Faster Computations with Generative Expressions

Jonas Östlund

February 23, 2019

This talk

- Background and motivation
- Do more during code generation, less at runtime
- How it works and how to use it: Numerical optimization example
- Results

About Jonas

- Software engineer at MindMaze in Lausanne
- Co-founder of Anemomind
- Research in computer vision
- This project in my spare time
- Github: `https://github.com/jonasseglare`
- Blog: `http://ostlund.rocks/`

Squaring a number in Geex

Example: $5^2 = 5 \cdot 5 = 25$
Using Generative Expressions (Geex)

```
(require '[geex.java :as java]
         '[geex.common :as c])

(java/typed-defn geex-square [Double/TYPE x]
   (c/* x x))
```

Squaring a number in Geex

Example: $5^2 = 5 \cdot 5 = 25$
Using Generative Expressions (Geex)

```
(require '[geex.java :as java]
         '[geex.common :as c])

(java/typed-defn geex-square [Double/TYPE x]
   (c/* x x))
```

```
(geex-square 5)
```

Squaring a number in Geex

Example: $5^2 = 5 \cdot 5 = 25$
Using Generative Expressions (Geex)

```
(require '[geex.java :as java]
         '[geex.common :as c])

(java/typed-defn geex-square [Double/TYPE x]
   (c/* x x))
```

```
(geex-square 5)
```

```
25.0
```

Println debugging

```
(java/typed-defn geex-square [Double/TYPE x]
  (println "Input:" x)
  (let [output (c/* x x)]
    (println "Output:" output)
    output))
```

Println debugging

```
(java/typed-defn geex-square [Double/TYPE x]
  (println "Input:" x)
  (let [output (c/* x x)]
    (println "Output:" output)
    output))
```

```
Input: #object[geex.DynamicSeed 0x22351330 ISeed(type=double, id=5 ...
Output: #object[geex.DynamicSeed 0x2158d1d1 ISeed(type=double, id= ...
```

Println debugging

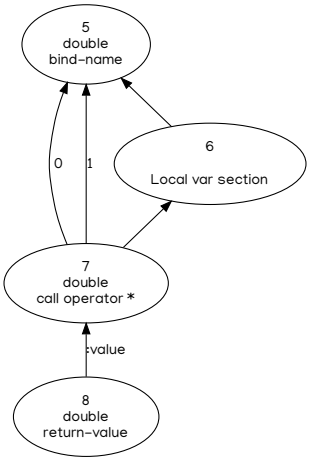
```
(java/typed-defn geex-square [Double/TYPE x]
  (println "Input:" x)
  (let [output (c/* x x)]
    (println "Output:" output)
    output))
```

```
Input: #object[geex.DynamicSeed 0x22351330 ISeed(type=double, id=5 ...
Output: #object[geex.DynamicSeed 0x2158d1d1 ISeed(type=double, id= ...
```

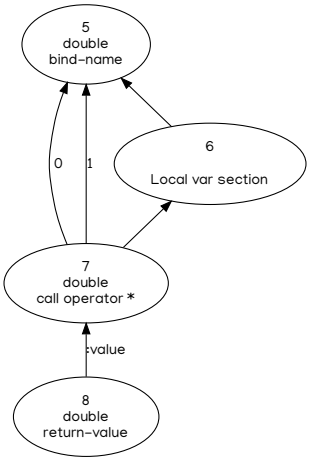
```
#'cljd.square/geex-square
```

Function body executed when loading the code

Computational graph



Computational graph



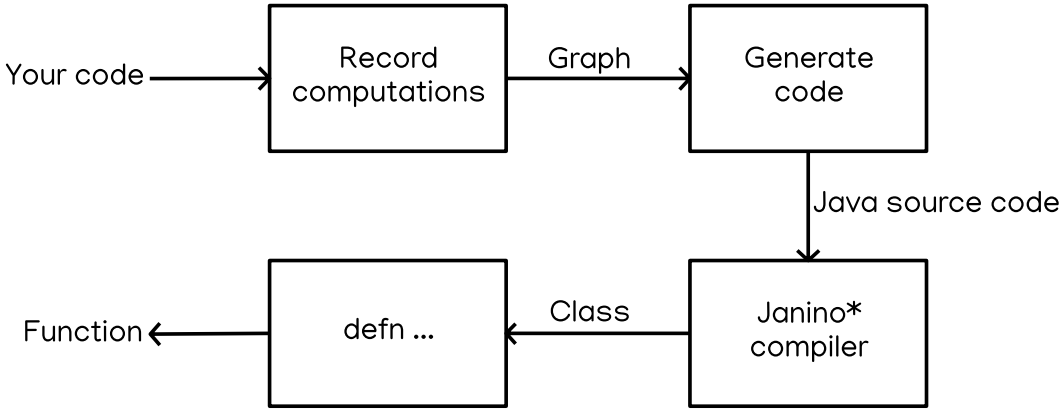
Seeds

- Vertices in the graph
- Replace unknown runtime values
- Generate code
- Contain type information

Generated code

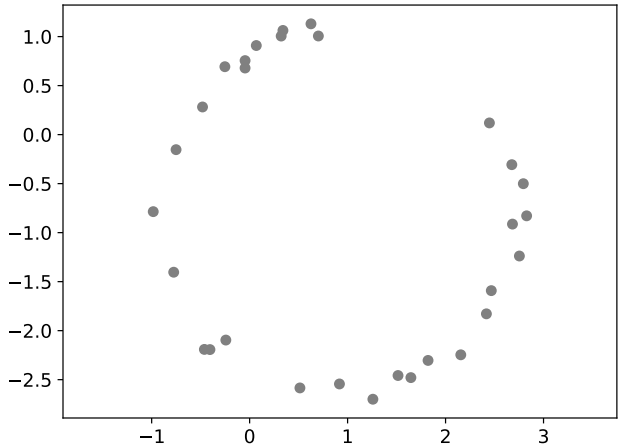
```
package cljd_psquare;

public class TypedDefn__geex_dsquare {
  /* Various definitions */
  public double apply(final double arg00) {
    return (arg00 * arg00);
  }
}
```

Code generation pipeline

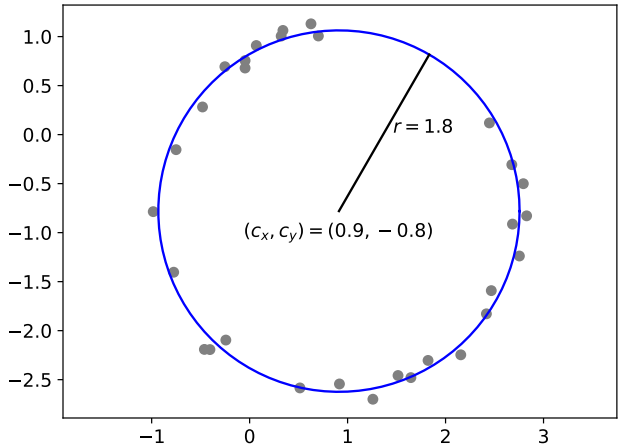


* https://janino-compiler.github.io/janino/

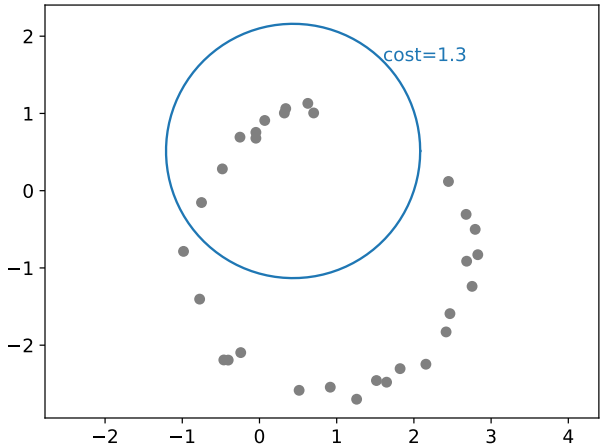
Circle fitting



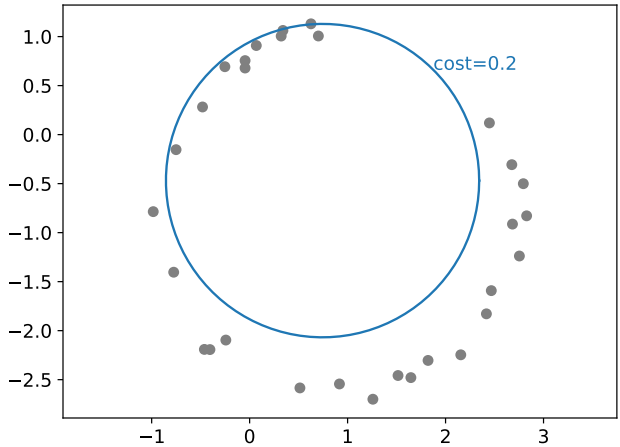
Circle fitting



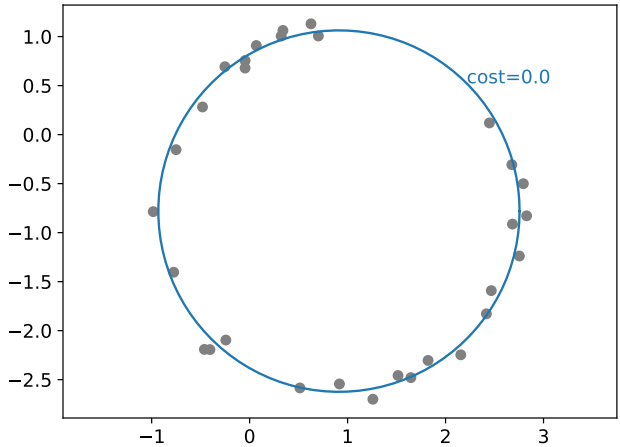
Optimization problem



Optimization problem

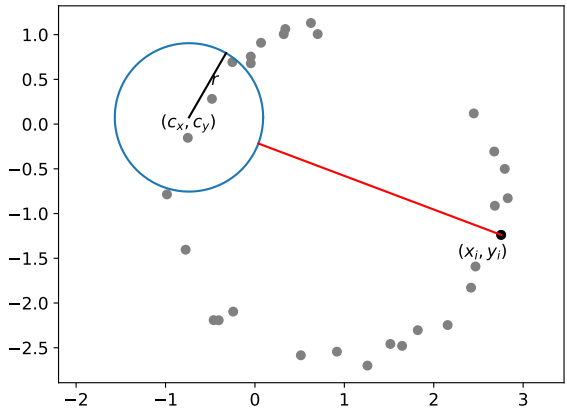


Optimization problem



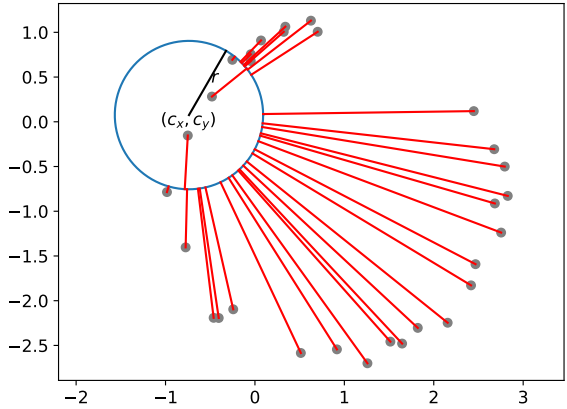
Cost of a single point

$$f_i(c_x, c_y, r) = \left(\underbrace{\sqrt{(c_x - x_i)^2 + (c_y - y_i)^2} - r}_{\text{distance from point to circle}} \right)^2$$



Cost of all points

$$f(c_x, c_y, r) = \frac{1}{N} \sum_{i=1}^{N} \left(\sqrt{(c_x - x_i)^2 + (c_y - y_i)^2} - r \right)^2$$



Cost of a single point

$$\underbrace{f_i(c_x, c_y, r)}_{\texttt{evaluate-point}} = \left(\sqrt{(c_x - x_i)^2 + (c_y - y_i)^2} - r\right)^2$$

```
(require '[geex.common :as c])

(defn sqr [x] (c/* x x))

(defn evaluate-point [{:keys [cx cy r]} ;; ← Circle parameters
                      [x y]] ;; ← The point
  (let [dist-to-centre (c/sqrt (c/+ (sqr (c/- x cx))
                                    (sqr (c/- y cy))))
        dist-to-circle (c/- dist-to-centre r)]
    (sqr dist-to-circle)))
```

Objective function

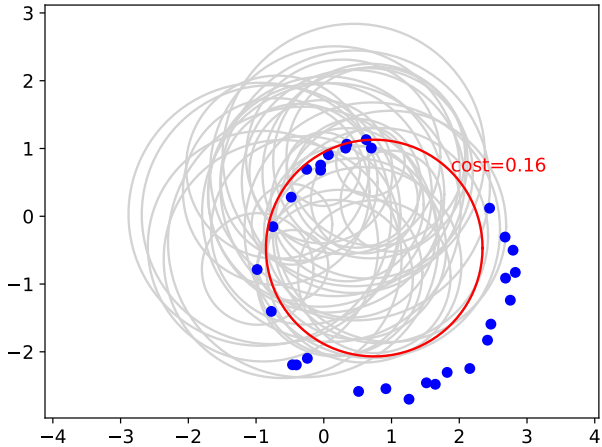
$$f(c_x, c_y, r) = \frac{1}{N} \sum_{i=1}^{N} \left(\sqrt{(c_x - x_i)^2 + (c_y - y_i)^2} - r \right)^2$$

```
(defn circle-fitness-cost [circle-params point-array init-cost]
  (let [n (c/quot (c/cast Long/TYPE (c/count point-array)) 2)]
    (c/* (c// 1.0 n)
         (c/transduce
          (c/map (comp (partial evaluate-point circle-params)
                       (partial get-point-2d point-array)))
          c/+
          init-cost ;; ← Typically 0
          (c/range n)))))
```

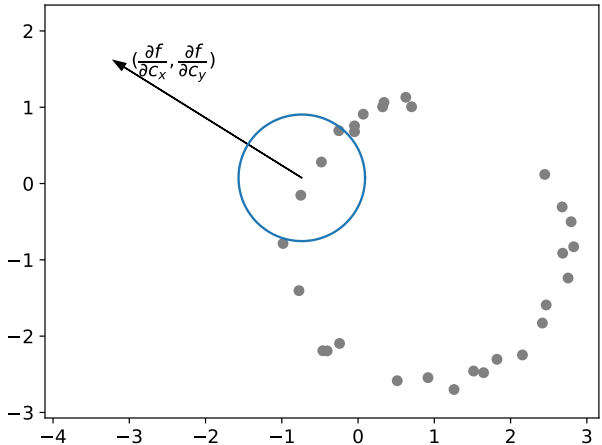
Generated objective function

```
// ... 26 lines of code not shown
    while (true) {
      final double s0041 = lvar0;
      final long s0042 = lvar1;
      final long s0043 = lvar2;
      final long s0044 = lvar3;
      if ((s0043 <= 0)) {
        lvar4 = s0041;
      } else {
        final long s0055 = (2 * s0042);
        final double s0059 = (arg01[((int) (s0055 + 0))]);
        final double s0063 = (arg01[((int) (s0055 + 1))]);
        final double s0064 = (s0059 - s0010);
        final double s0066 = (s0063 - s0015);
        final double s0070 = ((java.lang.Math.sqrt(((s0064 * s0064) + (s0066 * s0066)))) - s0020);
        lvar0 = (s0041 + (s0070 * s0070));
        lvar1 = (s0042 + s0044);
        lvar2 = (s0043 - 1);
        lvar3 = s0044;
        continue;
      }
      final double s0085 = lvar4;
      lvar5 = s0085;
      break;
    }
    final double s0091 = lvar5;
    return ((1.0 / s0026) * s0091);
// ... 2 lines of code not shown
```

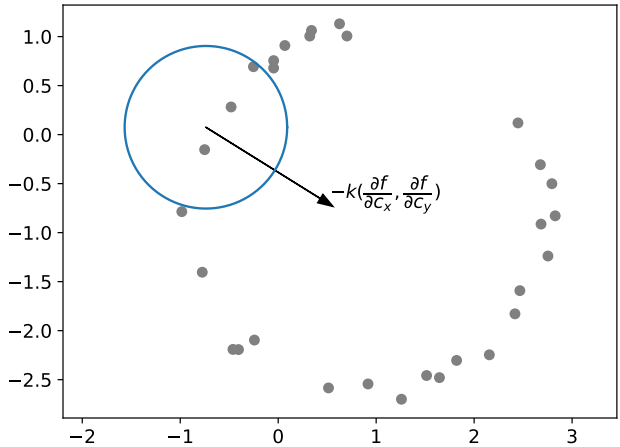
Minimizing the cost (naïve approach)



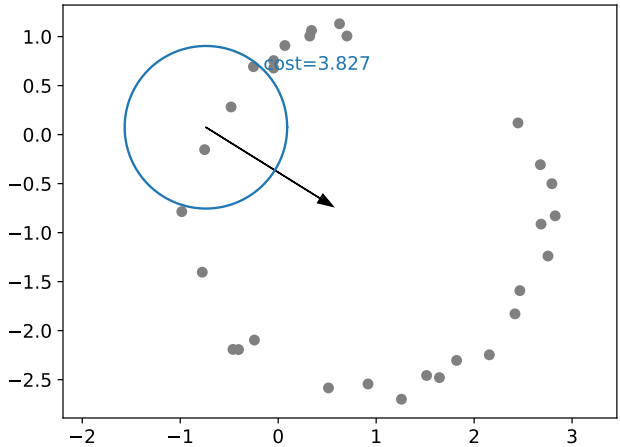
Gradient descent



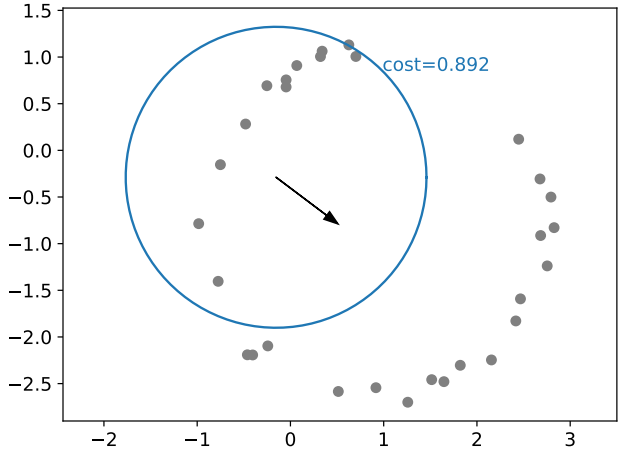
Gradient descent



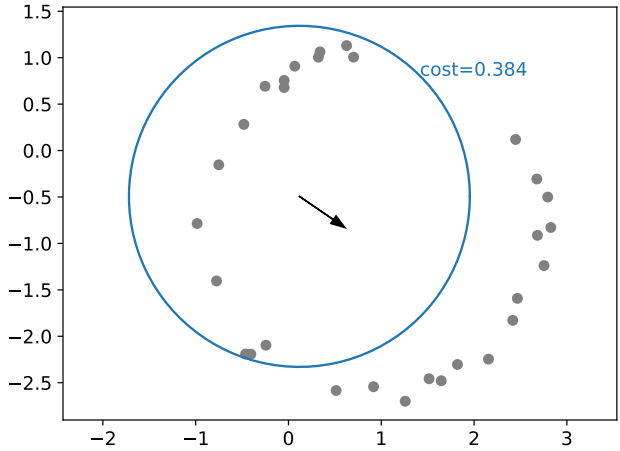
Gradient descent



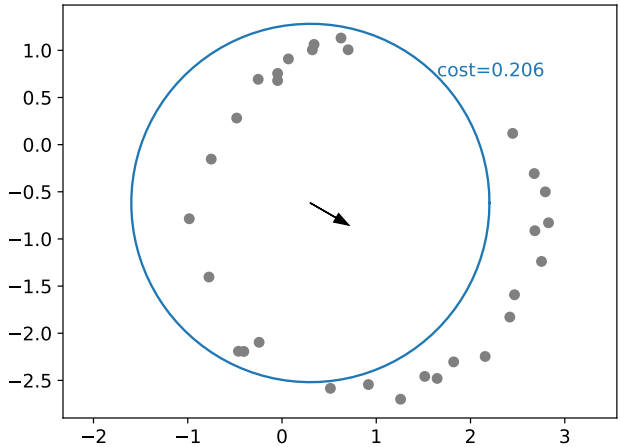
Gradient descent



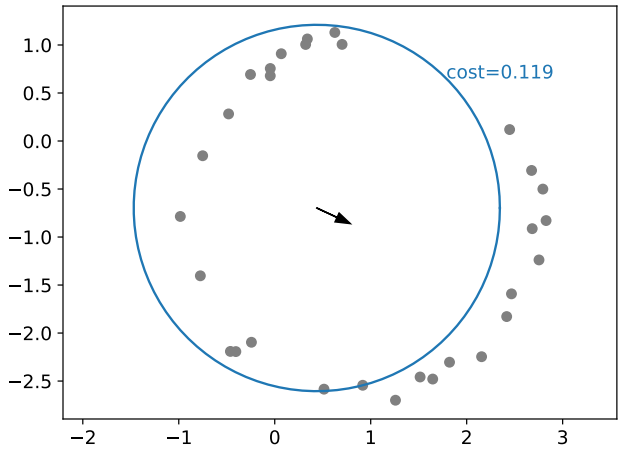
Gradient descent



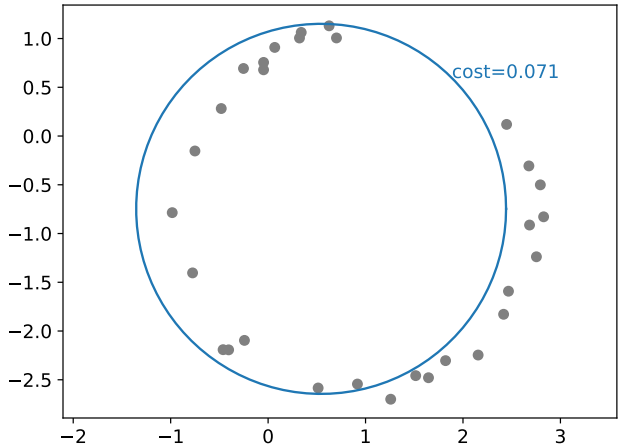
Gradient descent



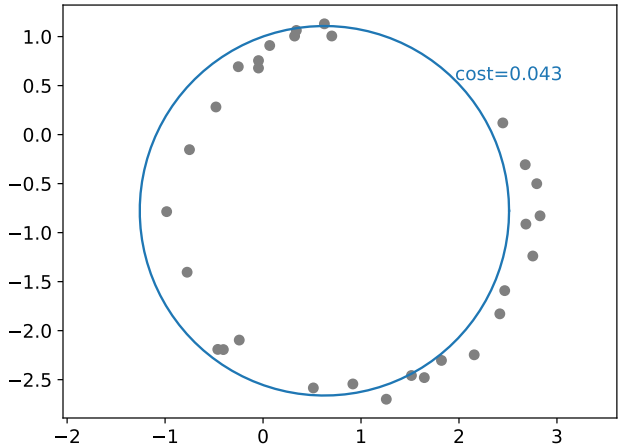
Gradient descent



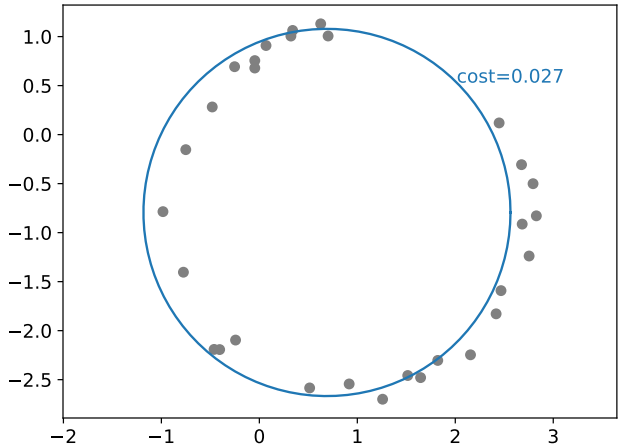
Gradient descent



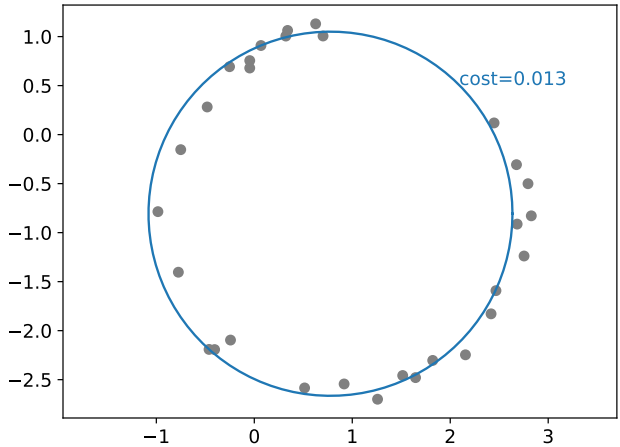
Gradient descent



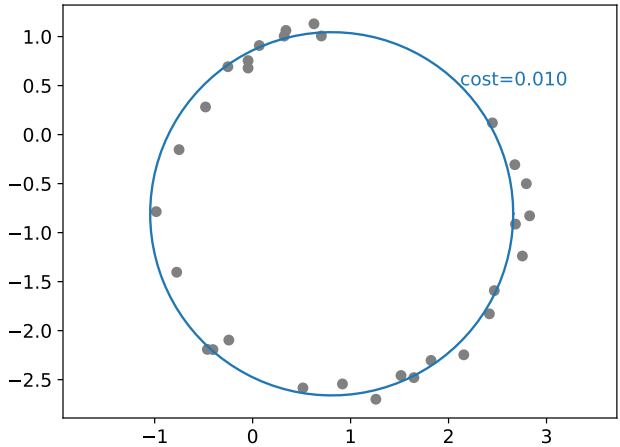
Gradient descent



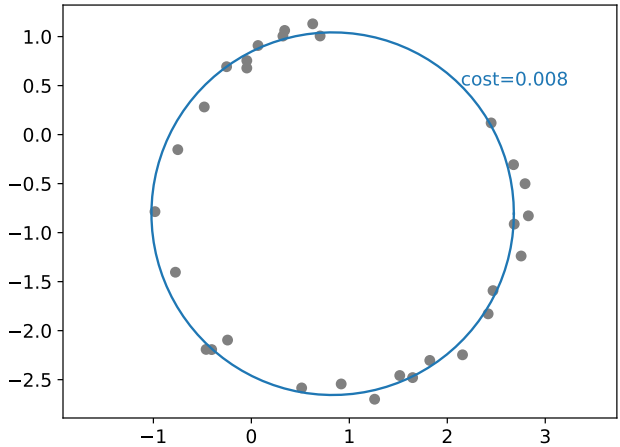
Gradient descent



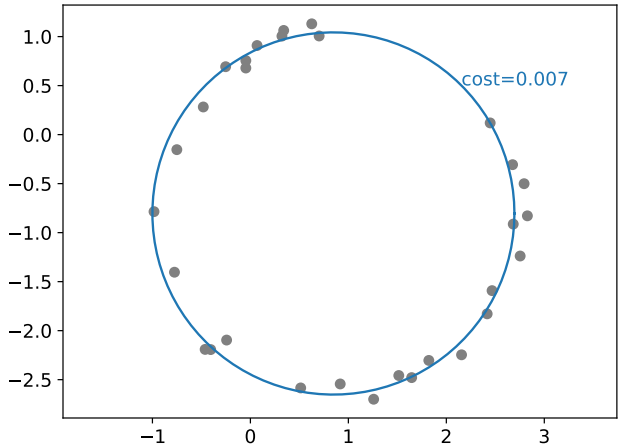
Gradient descent



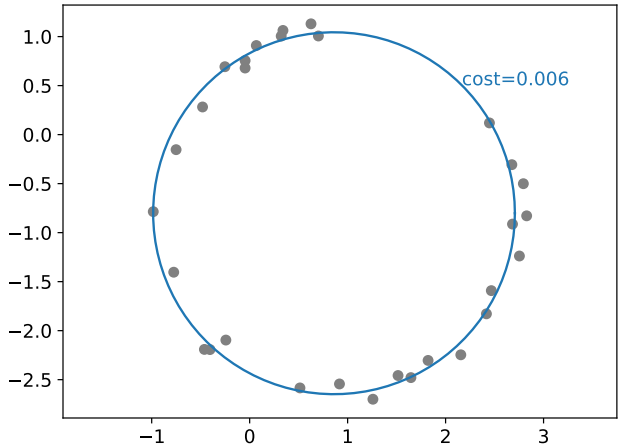
Gradient descent



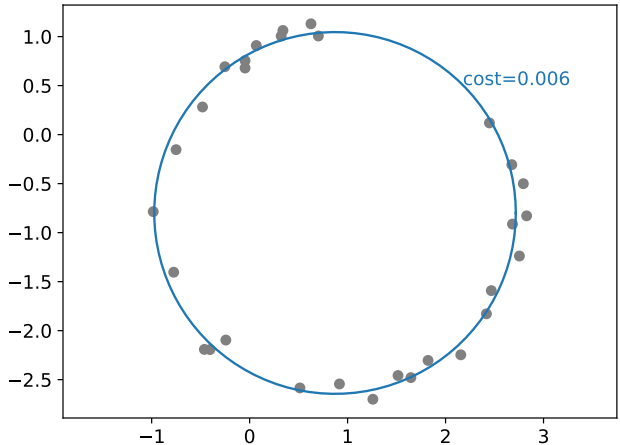
Gradient descent



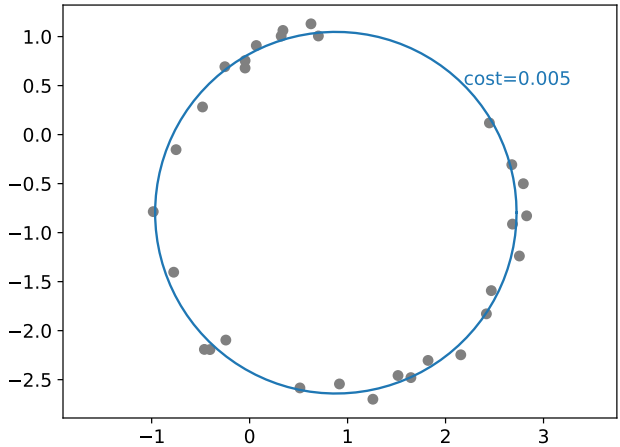
Gradient descent



Gradient descent

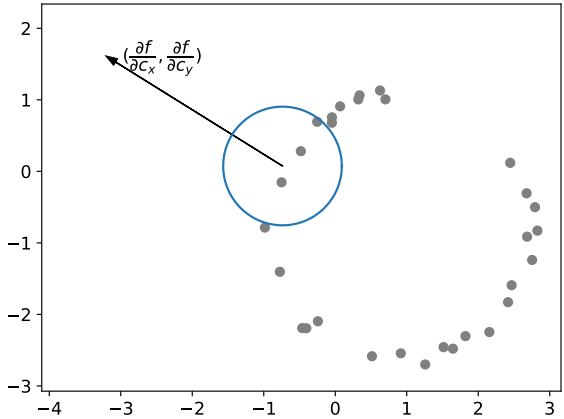


Gradient descent (good enough)



Computing the gradient

Differentiate the cost with respect to circle centre and radius: $\frac{\partial f}{\partial c_x}$, $\frac{\partial f}{\partial c_y}$ and $\frac{\partial f}{\partial r}$



A new numeric “type”: AD numbers

```clojure
(defn variable [x]
  {:value x
   :deriv 1.0}) ;; dx/dx = 1
```

```clojure
(defn constant [c]
  {:value c
   :deriv 0.0}) ;; dc/dx = 0, c being a constant
```

AD stands for a “automatic differentiation”.

Example-based multiple dispatch

Register our type with name `::ad`

```
(require '[bluebell.utils.ebmd :as ebmd])

(defn ad-number? [x]
  (and (map? x) (contains? x :value) (contains? x :deriv)))

(ebmd/def-arg-spec
  ::ad ;; ← Name of the spec
  {:pred ad-number? ;; ← Predicate function

   ;; Examples disambiguate overlapping predicates:
   :pos [(variable 3.0) (constant 5.0)] ;; ← Matching examples
   :neg [2.0 :kwd {:kattskit 119}]}) ;; ← Non-matching examples
```

Extending multiplication for automatic differentiation

```
(ebmd/def-poly c/binary-mul [::ad a
                             ::ad b]
  {:value (c/* (:value a) (:value b))

   ;; Recall that (a · b)' = a' · b + a · b'
   :deriv (c/+ (c/* (:value a)
                    (:deriv b))
               (c/* (:deriv a)
                    (:value b)))})
```

The product of two AD numbers is a new AD number.

Derivative of a cube

$g(x) = x^3$ e.g. $g(9) = 729$

Derivative of a cube

$g(x) = x^3$ e.g. $g(9) = 729$

$g'(x) = 3x^2$ e.g. $g'(9) = 243$

Derivative of a cube

$g(x) = x^3$ e.g. $g(9) = 729$

$g'(x) = 3x^2$ e.g. $g'(9) = 243$

```
(let [x (variable 9.0)]
  (c/* x x x))
```

Derivative of a cube

$g(x) = x^3$ e.g. $g(9) = 729$

$g'(x) = 3x^2$ e.g. $g'(9) = 243$

```
(let [x (variable 9.0)]
  (c/* x x x))
```

```
{:value 729.0, :deriv 243.0}
```

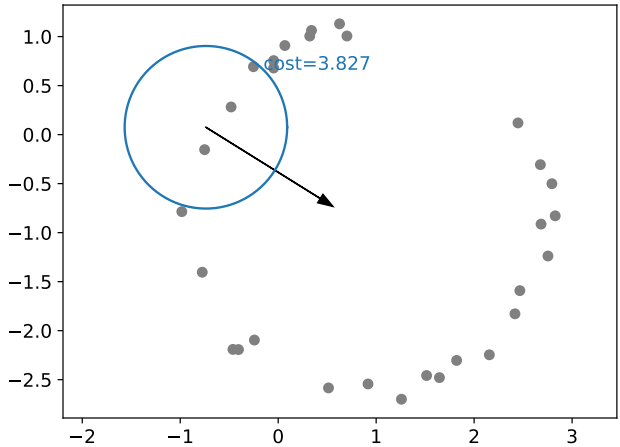
Gradient code generation

```
(defn derivative-for-key [circle-params points k]
  (:deriv (circle-fitness-cost ;; ← Derivative of objective function
           (update circle-params k variable) ;; ← 'k' is a variable
           points
           (constant 0.0))))

(java/typed-defn gradient
   [{:cx Double/TYPE :cy Double/TYPE :r Double/TYPE} circle-params
    (c/array-class Double/TYPE) points]
   {:cx (derivative-for-key circle-params points :cx) ;; ← df/dc_x
    :cy (derivative-for-key circle-params points :cy) ;; ← df/dc_y
    :r (derivative-for-key circle-params points :r)}) ;; ← df/dr
```

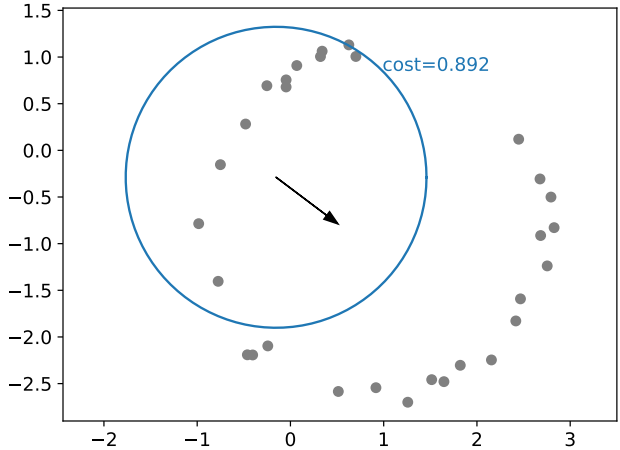
Generated gradient code

```
// ... 50 lines of code not shown
    while (true) {
      final double s0043 = lvar0;
      final double s0044 = lvar1;
      final long s0045 = lvar2;
      final long s0046 = lvar3;
      final long s0047 = lvar4;
      if ((s0046 <= 0)) {
        lvar5 = s0043;
        lvar6 = s0044;
      } else {
        final long s0059 = (2 * s0045);
        final double s0063 = (arg01[((int) (s0059 + 0))]);
        final double s0067 = (arg01[((int) (s0059 + 1))]);
        final double s0068 = (s0063 - s0010);
        final double s0075 = (s0067 - s0015);
        final double s0080 = (java.lang.Math.sqrt(((s0068 * s0068) + (s0075 * s0075))));
        final double s0084 = (s0080 - s0020);
        final double s0086 = ((((0.5 / s0080) * (((s0068 * -1.0) + (-1.0 * s0068)) + 0.0)) - 0.0);
        lvar0 = (s0043 + ((s0084 * s0086) + (s0086 * s0084)));
        lvar1 = (s0044 + (s0084 * s0084));
        lvar2 = (s0045 + s0047);
        lvar3 = (s0046 - 1);
        lvar4 = s0047;
        continue;
      }
      final double s0106 = lvar5;
      final double s0107 = lvar6;
      lvar7 = s0106;
      lvar8 = s0107;
      break;
    }
// ... 89 lines of code not shown
```

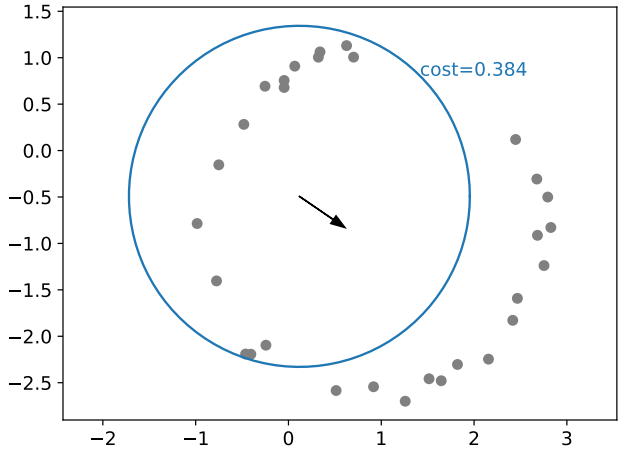
Gradient descent



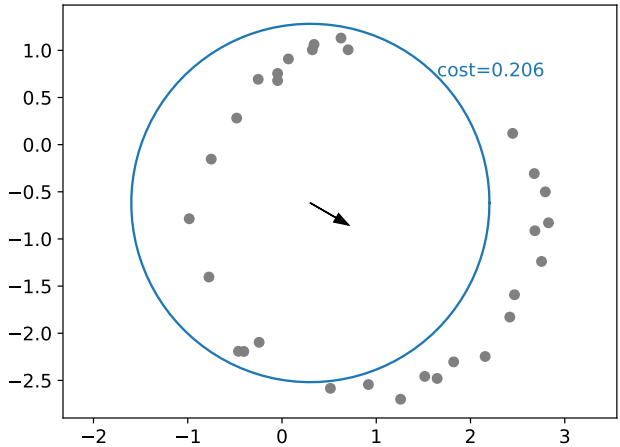
Gradient descent



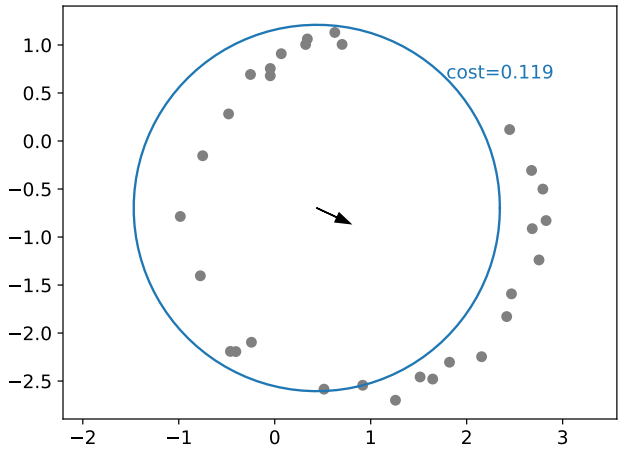
Gradient descent



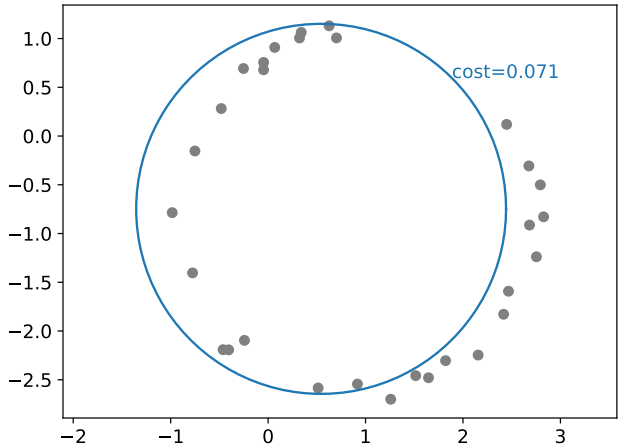
Gradient descent



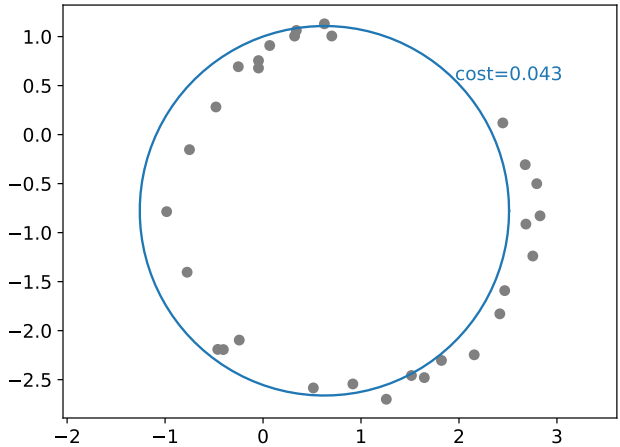
Gradient descent



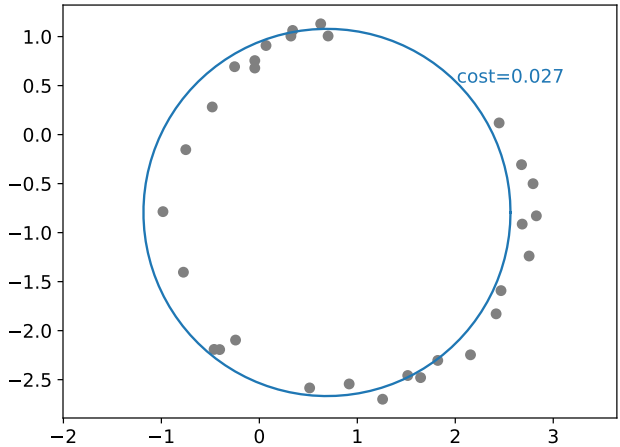
Gradient descent



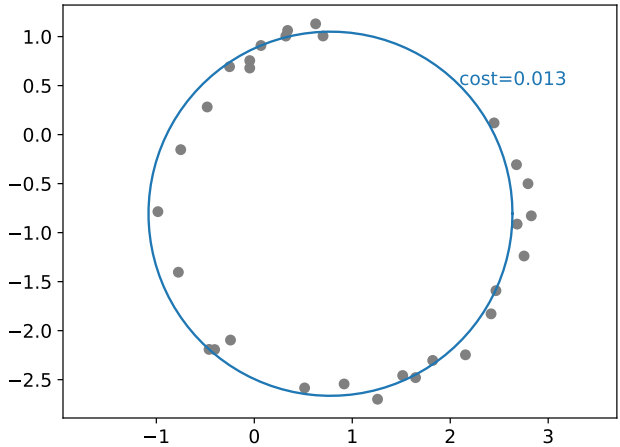
Gradient descent



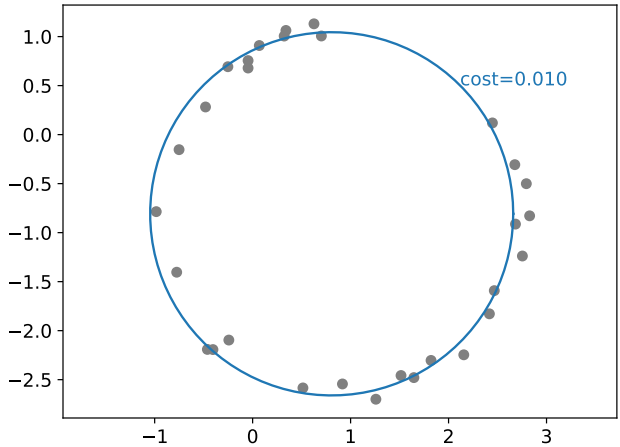
Gradient descent



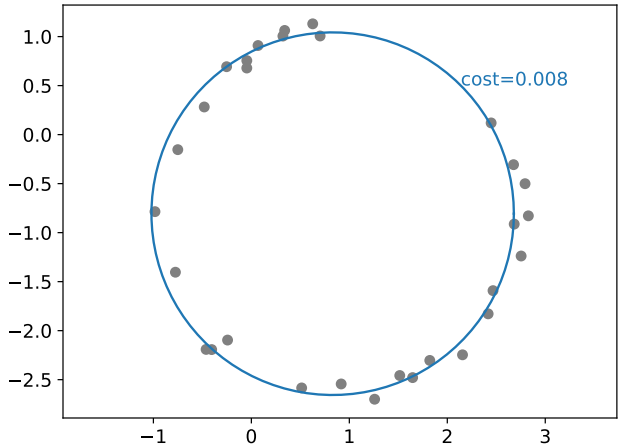
Gradient descent



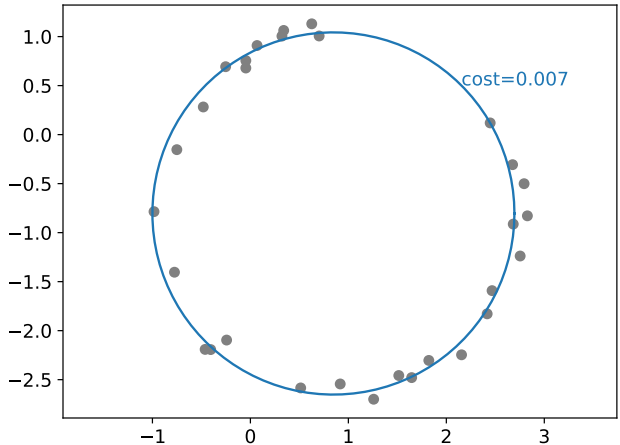
Gradient descent



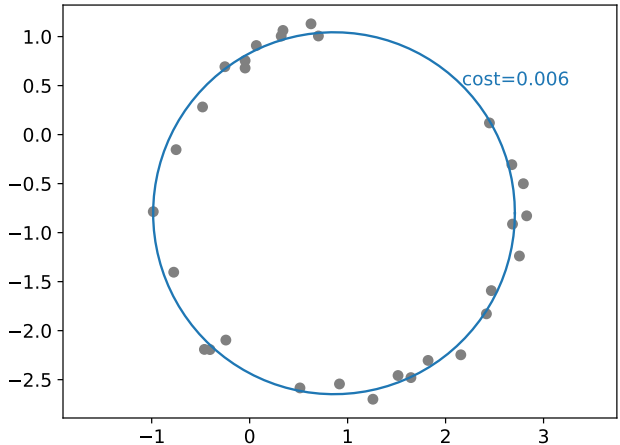
Gradient descent



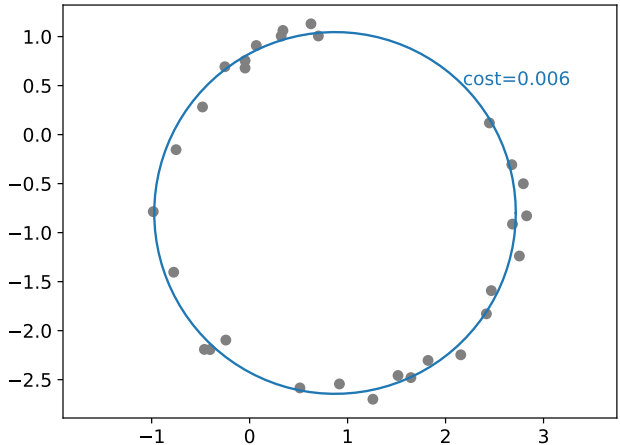
Gradient descent



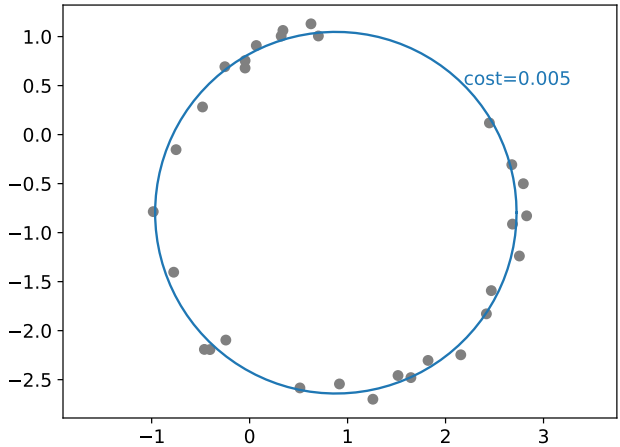
Gradient descent



Gradient descent



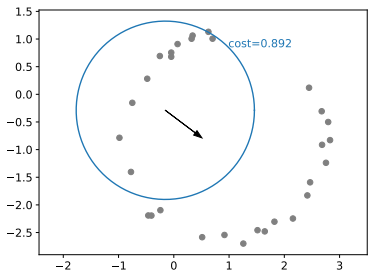
Gradient descent (good enough)



Measuring the time

Repeatedly optimize the circle parameters using gradient descent and automatic differentiation.

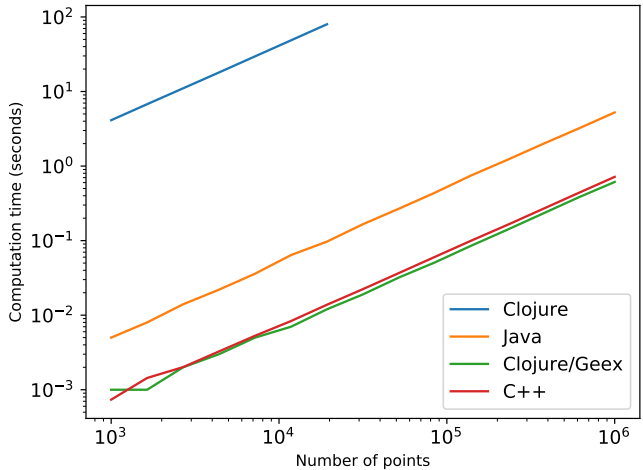
- Vary the number of points that we fit to
- Measure the computation time
- High-level implementations in
  - Clojure
  - Java
  - Clojure with Geex (what we just implemented)
  - C++



Example: Java implementation

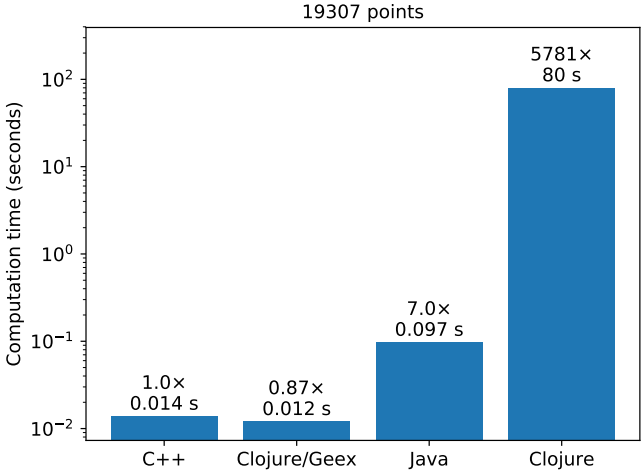
```
// ... 24 lines of code not shown
    public ADNumber evaluate(double[] data, ADCircleParameters params) {
        int N = data.length/2;
        ADNumber sum = ADNumber.constant(0.0);

        for (int i = 0; i < N; i++) {
            int at = 2*i;
            ADNumber x = ADNumber.constant(data[at + 0]);
            ADNumber y = ADNumber.constant(data[at + 1]);

            ADNumber distToCentre = x.sub(params.cx).square()
                .add(y.sub(params.cy).square())
                .sqrt();
            ADNumber distToCircle = distToCentre.sub(params.r);
            sum = sum.add(distToCircle.square());
        }

        return sum.mul(ADNumber.constant(1.0/N));
// ... 30 lines of code not shown
```

Circle fitness optimization problem



Comparison of high-level implementations that have not been optimized.

Circle fitness optimization problem



Comparison of high-level implementations that have not been optimized.

Conclusion

- Tool for code generation
- Write high-level code
- Write fast code
- General purpose
- Complement to Clojure
- Just another library
- To be published in the next few days:
  - Geex source code: `https://github.com/jonasseglare/geex`
  - This presentation: `https://github.com/jonasseglare/cljd2019`

Conclusion

- Tool for code generation
- Write high-level code
- Write fast code
- General purpose
- Complement to Clojure
- Just another library
- To be published in the next few days:
  - Geex source code: `https://github.com/jonasseglare/geex`
  - This presentation: `https://github.com/jonasseglare/cljd2019`

Thank you for listening!