

# Approximating square root iteratively

How to compute values iteratively instead of symbolically

Terence Parr  
MSDS program  
**University of San Francisco**

# Iterative computing

- There are lots of useful functions with no closed form solution, meaning we can't just do a computation and return the value
- Instead, we approximate function values with iterative methods
- Examples include:
  - sine (with Taylor series expansion)
  - square root (as we'll do in this lecture)
  - optimize a cost or error function (e.g., gradient descent in the introduction to machine learning course)

# Babylonian method for sqrt

- The idea: pick an initial estimate,  $x_0$ , and then iterate with better and better estimates,  $x_i$ , using the ([Babylonian method](#)) recurrence relation:

$$x_{i+1} = \frac{1}{2} \left( x_i + \frac{n}{x_i} \right)$$

- This process relies on the midpoint of  $x_i$  and  $\frac{n}{x_i}$  getting closer and closer to the square root of  $n$
- The amazing thing is that the iteration converges very quickly

# The goal

- Our goal is to write a function that takes a single number and returns its square root
- What do we know about the function before thinking about code?
- Well, we have a clear description of the problem per the recurrence relation, and we also have the function signature:

```
def sqrt(n): ...
```

- Because we are implementing a recurrence relation, we know that we will have a loop that computes  $x_{i+1}$  from  $x_i$

# Convergence

$$x_{i+1} = \frac{1}{2} \left( x_i + \frac{n}{x_i} \right)$$

- The terminating condition of the loop is when we have reached convergence or close to it
- Convergence just means that  $x_{i+1}$  is pretty close to  $x_i$
- Because we can never compare to real numbers for equality, we have to check for the difference being smaller than some precision like 0.00000001

# Algorithm

- Iterative methods all share the same basic outline
- Python does not have a repeat-until loop so we fake it with an infinite loop containing a conditional that breaks out upon convergence

*set  $x_0$  to initial value*

*repeat:*

$x_{i+1} = \text{function-giving-next-value}(x_i)$

*until  $\text{abs}(x_{i+1} - x_i) < \text{precision}$*

*return  $x_{i+1}$*

*set  $x_0$  to initial value*

*while True:*

$x_{i+1} = \text{function-giving-next-value}(x_i)$

*if  $\text{abs}(x_{i+1} - x_i) < \text{precision}$*

*return  $x_{i+1}$*

# Implementation

- The translation to Python is fairly straightforward but notice that we don't need to track all  $x_i$ ; we just need the previous/current

```
def sqrt(n):  
    "compute square root of n"  
    PRECISION = 0.00000001 # stop iterating when we converge with this delta  
    x_0 = 1.0 # pick any old initial value  
    x_prev = x_0  
    while True: # Python doesn't have repeat-until loop so fake it  
        #print(x_prev)  
        x_new = 0.5 * (x_prev + n/x_prev)  
        if abs(x_new - x_prev) < PRECISION:  
            return x_new  
        x_prev = x_new # x_{i+1} becomes x_i (previous value)
```

```
sqrt(100)
```

```
10.0
```

# Testing our implementation

- To test our square root approximation, we can compare it to **math.sqrt()** and use numpy's **isclose()** to do the comparison

```
import numpy as np
def check(n):
    assert np.isclose(sqrt(n), np.sqrt(n))
def test_sqrt():
    check(125348)
    check(89.2342)
    check(100)
    check(1)
    check(0)

test_sqrt()
```



# Exercise

- Go to the [notebook version of this lecture](https://github.com/parr/msds501/blob/master/labs/sqrt.ipynb)[1] and do the exercise at the bottom
- Try not to cut and paste the code; see if you can implement the recurrence relation yourself
- Then, had a print statement so you can track how the  $x_i$  values converge as we iterate

```
sqrt(125348.0)
1.0
62674.5
31338.249992
15671.1249162
7839.56178812
3927.77547356
1979.84435152
1021.5781996
572.139273508
395.612894667
356.228988269
354.051888518
354.045194918
354.045194855
```

[1] <https://github.com/parr/msds501/blob/master/labs/sqrt.ipynb>