Kotlin∇: A Shape-safe DSL for Differentiable Programming

Breandan Considine, Michalis Famelis, Liam Paull

Main Idea

- We create an embedded DSL for differentiable programming in Kotlin
- Supports shape checking and inference for multi-dimensional arrays
- Implementable in any language with first-class functions and generics

Shape errors

There are three broad strategies for handling shape errors:

- Perform type coercion by implicitly broadcasting or reshaping arrays
- Raise a runtime error (e.g. `tf.errors.InvalidArgumentError`)
- Do not allow programs which can result in a shape error to compile

In Kotlin∇, we prefer the last strategy. Consider the following scenario:

```
a = np.array('0 1 2 3 4 5')
b = np.array('6 7 8 9')
c = a + b
```

```
val a = Vec(0, 1, 2, 3, 4, 5)
val b = Vec(6, 7, 8, 9)
val c = a + b
```

Similarly, when the inner dimensions of two matrices do not match:

```
a = np.matrix('0 1 2 ; 3 4 5')
b = np.matrix('6 7; 8 9')
c = a @ b
```

```
val a = Mat2x3(0, 1, 2, 3, 4, 5)
val b = Mat2x2(6, 7, 8, 9)
val c = a * b
```

We can detect the presence and location of the error within the editor.

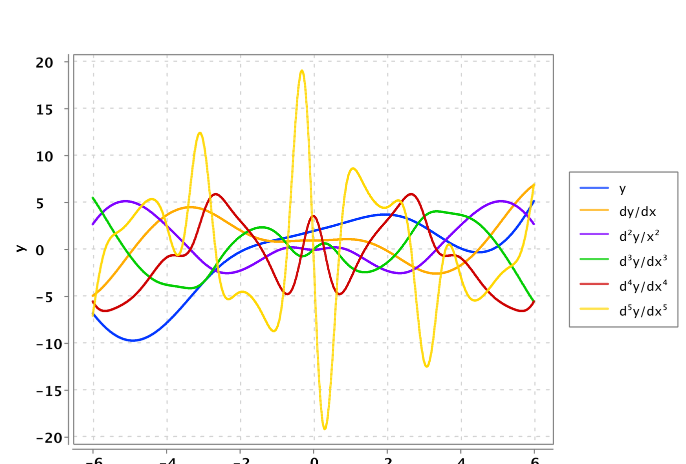
Type system

Math	Infix	Prefix	Postfix	Operator Type Signature
$\mathbf{A}(\mathbf{B}), \mathbf{A} \circ \mathbf{B}$	`a(b)`			(a: $\mathbb{R}^\tau \to \mathbb{R}^\pi$, b: $\mathbb{R}^\lambda \to \mathbb{R}^\tau$) $\to (\mathbb{R}^\lambda \to \mathbb{R}^\pi)$
$\mathbf{A} \pm \mathbf{B}$	`a + b`, `a - b`	`plus(a, b)`		(a: $\mathbb{R}^\tau \to \mathbb{R}^\pi$, b: $\mathbb{R}^\lambda \to \mathbb{R}^\pi$) $\to (\mathbb{R}^? \to \mathbb{R}^\pi)$
$\mathbf{AB}$	`a * b`, `a.times(b)`	`times(a, b)`		(a: $\mathbb{R}^\tau \to \mathbb{R}^{m\times n}$, b: $\mathbb{R}^\lambda \to \mathbb{R}^{n\times p}$) $\to (\mathbb{R}^? \to \mathbb{R}^{m\times p})$
$\frac{\mathbf{A}}{\mathbf{B}}$ $\mathbf{AB}^{-1}$	`a / b` `a.div(b)`	`div(a, b)`		(a: $\mathbb{R}^\tau \to \mathbb{R}^{m\times n}$, b: $\mathbb{R}^\lambda \to \mathbb{R}^{p\times n}$) $\to (\mathbb{R}^? \to \mathbb{R}^{m\times p})$
$\log_b \mathbf{A}$	`a.log(b)`	`log(a, b)`		(a: $\mathbb{R}^\tau \to \mathbb{R}^{m\times m}$, b: $\mathbb{R}^\lambda \to \mathbb{R}^{m\times m}$) $\to (\mathbb{R}^? \to \mathbb{R})$
$\mathbf{A}^b$	`a.pow(b)`	`pow(a, b)`		(a: $\mathbb{R}^\tau \to \mathbb{R}^{m\times m}$, b: $\mathbb{R}^\lambda \to \mathbb{R}$) $\to (\mathbb{R}^? \to \mathbb{R}^{m\times m})$
∇a		`grad(a)`	`a.grad()`	(a: $C(\mathbb{R}^\tau \to \mathbb{R})$) $\to (\mathbb{R}^\tau \to \mathbb{R}^\tau)$
$\nabla_{\mathbf{B}} a$	`a.d(b)` `a.grad(b)`	`grad(a, b)`		(a: $C(\mathbb{R}^\tau \to \mathbb{R})$, b: $C(\mathbb{R}^\lambda \to \mathbb{R}^n)$) $\to (\mathbb{R}^? \to \mathbb{R}^n)$

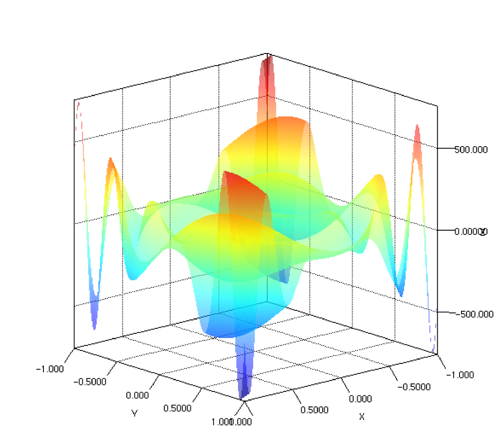
Visualization

Kotlin∇ is capable of computing arbitrarily high order derivatives.

```
val y = sin(sin(sin(x))) / x
val z = y + sin(x) * x + cos(x) + x
val d1 = d(z) / d(x)
val d2 = d(d1) / d(x)
val d3 = d(d2) / d(x)
val d4 = d(d3) / d(x)
val d5 = d(d4) / d(x)
plot2D(-6..6, z, d1, d2, d3, d4, d5)
```

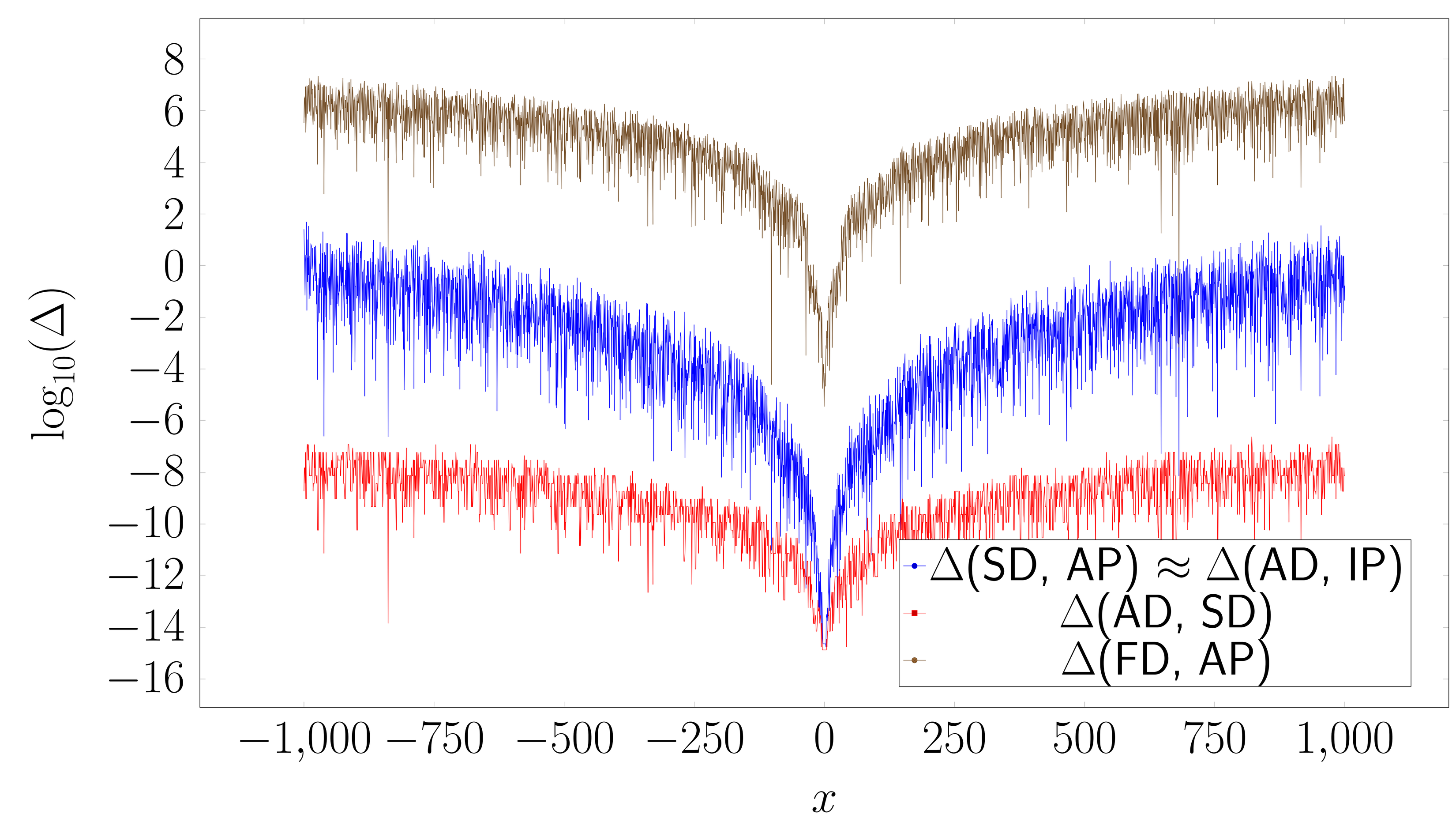


```
val Z = 10 * (x * x + pow(y, 2))
val sinZ = sin(Z)
val sinZ_10 = sinZ / 10
val dZ_dx = d(sinZ_10) / d(x)
val d2Z_dxdy = d(dZ_dx) / d(y)
val d3Z_d2xdy = grad(d2Z_dxdy)[x]
plot3D(-1..1, d3Z_d2xdy)
```

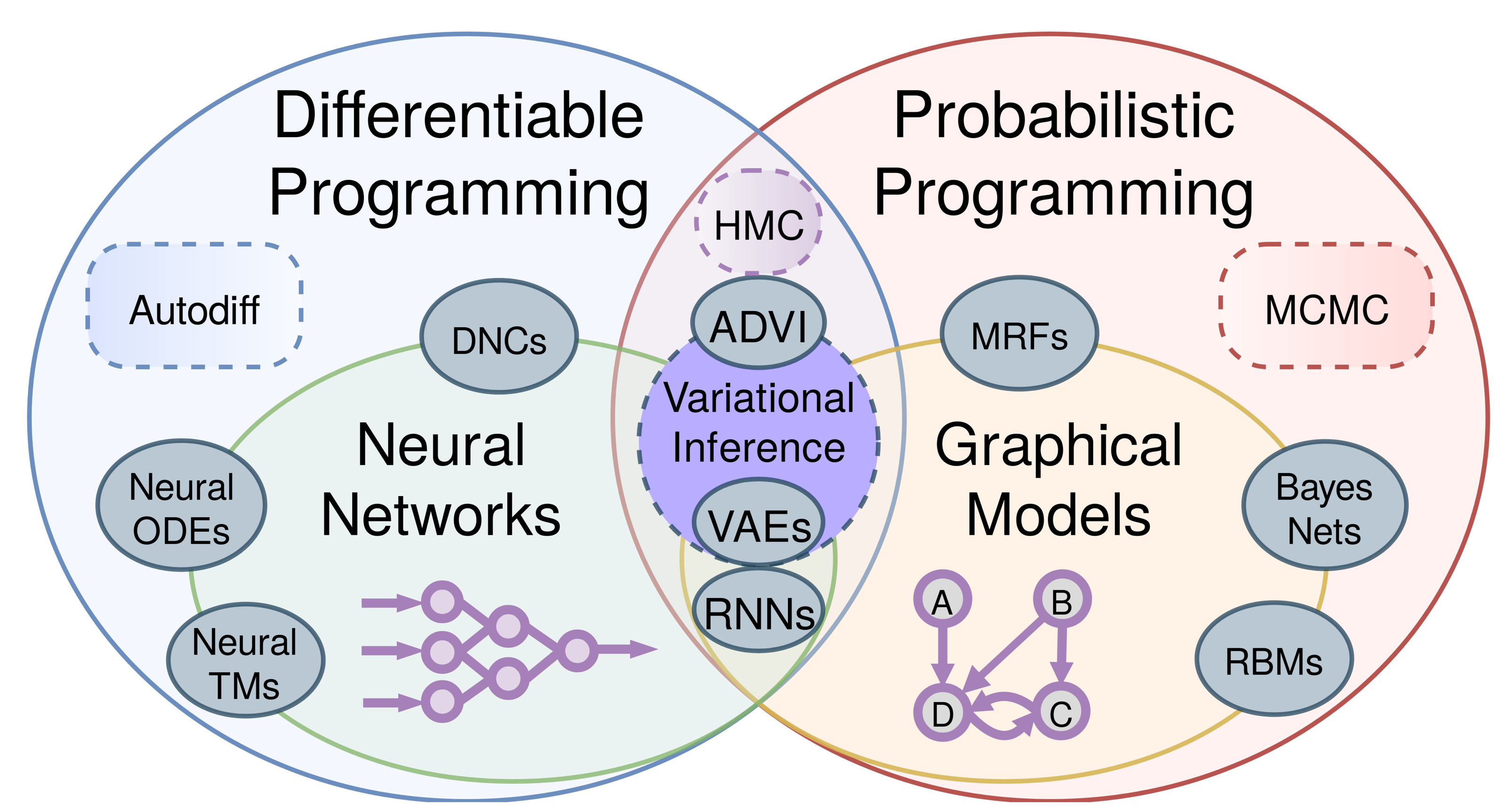


Testing

Log errors evaluating $f(x) = \frac{\sin(\sin(\sin(x)))}{x} + x\sin(x) + \cos(x) + x$



Differentiable Programming



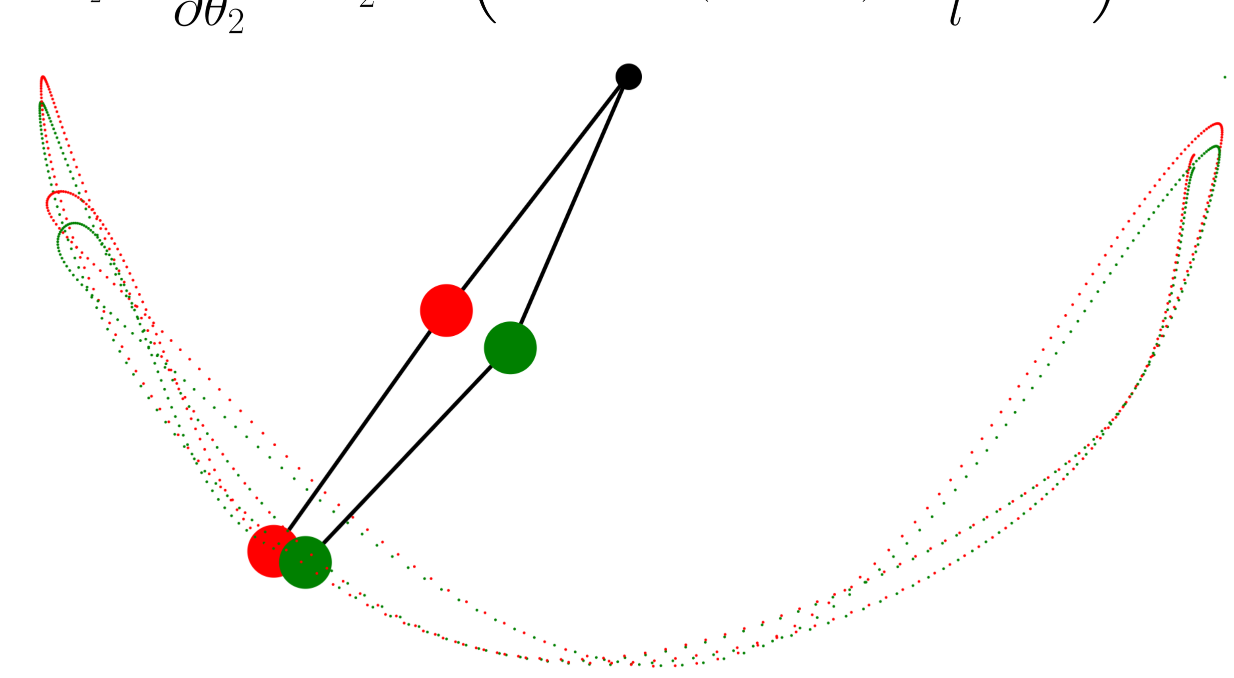
Double Pendulum

$$\dot{\theta}_1 = \frac{6}{ml^2} \frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$

$$\dot{\theta}_2 = \frac{6}{ml^2} \frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}.$$

$$\dot{p}_{\theta_1} = \frac{\partial L}{\partial \theta_1} = -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right)$$

$$\dot{p}_{\theta_2} = \frac{\partial L}{\partial \theta_2} = -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right).$$



Université de Montréal

Mila