Numerical Integration with Clojure

- Find the area under a curve by dividing area into trapeziums

Numerical Integration with Clojure

- Basic functions
- Simpsons rule
- Extend to multi-dimensional

Numerical Integration with Clojure

- Disclaimer
  - There are definitely better ways to do this
  - I am not yet a Clojure expert (so don't ask hard questions)

Sequence Functions

```
(first [0 4])   0

(second [0 4])   4

(range 0 4)   (0 1 2 3)

(count (range 0 4))   4

(partition 1 (range 0 4))  ((0) (1) (2) (3))

(partition 2 1 (range 0 4))  ((0 1) (1 2) (2 3))

(map inc (range 0 4))  (1 2 3 4)

(reduce + (range 0 4)) 6
```

Extra Sequence Function

```
(range-inclusive [0 4] 1)    (0 1 2 3 4)   (my addition)
```

Anonymous Functions

```
(map #(* % %) (range 0 4))   (0 1 4 9)

(map #(+ %1 %2) (range 0 4)  (range 0 4))  (0 2 4 6)
```

apply Function

```
(+ 0 1 2 3)  6
```

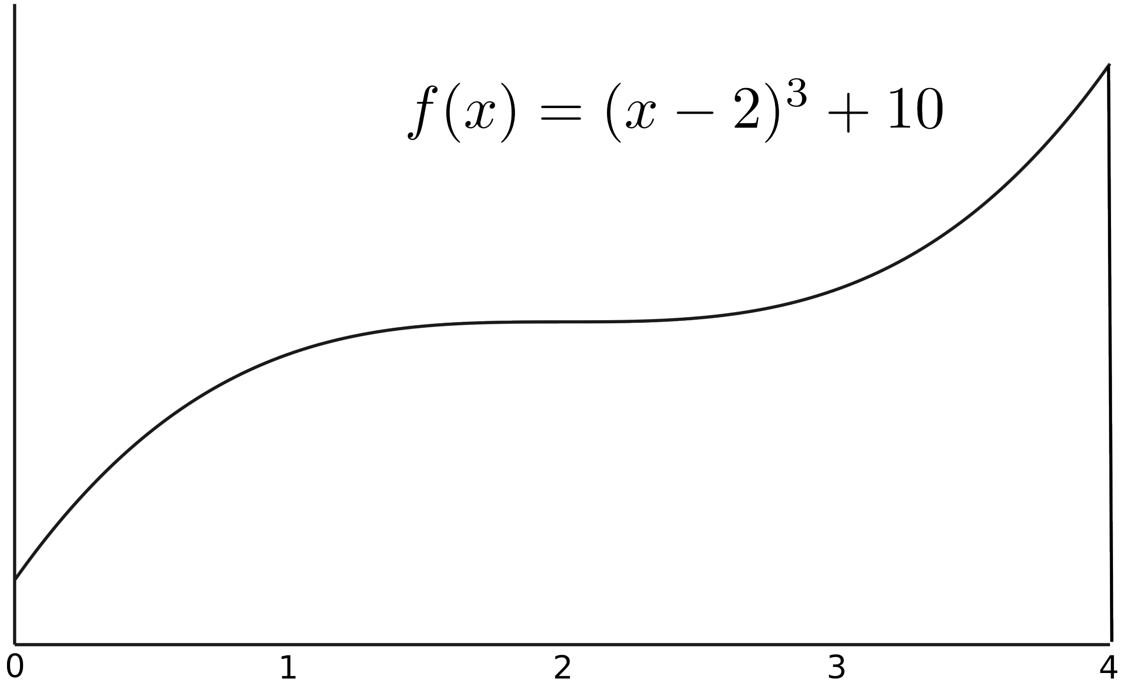
```
(+ [0 1 2 3])

ClassCastException Cannot cast
clojure.lang.PersistentVector to java.lang.Number
```

```
(apply + [0 1 2 3]) 6
```

Simpsons Rule

- We want to find the area under curve between 0 and 4

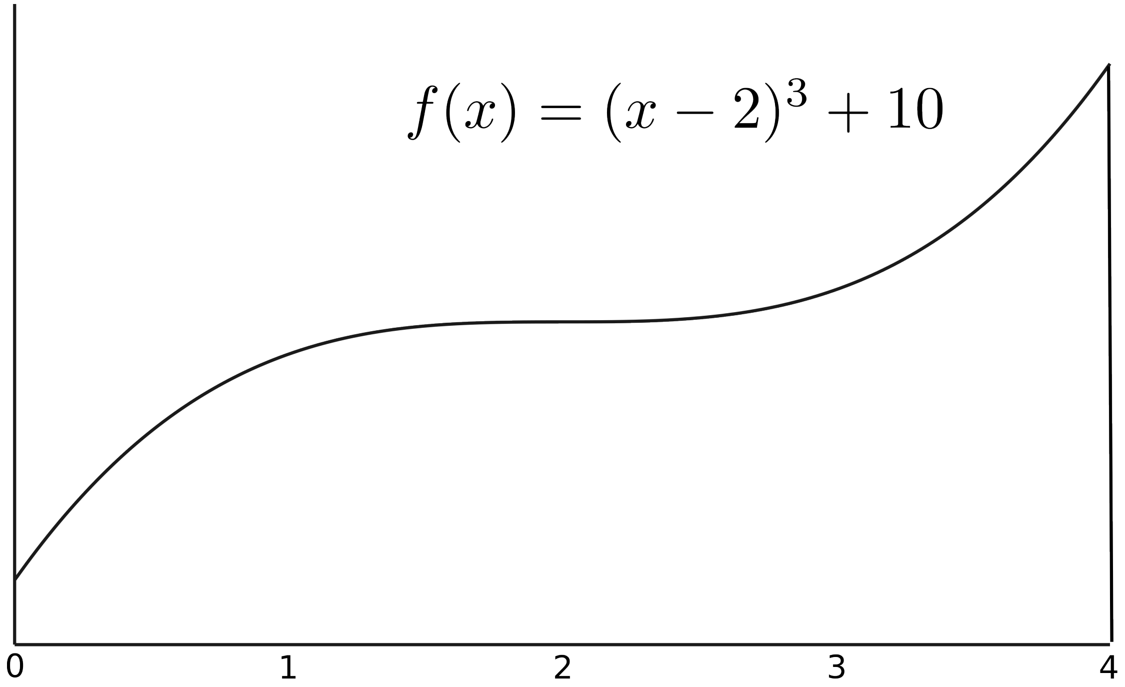


fn-1D

```
(defn
fn-1D
  "1D test function (x-2)^3 +10"
  [x]
  (+ 10 (Math/pow (- x 2) 3)))
```

Simpsons Rule

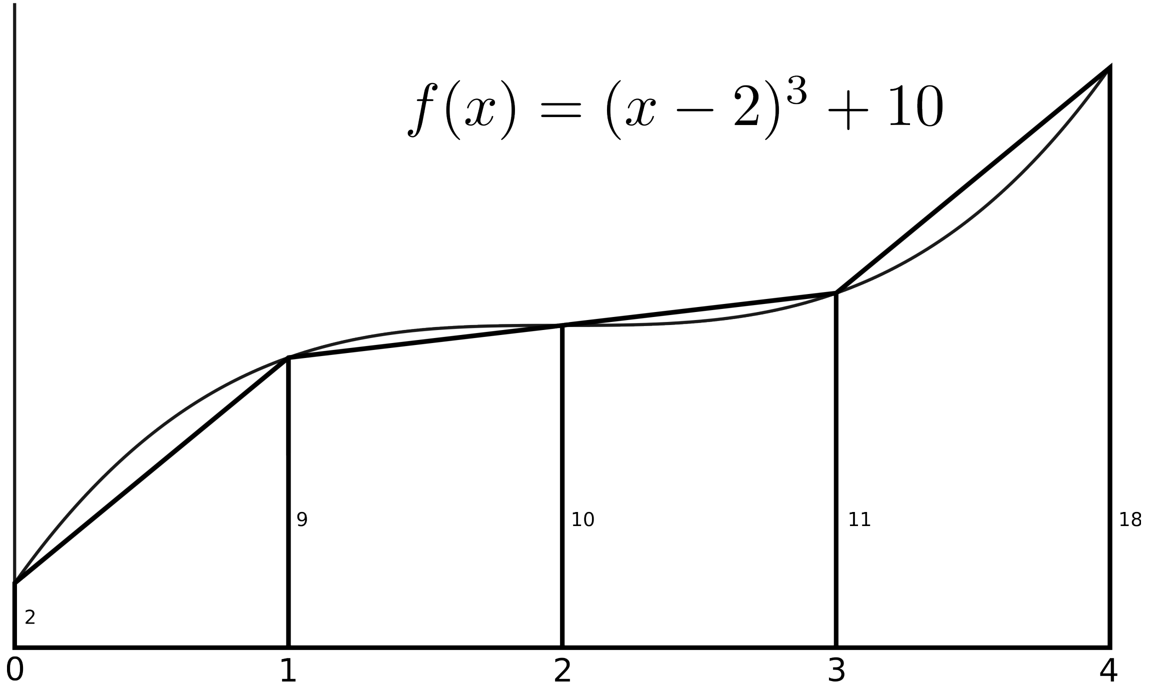
- Inputs?



fn-1D	(x – 2)^3 + 10
bounds	[0 4]
step	1

Simpsons Rule

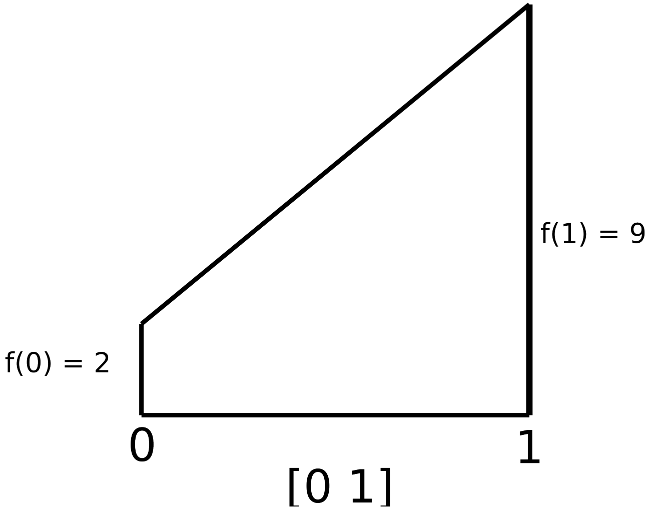
- Divide into trapezium segments.



Simpsons Rule

- Area of trapezium = base * average of heights

$$f(x) = (x-2)^3 + 10$$



Area = 1 * ½ (2 + 9)

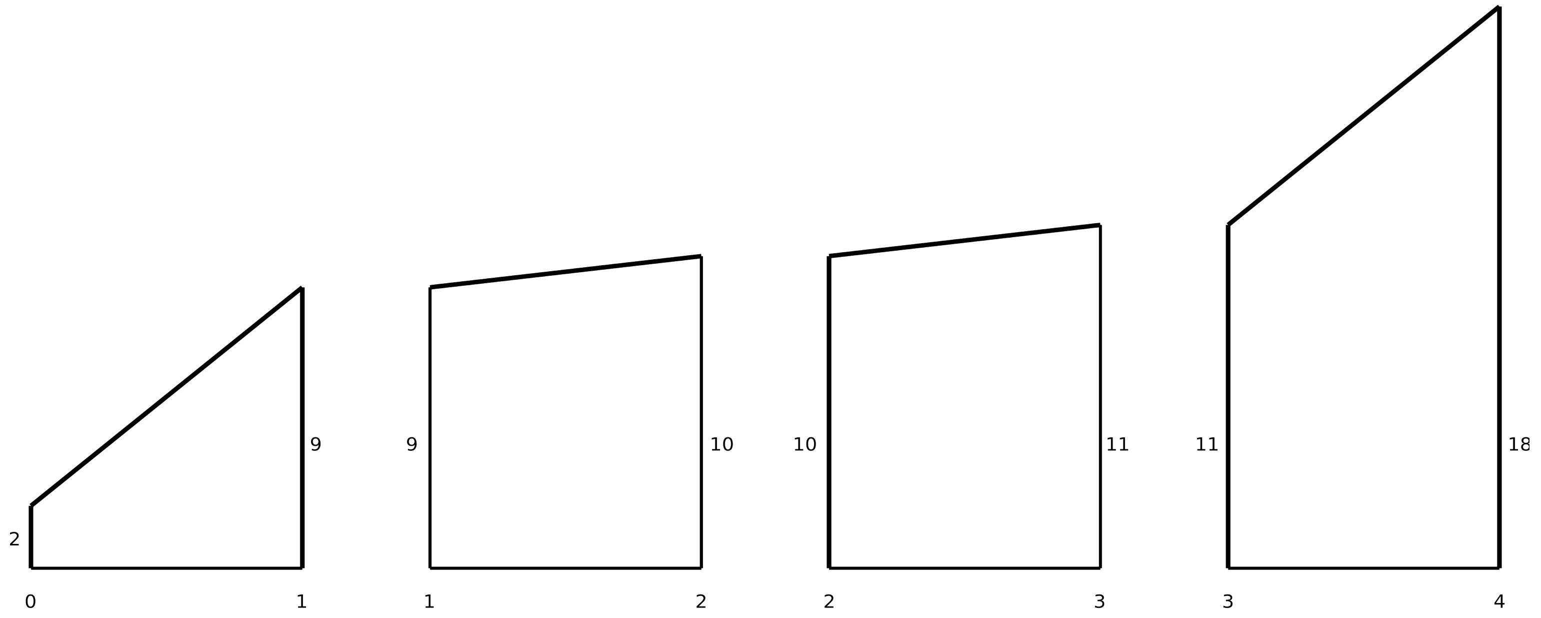
trapezium-1D

```clojure
(defn
trapezium-1D
  "The area of a trapezium is the base * the average of the
heights"
  [f bound]
  (let [
        average-of-heights (/ (reduce + (map f bound)) 2 )
        base (- (second bound) (first bound))
        ]
    (* base average-of-heights)))
```

```clojure
(trapezium-1D fn-1D [0 1])
```

Simpsons Rule

- Decompose
  - ▶ area: sum of
    - ▶ area of trapezium : computed from
      - ▶ fn-1D
      - ▶ bounds of trapezium



split-bounds-1D

```clojure
(defn
split-bounds-1D
  "Get a list of bounds split by step"
  [bound step]
  (partition 2 1 (range-inclusive bound step)))
```

```clojure
(split-bounds-1D [0 4] 1)
```

volume-1D

```
(defn
volume-1D
  "Sum the areas of all the trapeziums."
  [f bound step]
  (reduce + (map #(trapezium-1D f %)
                      (split-bounds-1D bound step))))
```

```
(volume-1D fn-1D [0 4] 1)
```

Multi-dimensional

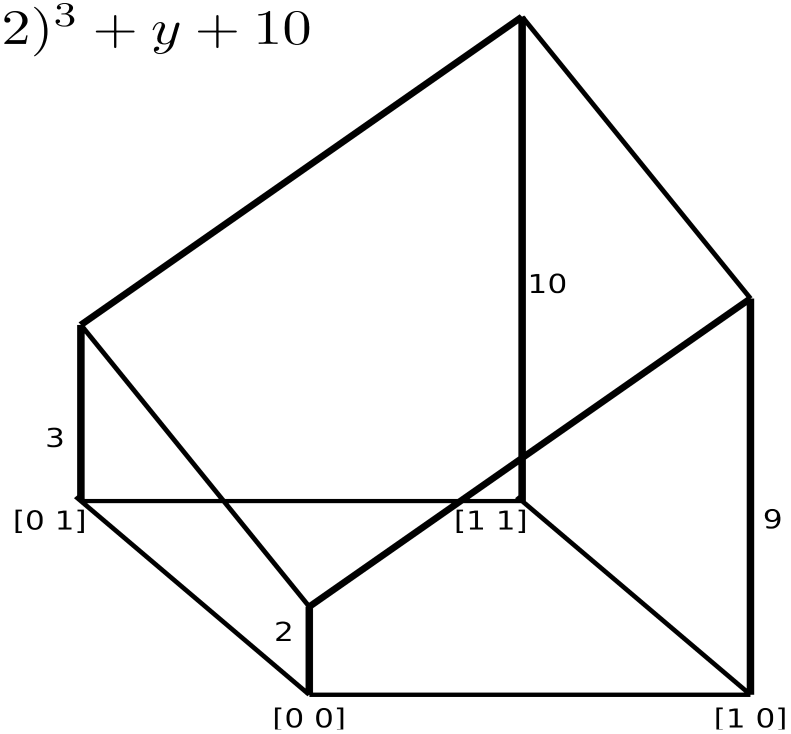
- Redefine parameters

fn-1D	=>	fn-2D		- (x-2)^3 + y +10
bound [min max]	=>	bounds	[[0 4] [0 4]]	- one pair for each dimension
step	=>	step	[1 1]	- list of scalars
x	=>	point	[1 1]	- list of scalars

Multi-dimensional

- Decompose
  - volume: sum of
    - volume of trapezium : computed from
      - fn-ND
      - bounds of trapezium in both dimensions

$$f(x, y) = (x-2)^3 + y + 10$$



bounds [[0 1] [0 1]]

Combination Functions

```
(cartesian-product [0 1]) ((0) (1))
(cartesian-product [0 1] [0 1]) ((0 0) (0 1) (1 0) (1 1))
```

trapezium-ND

```
(defn
  trapezium-ND
  "The volume of a trapezium is the base * the average of the heights"
  [f bounds]
  (let [
        points (apply cartesian-product bounds)
        average-of-heights (/ (reduce + (map f points)) (count points) )
        base (reduce * (map #(- (second %) (first %)) bounds))
       ]
    (* base average-of-heights)
    )
  )
```

```
(trapezium-ND fn-2D [[0 1] [0 1]])
```

Multi-dimensional

- Decompose
    - volume: sum of
        - volume of trapezium : computed from
            - fn-ND
            - bounds of trapezium in both dimensions

split-bounds-ND

```
(defn
split-bounds-ND
  "Split the nd bounds up."
  [bounds step]
  (apply
     cartesian-product
       (map #(partition 2 1 (range-inclusive %1 %2))
         bounds step)))
```

```
(split-bounds-ND [[0 2] [0 2]] [1 1])
```

volume-ND

```
(defn
volume-ND
  "Sum the nd volumes of all trapeziums."
  [f bounds step]
  (reduce + (map #(trapezium-ND f %)
                         (split-bounds-ND bounds step))))
```

```
(volume-ND fn-2D [[0 1] [0 1]] [1 1])
```

Ok Bye