A CLAMP FUNCTION WITH ENHANCED DIFFERENTIABILITY

EuroAD 28 @ CERN

Cord Bleibaum[1], Christian Sattler[1,2]

[1]DLR, Institute of Future Fuels [2]RWTH Aachen

Introduction – About Me

Hello,

I‘m Cord, let me introduce myself shortly:

- PhD Student at DLR Institute of Future Fuels
- Specializing on differentiable ray tracing for solar optics applications
- Working mainly in Jülich, next to the Synlight solar simulator
- And also I love optimizing things, from code to functions

Introduction – Where do I need this?

Synlight Solar Simulator Jülich

I’m building a differentiable raytracer for solar research, simulating and developing optimization software for facilities like Synlight [8]

This needs many different physical constraints, and thus I need a pattern to model these better in a differentiable way

Trying to keep the code as close as I can to what I would do for a non-differentiable software

Introduction – The classic clamp function

Constrains a value between two limits, stopping the value at the edges (for example in OpenGL [6]):

$$\mathrm{clamp}(x, a, b) = \begin{cases} a & \text{if } x < a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x > b \end{cases}$$

Important properties:
- Constrains values to the range
- Rising in the range between a and b
- Value can reach the limit values

Problem:
- Derivative outside $x \in [a, b]$ equals zero
- No useful partial derivatives towards a and b

Solution: building a modified function that behaves like a clamp as far as feasible

First idea – Trigonometric Clamp

A naive improvement would be to use a trigonometric function:

$$clamp_{sin}(x, a, b) = a + (b - a)\left(\frac{1}{2}\sin(x) + \frac{1}{2}\right)$$

Why not?

Trigonometric functions are known to be slow on CPUs, and it can be done faster ☺

(benchmark comparison will follow later)

⇒ Next: a better solution

Modified Clamp (1)

We are thus looking to build a function $\mathrm{clamp}_{\mathrm{mod}}(x, a, b)$ with the following properties:

- No regions where the derivative stays at zero (isolated critical points are acceptable in our use case)
- Constrains output values to the desired region
- Value rising between the limit values
- Value can reach the limits of a and b
- We need a way to set a specific result value ⇒ Needs to be invertible for $x \in [a, b]$

⇒ A periodic function would work for this, with a period of $b - a$

To do this, we're combining two components, a polynomial and a modulo

details on the next slides

Modified Clamp (2)

To combine with a modulo into a periodic function, and be differentiable, we are looking for a function, that could have these properties:

- $f(a) = a$
- $f(b) = b$
- $f'(a) = 0$ and $f'(b) = 0$, so that we can stitch the ends together

There is a already function that solves this equation for $a = 0$ and $b = 1$, a smoothstep function [2]:

$$S_1(x) = -2x^3 + 3x^2$$

This can then be stretched to $[a, b]$ using $a + (b - a)\, S_1(x)$

We're only scaling the output value, not the input value, as this is still invertible in $[a, b]$

Modified Clamp (3)

Last component needed is a suitable modulo-based function to combine with, one that is suitable here is a sawtooth-wave with a period of $p = 2$, assuming modulo implemented by floor division [3]:

$$x(t) = \left| \frac{t}{2} - \left\lfloor \frac{t}{2} + \frac{1}{2} \right\rfloor \right| = |x \bmod 2 \ - 1|$$

Then combining these components we get the modified clamp:

$$\mathrm{clamp}_{\mathrm{mod}}(x_{inv}, a, b) = a + (b - a)(3(x_{inv} \bmod 2 \ - 1)^2 - 2|x_{inv} \bmod 2 \ - 1|^3)$$

This now restricts values to a range of $[a, b]$ similar to a usual clamp function. A notable change is, that the parameter x needs to be encoded into x_{inv} as it is not directly output anymore, and thus the need for a partial inverse.

Modified Clamp - Inverse

To set the result to a specific value, we need a suitable inverse. For the smoothstep function there is already a known inverse (a solution of the cubic equation [4]) in the region of $\left[-\frac{1}{2}, \frac{3}{2}\right]$, which is sufficient:

$$S_1^{-1}(x) = \frac{1}{2}\sin\left(\frac{\sin^{-1}(1-2x)}{3}\right)$$

This can then again be stretched towards the limits of $[a, b]$:

$$clamp_{mod}^{-1}(x, a, b) = S_1^{-1}\left(\frac{x}{b-a} - a\right) = \frac{1}{2}\sin\left(\frac{\sin^{-1}\left(1-2\left(\frac{x}{b-a}-a\right)\right)}{3}\right)$$

Due to the periodicity of $\mathrm{clamp}_{\mathrm{mod}}$ this is not the only solution, there are, for example, further solutions adding $2n(b-a),\ \ n \in \mathbb{Z}$

This is usually only executed once to set a specific starting value. We can then move x_{inv} directly in an optimizer, without the result value moving out of range.

Modified Clamp - Derivatives

We can now differentiate this towards the boundaries, which wasn't possible with the classic version:

$$\frac{\partial \mathrm{clamp}_{\mathrm{mod}}}{\partial a} = 2|x \bmod 2 \ - 1|^3 - 3(x \bmod 2 \ - 1)^2 + 1$$

$$\frac{\partial \mathrm{clamp}_{\mathrm{mod}}}{\partial b} = 3(x \bmod 2 \ - 1)^2 - 2|x \bmod 2 \ - 1|^3$$

This can also be differentiated towards x:

$$\frac{\partial \mathrm{clamp}_{\mathrm{mod}}}{\partial x} = \begin{cases} 6(a \ - \ b)((x \bmod 2)2 \ - \ 3(x \bmod 2) \ + \ 2) & if \ x \bmod 2 > 1 \\ -6(a \ - \ b)((x \bmod 2)2 \ - \ x \bmod 2) & if \ x \bmod 2 < 1 \\ 0 & if \ x \bmod 2 = 1 \end{cases}$$

The 0 at $x \bmod 2 = 1$ is due to both sides having the same value, and derivative of 0 there (written as a case here, as this is the way we currently implement this)

Optimization Example

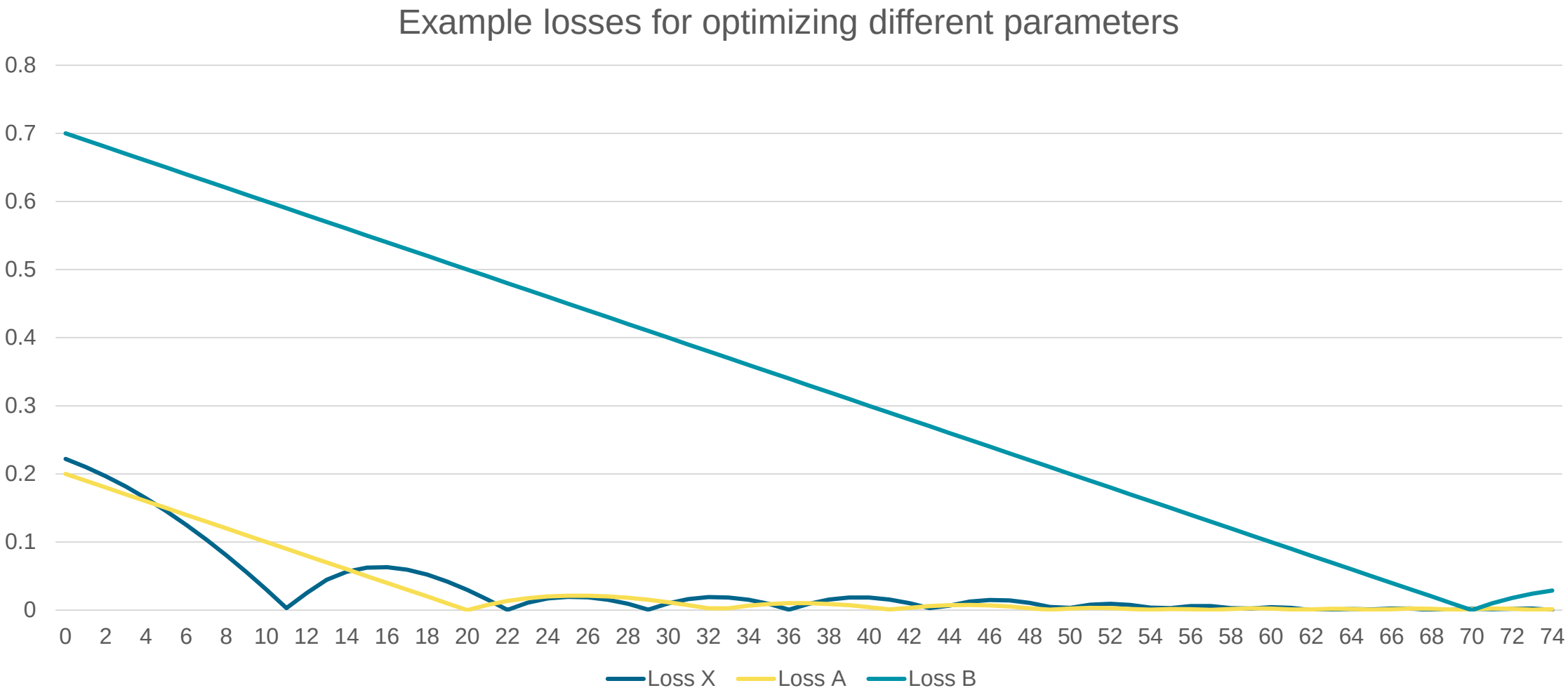
This can be plugged into, for example, PyTorch[1], for a short example of learning all three value, for example:

- Stochastic gradient descent [5] based optimizer (in this case Adam [9]), $\eta = 0.02$ (other parameters at PyTorch defaults)
- Trying to adjust the free parameter, such that: $clamp_{mod}(x, a, b) = 0.75$
- Absolute Error Loss: $|x - 0.75|$
- 75 iterations
- Starting value for variable: 0.1, other values (if they are not the variable in that case):

$$x = 0.5, a = 0, b = 1$$

Three cases, one for x, a, and b (results on next slide)

Optimization Example



Performance – Setup

To test the performance, we test both using Google Benchmark. System setup:

- Intel Core i9 12900k @ 4.9 GHz, disabled frequency scaling
- 64gb DDR5 @ 4800 MHz
- Linux 6.17 Kernel
- Code compiled with GCC 15.2, flags: `-O3 -march=native`
- Simplified to $a = 0$ and $b = 1$, as this part is the same for both cases

Benchmark code (clamp implementations on the next slides):

```
static void BM_Clamp(benchmark::State& state) {
    volatile double x = 0.5, out = 0;
    for (auto _ : state){
        out = clamp(x);
        for (int j = 0; j < 32; ++j) out = clamp_dx(x);
    }
}
BENCHMARK(BM_Clamp);
```

Performance – Implementations(1)

```
inline double sin_clamp(double x) { return 0.5 * std::sin(x) + 0.5; }
inline double sin_clamp_dx(double x) { return 0.5 * std::cos(x); }
```

Performance – Implementations(2)

```
inline double fast_mod(double x, int y) {  return x-((int)(x/y))*y; }

inline double mod_clamp(double x) {
    const double modx1 = fast_mod(x, 2) - 1.0;
    const double pow2 = modx1 * modx1;
    const double pow3 = modx1 * modx1 * std::fabs(modx1);
    return 3.0*pow2 - 2.0*pow3;
}

inline double mod_clamp_dx(double x) {
    const double modx = fast_mod(x, 2);
    const double pow2 = modx * modx;
    return modx > 1 ? -6.0*(pow2 - 3.0*modx + 2.0) : 6.0*(pow2 - modx);
}
```

Performance – Results

As seen below, the modulo-based version is round 3x faster

Version	CPU time per iteration
Modulo	31 ns
Trigonometric	102 ns

This is due to trigonometric instructions being very slow on CPUs, with
a latency of around 60-120 cycles, versus for example around 4 cycles for a multiplication,
and also vastly different throughputs (on Intel IceLake) [7]

Summary&Questions

$$\text{clamp}(x, a, b) = \begin{cases} a & \text{if } x < a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x > b \end{cases}$$

$$\text{clamp}_{\text{mod}}(x_{inv}, a, b) = a + (b - a)(3(x_{inv} \bmod 2\ - 1)^2 - 2|x_{inv} \bmod 2\ - 1|^3)$$

Built a clamp function with better differentiability
Faster than first naïve attempts

interesting further directions

Can we be even faster?
More patterns for differentiable programming?
And GPUs?

Also I'd like to thank my supervisors, Dmitrij Laaber and Daniel Maldonado-Quinto, and my department leader Dennis Thomey, for their advice and support

Feel free to ask any questions ☺

References

[1] A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” [Online]. Available: https://arxiv.org/abs/1912.01703

[2] “HLSL: smoothstep.” [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-smoothstep?redirectedfrom=MSDN

[3] D. E. Knuth, The art of computer programming, volume 1 (3rd ed.): fundamental algorithms. USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[4] I. Quilez, “inverse smoothstep.” [Online]. Available: https://iquilezles.org/articles/ismoothstep/

[5] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” Psychological review, pp. 386–408, 1958, [Online]. Available: https://api.semanticscholar.org/CorpusID:12781225

[6] https://registry.khronos.org/OpenGL-Refpages/gl4/

[7] Fog, Agner. Instruction tables. Technical University of Denmark, various years (e.g., last updated 2016 or 2025), https://www.agner.org/optimize/instruction_tables.pdf

[8] Wieghardt, Kai & Funken, Karl-Heinz & Dibowski, Gerd & Hoffschmidt, Bernhard & Laaber, Dmitrij & Hilger, Patrick & Eßer, Kai-Peter. (2016). SynLight – the world’s largest artificial sun. 1734. 030038. 10.1063/1.4949090.

[9] Kingma, Diederik P. and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” CoRR abs/1412.6980 (2014): n. pag.

Imprint

Topic:	A Clamp function with enhanced differentiability
Date:	2025-12-08
Author:	Cord Bleibaum
Institute:	Future Fuels
Image sources:	All images “DLR (CC BY-NC-ND 3.0)” unless otherwise stated