Rigorous Error Analysis for Logarithmic Number Systems

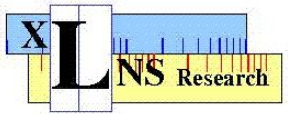
Presented at the 32nd IEEE International Symposium on Computer Arithmetic
ARITH 2025

Thanh Son Nguyen
University of Utah
thahnson@cs.utah.edu

Alexey Solovyev
University of Utah
solovyev.alexey@gmail.com

Ganesh Gopalakrishnan
University of Utah
ganesh@cs.utah.edu

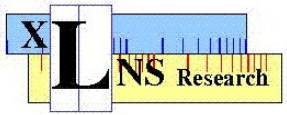
Mark G. Arnold
Lehigh University
maab@lehigh.edu



https://github.com/soarlab/RigorousErrorLNS

www.xlnsresearch.com
github.com/xlnsresearch

A word about “ xlns” as relevant to this paper

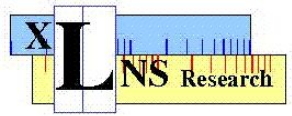


www.xlnsresearch.com

github.com/xlnsresearch

A word about “ xlns” as relevant to this paper

www.xlnsresearch.com website (also xlnsresearch.github.io) = Logarithmic Number Systems (LNS) papers



www.xlnsresearch.com

github.com/xlnsresearch

A word about “ xlns” as relevant to this paper

www.xlnsresearch.com website (also xlnsresearch.github.io) = Logarithmic Number Systems (LNS) papers

github.com/xlnsresearch open-source repositories for LNS

xlns configurable Python library for LNS tensors (analogous to Numpy)

- **available as PyPi wheel via** `pip install xlns`
- **play with it in Python** `import xlns as xl`

www.xlnsresearch.com
github.com/xlnsresearch

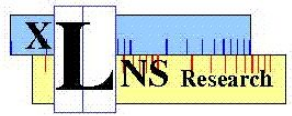
A word about “ xlns” as relevant to this paper

www.xlnsresearch.com website (also xlnsresearch.github.io) = Logarithmic Number Systems (LNS) papers

github.com/xlnsresearch open-source repositories for LNS

xlns configurable Python library for LNS tensors (analogous to Numpy)

- available as PyPi wheel via `pip install xlns`
- play with it in Python `import xlns as xl`
- supports plug-in configurations for novel LNS arithmetic algorithms
- actively seeks open-source contribution of these
- *algorithms used in this paper* are available `import xlnsconf.utah_tayco_ufunc`



www.xlnsresearch.com
github.com/xlnsresearch

A word about “ xlns” as relevant to this paper

www.xlnsresearch.com website (also xlnsresearch.github.io) = Logarithmic Number Systems (LNS) papers

github.com/xlnsresearch open-source repositories for LNS

xlns configurable Python library for LNS tensors (analogous to Numpy)

- **available as PyPi wheel via** `pip install xlns`
- **play with it in Python** `import xlns as xl`
- **supports plug-in configurations for novel LNS arithmetic algorithms**
- **actively seeks open-source contribution of these**
- ***algorithms used in this paper* are available** `import xlnsconf.utah_tayco_ufunc`

```
>>> import xlns as xl
>>> xl.xlnsnp([1,2,3])+4
 xlnsnp([xlns(5.00000016087236) xlns(5.999999933686084) xlns(7.000000011403832)])
```

www.xlnsresearch.com
github.com/xlnsresearch

A word about “ xlns” as relevant to this paper

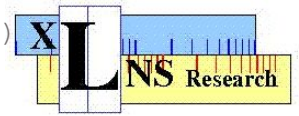
www.xlnsresearch.com website (also xlnsresearch.github.io) = Logarithmic Number Systems (LNS) papers

github.com/xlnsresearch open-source repositories for LNS

xlns configurable Python library for LNS tensors (analogous to Numpy)

- **available as PyPi wheel via** `pip install xlns`
- **play with it in Python** `import xlns as xl`
- **supports plug-in configurations for novel LNS arithmetic algorithms**
- **actively seeks open-source contribution of these**
- ***algorithms used in this paper* are available** `import xlnsconf.utah_tayco_ufunc`

```
>>> import xlns as xl
>>> xl.xlnsnp([1,2,3])+4
 xlnsnp([xlns(5.000[illegible]0016[illegible]87236) xlns(5.999999933686084) xlns(7.000000011403832)])
>>> import xlnsconf.utah_tayco_ufunc
>>> xl.xlnsnp([1,2,3])+4
 xlnsnp([xlns(4.999[illegible]9974[illegible]724449) xlns(5.999999933686084) xlns(6.999999432996774)]
>>> xl.xlnsset(9)
>>> xl.xlnsnp([1,2,3])+4
 xlnsnp([xlns(4.994403908756931) xlns(5.995933468164773) xlns(7.006013412127704)])
```

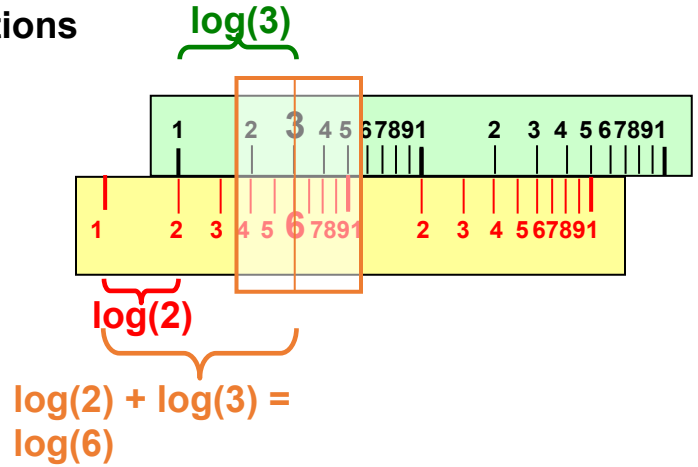


www.xlnsresearch.com
github.com/xlnsresearch

Advantages of LNS

Cheaper multiply, divide, square root

Good for applications with high proportion of multiplications

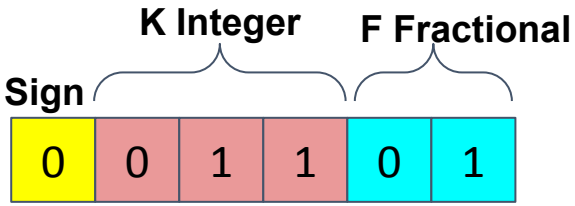


$$\log_2(2^p \times 2^q) = \log_2 2^{p+q} = p + q$$

$$\log_2(2^p / 2^q) = \log_2 2^{p-q} = p - q$$

XLNS Research

Floating Point versus LNS



Different Interpretations!

Floating-point:

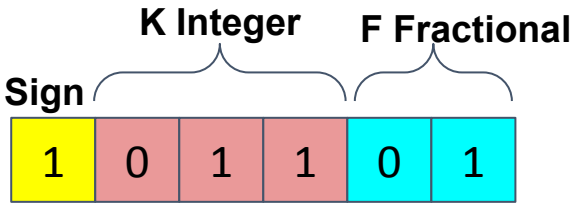
$$+2^{3} \cdot 1.25 = +10$$

LNS:

$$+2^{3.25} = +9.51\ldots$$

Usually, base b = 2

Floating Point versus LNS



Different Interpretations!

Floating-point:

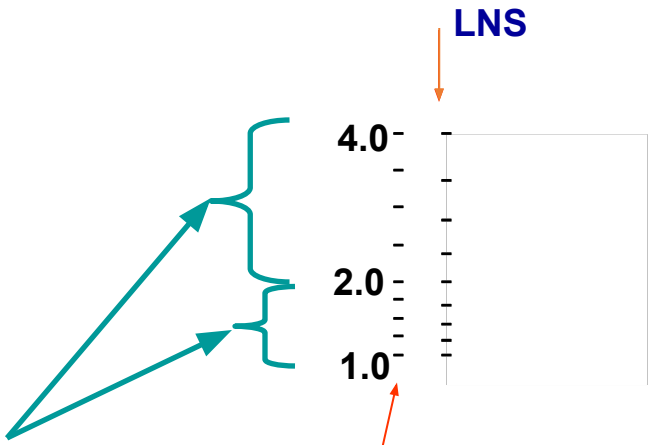
$-2^{3} \cdot 1.25 = -10$

LNS:

$-2^{3.25} = -9.51\ldots$

Usually, base b = 2

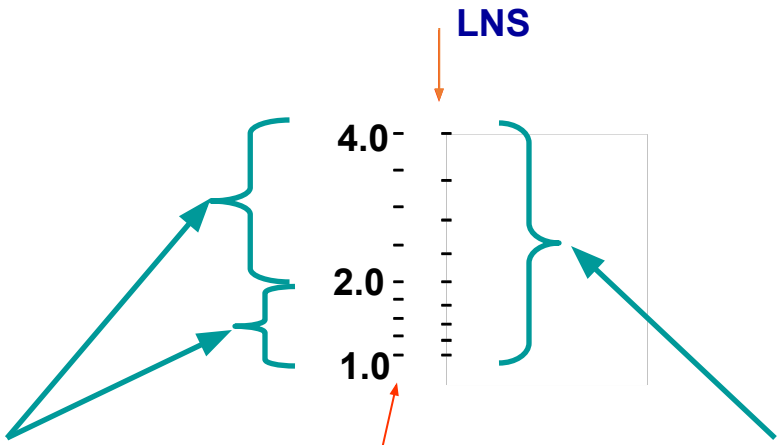
Floating Point versus LNS



Discrete change in distance
causes wobble in relative precision

Floating
Point

Floating Point versus LNS

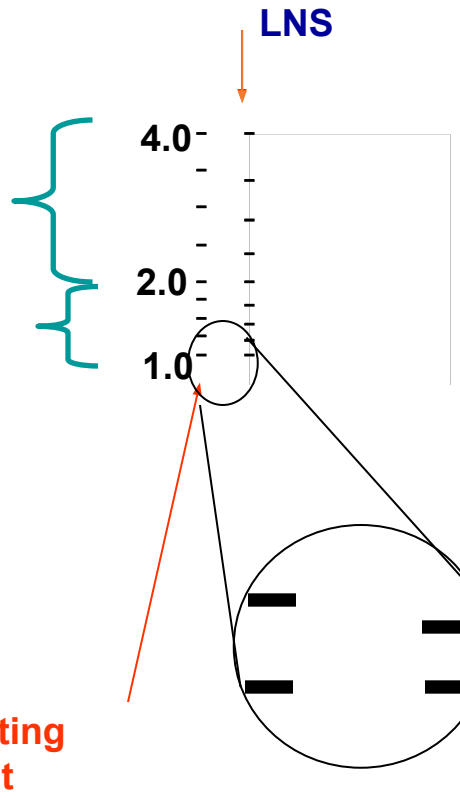


Discrete change in distance
causes wobble in relative precision

Continuous change in distance
means constant relative precision

Floating
Point

Floating Point versus LNS



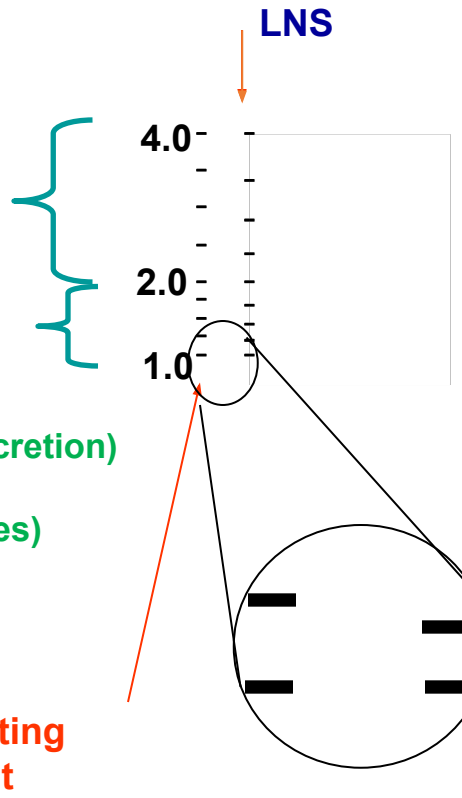
Lewis' Observation:
Round to Nearest LNS
In(2) Better Than FP!
Margin for round error
yet still be *BTFP*

Floating Point versus LNS

Many possible rounding rules:

a) Round to nearest (difficult with LNS)
b) Up
c) Down
d) *Faithful* (either up/down implementor's discretion)
e) Stochastic (like faithful but random)
f) BTFP (like nearest only faithful in hard cases)
g) Weaker than faithful (tableless Mitchell)

Faithful means machine unit $\varepsilon=2^{-F}$

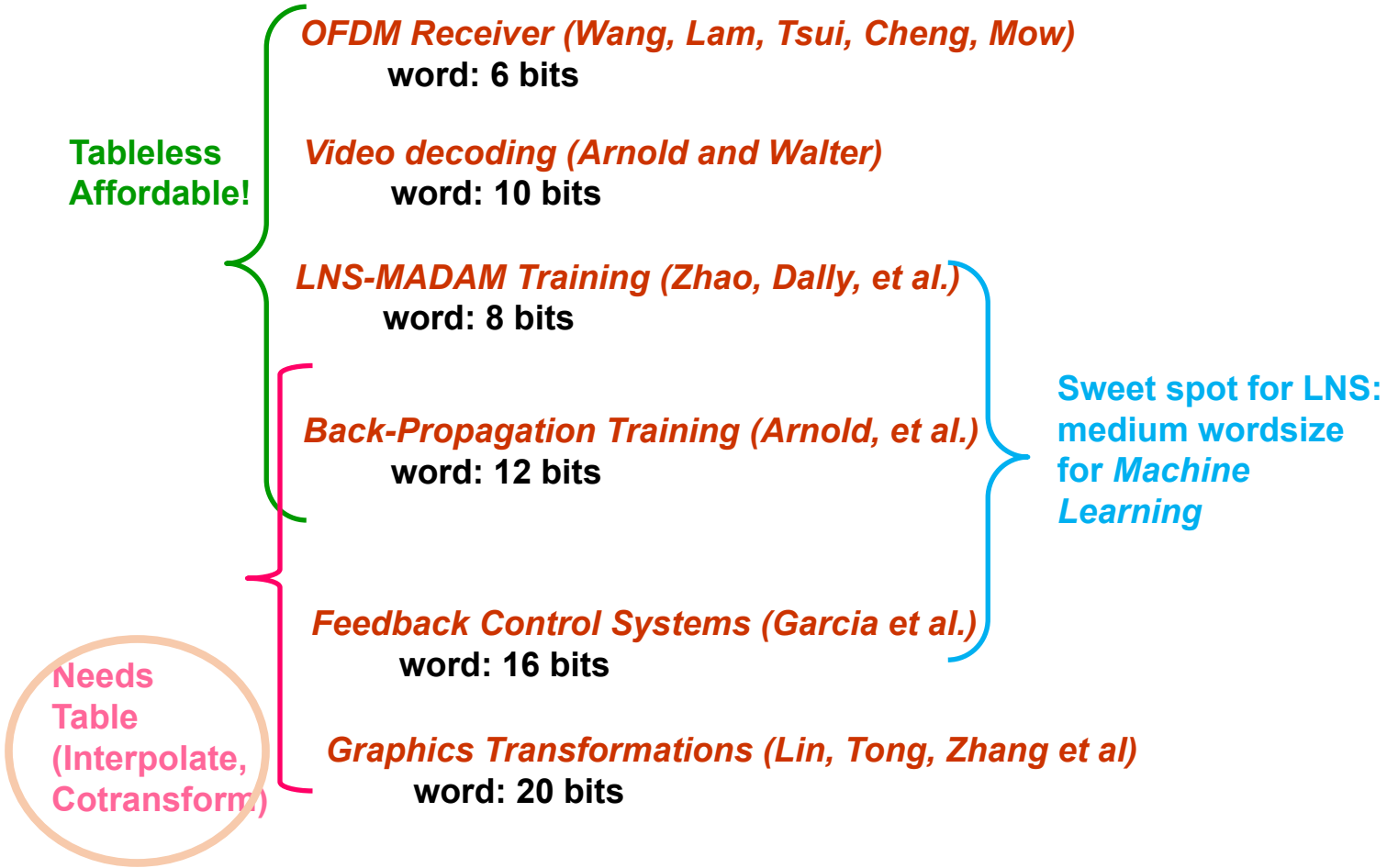


Lewis' Observation:
Round to Nearest LNS
In(2) Better Than FP!
Margin for round error
yet still be *BTFP*

Commercial Interest in LNS

European Union:	European Logarithmic Microprocessor (ELM)
Motorola:	LNS chip for satellite network
Yamaha:	Music Synthesizer
Boeing:	Aircraft controls
Interactive Machines,Inc.:	IMI-500: Animation for *Jay Jay the Jet Plane*
Advanced Rendering Technologies:	Hardware Ray-Tracing Engine
Cambridge/Microsoft:	HTK Hidden Markov Model Toolkit
Univ. of Tokyo:	N-body Gravity Pipeline (GRAPE): Gordon Bell Prize
NVIDIA:	LNS-MADAM and novel LNS summation

LNS Wordsize for Applications

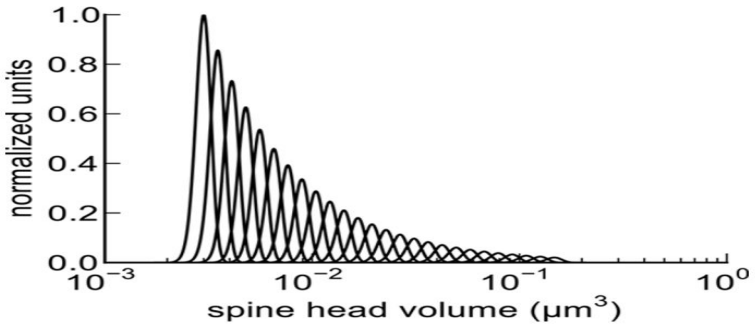


Biological Intelligence uses Logarithms

Weber–Fechner law: … response to sensory stimulus (light, sound,…) proportional to the logarithm of the stimulus.

Mental numbers uses logarithmic scale.

"biological synapses…26 levels in a **logarithmic number system**, thus storing … 5 bits"



G. Buzsáki, K. Mizusek, "The log-dynamic brain: how skewed distributions affect network operations", Nat Rev Neurosci. 15(4):264-78, Apr. 2014.doi: 10.1038/nrn3687.

J. E. Volk, B Parhami, "Number Representation and Arithmetic in the Human Brain", https://web.ece.ucsb.edu/~parhami/pubs_folder/parh20-iemcon-arithmtic-human-brain-final.pdf

T. M. Bartol, Jr., et al., "Nanoconnectomic upper bound on the variability of synaptic plasticity", *eLife,* 2015. 10.7554/eLife.10778.002

Number Systems for Deep Neural Network Architectures: A Survey

Ghada Alsuhli[1], Vasileios Sakellariou[1], Hani Saleh, *Senior Member, IEEE,*[1], Mahmoud Al-Qutayri, *Senior Member, IEEE,*[1], Baker Mohammad, *Senior Member, IEEE,*[1], and Thanos Stouraitis[1]

[47] M. G. Arnold, T. A. Bailey, J. J. Cupal, and M. D. Winkel, "On the cost effectiveness of logarithmic arithmetic for backpropagation training on simd processors," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 2. IEEE, 1997, pp. 933–936.

[48] B. Parhami, "Computing with logarithmic number system arithmetic: Implementation methods and performance benefits," *Computers & Electrical Engineering*, vol. 87, p. 106800, 2020.

[49] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

[50] A. Sanyal, P. A. Beerel, and K. M. Chugg, "Neural network training with approximate logarithmic computations," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 3122–3126.

[51] S. A. Alam, J. Garland, and D. Gregg, "Low-precision logarithmic number systems: Beyond base-2," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 4, pp. 1–25, 2021.

[52] I. Kouretas and V. Paliouras, "Logarithmic number system for deep learning," in *International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018, pp. 1–4.

[53] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[54] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[55] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[56] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 928–931.

[57] R. Pilipović and P. Bulić, "On the design of logarithmic multiplier using radix-4 Booth encoding," *IEEE access*, vol. 8, pp. 64 578–64 590, 2020.

[58] L. Harsha, B. R. Jammu, N. Bodasingi, S. Veeramachaneni, and N. M. SK, "A low error, hardware efficient logarithmic multiplier," *Circuits, Systems, and Signal Processing*, vol. 41, no. 1, pp. 485–513, 2022.

[59] M. S. Kim, A. A. Del Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 617–622.

[60] M. S. Kim, A. A. Del Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 660–675, 2018.

[61] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 145–150.

[62] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 418–425.

[63] U. Lotrič and P. Bulić, "Logarithmic multiplier in hardware implementation of neural networks," in *International Conference on Adaptive and Natural Computing Algorithms*. Springer, 2011, pp. 158–168.

[64] H. Kim, M. S. Kim, A. A. Del Barrio, and N. Bagherzadeh, "A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 108–111.

[65] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[66] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 614–625, 2020.

[67] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018.

[68] T.-B. Juang, C.-Y. Lin, and G.-Z. Lin, "Area-delay product efficient design for convolutional neural network circuits using logarithmic number systems," in *International SoC Design Conference (ISOCC)*. IEEE, 2018, pp. 170–171.

[69] T.-B. Juang, H.-L. Kuo, and K.-S. Jan, "Lower-error and area-efficient antilogarithmic converters with bit-correction schemes," *Journal of the Chinese Institute of Engineers*, vol. 39, no. 1, pp. 57–63, 2016.

[70] T.-B. Juang, P. K. Meher, and K.-S. Jan, "High-performance logarithmic converters using novel two-region bit-level manipulation schemes," in *Proceedings of 2011 [illegible] on VLSI Design, Automation and Test*. IEEE, 2011, pp. 1–4.

[71] J. Zhao, S. Dai, R. Venkatesan, M.-Y. Liu, B. Khailany, B. Dally, and A. Anandkumar, "Low-precision training in logarithmic number system using multiplicative weight update," *arXiv preprint arXiv:2106.13914*, 2021.

[72] S. [illegible], M. Liang, A. Guntoro, W. Stechele, and G. Asch[illegible], "Efficient hardware [illegible] logarithmic data representation with arbitrary log-base," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[73] T.-Y. Lu, H.-H. Chin, H.-I. Wu, and R.-S. Tsay, "A very compact embedded CNN processor design based on logarithmic computing," *arXiv preprint arXiv:2010.11686*, 2020.

[74] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logarithmic computation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5900–5904.

[75] J. Xu, Y. Huan, L.-R. Zheng, and Z. Zou, "A low-power arithmetic element for multi-base logarithmic computation on deep neural networks," in *IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018, pp. 43–48.

[76] J. Xu, Y. Huan, Y. Jin, H. Chu, L.-R. Zheng, and Z. Zou, "Base-reconfigurable segmented logarithmic quantization and hardware design for deep neural networks," *Journal of Signal Processing Systems*, vol. 92, no. 11, pp. 1263–1276, 2020.

[77] T. Ueki, K. Iwai, T. Matsubara, and T. Kurokawa, "Learning accelerator of deep neural networks with logarithmic quantization," in *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*. IEEE, 2018, pp. 634–638.

Number Systems for Deep Neural Network Architectures: A Survey

Ghada Alsuhli[1], Vasileios Sakellariou[1], Hani Saleh, *Senior Member, IEEE,*[1], Mahmoud Al-Qutayri, *Senior Member, IEEE,*[1], Baker Mohammad, *Senior Member, IEEE,*[1], and Thanos Stouraitis[1]

[47] M. G. Arnold, T. A. Bailey, J. J. Cupal, and M. D. Winkel, "On the cost effectiveness of logarithmic arithmetic for backpropagation training on simd processors," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 2. IEEE, 1997, pp. 933–936.

[48] B. Parhami, "Computing wi... Implementation methods ar... *Electrical Engineering*, vol.

[49] D. Miyashita, E. H. Lee, ... ral networks using logarith... *arXiv:1603.01025*, 2016.

[50] A. Sanyal, P. A. Beerel, and... with approximate logarithm... *Conference on Acoustics, S...* IEEE, 2020, pp. 3122–3126.

[51] S. A. Alam, J. Garland, an... number systems: Beyond ba... *and Code Optimization (TA...*

[52] I. Kouretas and V. Palioura... learning," in *International C... Technologies (MOCAST)*. I...

[53] H. Saadat, H. Bokhari, an... multipliers for approximate ... *IEEE Transactions on Comp... and Systems*, vol. 37, no. 11...

[54] J. N. Mitchell, "Computer ... logarithms," *IRE Transaction...* 517, 1962.

[55] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[56] M. S. Ansari, B. F. Cockburn, and J. Han, "A hardware-efficient logarithmic multiplier with improved accuracy," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 928–931.

[57] R. Pilipović and P. Bulić, "On the design of logarithmic multiplier using radix-4 Booth encoding," *IEEE access*, vol. 8, pp. 64 578–64 590, 2020.

... in *Proceedings of 201... on VLSI Design, Aut... and Test*. IEEE, 2011, pp. 1–4.

[... J. Zhao, S. Dai, R. Venkatesan, M.-Y. Liu, B. Khailany, B. Dally, and A. Anandkumar, "Low-precision training in logarithmic number system using multiplicative weight update," *arXiv preprint arXiv:2106.13914*, 2021.

[72] S... M. Liang, A. Guntoro, W. Stechele, and G. Asch... "Efficient hardware ... ... data repre-sentation with arbitrary log-base," in *Proceedings of the International*

Jiawei Zhao, Steve Dai, Rangharajan Venkatesan, Brian Zimmer, Mustafa Ali, Ming-Yu Liu, Brucek Khailany, William J. Dally, Anima Anandkumar, "LNS-Madam: Low-Precision Training in Logarithmic Number System using Multiplicative Weight Update" arXiv preprint 2106.13914v3

Appeared in IEEE Trans. Comput. vol. 71, no. 12, pp. 3179-3190, 1 Dec. 2022, doi: 10.1109/TC.2022.3202747.

Authors are from NVIDIA, which has several related patent applications

Neural Network Accelerator Using Logarithmic-based Arithmetic
Dally; William James ; et al.
uspto.report › / patents › / NVIDIA Corporation › / Patent 16/549683

Inference Accelerator Using Logarithmic-based Arithmetic
Dally; William James ; et al.
uspto.report › / patents › / NVIDIA Corporation › / Patent 16/750823

Energy and Policy Considerations for Deep Learning in NLP

Emma Strubell **Ananya Ganesh** **Andrew McCallum**
College of Information and Computer Sciences
University of Massachusetts Amherst
{strubell, aganesh, mccallum}@cs.umass.edu

Abstract

nt progress in hardware and methodol-
or training neural networks has ushered
new generation of large networks trained
undant data. These models have ob-
d notable gains in accuracy across many
tasks. However, these accuracy improve-
s depend on the availability of exception-
arge computational resources that neces-
similarly substantial energy consump-
As a result these models are costly to
and develop, both financially, due to the
f hardware and electricity or cloud com-
ime, and environmentally, due to the car-

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

[47] N
e
c
N
[48] E
I
E
[49] D
r
[50] A
[51] S
[52] I
l
T
[53] P
n
I
and Systems, vol. 37, no. 11
[54] J. N. Mitchell, "Computer
logarithms," *IRE Transaction*
517, 1962.

VLSI Design,
Khailany, B. Dally, and
arithmic number system
rint arXiv:2106.13914,
ele, and G. Asch
mic data repre-
ings of the International

rian Zimmer, Mustafa
hima Anandkumar,
Number System using
914v3

. 3179-3190, 1 Dec.

baten applications
mic-based Arithmetic

mic-based Arithmetic
Dally; William James ; et al.
uspto.report › / patents › / NVIDIA Corporation › / Patent 16/750823

12-1-2024

Logarithmic Number System Is Optimal for AI Computations: Theoretical Explanation of Empirical Success

Olga Kosheleva
The University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Christoph Lauter
The University of Texas at El Paso, cqlauter@utep.edu

Kristalys Ruiz-Rohena
The University of Texas at El Paso, kruizrohena@miners.utep.edu

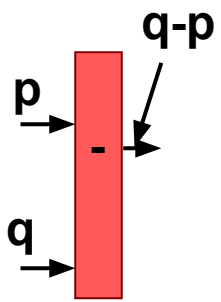
517, 1962.

LNS Addition via Gaussian Logs

Given $p = \log_b|P| < q = \log_b|Q|$:
1. Let $x = q - p$

Why it works:
1. $x = \log_b|Q/P|$

Hardware:
1 min/max unit (so p<q)
1 subtractor

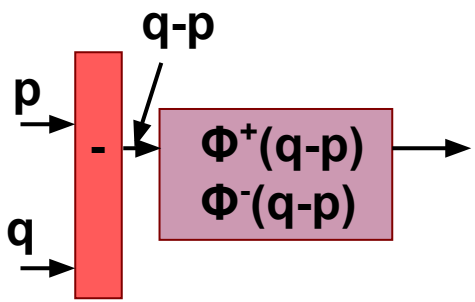


LNS Addition via Gaussian Logs

Given $p = \log_b|P| < q = \log_b|Q|$:
1. Let $x = q - p$
2. Lookup $\Phi^+(x) = \log_b(1+b^x)$
   or $\Phi^-(x) = \log_b|1-b^x|$

Why it works:
1. $x = \log_b|Q/P|$
2. $\Phi^+(x) = \log_b(1+|Q/P|)$
   $\Phi^-(x) = \log_b|1-|Q/P||$

Hardware:
1 min/max unit (so p<q)
1 subtractor
1 function approximation unit

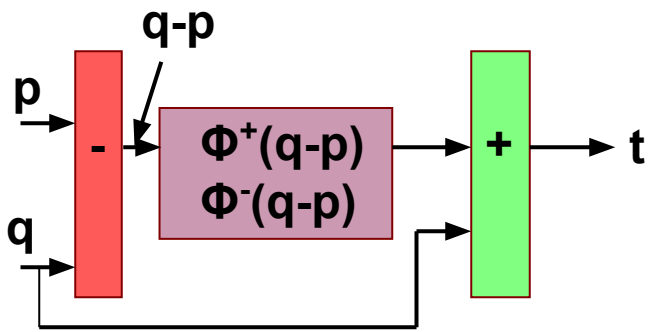


LNS Addition via Gaussian Logs

Given $p = \log_b|P| < q = \log_b|Q|$:
1. Let $x = q - p$
2. Lookup $\Phi^+(x) = \log_b(1+b^x)$
   or $\Phi^-(x) = \log_b|1-b^x|$
3. $t = p + \Phi(x)$

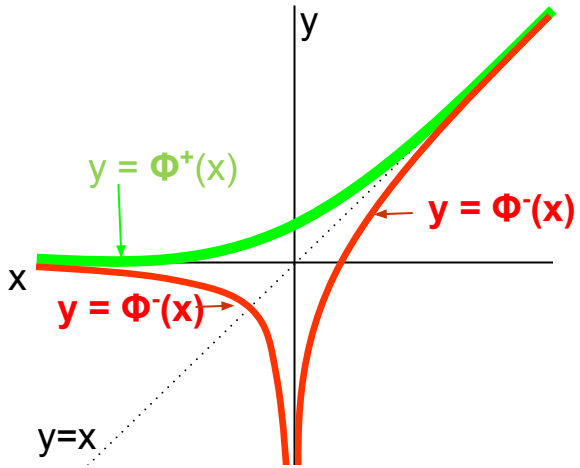
Why it works:
1. $x = \log_b|Q/P|$
2. $\Phi^+(x) = \log_b(1+|Q/P|)$
   $\Phi^-(x) = \log_b|1-|Q/P||$
3. $t = \log_b(|P|(1\pm|Q|/|P|)$

Hardware:
1 min/max unit (so p<q)
1 subtractor
1 function approximation unit
1 adder



Gaussian Logs

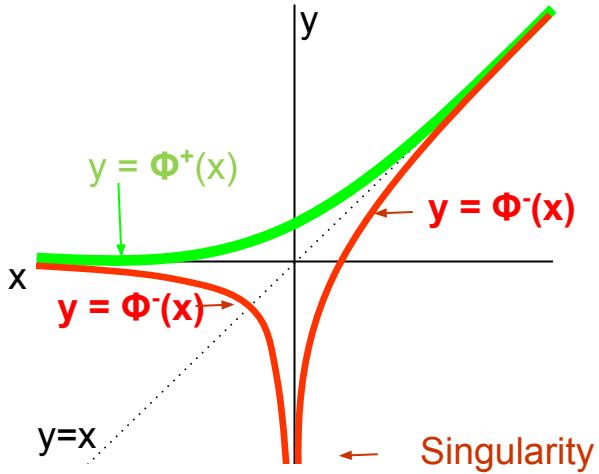
Use: $\Phi^+(x)$ when signs are same, also known as $s_b(x) = \log_b(1+b^z)$
$\Phi^-(x)$ when signs are different, also known as $d_b(x) = \log_b|1-b^z|$
$\Phi(x)$ for generic Gaussian Log



Gaussian Logs

Use: $\Phi^+(x)$ when signs are same, also known as $s_b(x) = \log_b(1+b^z)$
$\Phi^-(x)$ when signs are different, also known as $d_b(x) = \log_b|1-b^z|$
$\Phi(x)$ for generic Gaussian Log

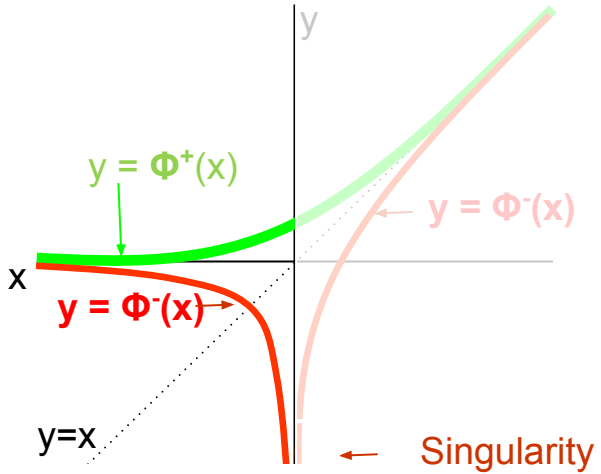
$\Phi^-(x)$ *harder to approximate*
due to singularity near x=0



Gaussian Logs

Use: $\Phi^+(x)$ when signs are same, also known as $s_b(x) = \log_b(1+b^z)$
$\Phi^-(x)$ when signs are different, also known as $d_b(x) = \log_b|1-b^z|$
$\Phi(x)$ for generic Gaussian Log

$\Phi^-(x)$ *harder to approximate*
due to singularity near x=0

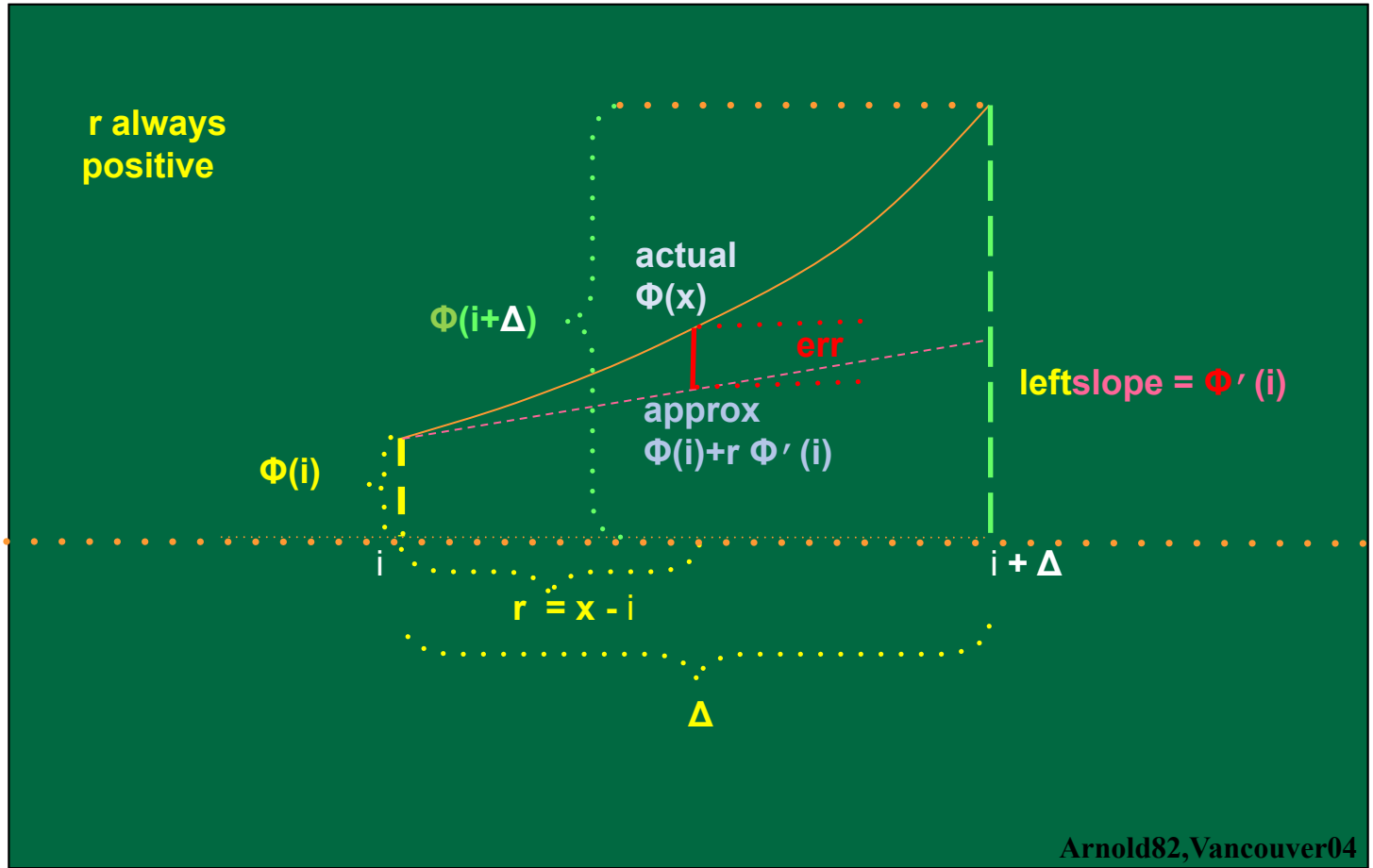


Commutativity: only use x < 0

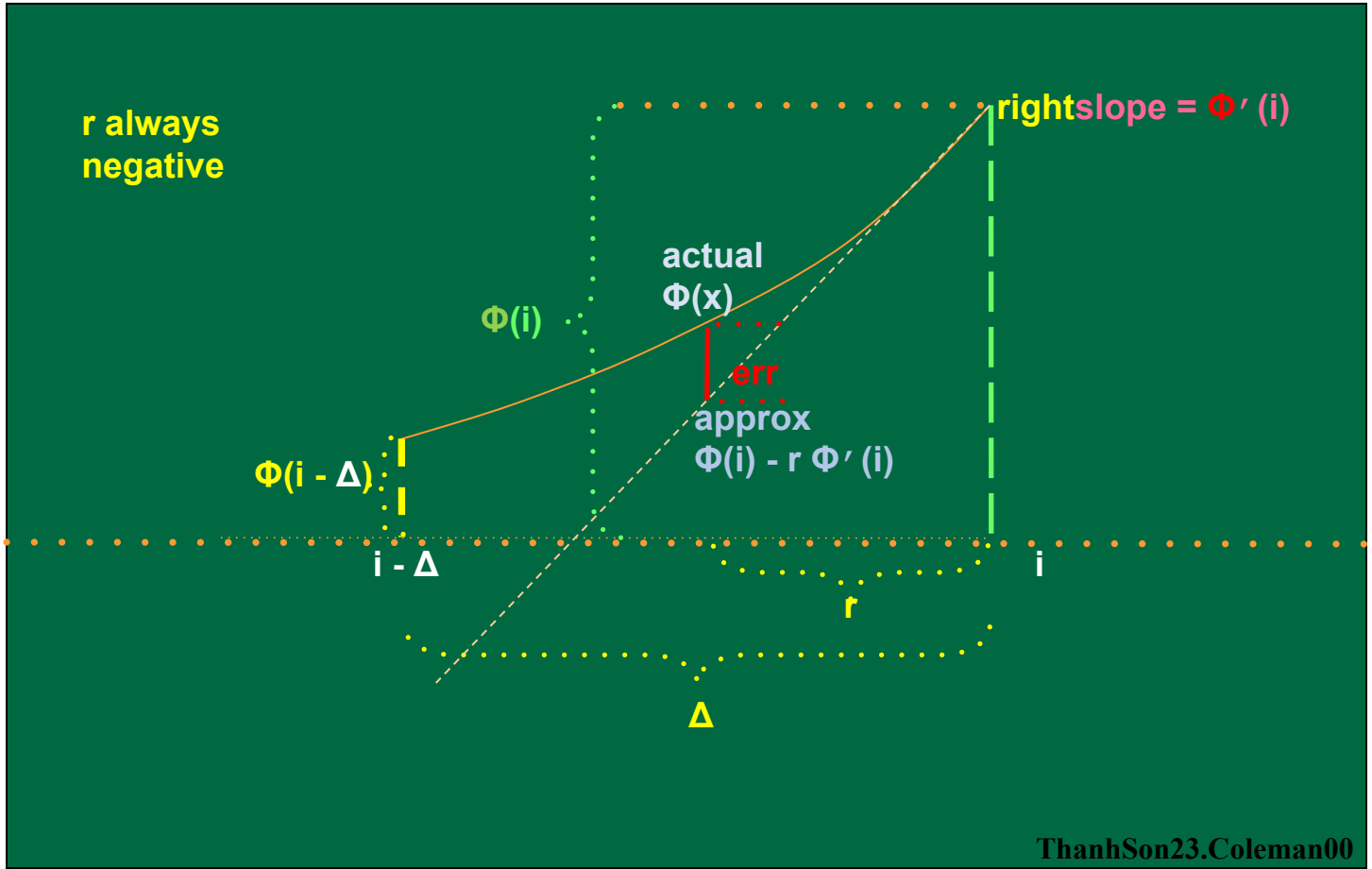
$\Phi(x) = \Phi(-x) + x$

$1+X = X+1 = (1+1/X)\ X$

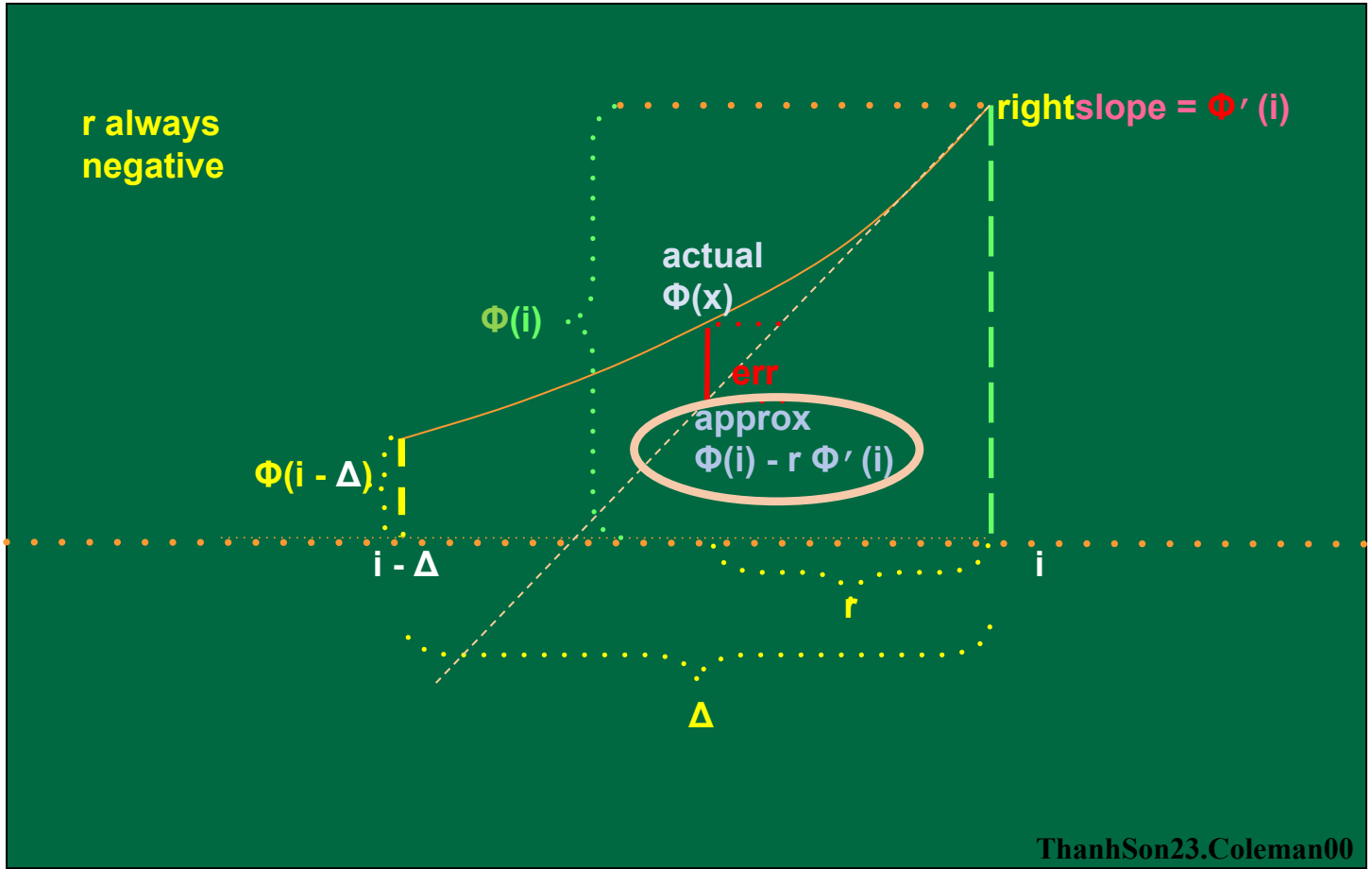
Left Linear Taylor Interpolation



Right Linear Taylor Interpolation



Right Linear Taylor Interpolation



Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1990]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1990]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

3. Use co-transformation to

a) Convert Φ^- with $x = r_b + r_a$ to Φ^+ [Arnold 1998] :

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^+(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a > 0$ and $r_b > 0$

Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1990]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

3. Use co-transformation to

a) Convert Φ^- with $x = r_b + r_a$ to Φ^+ [Arnold 1998] :

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^+(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a > 0$ and $r_b > 0$

b) Convert Φ^- with $x = r_b - r_a$ near 0 to Φ^- with arg away 0 [Coleman 2000]:

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^-(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a < 0$ and $r_b > 0$

Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1990]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

3. Use co-transformation to

a) Convert Φ^- with $x = r_b + r_a$ to Φ^+ [Arnold 1998] :

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^+(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a > 0$ and $r_b > 0$

b) Convert Φ^- with $x = r_b - r_a$ near 0 to Φ^- with arg away 0 [Coleman 2000]:

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^-(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a < 0$ and $r_b > 0$

Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1990]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

3. Use co-transformation to

a) Convert Φ^- with $x = r_b + r_a$ to Φ^+ [Arnold 1998] :

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^+(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a > 0$ and $r_b > 0$

b) Convert Φ^- with $x = r_b - r_a$ near 0 to Φ^- with arg away 0 [Coleman 2000]:

Commutativity:

$\Phi(- r_a) = - r_a + \Phi(r_a)$

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^-(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a < 0$ and $r_b > 0$

$= \Phi^-(r_b) + \Phi^-(r_b - r_a + \Phi^-(r_a) - \Phi^-(r_b))$

$= \Phi^-(r_b) + \Phi^-(x + \Phi^-(r_a) - \Phi^-(r_b))$

Problem: still needs moderate-sized table of $\Phi^-(r_a)$ and $\Phi^-(r_b)$

Prior solutions for Φ^- singularity

1. Partition range of x with non-uniform Δ [Lewis 1989]

F (precision)	15	17	19	21	23
Φ^+ bits	0.3K	1K	2K	5K	10K
Φ^- bits	1K	4K	10K	24K	60K

Problem: Φ^- takes most of the ROM

2. Separate non-interpolative computation of $1+b^x$ and $\log_b$ [Chen 2003]

Problem: Number of stages proportional to precision

3. Use co-transformation to

a) Convert Φ^- with $x = r_b + r_a$ to Φ^+ [Arnold 1998] :

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^+(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a > 0$ and $r_b > 0$

b) Convert Φ^- with $x = r_b - r_a$ near 0 to Φ^- with arg away 0 [Coleman 2000]:

Commutativity:

$\Phi(-r_a) = -r_a + \Phi(r_a)$

$\Phi^-(r_b + r_a) = \Phi^-(r_b) + \Phi^-(r_b + \Phi^-(r_a) - \Phi^-(r_b))$, where $r_a < 0$ and $r_b > 0$

$= \Phi^-(r_b) + \Phi^-(r_b - r_a + \Phi^-(r_a) - \Phi^-(r_b))$

$= \Phi^-(r_b) + \Phi^-(x + \Phi^-(r_a) - \Phi^-(r_b))$

Problem: still needs moderate-sized table of $\Phi^-(r_a)$ and $\Phi^-(r_b)$

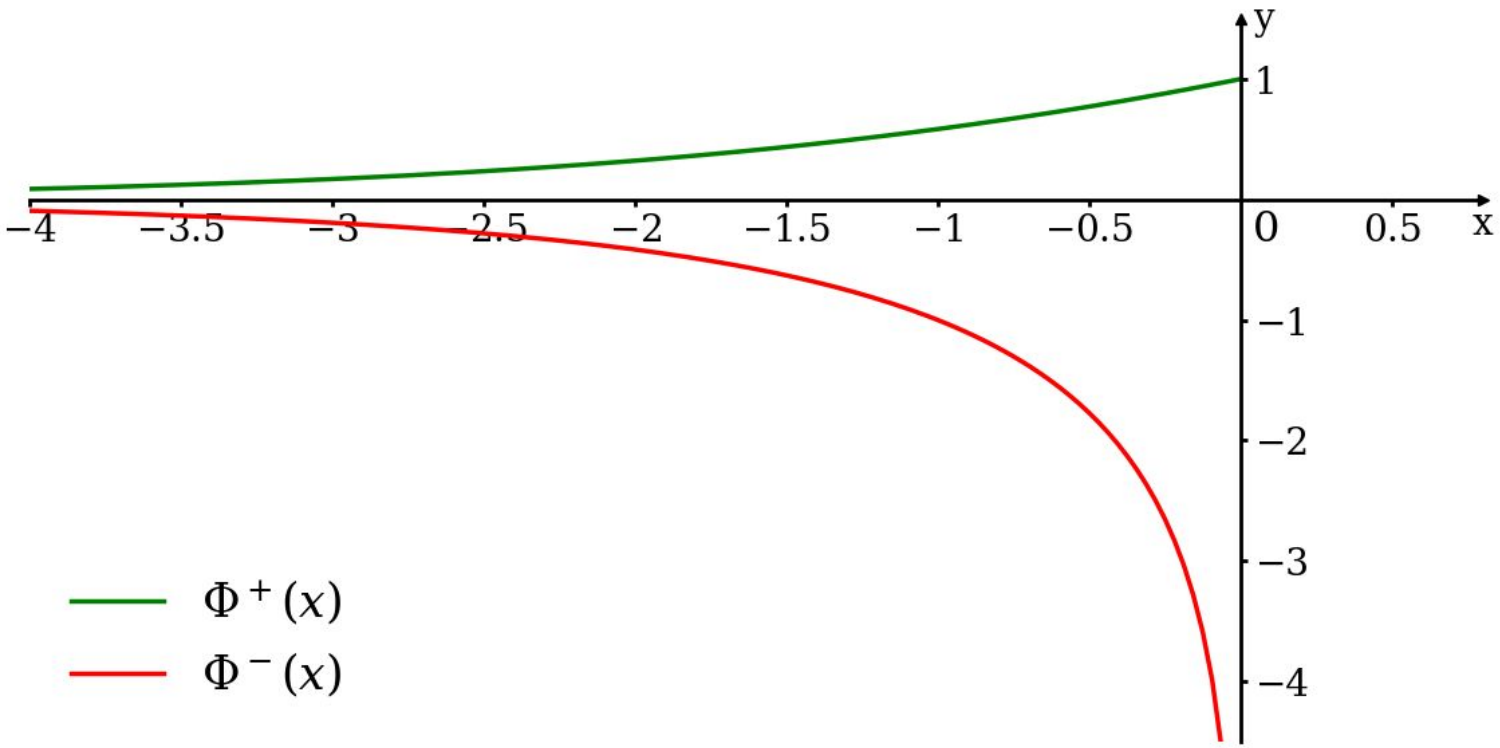
4. Use higher-order co-transformation involving r_a, r_b, r_c ...:

a) Recursive application of 3a) [Arnold 1997] or

b) Recursive application of 3b) [Coleman 2011]

Problem: challenging to formalize

Formally verified error bounds for Φ^+ and Φ^- approximations



Lean4: A programming language and a proof assistant

- **Built on dependent type theory**
  - Enables precise mathematical definitions and rigorous proof construction.
- **Mathlib: A comprehensive mathematics library**
  - Includes definitions and theorems crucial for formalizing diverse areas like calculus, algebra, and analysis.
- **Metaprogramming capabilities**
  - Enables custom tactics and automation to streamline the proof process.

Mathlib support for formalizing calculus-based proofs

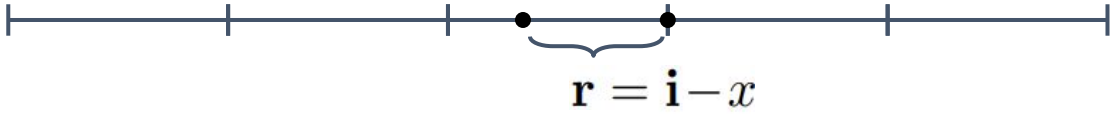
- Limits of Functions
  - `Filter.Tendsto.add, Filter.Tendsto.rpow`
  - `HasDerivAt.lhopital_zero_right_on_Ioo`
- Derivatives
  - `deriv_add, deriv_sub, deriv_mul, deriv_div`
- Continuity and Differentiability
  - `Continuous.add, ContinuousOn.add, ContinuousAt.add`
  - `Differentiable.add, DifferentiableOn.add, DifferentiableAt.add`
- Monotonicity and Antitonicity
  - `monotone_of_deriv_nonneg, antitone_of_deriv_nonpos`
  - `strictMono_of_deriv_pos, strictAnti_of_deriv_neg`
- Tactics
  - `fun_prop, continuity`
  - `norm_num, positivity, linarith`
  - `field_simp, ring_nf`

First-order Taylor approximation

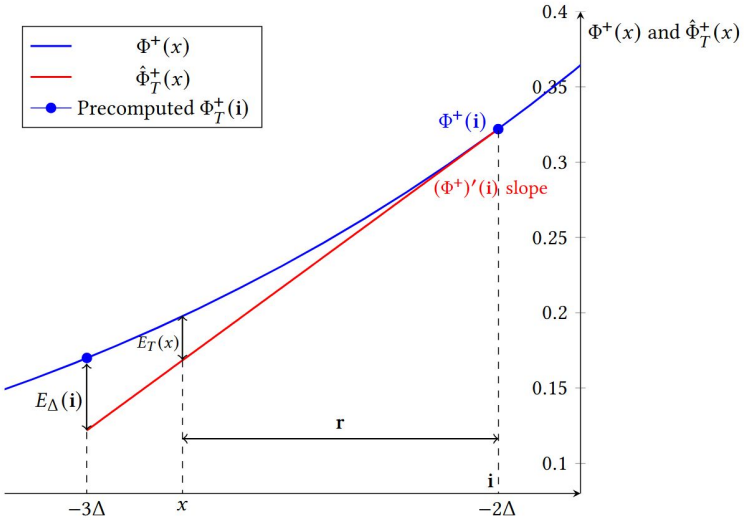
Let Δ be the distance of adjacent values in the look-up tables

Want to compute x

Pre-computed and stored $\mathbf{i} = \Delta(x \text{ div } \Delta)$



$\mathbf{r} = \mathbf{i} - x$



$$\hat{\Phi}_T(x) = \Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i})$$

Look-up Tables

Error sources

REAL

$$\hat{\Phi}_T(x) = \Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i})$$

Approximation Error

Fixed-point rounding Error

FIXED-POINT

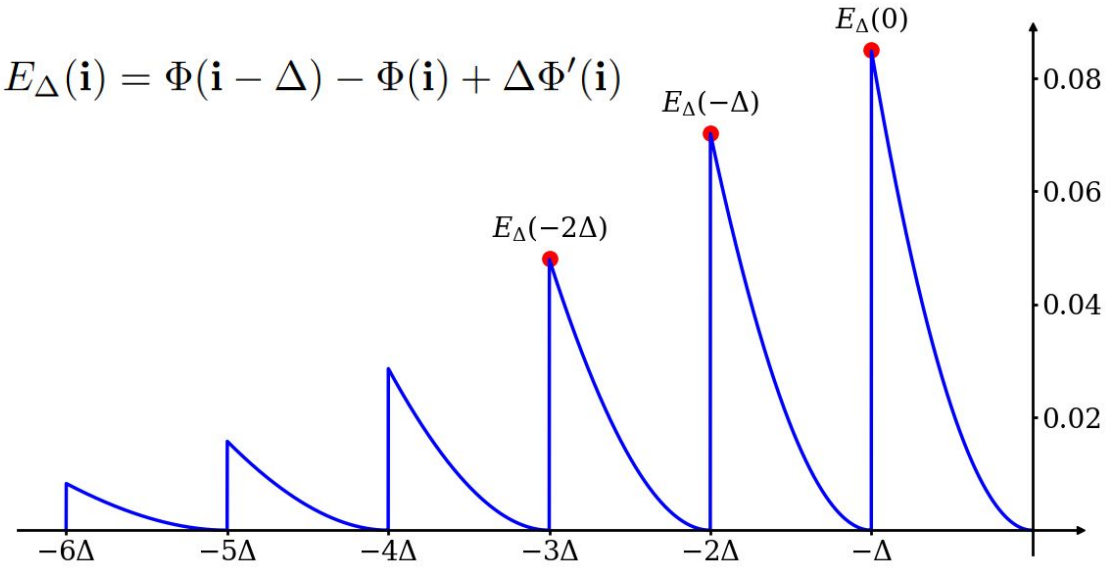
$$\tilde{\Phi}_T(x) = \overline{\Phi}(\mathbf{i}) - \mathrm{rnd}(\mathbf{r}\overline{\Phi'}(\mathbf{i}))$$

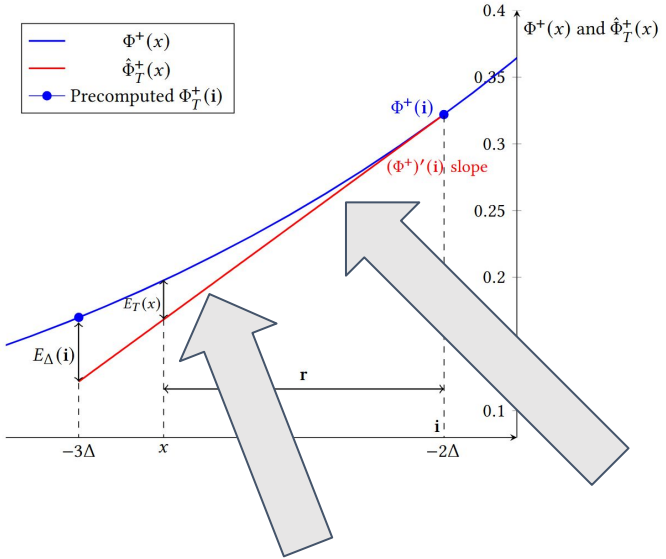
Error of first-order Taylor approximation of Φ^+

$$E(x) = \Phi(x) - \hat{\Phi}_T(x) = \Phi(\mathbf{i} - \mathbf{r}) - (\Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}))$$

Y-Axis : Error of first-order Taylor approximation at x

$$E_\Delta(\mathbf{i}) = \Phi(\mathbf{i} - \Delta) - \Phi(\mathbf{i}) + \Delta\Phi'(\mathbf{i})$$





Error increases wiith **r** and vanishes at table entries

Main proof ideas

- Consider the error function

$$E(\mathbf{i}, \mathbf{r}) = \Phi(\mathbf{i} - \mathbf{r}) - (\Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}))$$

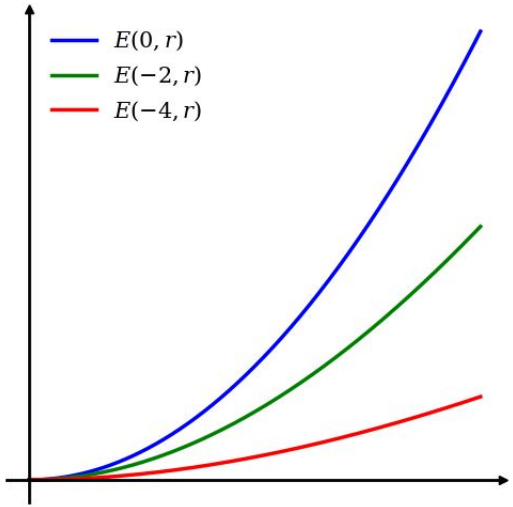
as a function of two independent real-valued variable $\mathbf{i} \leq 0$ and $\mathbf{r} \geq 0$.

Main proof ideas

- Consider the error function

$$E(\mathbf{i}, \mathbf{r}) = \Phi(\mathbf{i} - \mathbf{r}) - (\Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}))$$

  as a function of two independent real-valued variable **i** ≤ 0 and **r** ≥ 0.
- Show that $E(\mathbf{i}, \mathbf{r})$ is a **strictly increasing** function of **r** for fixed **i** ≤ 0.

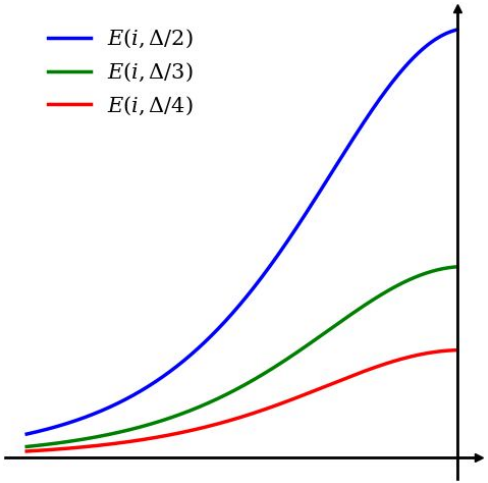


Main proof ideas

- Consider the error function

$$E(\mathbf{i}, \mathbf{r}) = \Phi(\mathbf{i} - \mathbf{r}) - (\Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}))$$

  as a function of two independent real-valued variable **i** ≤ 0 and **r** ≥ 0.
- Show that $E(\mathbf{i}, \mathbf{r})$ is a **strictly increasing** function of **r** for fixed **i** ≤ 0.
- Show that $E(\mathbf{i}, \mathbf{r})$ is a **strictly increasing** function of **i** for fixed 0 ≤ **r**.

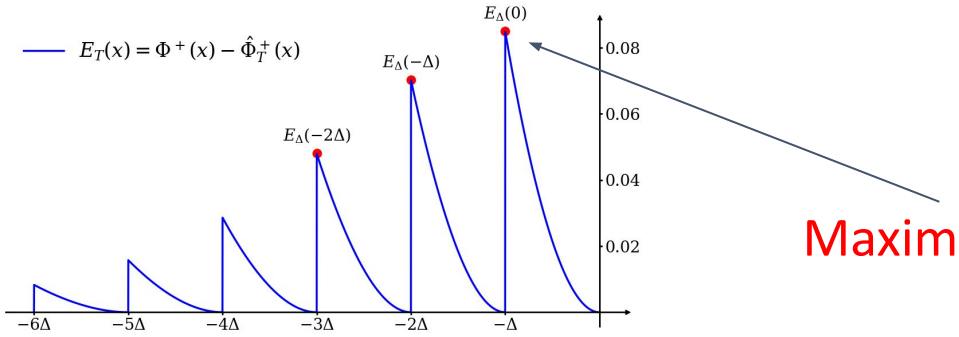


Main proof ideas

- Consider the error function

$$E(\mathbf{i}, \mathbf{r}) = \Phi(\mathbf{i} - \mathbf{r}) - (\Phi(\mathbf{i}) - \mathbf{r}\Phi'(\mathbf{i}))$$

  as a function of two independent real-valued variable **i** ≤ 0 and **r** ≥ 0.
- Show that $E(\mathbf{i}, \mathbf{r})$ is a **strictly increasing** function of **r** for fixed **i** ≤ 0.
- Show that $E(\mathbf{i}, \mathbf{r})$ is a **strictly increasing** function of **i** for fixed 0 ≤ **r**.
- The maximum error is reached when **i** = 0 and **r** = Δ.



Maximum error

Error bound for first-order Taylor approximation of Φ^+

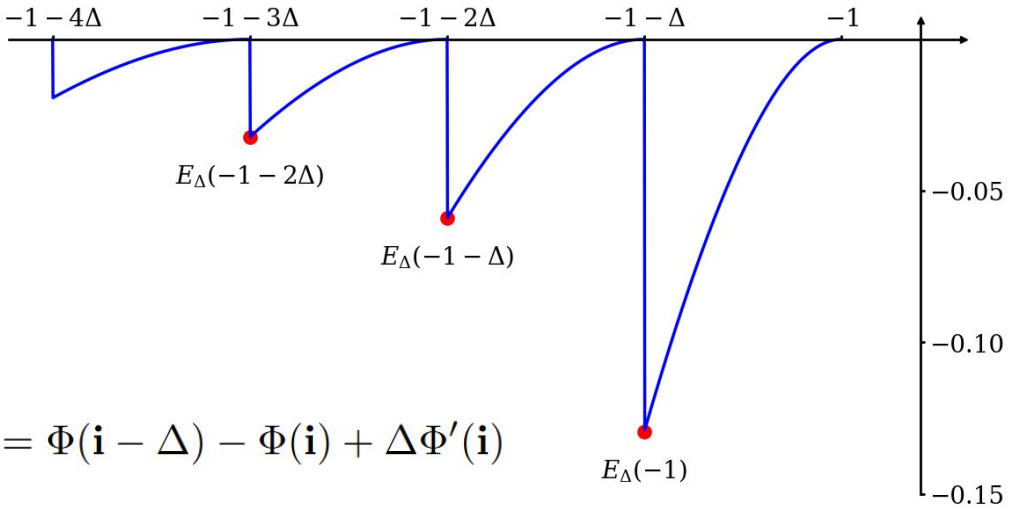
Lemma 1. *For all* $x \in (-\infty, 0]$,

$$|\Phi^+(x) - \hat{\Phi}^+_T(x)| \leq E^+_\Delta(0) = \frac{\ln 2}{8}\Delta^2 + O(\Delta^4).$$

```
lemma Ep_bound' (hi : i ≤ 0) (hr1 : 0 ≤ r) (hr2 : r ≤ Δ) : |Ep i r| ≤ Ep 0 Δ := by
  rw [abs_of_nonneg (Ep_r_nonneg hr1)]
  have ieq1 := Ep_i_monotoneOn hr1 hi Set.right_mem_Iic hi
  have ieq2 : Ep 0 r ≤ Ep 0 Δ := Ep_r_monotoneOn hr1 (by linarith : 0 ≤ Δ) hr2
  linarith
```

Error of first-order Taylor approximation of Φ

Y-Axis : Error of first-order Taylor approximation at x



$$E_\Delta(\mathbf{i}) = \Phi(\mathbf{i}-\Delta) - \Phi(\mathbf{i}) + \Delta\Phi'(\mathbf{i})$$

Error of first-order Taylor approximation of Φ^-

Lemma 2. *Suppose that* $\frac{1}{\Delta} \in \mathbb{N}$ *(e.g.,* $\Delta = 2^{-k}$ *for some natural number* k*). Then for all* $x \in (-\infty, -1]$*,*

$$|\Phi^-(x) - \hat{\Phi}^-_T(x)| \leq -E^-_\Delta(-1) = (\ln 2)\Delta^2 + O(\Delta^3).$$

```
lemma Em_bound' (hi₀ : i₀ < 0) (hi : i ≤ i₀) (hr1 : 0 ≤ r) (hr2 : r ≤ Δ) : |Em i r| ≤ Em i₀ Δ := by
  have hi0 : i < 0 := by linarith
  rw [abs_of_nonneg (Em_r_nonneg hi0 hr1)]
  have ieq1 := Em_i_monotoneOn hr1 (by simp only [Set.mem_Iio]; linarith) (by simp only [Set.mem_Iio, hi₀]) hi
  have ieq2 : Em i₀ r ≤ Em i₀ Δ := Em_r_monotoneOn hi₀ hr1 (by linarith : 0 ≤ Δ) hr2
  linarith
```

Total error bound of first-order Taylor approximation

Theorem 1. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration. Let $E_M^+ = E_\Delta^+(0)$, and $E_M^- = -E_\Delta^-(-1)$. Then*

$$|\Phi(x) - \tilde{\Phi}_T(x)| < E_M + (2 + \Delta)\epsilon.$$

```
/- A model of fixed-point arithmetic -/
structure FixedPoint where
  ε : ℝ
  rnd : ℝ → ℝ
  hrnd : ∀ x, |x - rnd x| ≤ ε
  rnd_mono : Monotone rnd
  rnd_1 : rnd 1 = 1
  rnd_0 : rnd 0 = 0
```

Approximation Error

Fixed-point rounding Error

$$\tilde{\Phi}_T(x) = \overline{\Phi}(\mathbf{i}) - \mathrm{rnd}(\mathbf{r}\overline{\Phi'}(\mathbf{i}))$$

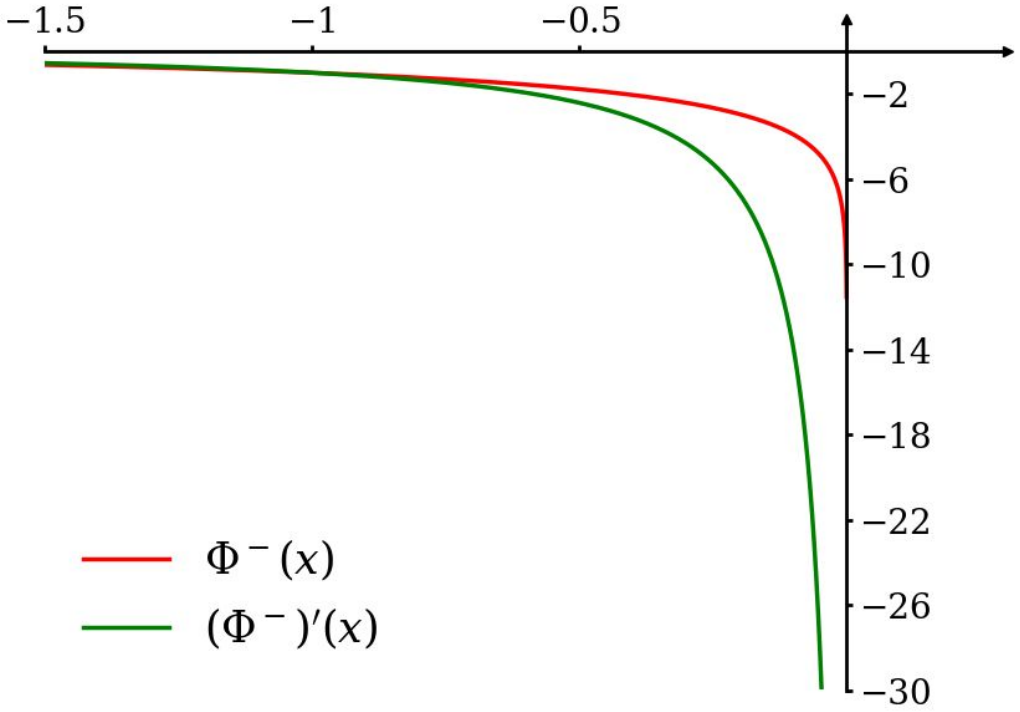
Total error bound of first-order Taylor approximation

Theorem 1. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration. Let $E_M^+ = E_\Delta^+(0)$, and $E_M^- = -E_\Delta^-(-1)$. Then*

$$|\Phi(x) - \tilde{\Phi}_T(x)| < E_M + (2 + \Delta)\epsilon.$$

```
theorem ΦTp_err_bound (fix : FixedPoint) {x Δ : ℝ} (hd : 0 < Δ) (hx : x ≤ 0) :
    |Φp x - ΦTp_fix fix Δ x| < Ep 0 Δ + (2 + Δ) * fix.ε := by
  have eq : Φp x - ΦTp_fix fix Δ x = Ep_fix fix (Iₓ Δ x) (Rₓ Δ x) := by
    unfold ΦTp_fix Ep_fix; rw [i_sub_r_eq_x]; ring_nf
  rw [eq]
  apply Ep_fix_bound fix _ (rx_nonneg hd x) (rx_lt_delta hd x)
  rw [← x_neg_iff_ix_neg hd]; exact hx
```

Subtraction requires special treatment for -1 < x < 0



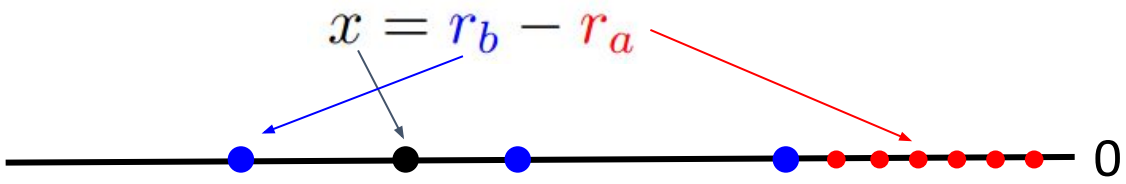
Computing $\Phi^-(x)$ when x is close to 0 by interpolation is quite inaccurate

Coleman's co-transformation

- Assume that x in $(-1, 0)$. Fix a small value $\Delta_a > 0$. Write $x = r_b - r_a$ with $r_b = i\,\Delta_a$ for some integer $i < 0$ such that $r_b < x$ and $-\Delta_a \leq r_a < 0$.

$$r_b = \left(\left\lceil \frac{x}{\Delta_a} \right\rceil - 1 \right) \Delta_a$$

$$r_a = r_b - x$$

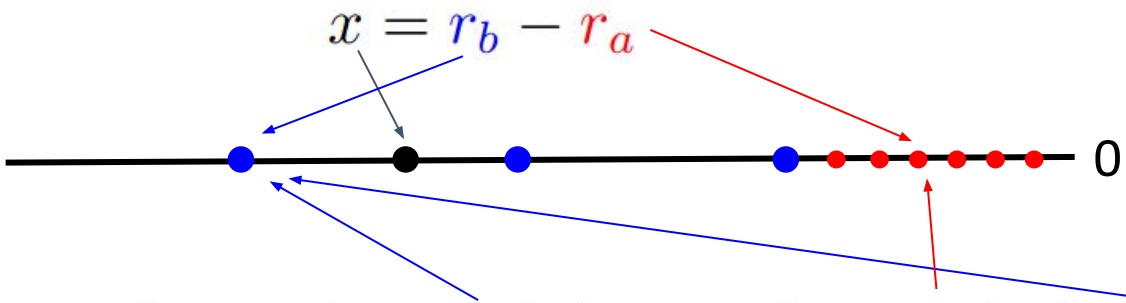


Coleman's co-transformation

- Assume that x in $(-1, 0)$. Fix a small value $\Delta_a > 0$. Write $x = r_b - r_a$ with $r_b = i\,\Delta_a$ for some integer $i < 0$ such that $r_b < x$ and $-\Delta_a \leq r_a < 0$.

$$r_b = \left(\left\lceil \frac{x}{\Delta_a} \right\rceil - 1 \right) \Delta_a$$
$$r_a = r_b - x$$

- Compute Φ^- as $\Phi^-(x) = \Phi^-(r_b - r_a) = \Phi^-(r_b) + \Phi^-(x + \Phi^-(r_a) - \Phi^-(r_b))$.

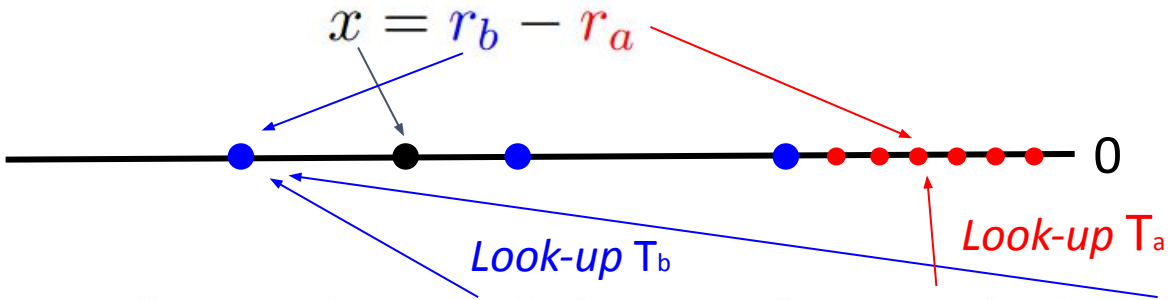


Coleman's co-transformation

- Assume that x in $(-1, 0)$. Fix a small value $\Delta_a > 0$. Write $x = r_b - r_a$ with $r_b = i\,\Delta_a$ for some integer $i < 0$ such that $r_b < x$ and $-\Delta_a \le r_a < 0$.

$$r_b = \left(\left\lceil \frac{x}{\Delta_a} \right\rceil - 1 \right) \Delta_a$$
$$r_a = r_b - x$$

- Compute Φ^- as $\Phi^-(x) = \Phi^-(r_b - r_a) = \Phi^-(r_b) + \Phi^-(x + \Phi^-(r_a) - \Phi^-(r_b))$.
- Requires 2 tables: T_a which contains all values of fixed-point values x in $[-\Delta_a, 0)$ and T_b which contains all integer multiples $r_b = i\,\Delta_a$ in $(-1, \Delta_a)$.

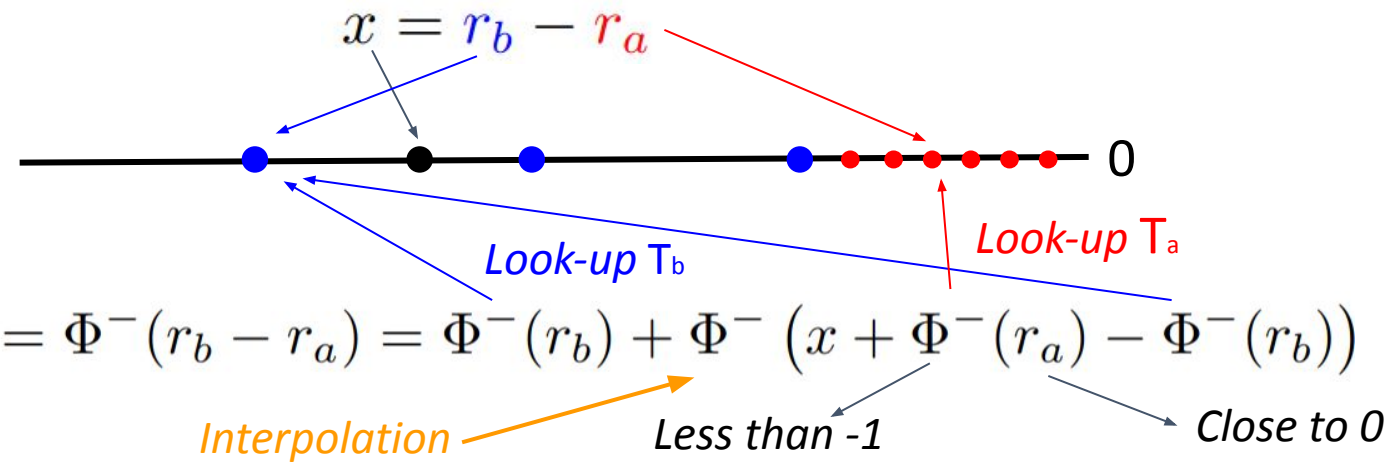


$$\Phi^-(x) = \Phi^-(r_b - r_a) = \Phi^-(r_b) + \Phi^-\left(x + \Phi^-(r_a) - \Phi^-(r_b)\right)$$

Coleman's co-transformation

For x in (-1, 0) consider **2** cases:

1) x in $[-\Delta_a, 0)$. Take the value of $\Phi^-(x)$ directly from table T_a.
2) x in $(-1, -\Delta_a)$. Compute r_b and r_a, $\Phi^-(r_b)$ is taken from table T_b, $\Phi^-(r_a)$ is taken from table T_a, and $x + \Phi^-(r_a) - \Phi^-(r_b) < -1$ is computed with other approximation techniques (e.g., Taylor approximation).



Coleman's co-transformation

- Problem: Too many values in tables T_a and T_b when ε is small.

  Example: $\varepsilon = 2^{-32}$, $\Delta_a = 2^{-16}$.
  Tables T_a and T_b have approximately 2^{16} values each.

Coleman's co-transformation

- Problem: Too many values in tables T_a and T_b when ε is small.

  Example: $\varepsilon = 2^{-32}$, $\Delta_a = 2^{-16}$.
  Tables T_a and T_b have approximately 2^{16} values each.

- Solution: Define $\Delta_b > \Delta_a$ and apply co-transformation technique twice.

  Compute r_c such that $x = r_c - r_{ab}$ with $r_c = j\,\Delta_b$ with $j < 0$ and $r_c < x$, $-\Delta_b \leq r_{ab} < 0$. Apply co-transformation to r_{ab} and get $r_{ab} = r_b - r_a$, $r_b < r_{ab}$, $-\Delta_a \leq r_a < 0$.

  Requires one more table T_c .
  Tables T_a, T_b and T_c have approximately 2^{11} values each.

Coleman's co-transformation

To compute $\Phi^{-}(x)$ consider **3** cases:

1) $-\Delta_a \le x < 0$: $\Phi^{-}(x)$ is taken from T_a.
2) x in $[-\Delta_b, -\Delta_a)$: $\Phi^{-}(x)$ is computed as $\Phi^{-}(r_b - r_a) = \Phi^{-}(r_b) + \Phi^{-}(x + \Phi^{-}(r_a) - \Phi^{-}(r_b))$.
3) x in $(-1, -\Delta_b)$: $\Phi^{-}(x)$ is computed as
   $\Phi^{-}(x) = \Phi^{-}(r_c - r_{ab}) = \Phi^{-}(r_c) + \Phi^{-}(x + \Phi^{-}(r_{ab}) - \Phi^{-}(r_c))$
   Here $\Phi^{-}(r_c)$ is taken from table T_c and $\Phi^{-}(r_{ab}) = \Phi^{-}(r_b - r_a)$ is computed as in case 2.

Coleman's co-transformation

To compute $\Phi^{-}(x)$ consider ~~3~~ **4** cases:

1) $-\Delta_a \leq x < 0$: $\Phi^{-}(x)$ is taken from T_a.
2) x in $[-\Delta_b, -\Delta_a)$: $\Phi^{-}(x)$ is computed as $\Phi^{-}(r_b - r_a) = \Phi^{-}(r_b) + \Phi^{-}(x + \Phi^{-}(r_a) - \Phi^{-}(r_b))$.
3) x in $(-1, -\Delta_b)$: $\Phi^{-}(x)$ is computed as
   $\Phi^{-}(x) = \Phi^{-}(r_c - r_{ab}) = \Phi^{-}(r_c) + \Phi^{-}(x + \Phi^{-}(r_{ab}) - \Phi^{-}(r_c))$
   Here $\Phi^{-}(r_c)$ is taken from table T_c and $\Phi^{-}(r_{ab}) = \Phi^{-}(r_b - r_a)$ is computed as in case 2.
4) x in $(-1, -\Delta_b)$ and $r_{ab} < -\Delta_a$. In this case, we cannot compute $\Phi^{-}(r_{ab})$ as in case 2 because $r_{ab} + \Phi^{-}(r_a) - \Phi^{-}(r_b)$ could be > -1. Take the value of $\Phi^{-}(r_{ab})$ directly from T_a.

Formalization revealed this additional case which was missing in our original informal proof.

Co-transformation error bound

Co-transformation formula with rounding:

$$\Phi^{-}(x) = \mathrm{rnd}\left(\Phi^{-}(r_b)\right) + \hat{\Phi}^{-}\left(x + \mathrm{rnd}\left(\Phi^{-}(r_a)\right) - \mathrm{rnd}\left(\Phi^{-}(r_b)\right)\right)$$

An approximation of $\Phi^{-}(x)$ for $x \leq -1$

A rounded value of $\Phi^{-}(x)$ stored in table T_b

A rounded value of $\Phi^{-}(x)$ stored in table T_a

Co-transformation error bound

Theorem 2. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration and E_{Φ^-} be the error bound of interpolating of Φ^- in the range $(-\infty, 1]$. Assume also that $\Delta_a \geq 4\epsilon$ and $\Delta_b \geq 8\epsilon + 2E_{\Phi^-}$. The error bound of computing $\Phi^-(x)$ when $x \in (-1, 0)$ using the co-transformation technique is:*

$$\Phi^-(-1 - E_{k_2}) - \Phi^-(-1) + E_{\Phi^-} + \epsilon$$

where

$$E_{k_2} = \Phi^-(-1 - 2\epsilon) - \Phi^-(-1) + E_{\Phi^-} + 2\epsilon.$$

```
theorem cotransformation_err_bound (fix : FixedPoint)
    (Φe : FunApprox Φm (Set.Iic (-1)))
    (ha : 0 < Δa) (hb : 0 < Δb)
    (hΔa : 4 * fix.ε ≤ Δa)                    /- Δa should be large enough -/
    (hΔb : 8 * fix.ε + 2 * Φe.err ≤ Δb) : /- Δb should be large enough -/
    let Ek2 := 2 * fix.ε + Φm (-1 - 2 * fix.ε) - Φm (-1) + Φe.err
    |Φm x - Cotrans fix Φe Δa Δb x| ≤ fix.ε + Φm (-1 - Ek2) - Φm (-1) + Φe.err
```

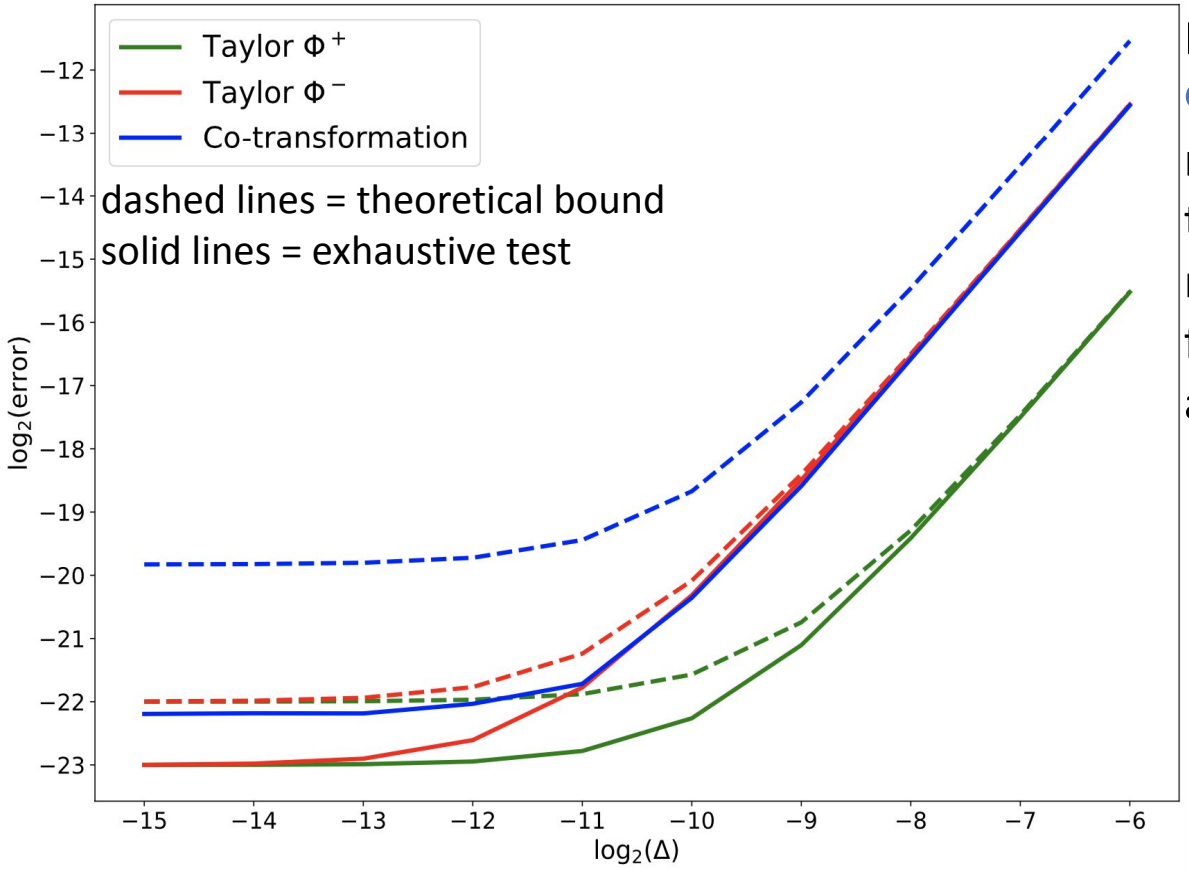
Co-transformation error bound (simplified)

Theorem 2. *Let ϵ be the machine-epsilon of the fixed-point representation of the LNS under consideration and E_{Φ^-} be the error bound of interpolating of Φ^- in the range $(-\infty, 1]$. Assume also that $\Delta_a \geq 4\epsilon$ and $\Delta_b \geq 8\epsilon + 2E_{\Phi^-}$. The error bound of computing $\Phi^-(x)$ when $x \in (-1, 0)$ using the co-transformation technique is:*

$$2E_{\Phi^-} + 5\epsilon$$

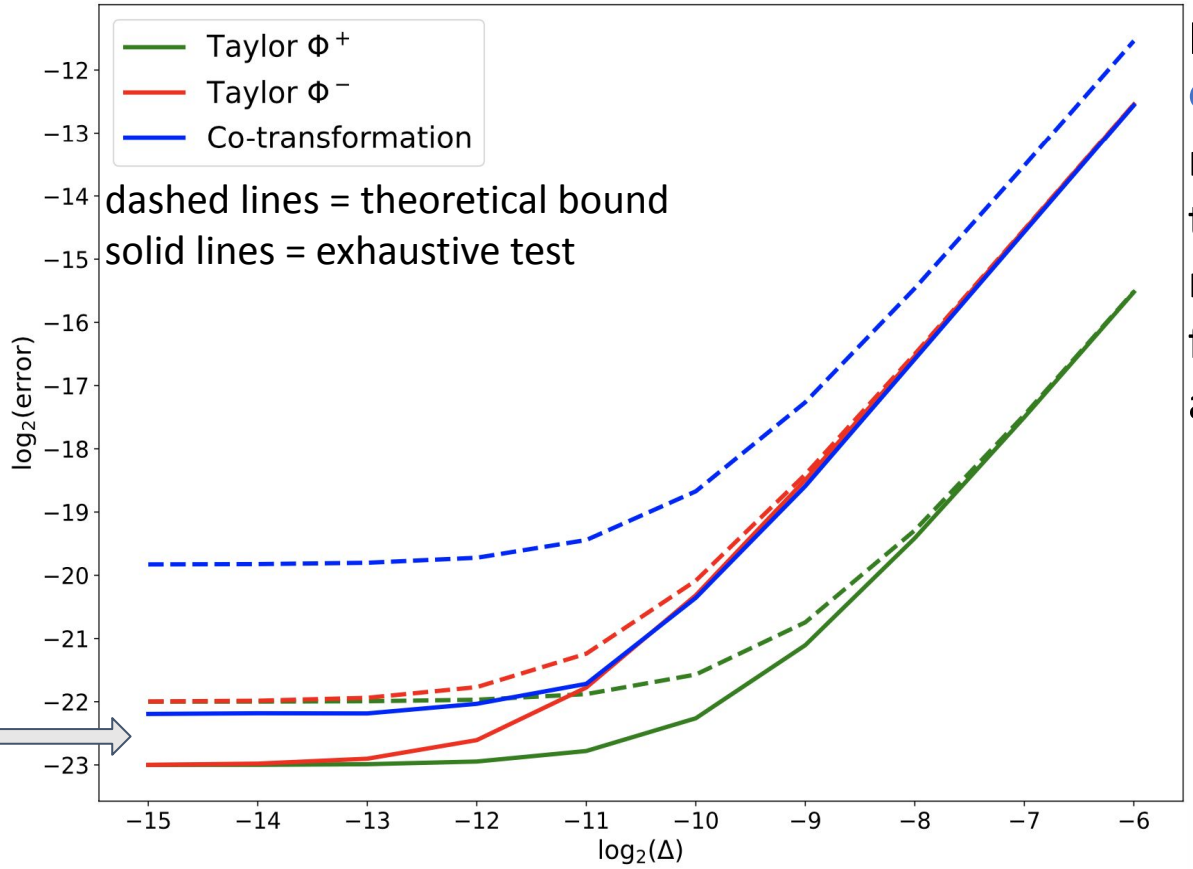
```
theorem cotransformation_err_bound' (fix : FixedPoint)
    (Φe : FunApprox Φm (Set.Iic (-1)))
    (ha : 0 < Δa) (hb : 0 < Δb)
    (hΔa : 4 * fix.ε ≤ Δa)                  /- Δa should be large enough -/
    (hΔb : 8 * fix.ε + 2 * Φe.err ≤ Δb) : /- Δb should be large enough -/
    |Φm x - Cotrans fix Φe Δa Δb x| ≤ 5 * fix.ε + 2 * Φe.err := by
```

Exhaustive verification



For given Δ, Φ^+ two bits more accurate than Φ^-: take more memory for same accuracy

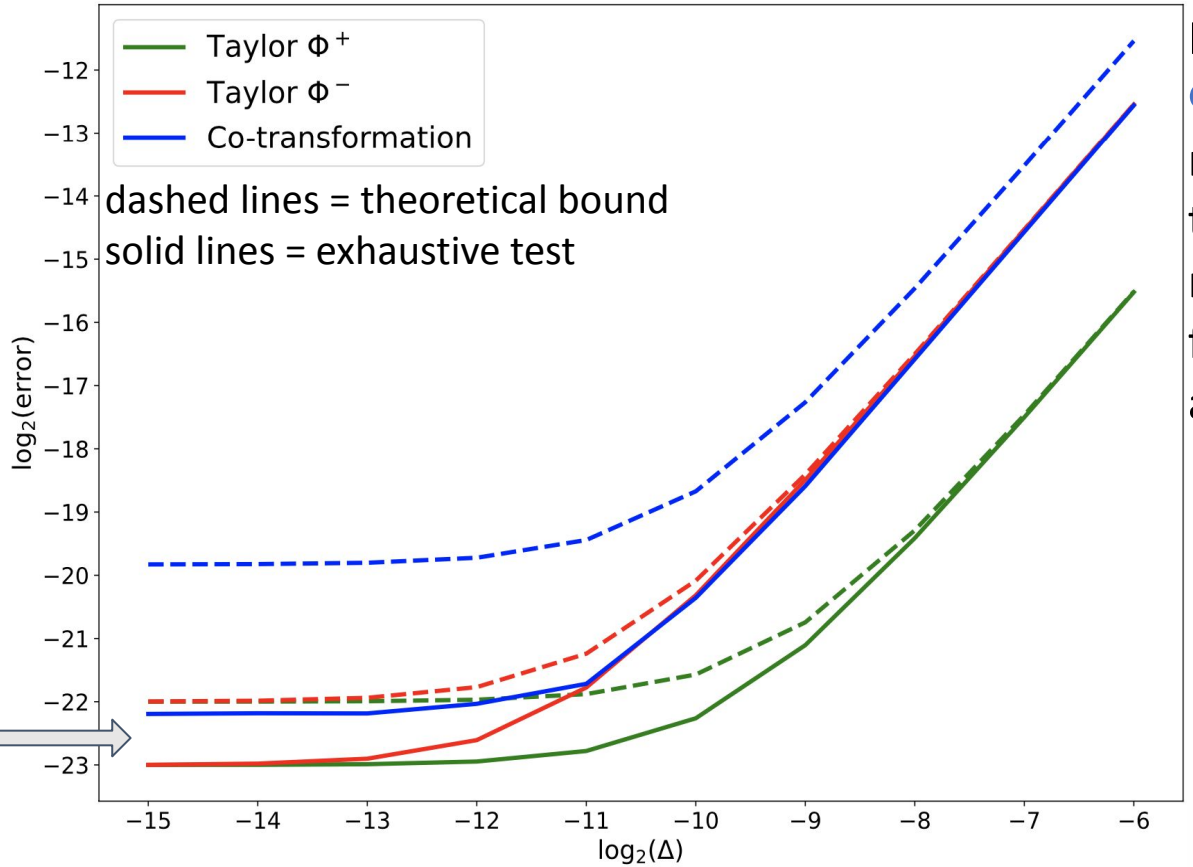
Exhaustive verification



For given Δ, Φ^+ two bits more accurate than Φ^-: take more memory for same accuracy

$\varepsilon = 2^{23}$ which explains the "hockey stick" shape

Exhaustive verification



For given Δ, Φ^+ two bits more accurate than Φ^-: take more memory for same accuracy

$\varepsilon = 2^{23}$ which explains the "hockey stick" shape

For $\Delta > \varepsilon$,

$\log_2(\text{err}) = 2\log_2(\Delta) = \log_2(\Delta^2)$ + constant: known quadratic err of linear interpolation

Results

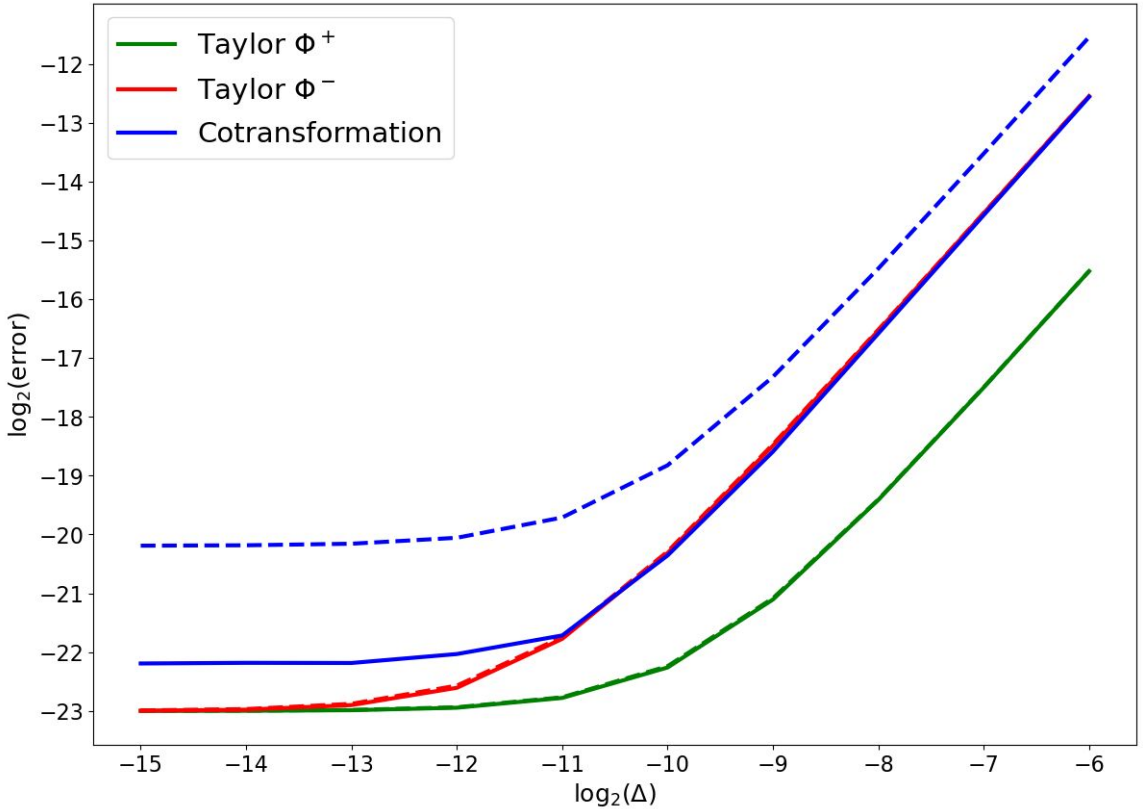
- Formally verified error bounds for first order Taylor approximations of Φ^+ and Φ^-.
- Formally verified error bounds for co-transformation techniques for computing Φ^- near zero.
- A library of Lean 4 definitions and lemmas for error analysis of LNS:

  `https://github.com/soarlab/RigorousErrorLNS`

Future work

- Developing additional Lean 4 tactics and error-analysis lemmas.
- Tighter bounds for co-transformation.
- Error analysis for LNS variations:
  - Error Correction (EC)/quadratic
  - Other co-transformation [Arnold 1997]
    - unlike [Coleman 2000] does not use less accurate Φ^- interpolation
  - Bases other than two
  - Guard bits
  - Varying Δs
  - Table-less LNS design

Exhaustive verification with directed rounding



Metaprogramming - developing tactics

```
elab "get_deriv" t:term loc:(deriv_at)? : tactic => do
  let realType := Expr.const ``Real []
  let realSetType ← mkAppM ``Set #[realType]
  withMainContext do
    let t ← Tactic.elabTerm t (some $ .forallE `x realType realType .default)
    let expr ← lambdaTelescope t (fun args e => toRExpr args e)
    let hyp := ← match loc with
                 | some loc => do
                   match loc with
                   | `(deriv_at| at $x) => do
                     let x ← Tactic.elabTerm x (some realType)
                     mkAppM ``expr_hasDerivAt #[expr, x]
                   | `(deriv_at| within $s) => do
                     let s ← Tactic.elabTerm s (some realSetType)
                     mkAppM ``expr_diff_deriv_on #[expr, s]
                   | _ => throwUnsupportedSyntax
                 | _ => mkAppM ``expr_diff_deriv #[expr]
    let hypType ← inferType hyp
    let (args, _, concl) ← forallMetaTelescope hypType
    liftMetaTactic fun mvarId => do
      let newId ← mvarId.assert `h concl (← mkAppM' hyp args)
      let (_, newId) ← newId.intro1P
      return args.toList.map Expr.mvarId! ++ [newId]
```