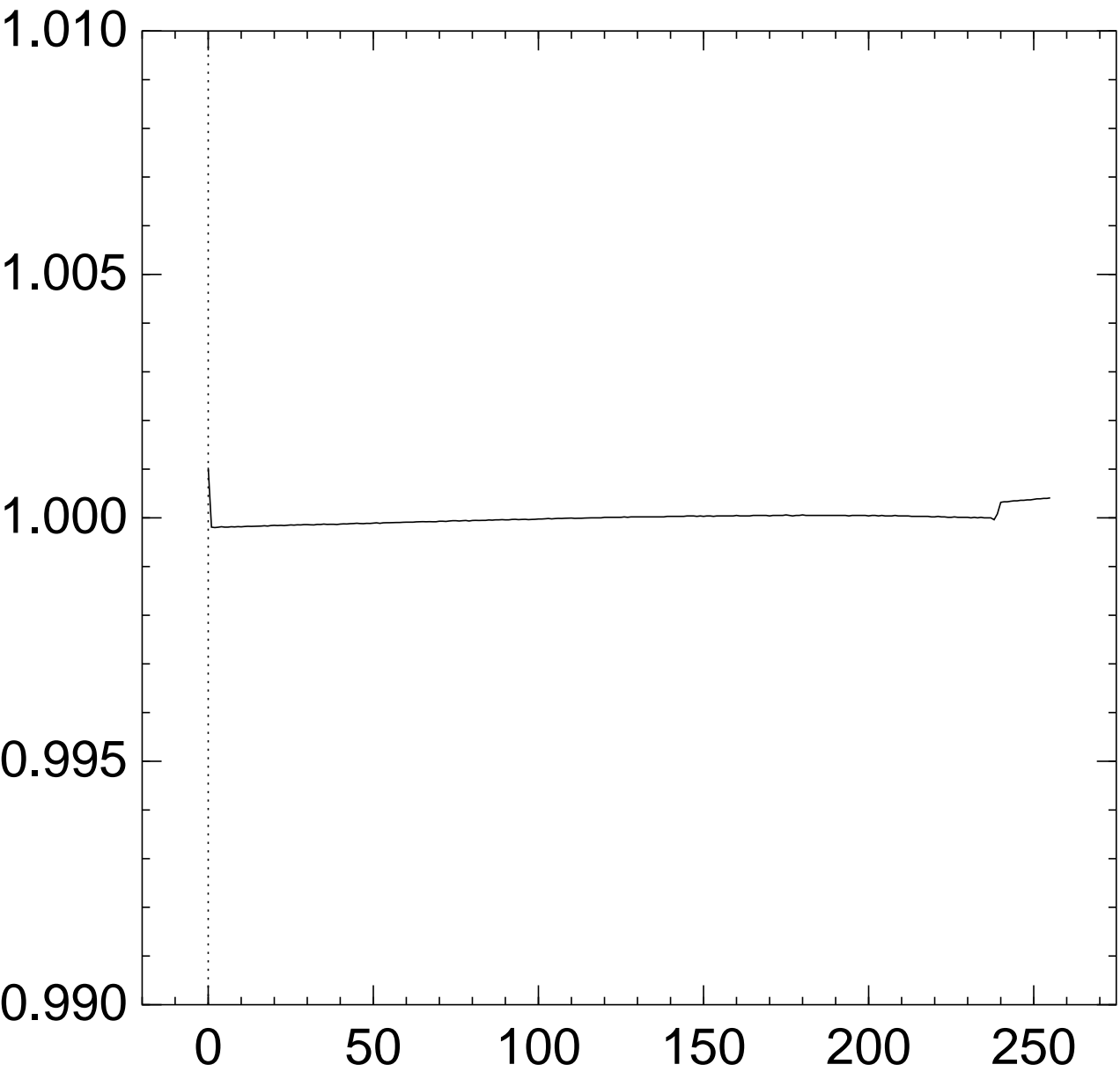
Graph of $256 \Pr[z_{237} = x]$:



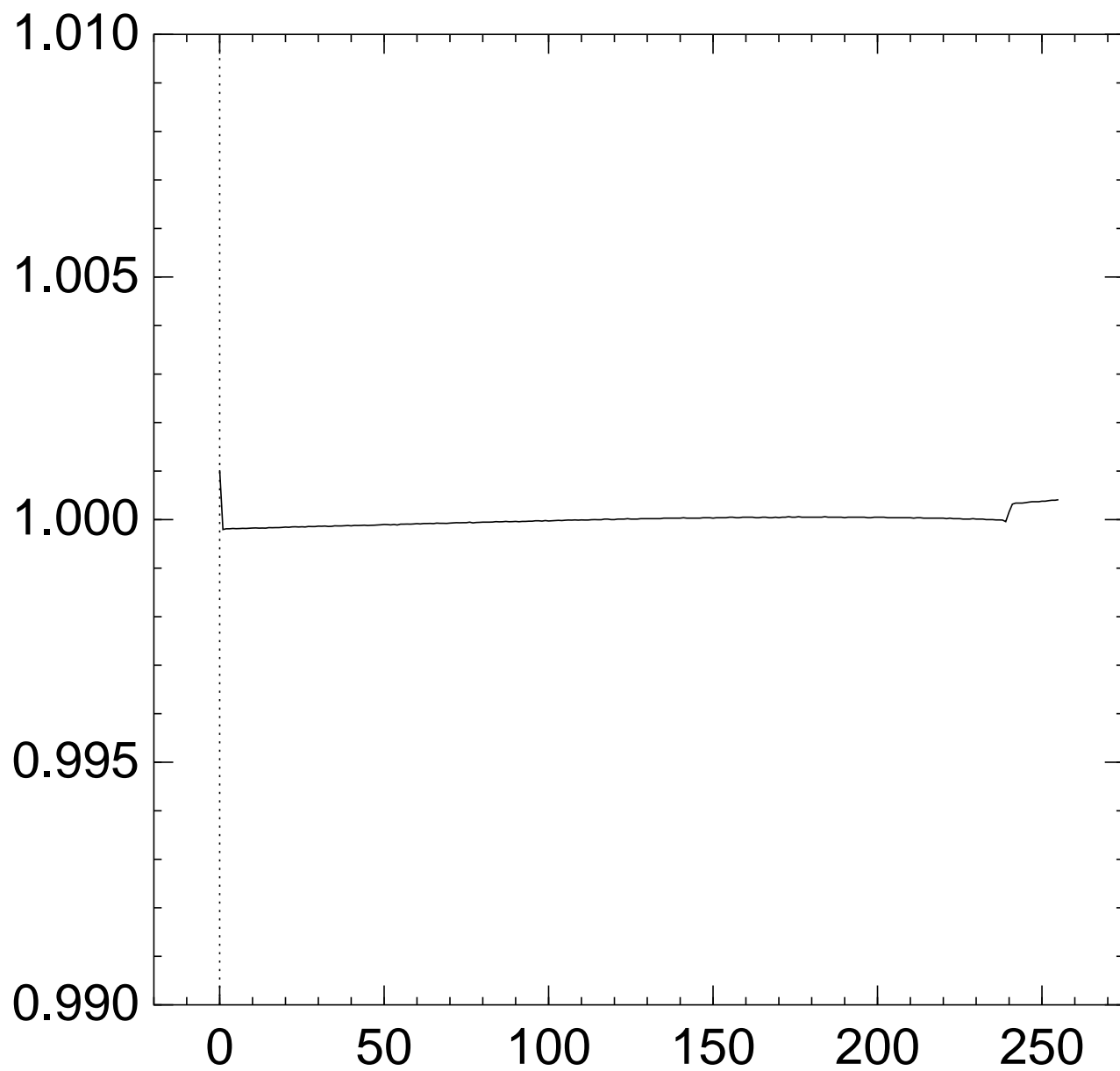
Graph of $256 \Pr[z_{238} = x]$:



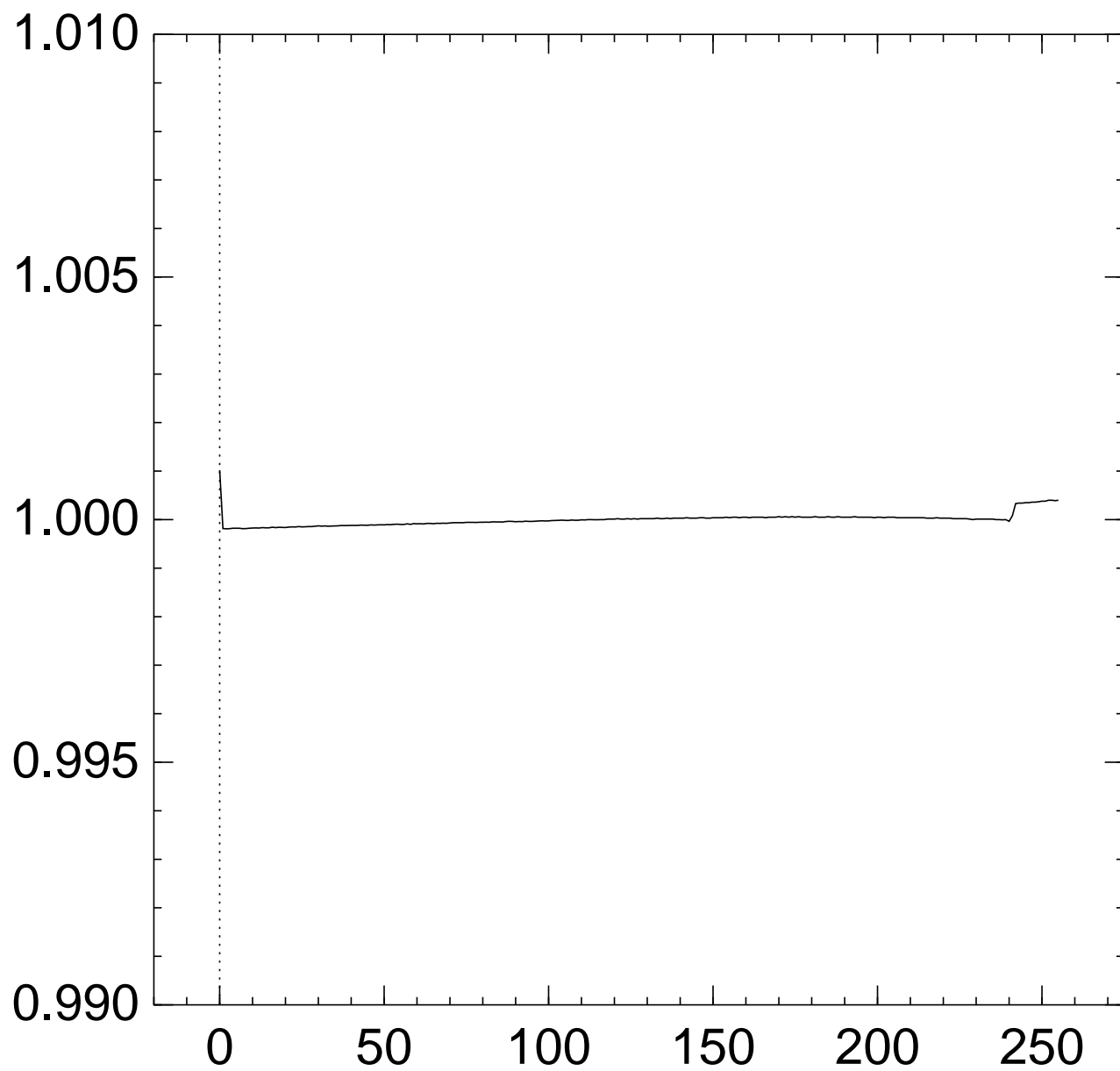
Graph of $256 \Pr[z_{239} = x]$:



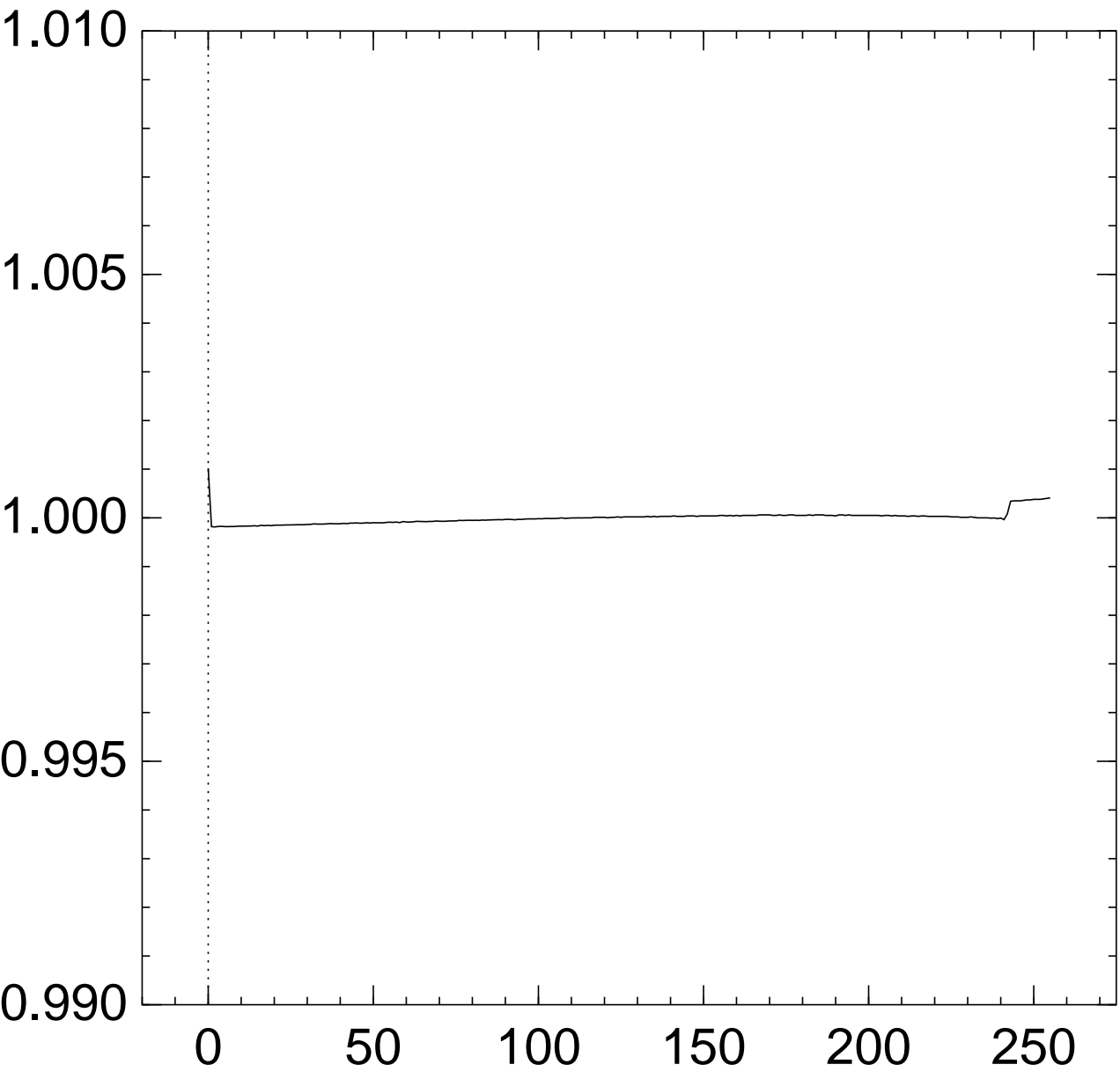
Graph of $256 \Pr[z_{240} = x]$:



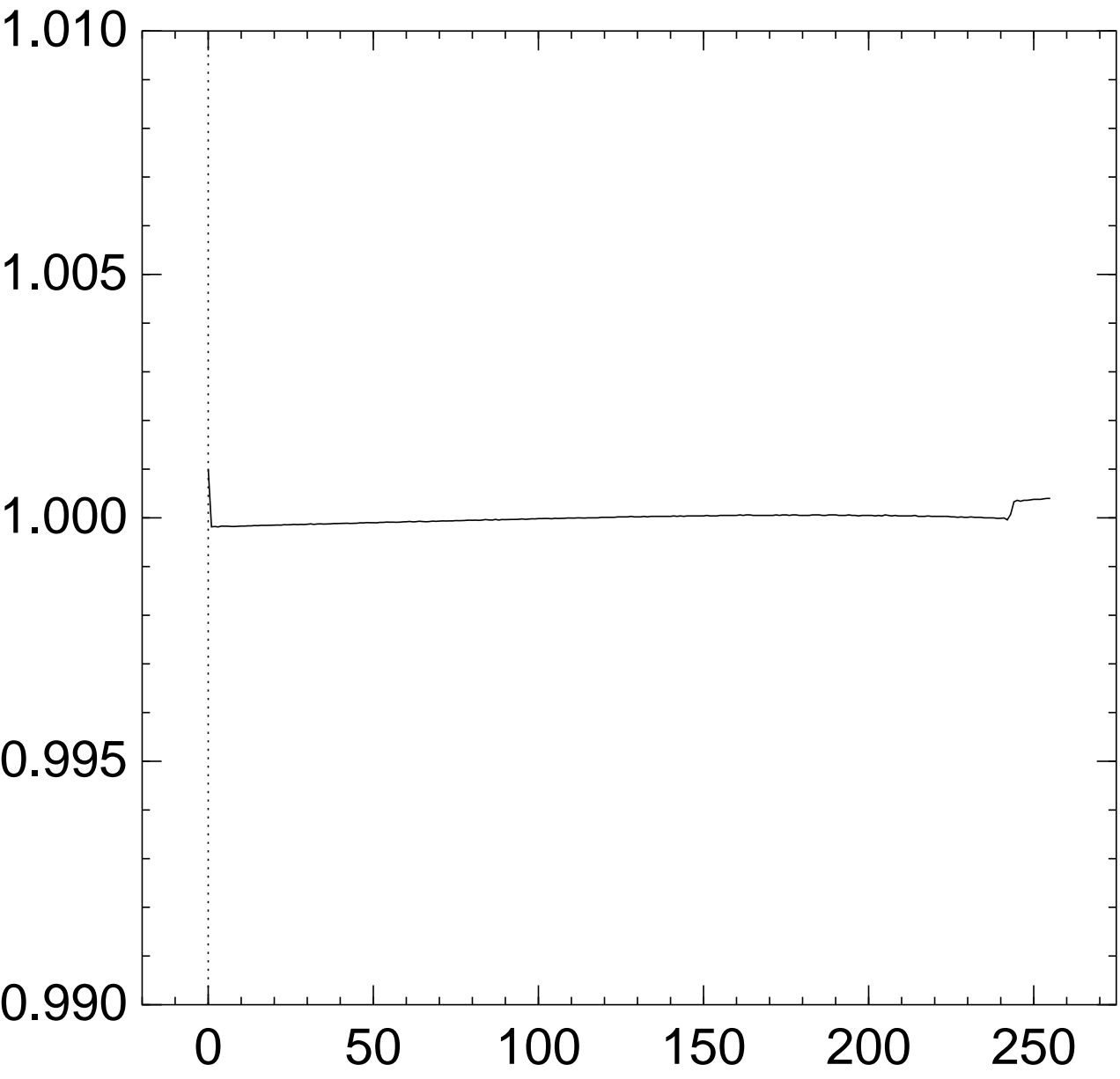
Graph of $256 \Pr[z_{241} = x]$:



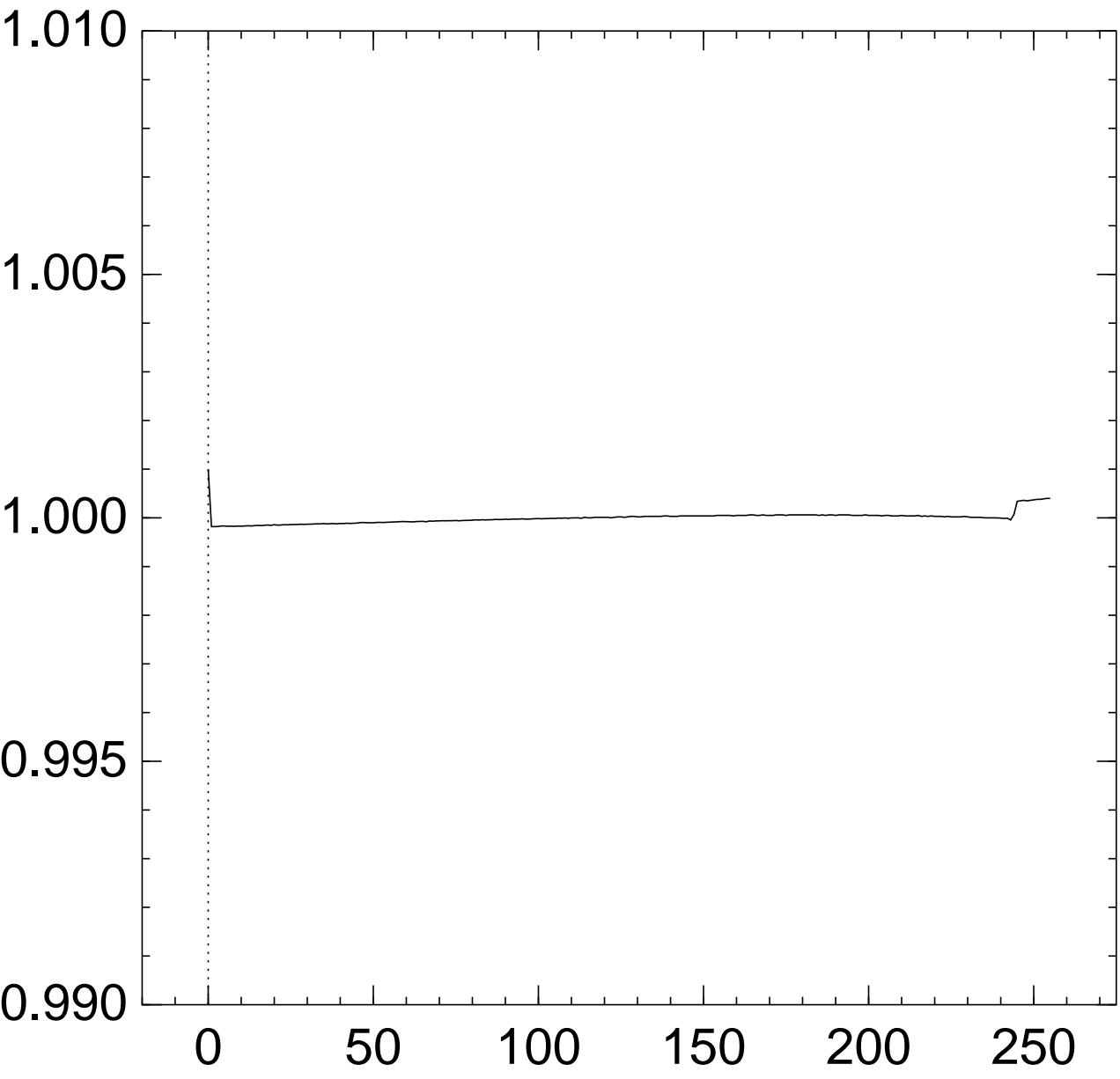
Graph of $256 \Pr[z_{242} = x]$:



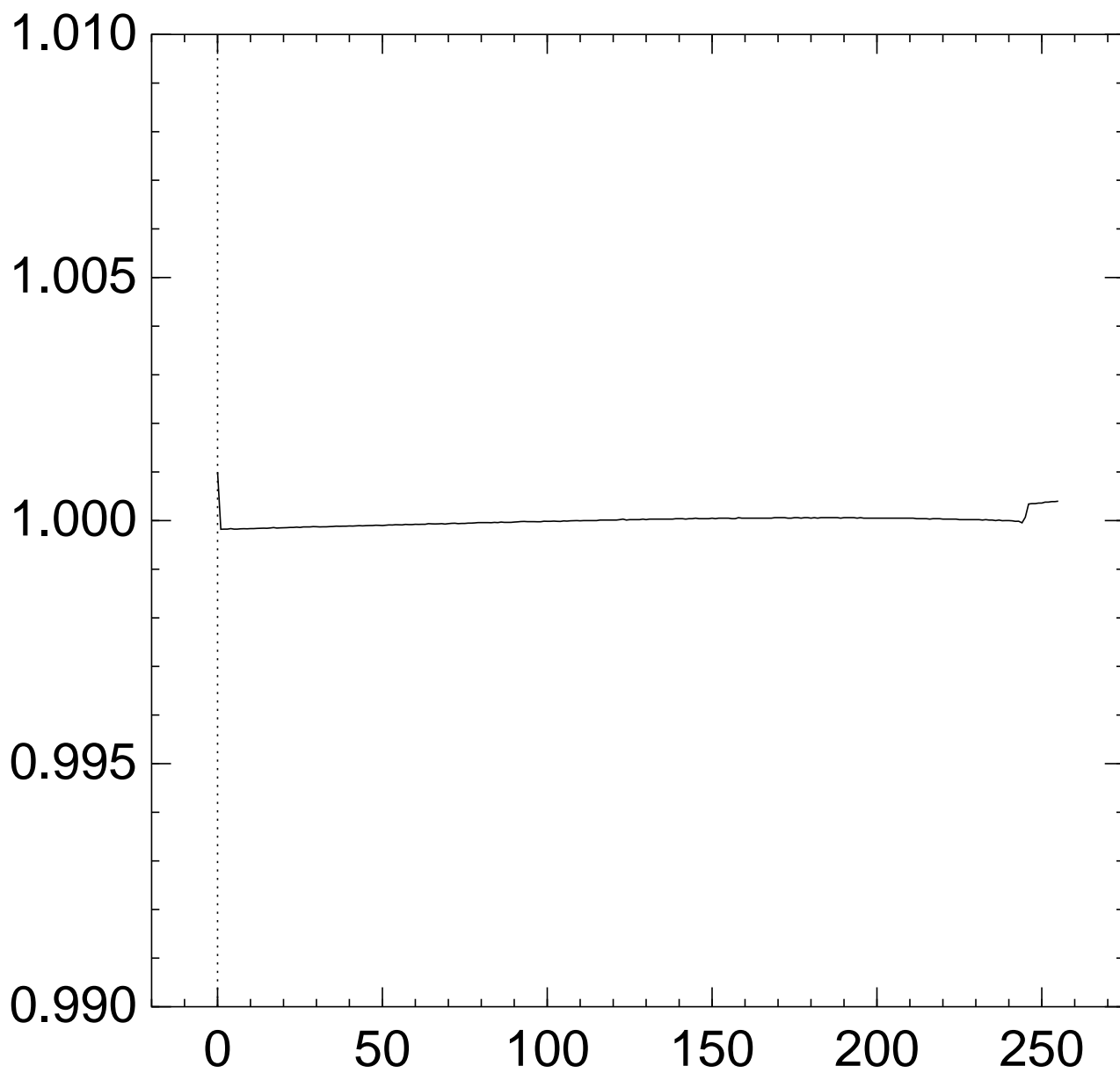
Graph of $256 \Pr[z_{243} = x]$:



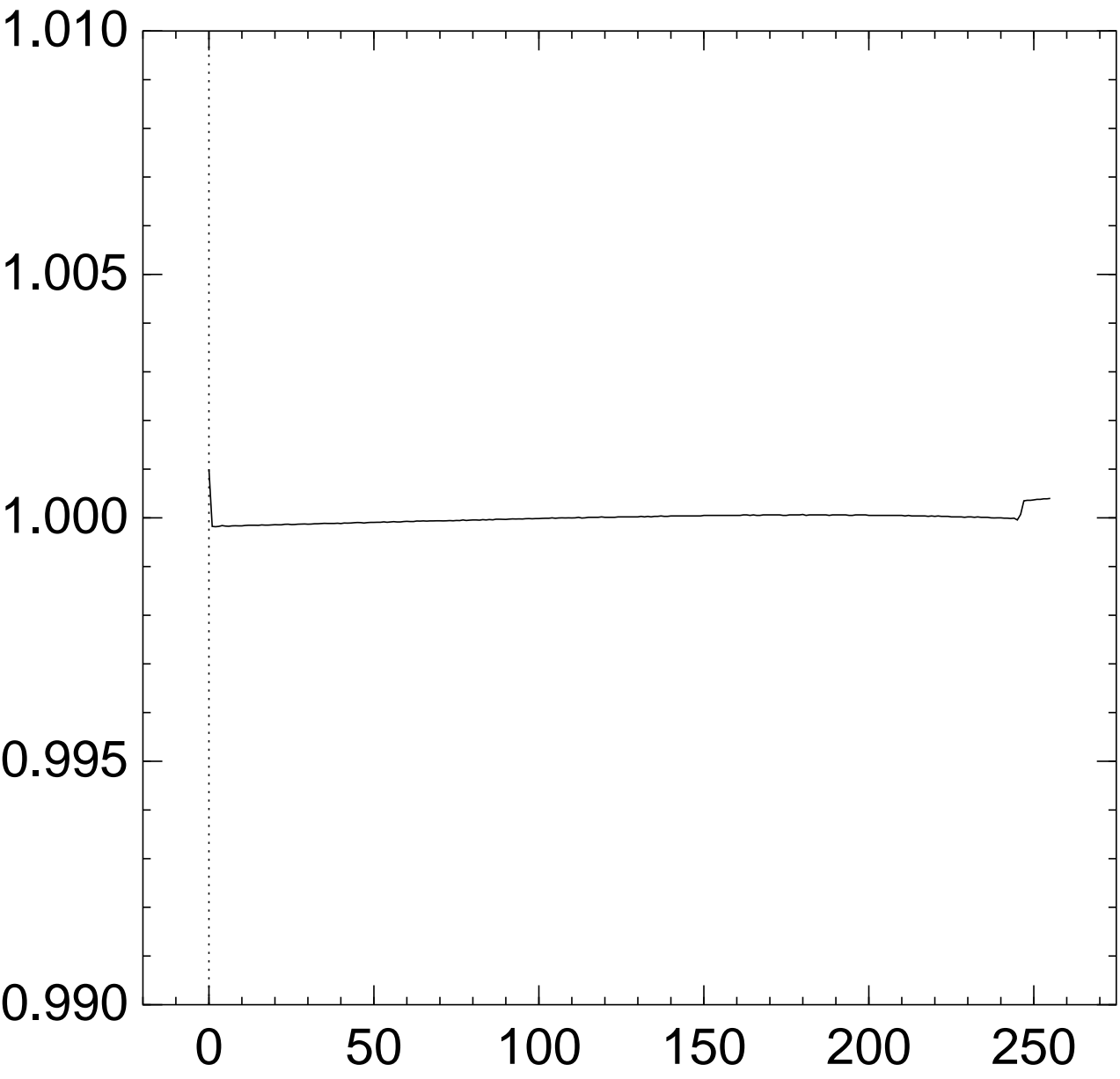
Graph of $256 \Pr[z_{244} = x]$:



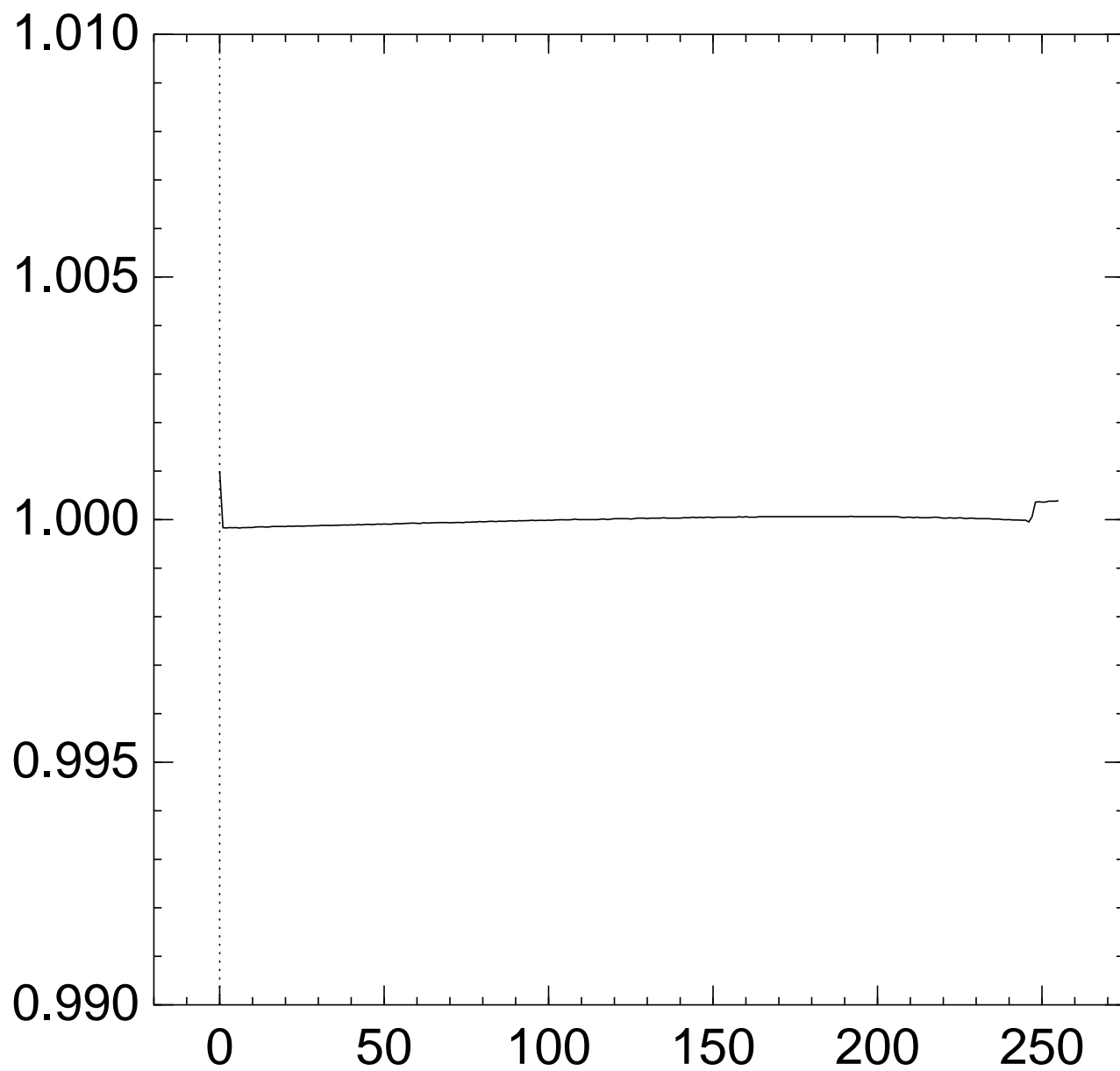
Graph of $256 \Pr[z_{245} = x]$:



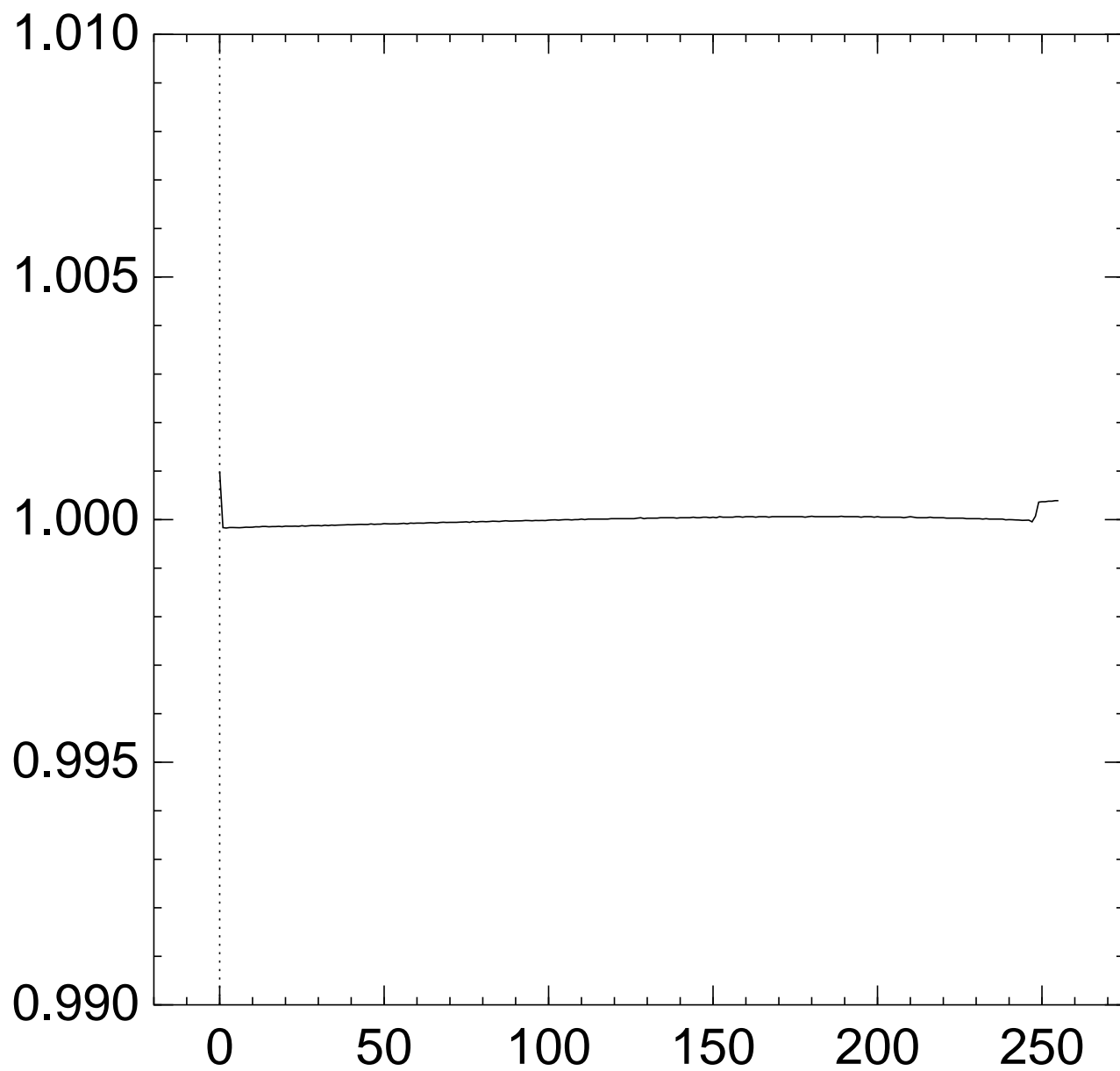
Graph of $256 \Pr[z_{246} = x]$:



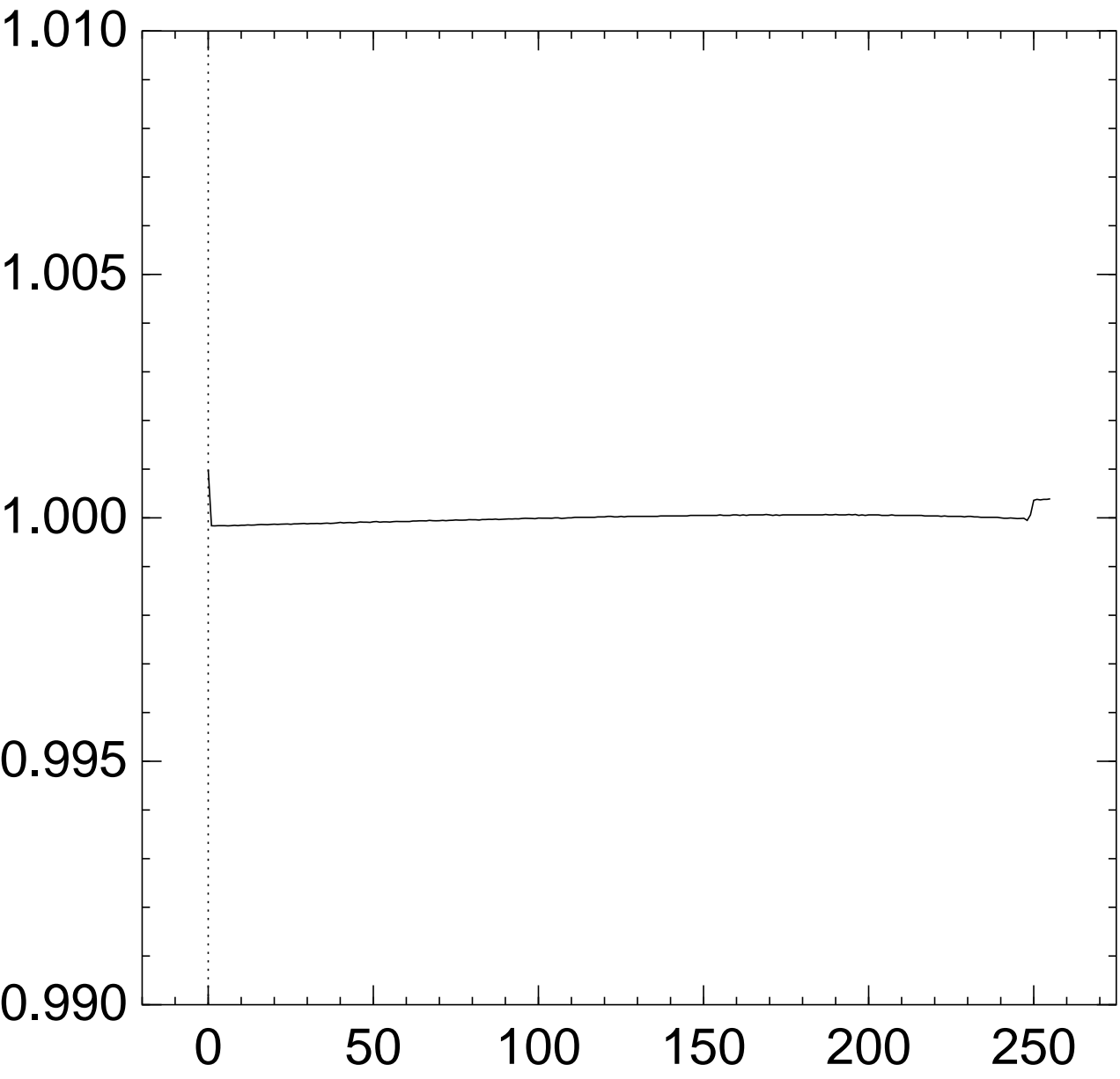
Graph of $256 \Pr[z_{247} = x]$:



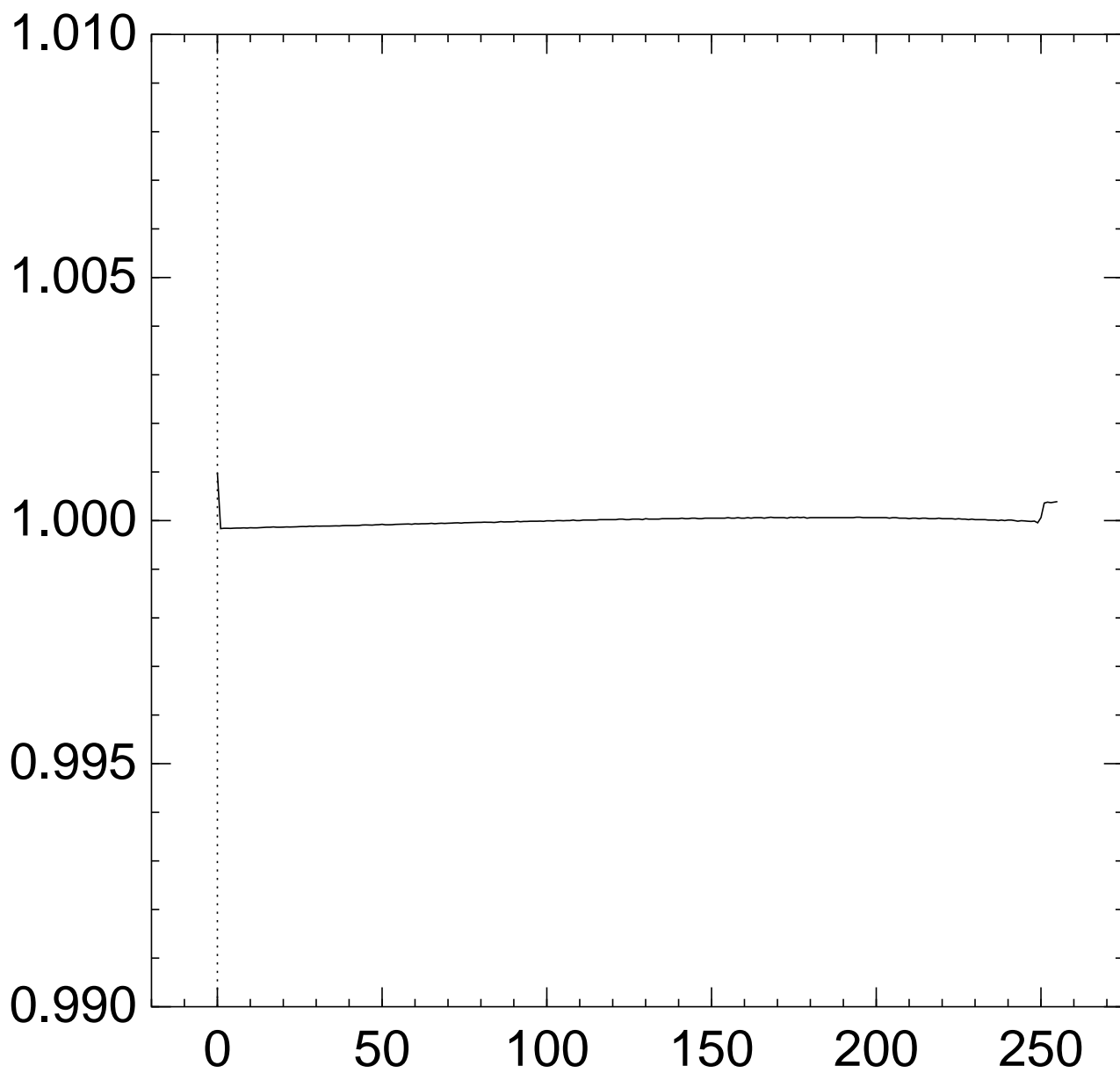
Graph of $256 \Pr[z_{248} = x]$:



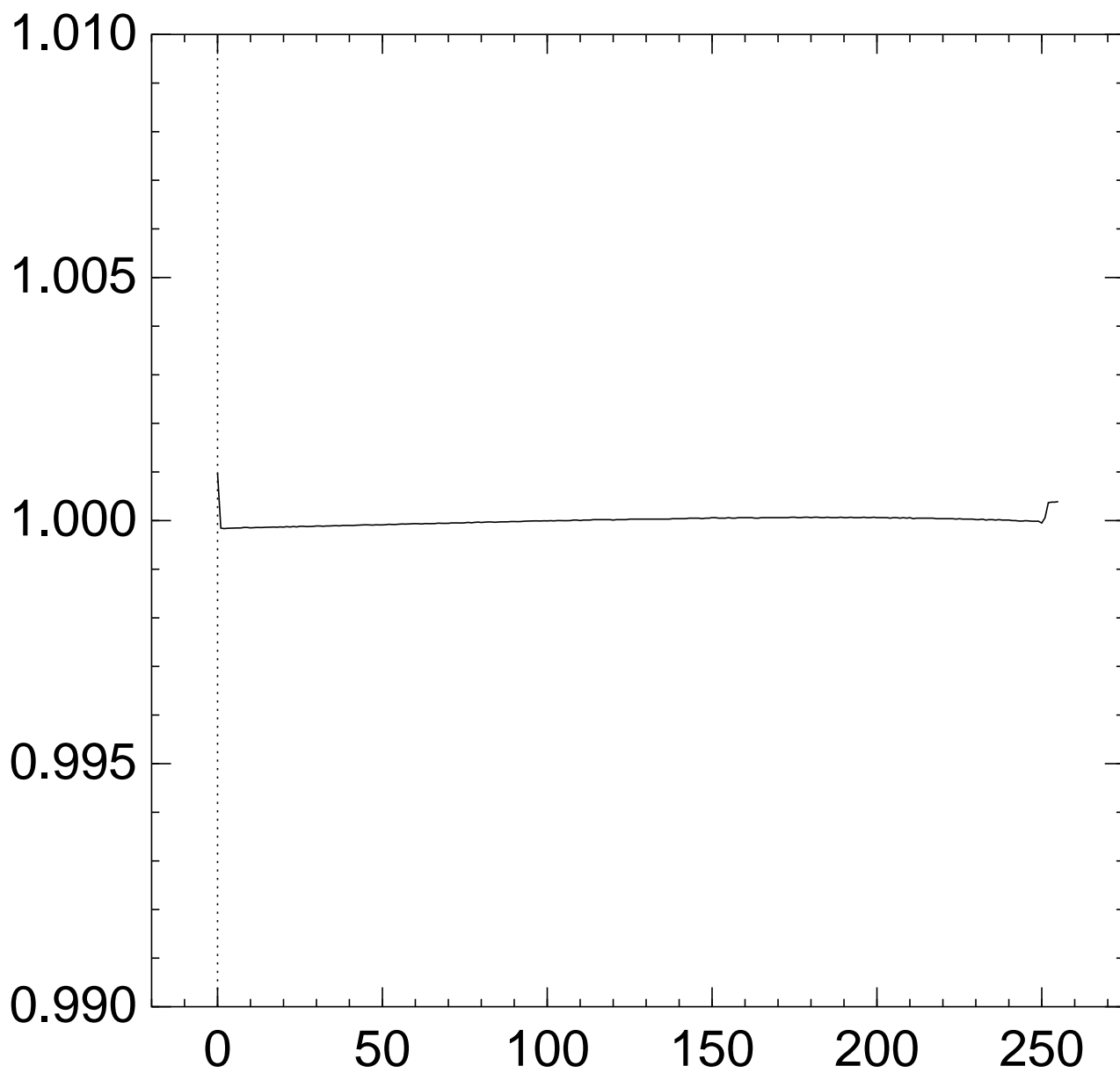
Graph of $256 \Pr[z_{249} = x]$:



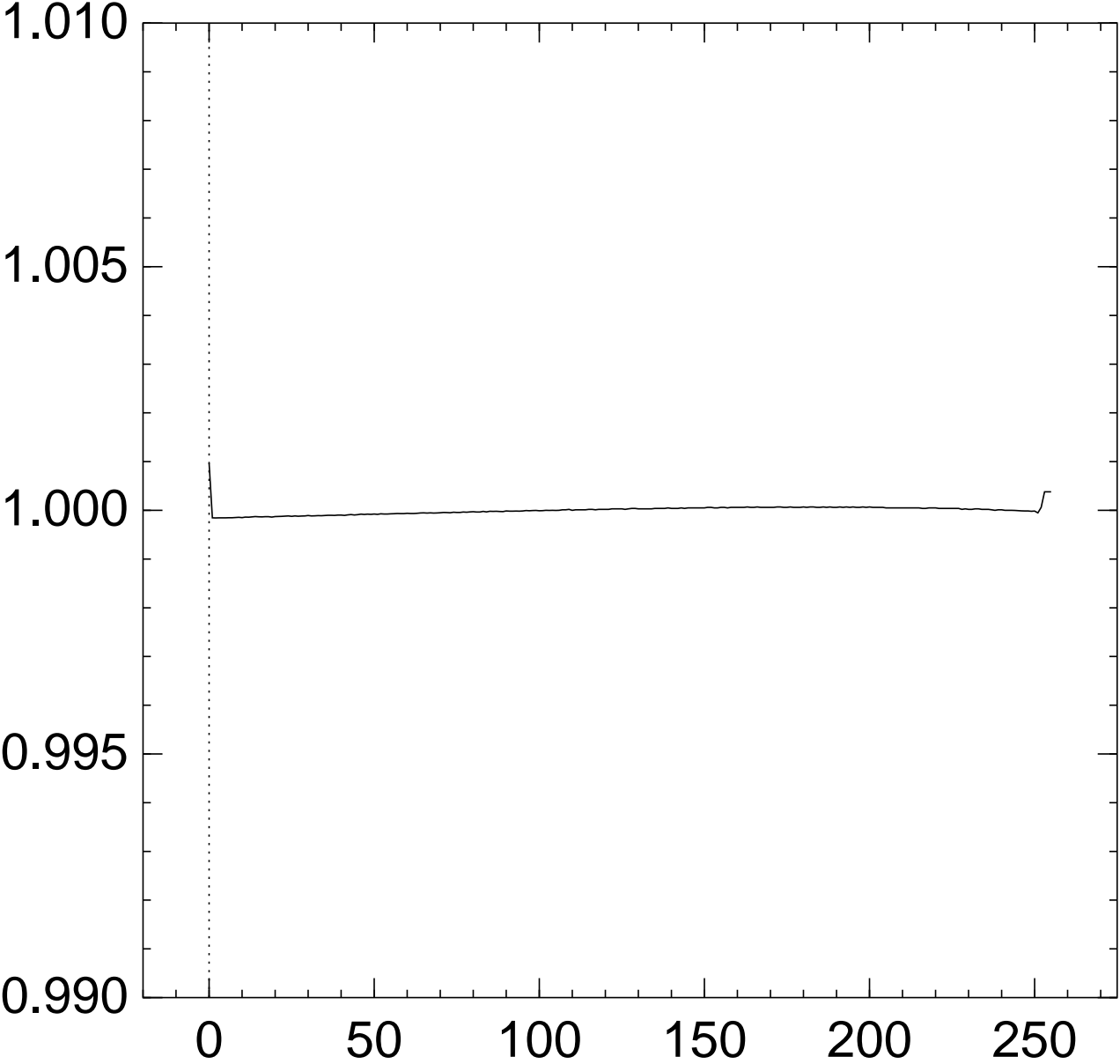
Graph of $256 \Pr[z_{250} = x]$:



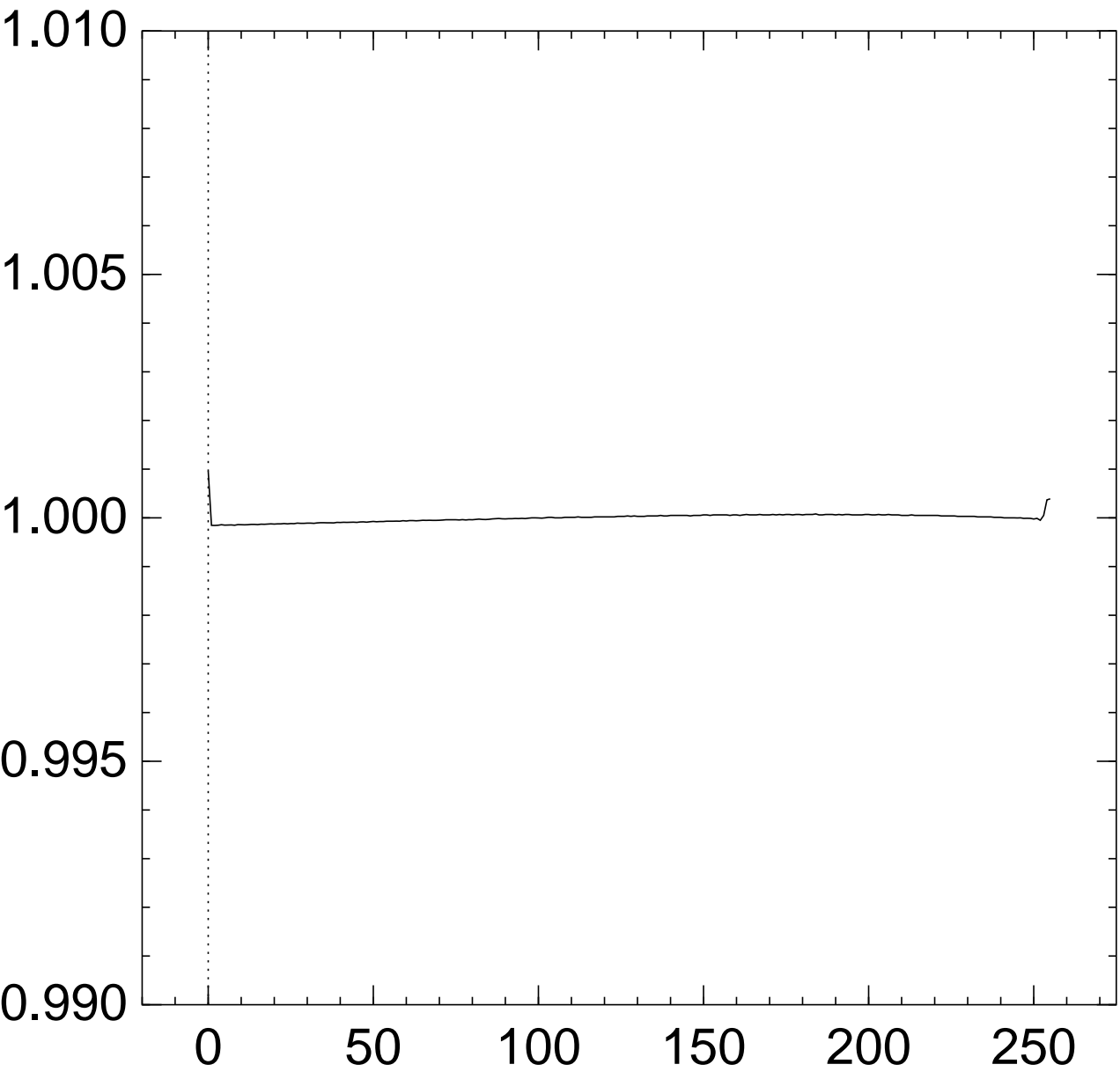
Graph of $256 \Pr[z_{251} = x]$:



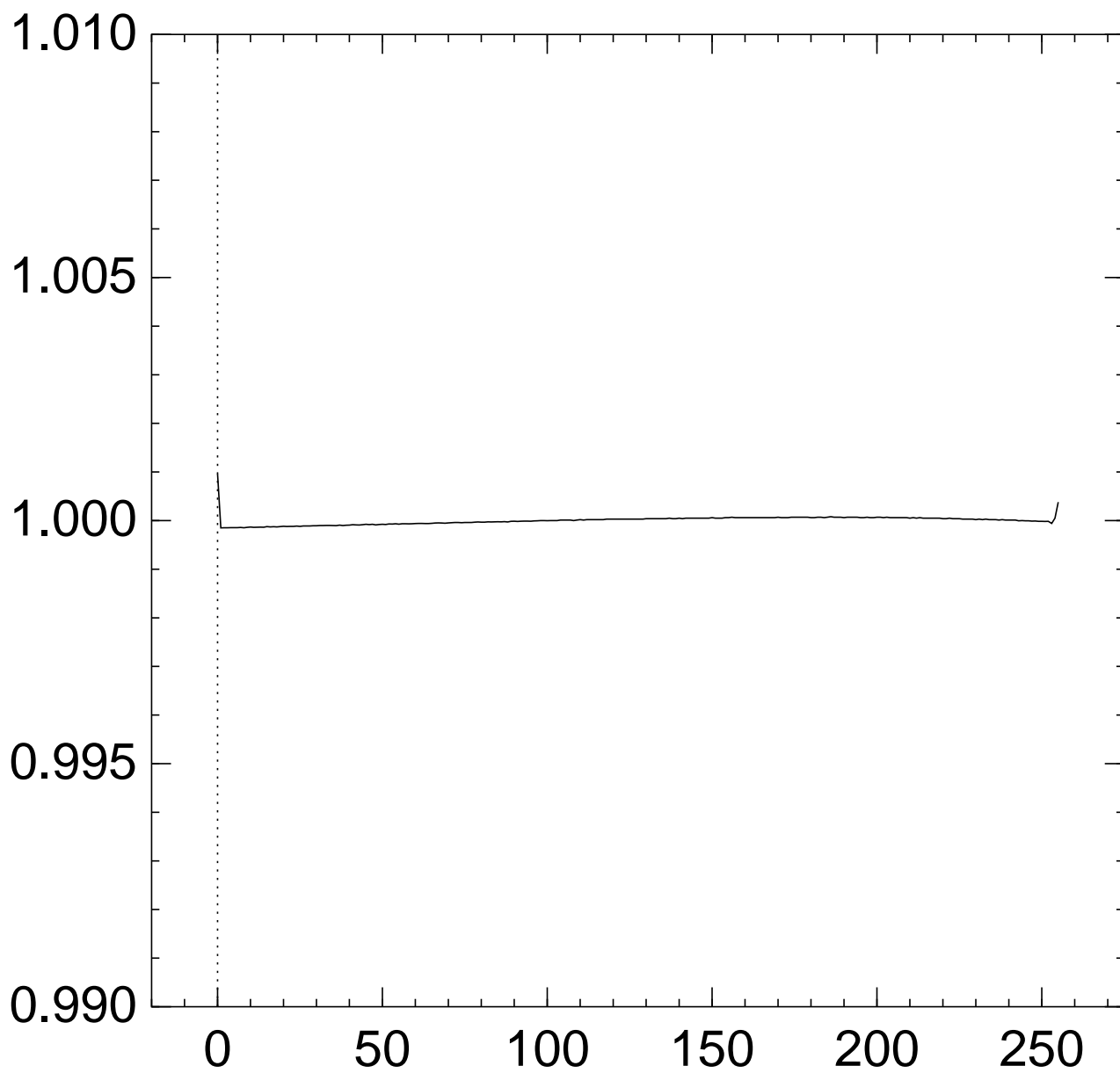
Graph of $256 \Pr[z_{252} = x]$:



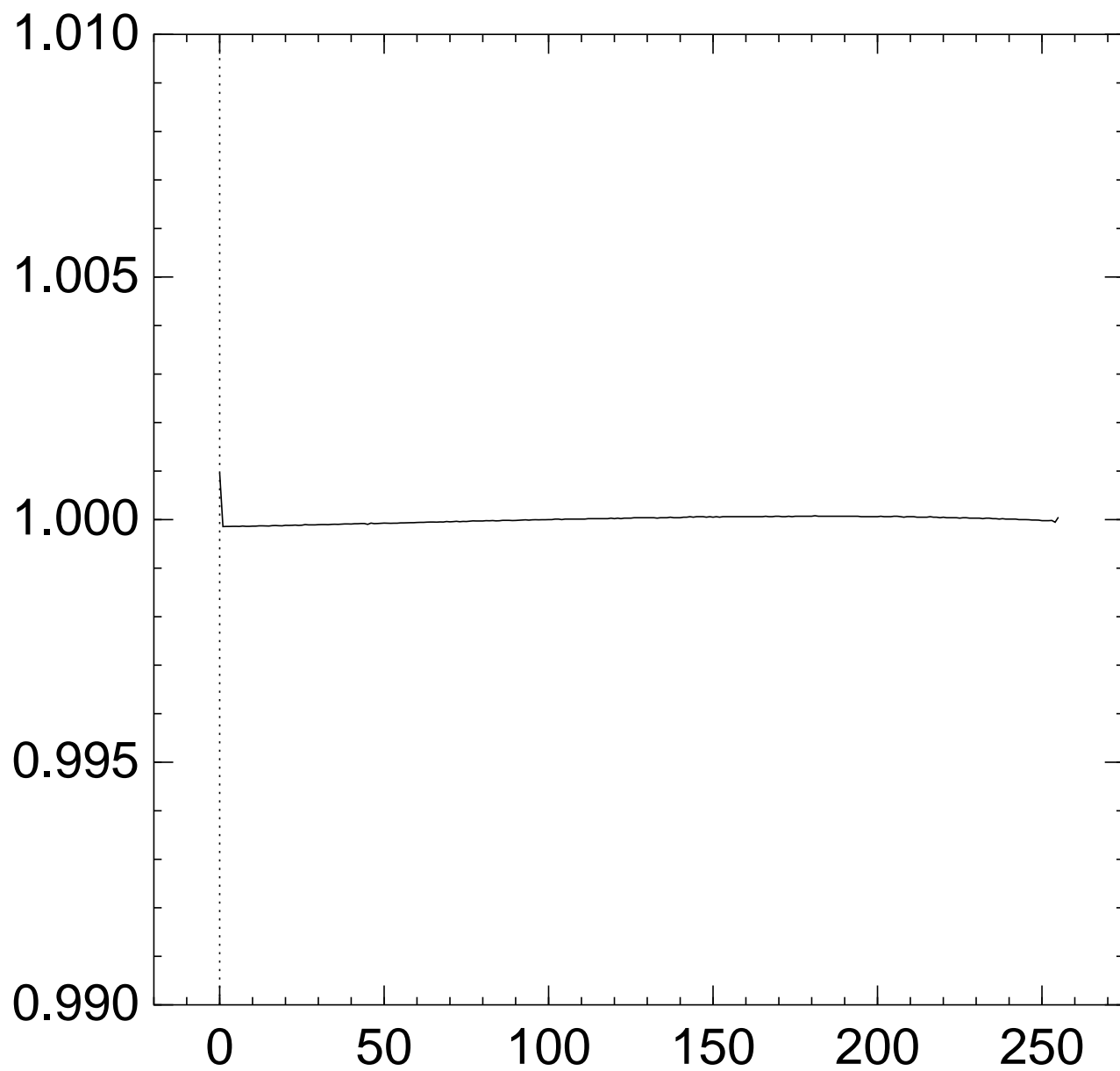
Graph of $256 \Pr[z_{253} = x]$:



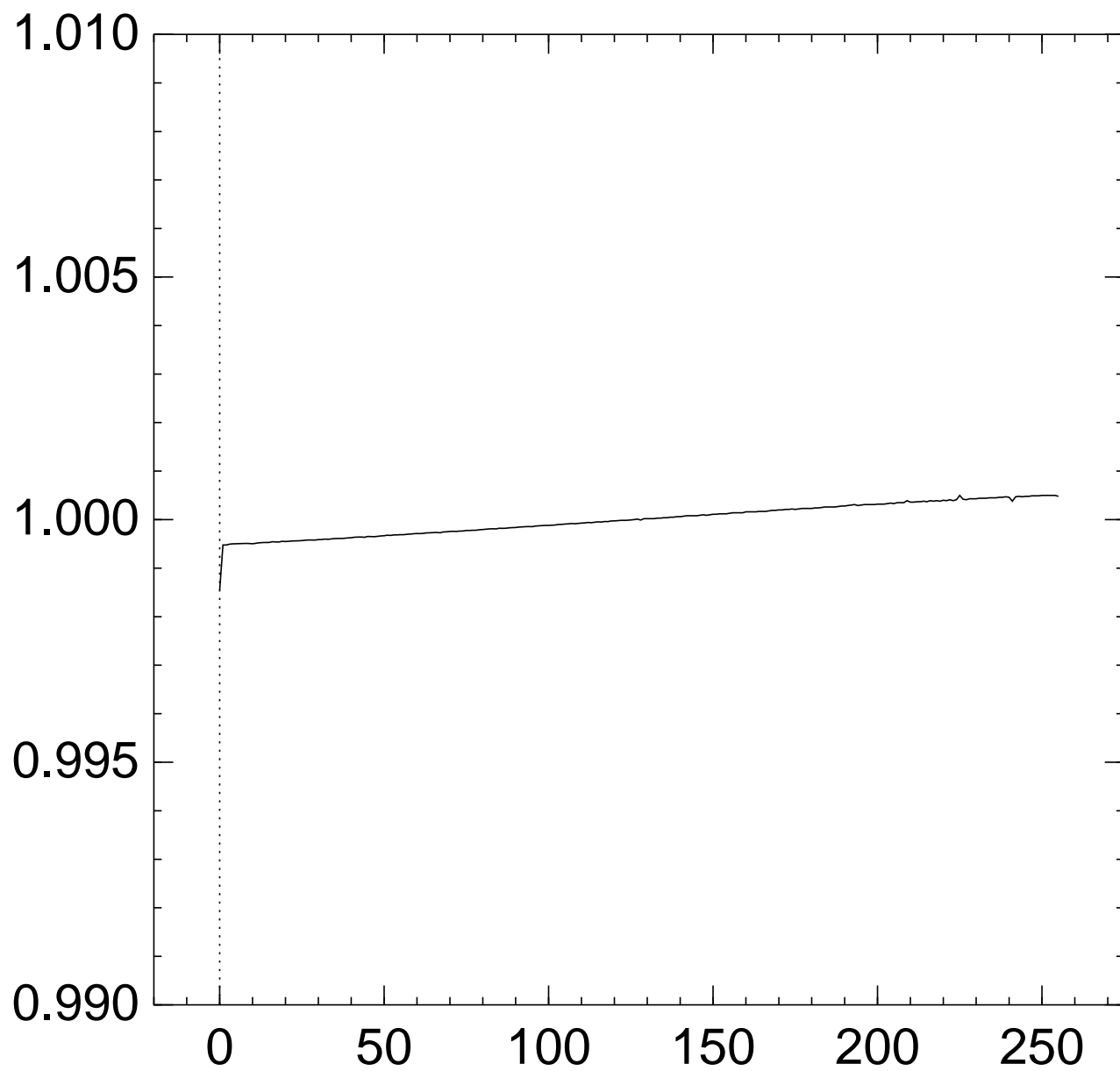
Graph of $256 \Pr[z_{254} = x]$:



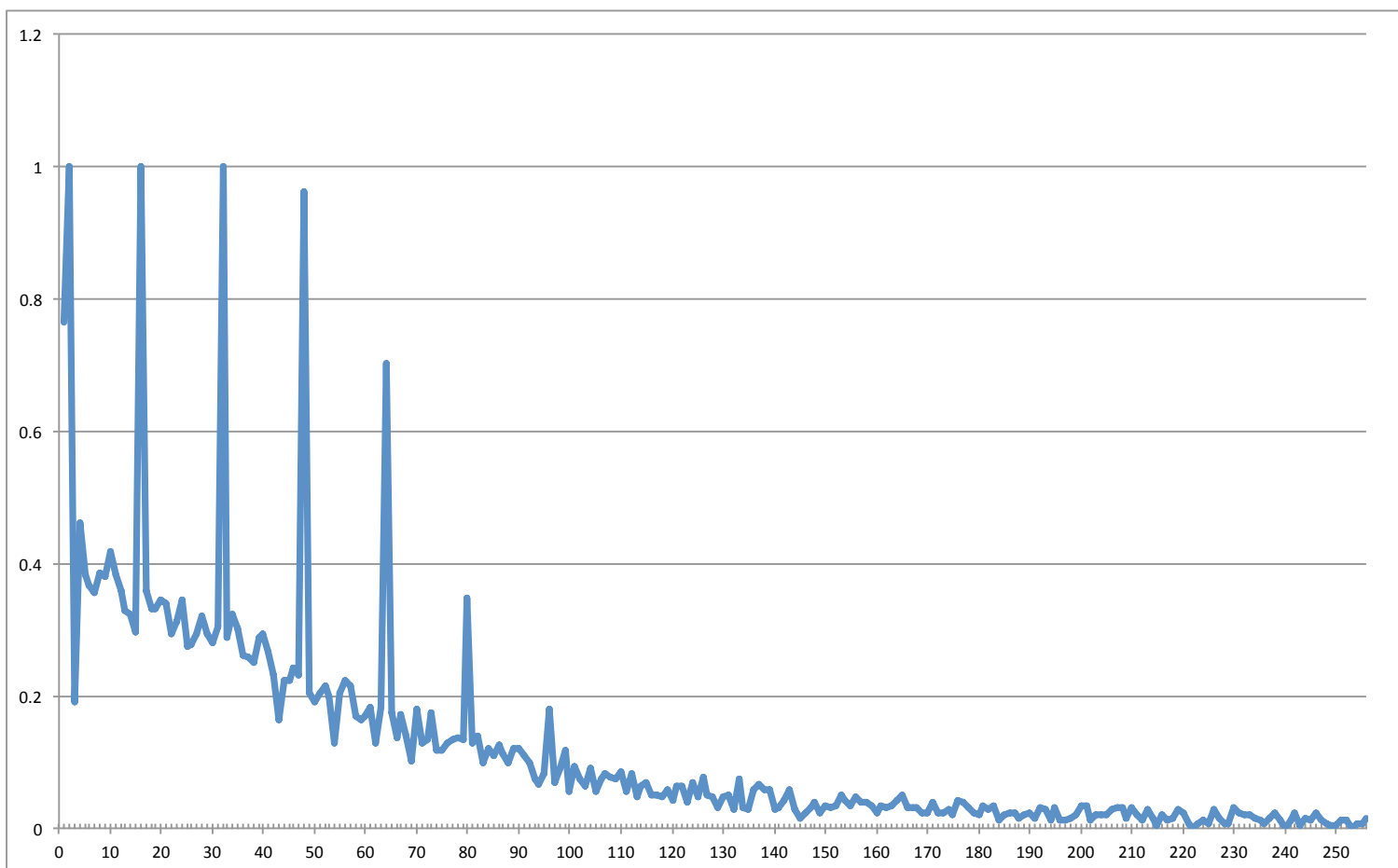
Graph of $256 \Pr[z_{255} = x]$:



Graph of $256 \Pr[z_{256} = x]$:

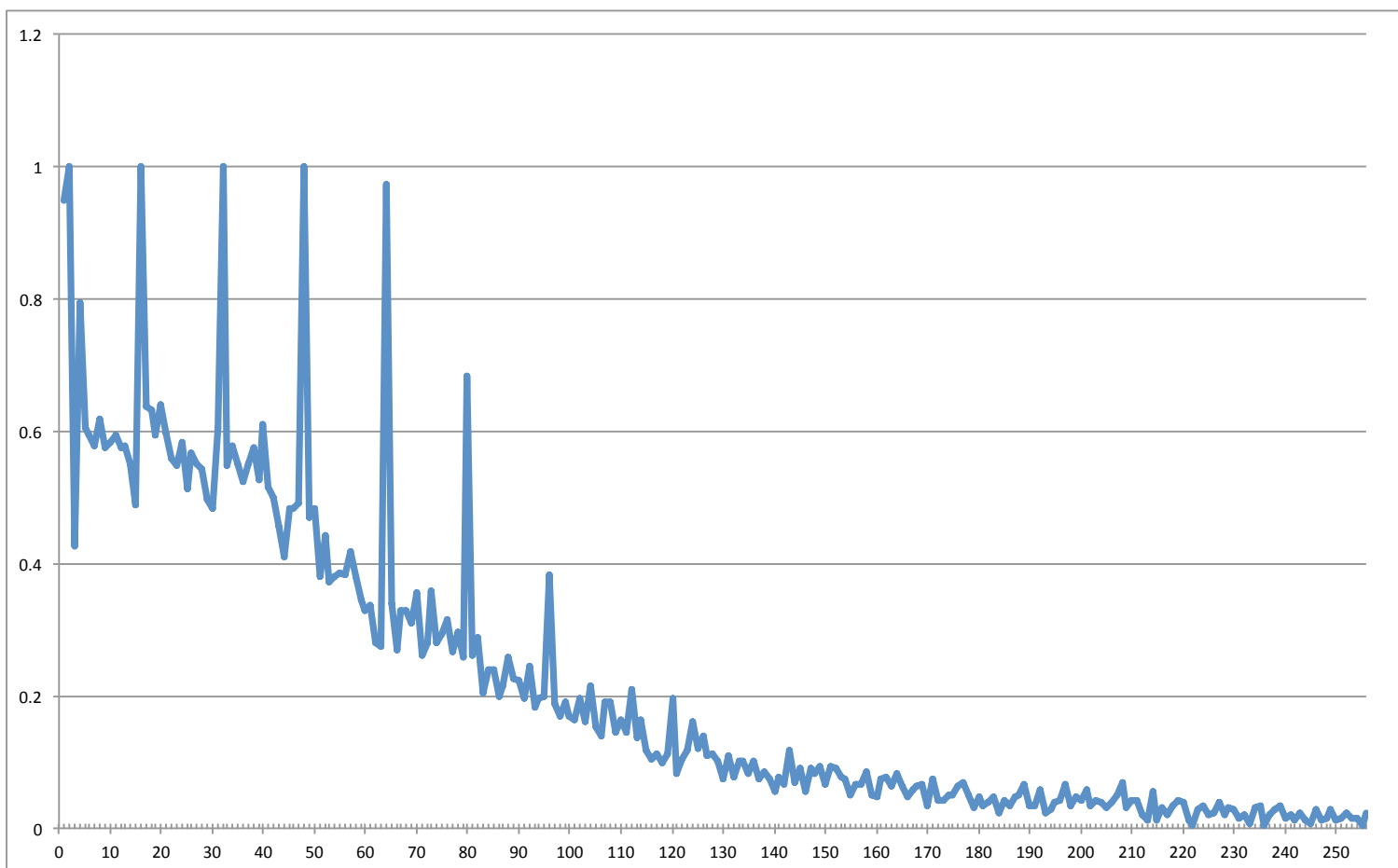


2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{24} ciphertexts (with
no prior plaintext knowledge):



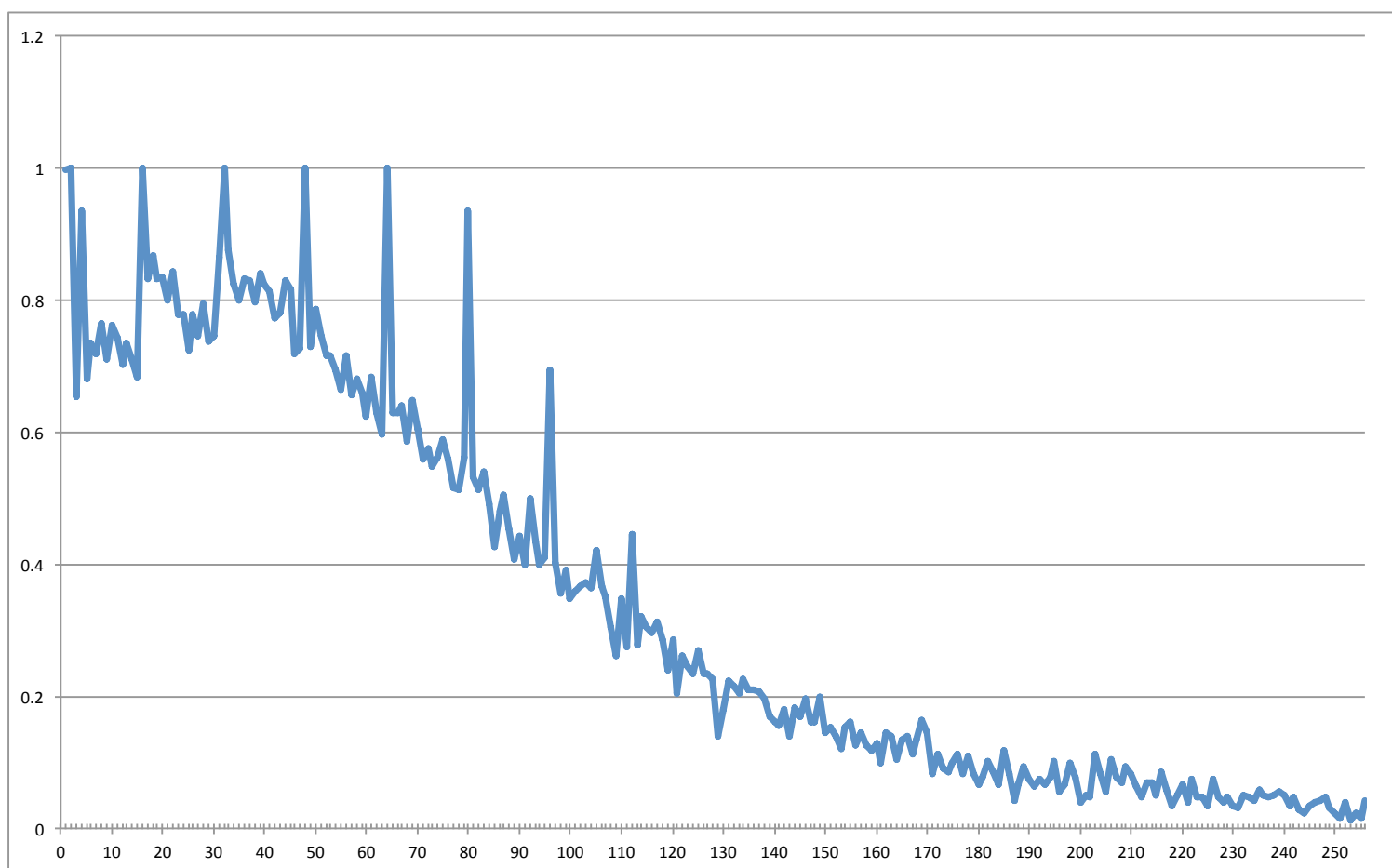
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{25} ciphertexts (with
no prior plaintext knowledge):



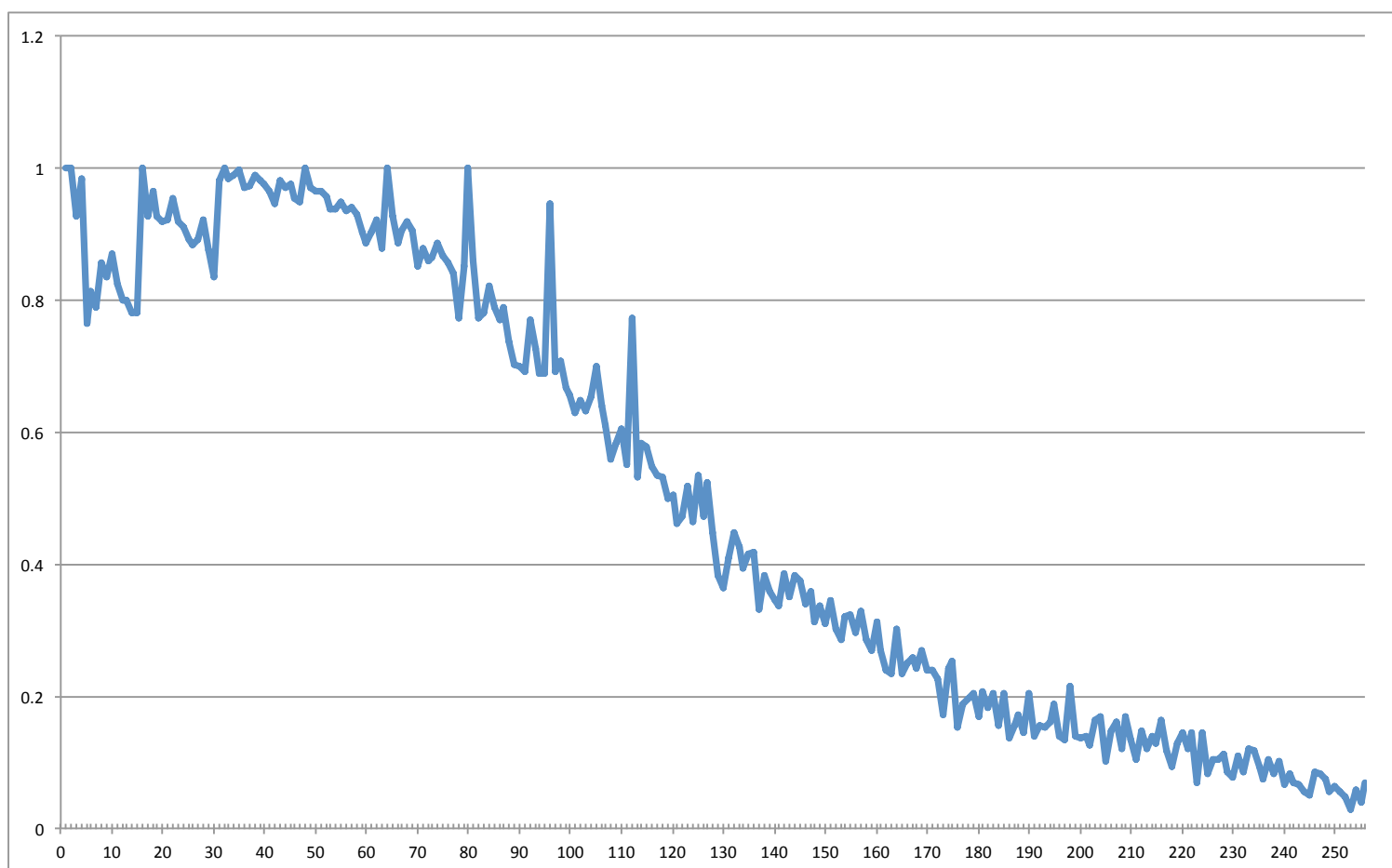
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{26} ciphertexts (with
no prior plaintext knowledge):



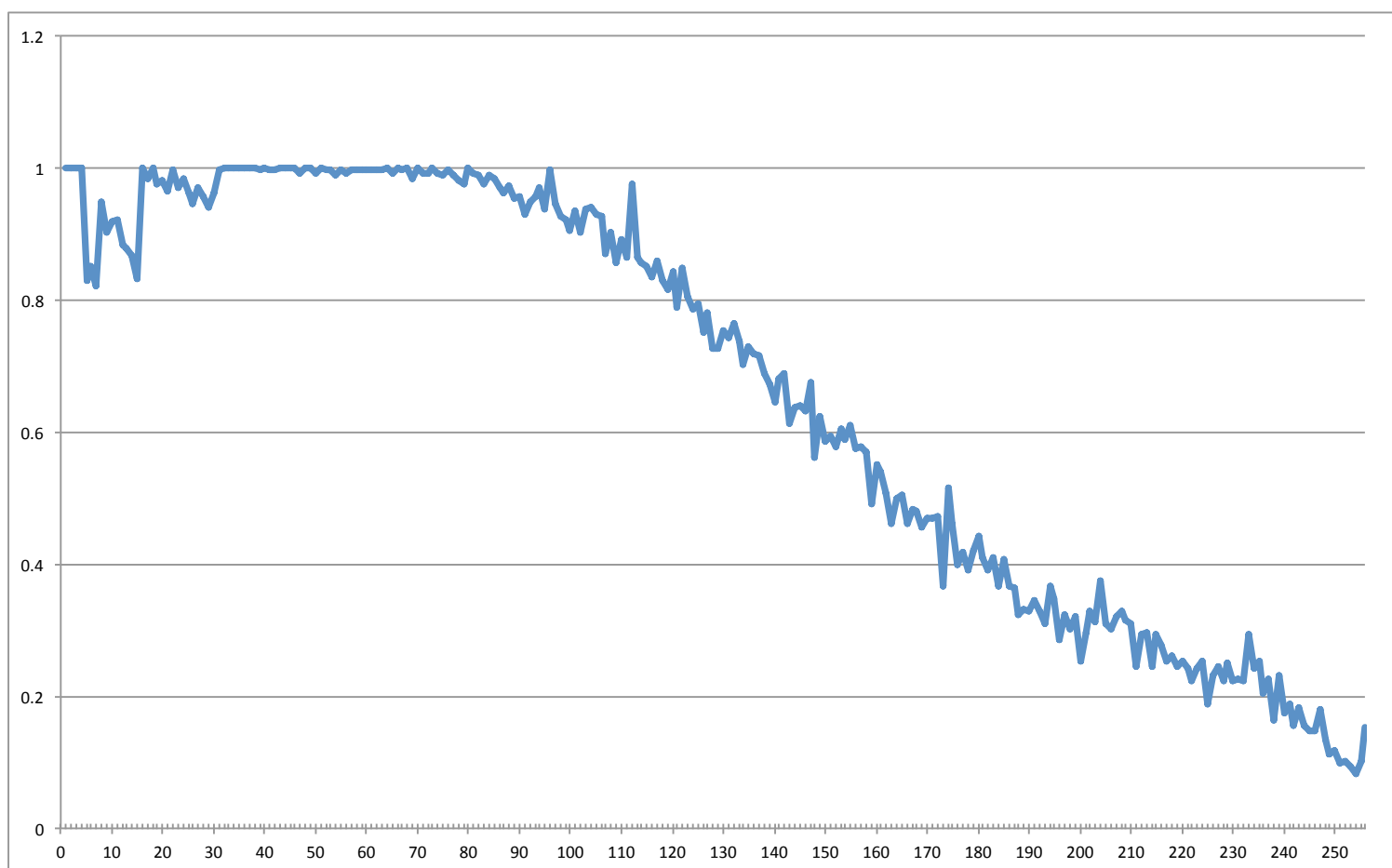
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{27} ciphertexts (with
no prior plaintext knowledge):



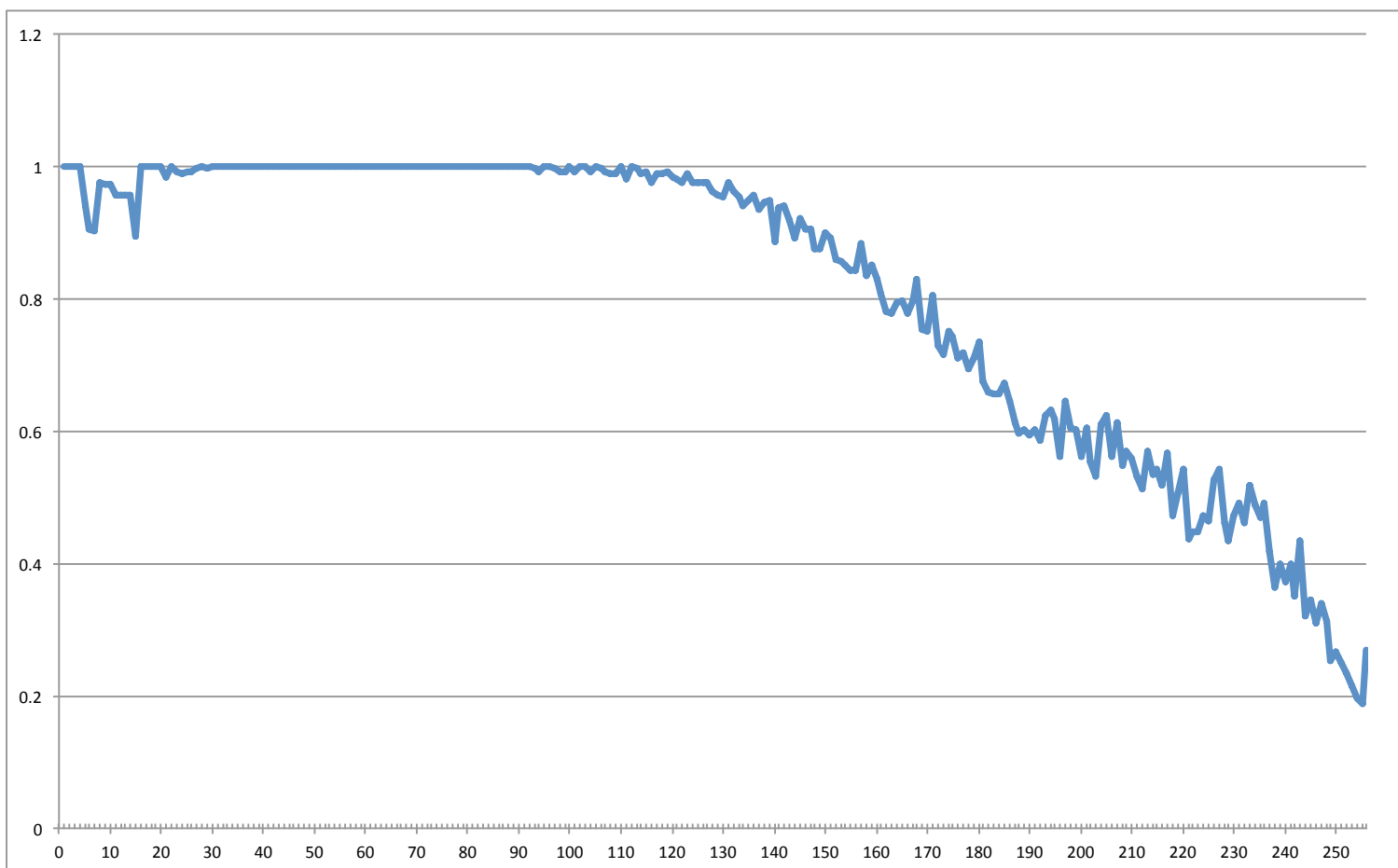
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{28} ciphertexts (with
no prior plaintext knowledge):



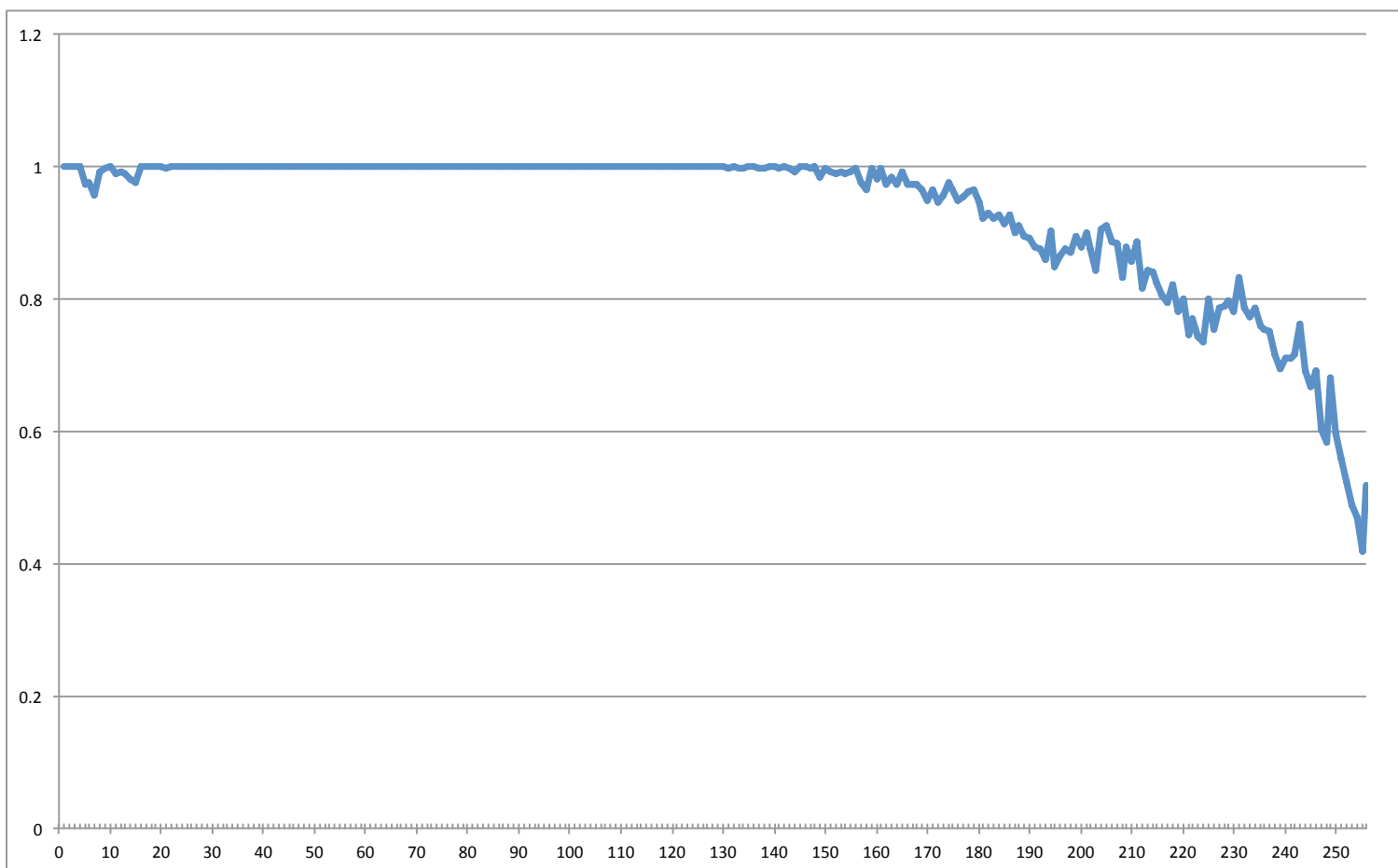
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{29} ciphertexts (with
no prior plaintext knowledge):



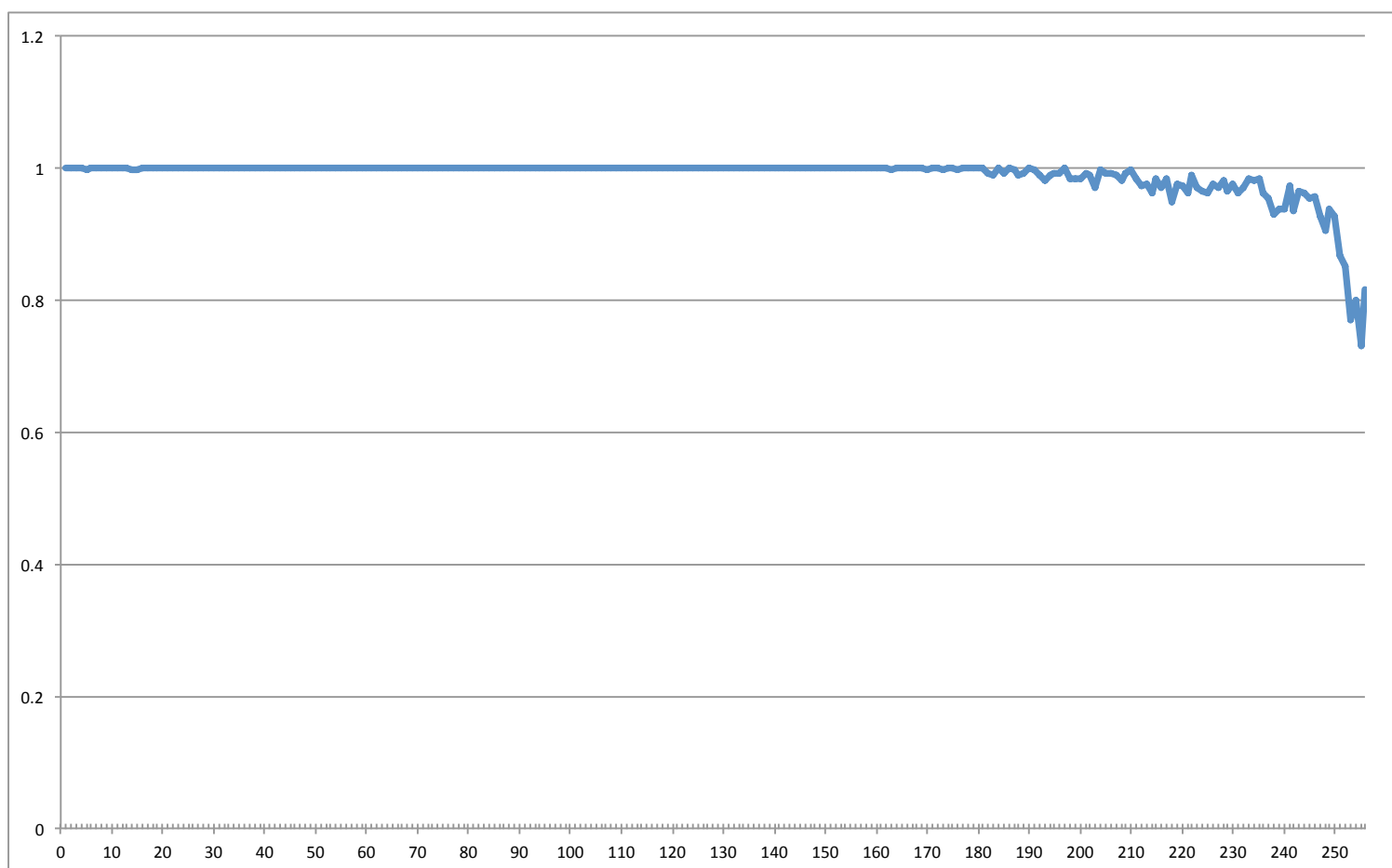
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{30} ciphertexts (with
no prior plaintext knowledge):



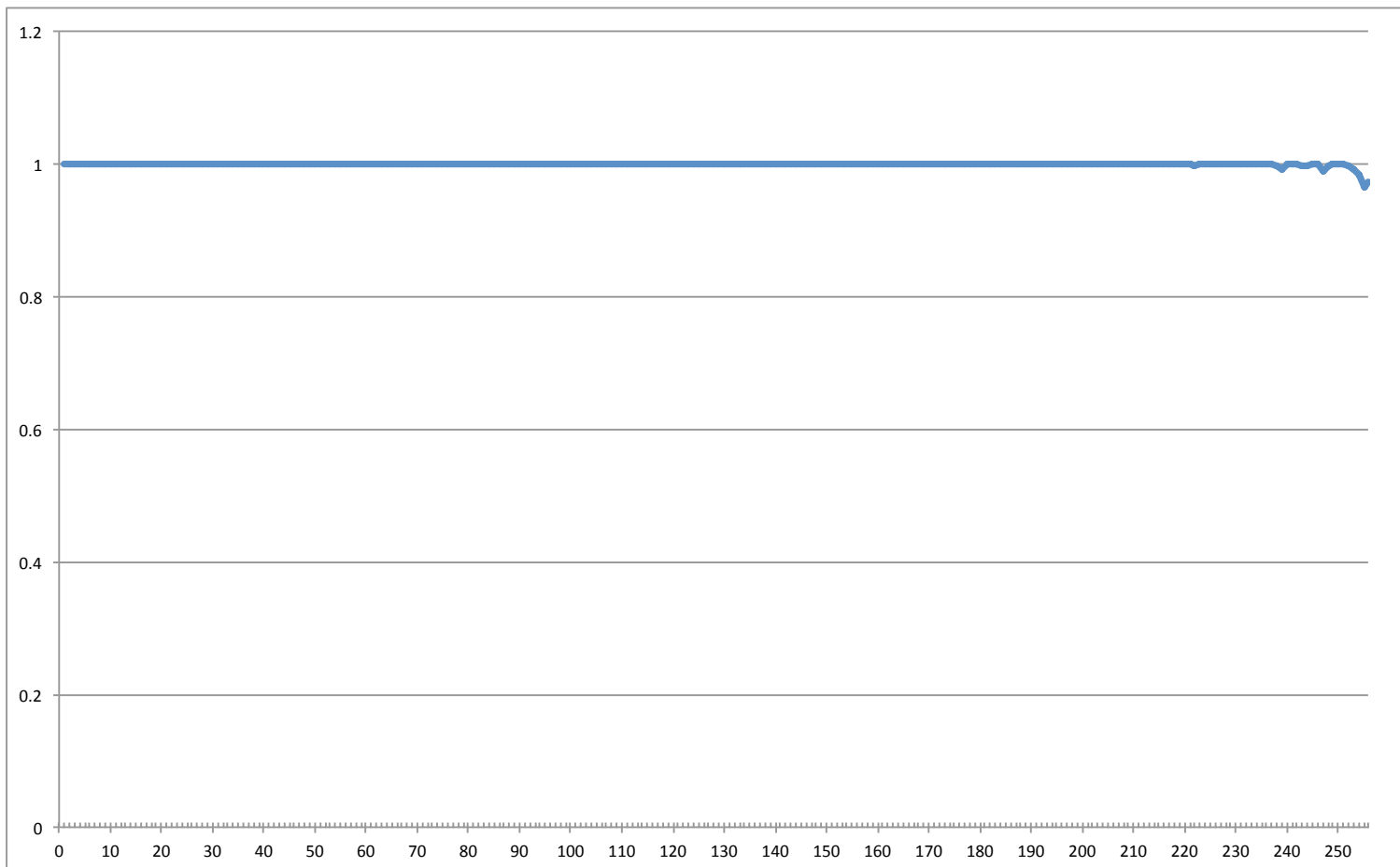
Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{31} ciphertexts (with
no prior plaintext knowledge):



Later bytes: see paper.

2013 AlFardan–Bernstein–
Paterson–Poettering–Schuldt
success probability (256 trials)
for recovering byte x of plaintext
from 2^{32} ciphertexts (with
no prior plaintext knowledge):



Later bytes: see paper.

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Maybe, maybe not.

Does AES-GCM actually do
what the users need?

Why does this happen?

For years we've had AES;
AES-GCM; defenses against
various side-channel attacks.

We simply have to educate the
software and hardware engineers
choosing crypto primitives, right?

Maybe, maybe not.

Does AES-GCM actually do
what the users need?

Often it doesn't.

Most obvious issue: performance.

e.g. 2001 Rivest: “The ‘heart’ of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems.”

e.g. OpenSSL still uses table-based implementations of AES for speed on most CPUs, leaking many key bits; see, e.g., [2012 Weiß–Heinz–Stumpf](#).

e.g. RFIDs need small ciphers.

Major research direction:
achieve better performance
than AES-GCM
without sacrificing security.

Fit into low power (watts),
low area (square micrometers),
sometimes low latency (seconds);
minimize area \times seconds/byte;
minimize energy (joules)/byte.

Many different CPUs, FPGAs,
ASIC manufacturing technologies.

Many different input sizes,
precomputation possibilities, etc.

Can one design do very well
in hardware *and* software?

Some inspirational examples:

Trivium and Keccak
are “hardware” designs
but not bad in software.

Can one design do very well
in hardware *and* software?

Some inspirational examples:

Trivium and Keccak
are “hardware” designs
but not bad in software.

Another approach:

replace ARX with “ORX”.

Skein-type mix doesn't work
but can imitate Salsa20:

compose $\hat{a} = ((b | c) \lll r)$.

Needs a few more rounds,
but friendlier to hardware.

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.

Limit impact of collisions?

Use larger blocks?

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.

Limit impact of collisions?

Use larger blocks?

Typical 128-bit pipe
is starting to feel too small.

Limit reforgeeries? Use wider pipe?

Another major research direction:
achieve better security
than AES-GCM
without sacrificing performance.

Typical 128-bit blocks
are starting to feel too small.

Limit impact of collisions?

Use larger blocks?

Typical 128-bit pipe

is starting to feel too small.

Limit reforgeeries? Use wider pipe?

Has anyone tried optimizing
192-bit/256-bit poly hashes?

(We've started some work.)

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

But user is surprised if repeated
message number leaks more
information, allows forgeries, etc.

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

But user is surprised if repeated
message number leaks more
information, allows forgeries, etc.

2006 Rogaway–Shrimpton:

first authenticate (n, m) ,
then use the authenticator
as a nonce to encrypt m .

Allow repeated message numbers?

User has to expect that
encrypting (n, m) and (n, m')
will tell attacker whether $m = m'$.

But user is surprised if repeated
message number leaks more
information, allows forgeries, etc.

2006 Rogaway–Shrimpton:

first authenticate (n, m) ,
then use the authenticator
as a nonce to encrypt m .

Is this protection compatible
with fast forgery rejection?

Many ciphers integrate
“free” message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Many ciphers integrate
“free” message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

Many ciphers integrate
“free” message authentication:
e.g., AES-OCB, Helix, Phelix.

Is this compatible
with repeated message numbers?

Is this compatible
with fast forgery rejection?

One approach: build
HFFH Feistel block cipher;
reuse first *H* for fast auth
with repeated message numbers;
reuse last *H* for another auth
with fast forgery rejection.

But this consumes bandwidth.

Many more directions
in authenticated ciphers.

AES-GCM is clearly not
the end of the story.

Can build better modes
using same MAC, cipher.

Can build better MACs,
combine with same cipher.

Can build better
block ciphers, stream ciphers.

Can build better integrated
authenticated ciphers.

CAESAR

“Competition for Authenticated Encryption: Security, Applicability, and Robustness”

competitions.cr.yp.to

Mailing list: crypto-competitions+subscribe@googlegroups.com

NIST is much too busy to run another competition but has generously provided a \$333099 “Cryptographic competitions” grant to UIC.