Lecture Notes
on
Orders of Convergence

by
Kenny Erleben
2016

Applying Numerical Methods

- Numerical methods are often iterative in nature. They take one computational step that yields the next iterate
- We wish to know if the iterations converge to a solution
- We wish to know how many steps we need to take to obtain a desired accuracy
- We wish to know how expensive the computational step is

Applying Numerical Methods

- Convergence theorems are going to give us “conditions” under which we know we will converge to a solution

- Convergence theorems are going to tells us the “rate” at which we will converge (how fast the error drops to zero). We will study the definitions of order of convergence rates to know what this means

- To study how errors behave little “o” and big “O” are tools often being used. Big “O” is also often used to analyse how expensive a computational step is

Applying Numerical Methods

- For now we have not yet “really” implemented any iterative numerical methods for solving any “problems”. However, we can imagine the iterative nature of such methods, and for each step the methods take we can imagine computing a measure of the current error (a real number).

- As computation goes on the iterative numerical method will create a sequence of such error values

- Hence, our learning will begin with infinite sequences of numbers.

Infinite Sequence of Numbers

$$\{x_n\} = \{x_0, x_1, x_2, x_3, \ldots\}$$

A Simple Example

$$x_n = \frac{1}{n}, \qquad \forall n > 0$$

Variation in Notation

$$x_n = \frac{1}{n}, \qquad n \geq 1$$

$$\{x_n\} = \left\{\frac{1}{n}\right\}, \qquad n \geq 1$$

$$\{x_n\}_1^K = \left\{\frac{1}{n}\right\}_1^K, \qquad K \geq 1$$

Analysing a Sequence of Numbers

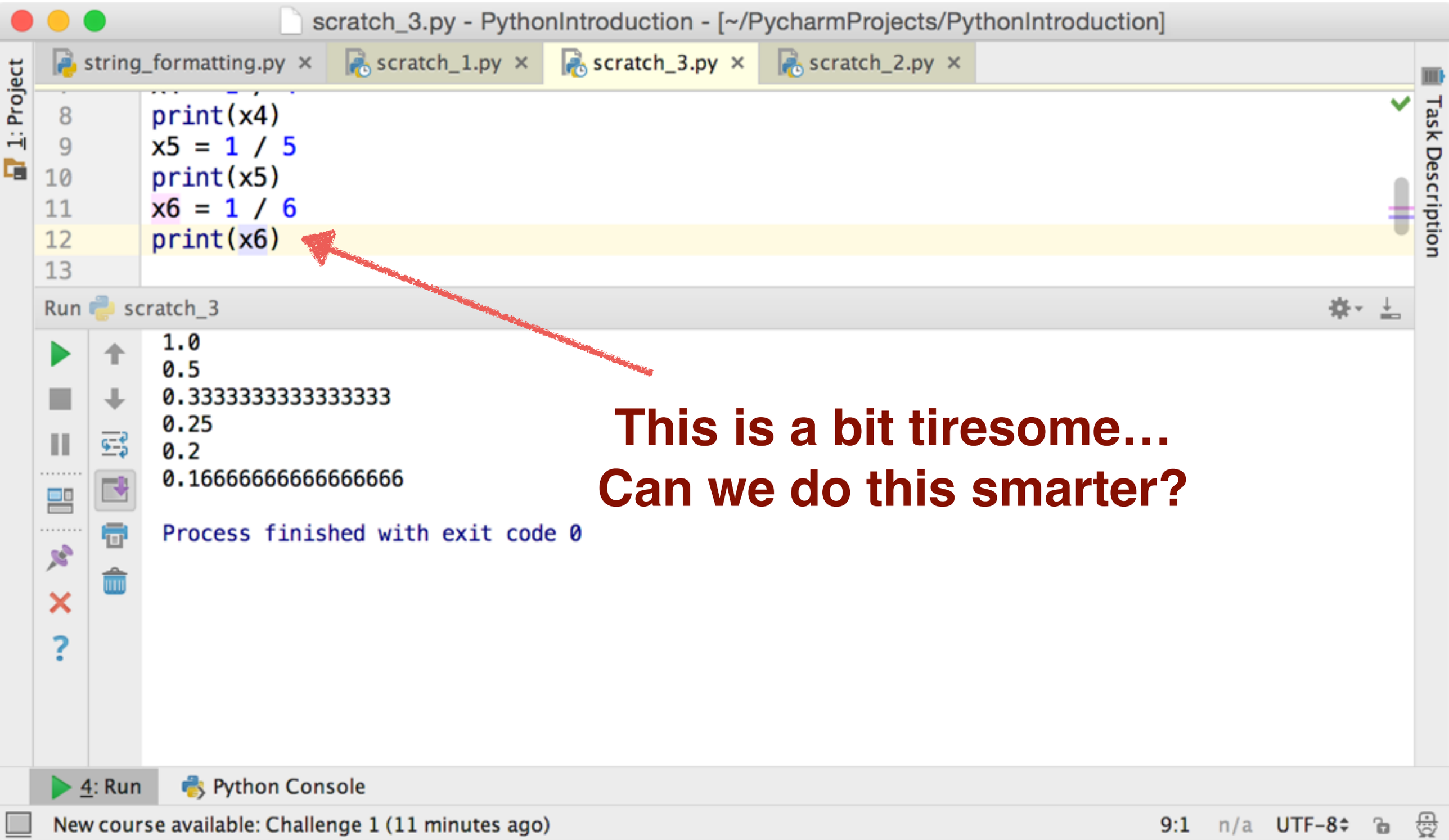
$$\lim_{n \to \infty} x_n = x^*$$

First Question: Does the sequence of numbers converge or not?

A Graphically Experimental Approach

Use a computer to compute and plot numbers to get an idea of the tendency

Sequence of Numbers in Python



Sequence of Numbers in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1 for n in range(1,100):
2     xn = 1 / n
3     print(xn)
4
```

Run scratch_3

```
0.011494252873563218
0.011363636363636364
0.011235955056179775
0.011111111111111112
0.010989010989010989
0.010869565217391304
0.010752688172043012
0.010638297872340425
0.010526315789473684
0.010416666666666666
0.010309278350515464
0.010204081632653061
0.010101010101010102

Process finished with exit code 0
```

Less code to write

4: Run | Python Console

New course available: Challenge 1 (today 12:25) 90:21 LF UTF-8

Sequence of Numbers in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1  x = [ 1 / n for n in range(1,100)]
2
3  for xn in x:
4      print(xn)
5
```

Run scratch_3

```
0.011494252873563218
0.011363636363636364
0.011235955056179775
0.011111111111111112
0.010989010989010989
0.010869565217391304
0.010752688172043012
0.010638297872340425
0.010526315789473684
0.010416666666666666
0.010309278350515464
0.010204081632653062
0.010101010101010102

Process finished with exit code 0
```

Even more compact

This output is not so easy to read

4: Run | Python Console

New course available: Challenge 1 (3 minutes ago) 102:1 n/a UTF-8

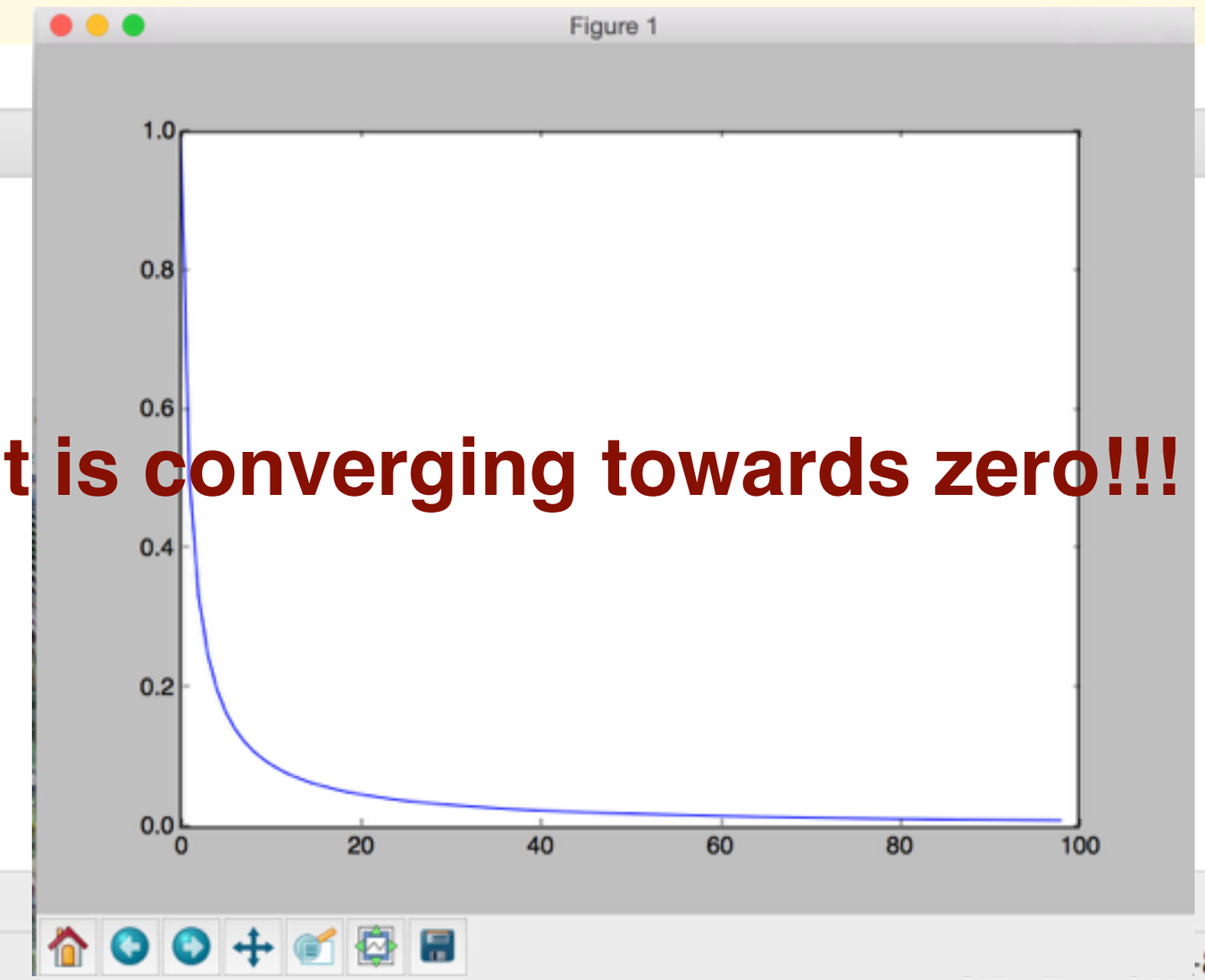
Plotting Sequence of Numbers in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1  import matplotlib.pyplot as plt
2
3  x = [1 / n for n in range(1, 100)]
4
5  plt.plot(x)
6  plt.show()
```

Let us plot output instead



!!!Looks like it is converging towards zero!!!

Run scratch_3

4: Run | Python Console

New course available: Challenge 1 (41 minutes ago)

Plotting Sequence of Numbers in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

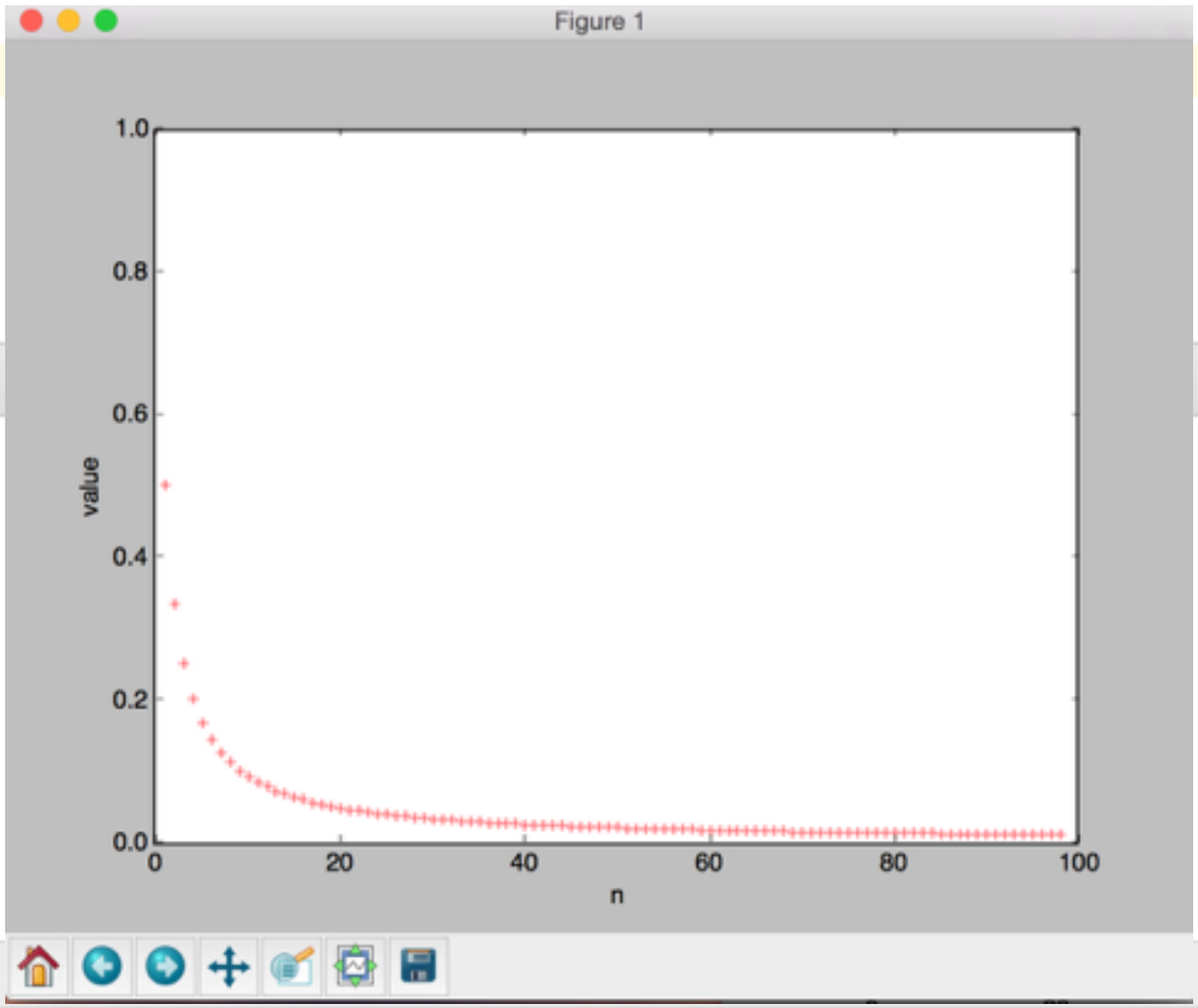
string_formatting.py | scratch_1.py | scratch_3.py | scratch_2.py

```
1  import matplotlib.pyplot as plt
2
3  x = [1/ n for n in range(1,100)]
4
5  plt.plot(x,'r+')
6  plt.ylabel('value')
7  plt.xlabel('n')
8  plt.show()
9
```

Sprinkles on top

Run: scratch_3 | scratch_3 | scratch_3

```
Process finished with exit code 0
```



4: Run | Python Console

New course available: Challenge 1 (today 12:25)

3:1 LF UTF-8

Formal Proofs

—

We need Limits

We write

$$\lim_{n\to\infty} x_n = c$$

If and only if

$$\forall \varepsilon > 0,\ \exists N > 0$$

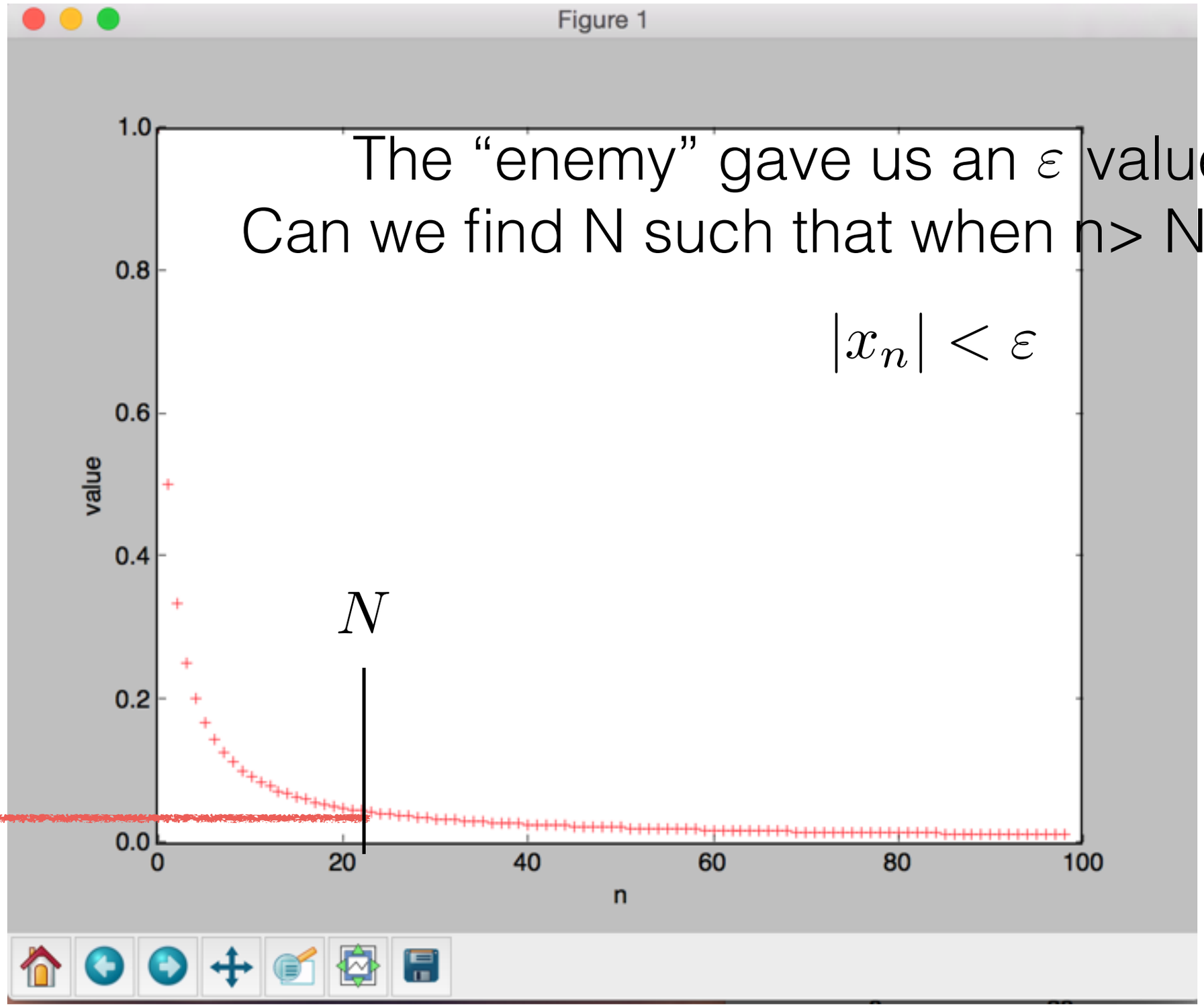
such that

$$n > N \quad \Rightarrow \quad |x_n - c| < \varepsilon$$

Recipe

1) **Somebody gives you an epsilon value**
2) **You prove you can find N-value so the inequality is true**

Idea for Proof



The “enemy” gave us an ε value

Can we find N such that when n> N then

$$|x_n| < \varepsilon$$

Formal Proof for Simple Example

Prove

$$\lim_{n\to\infty} \frac{1}{n} = 0$$

Given any $\varepsilon > 0$ and $n > N$ then

$$\left|\frac{1}{n}\right| = \frac{1}{n} < \frac{1}{N}$$

Guess a N-value

$$N > \frac{1}{\varepsilon}$$

Verify inequality from formal definition

$$\left|\frac{1}{n}\right| < \frac{1}{N} < \frac{1}{\frac{1}{\varepsilon}} = \varepsilon$$

“True” as wanted so our guess proved existence of N-value

Reflections on usefulness of convergence proofs

- Trivial (not the same as easy) to determine convergence either formally or experimentally

- Knowing we have convergence is nice…. but we want to use computers, so we want things to compute fast

- Hence, we often wish to categorize how fast we “converge” towards a solution. This is often termed “rate of convergence”

The Absolute Error by definition

$$e_n = |x_n - x^*|$$

This will make equations in definitions simpler to read

Convergence Rate Definitions

convergence constants…
smaller is better

Linear $e_{n+1} \leq c\, e_n, \quad \text{where } 0 < c < 1, n \geq N$

Super Linear $e_{n+1} \leq c_n\, e_n, \quad \text{where } c_n \to 0 \text{ for } n \to \infty < 1, n \geq N$

Quadratic $e_{n+1} \leq C\, e_n^2, \quad \text{where } C > 0, n \geq N$

p-order $e_{n+1} \leq C\, e_n^p, \quad \text{where } C > 0, p > 2, n \geq N$

The “equal” part is the worst behaviour

Holds for n big enough!

Linear Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

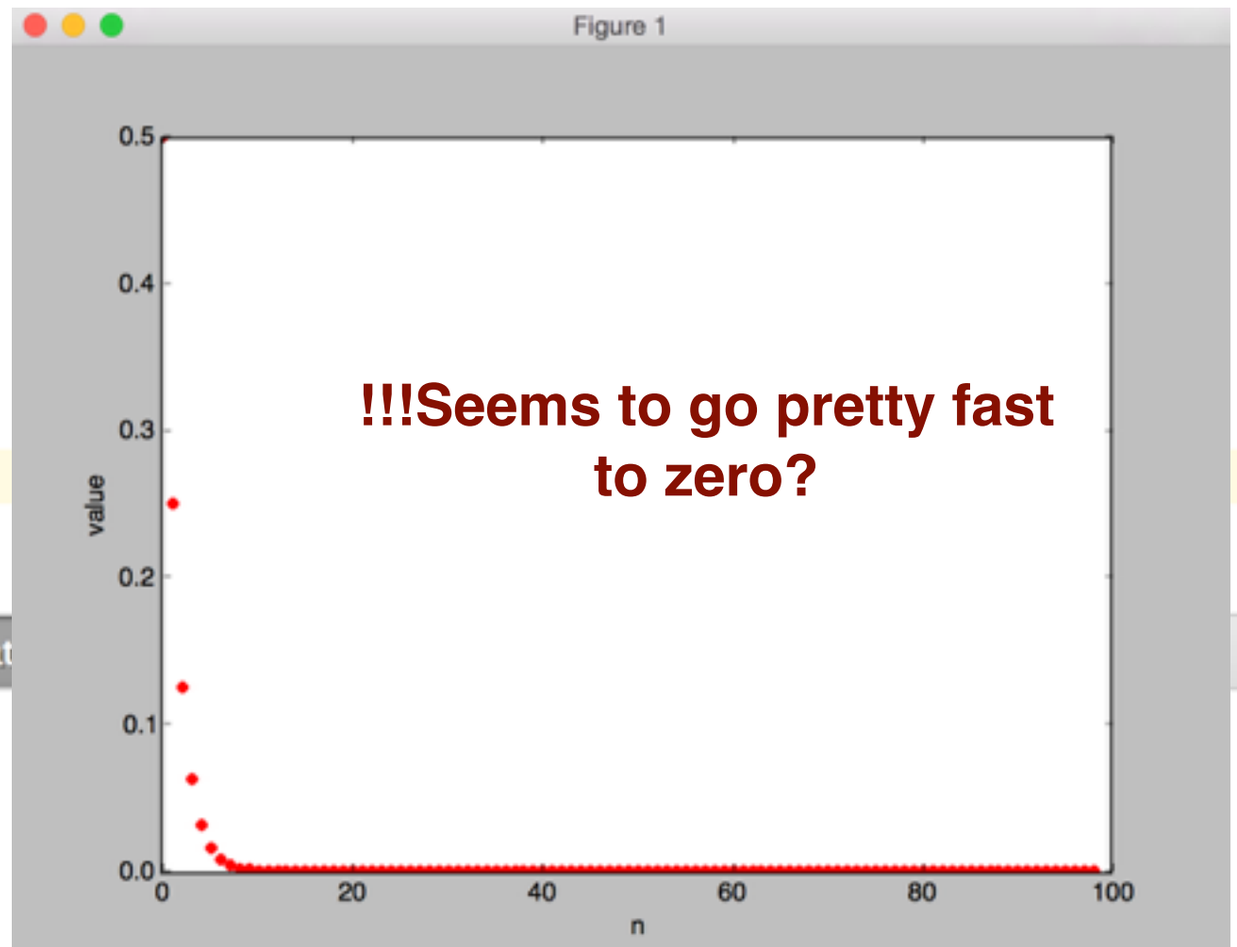
$$e_{n+1} = \frac{1}{2} e_n$$

Plot “Linear” Archetype Example

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py | scratch_1.py | scratch_3.py | scratch_2.py

```
1  import matplotlib.pyplot as plt
2
3
4  conv = []
5  e = 1.0
6  c   = 0.5
7  for n in range(1,100):
8      e = c * e
9      conv.append(e)
10 plt.plot(conv,'r.',markersize=10)
11 plt.ylabel('value')
12 plt.xlabel('n')
13 plt.show()
14
```



Run: scratch_3 | scratch_3 | scratch_3 | scrat

4: Run | Python Console

New course available: Challenge 1 (today 12:25)

13:11 LF UTF-8

Plot “Linear” Archetype Example
Convergence Constants Play a Big Role

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1  import matplotlib.pyplot as plt
2
3  constants = [100/n for n in range(101,200)]
4  for c in constants:
5      conv = []
6      e = 1.0
7      for n in range(1,100):
8          e = c * e
9          conv.append(e)
10     plt.plot(conv,linewidth=2)
11 plt.ylabel('value')
12 plt.xlabel('n')
13 plt.show()
14
```

Run: scratch_3 | scratch_3 | scratch_3 | scrat

```
Process finished with exit code 0
```

4: Run | Python Console

New course available: Challenge 1 (today 12:25) 3:1 LF UTF-8

1: Project | Task Description

Linear Convergence Rate Caveats

Given

$$e_{n+1} = \frac{1}{2} e_n$$

for $n \to \infty$ we have $e_n \to 0$

But if we have an initial error we also observe

$$e_n > 0, \ \forall n$$

Never going to reach zero error!

Super-Linear Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

$$e_{n+1} = c_n \, e_n$$

$$c_n = \frac{1}{n}$$

The error will get to zero if run long enough (theoretically)!
This is different from linear convergence rate

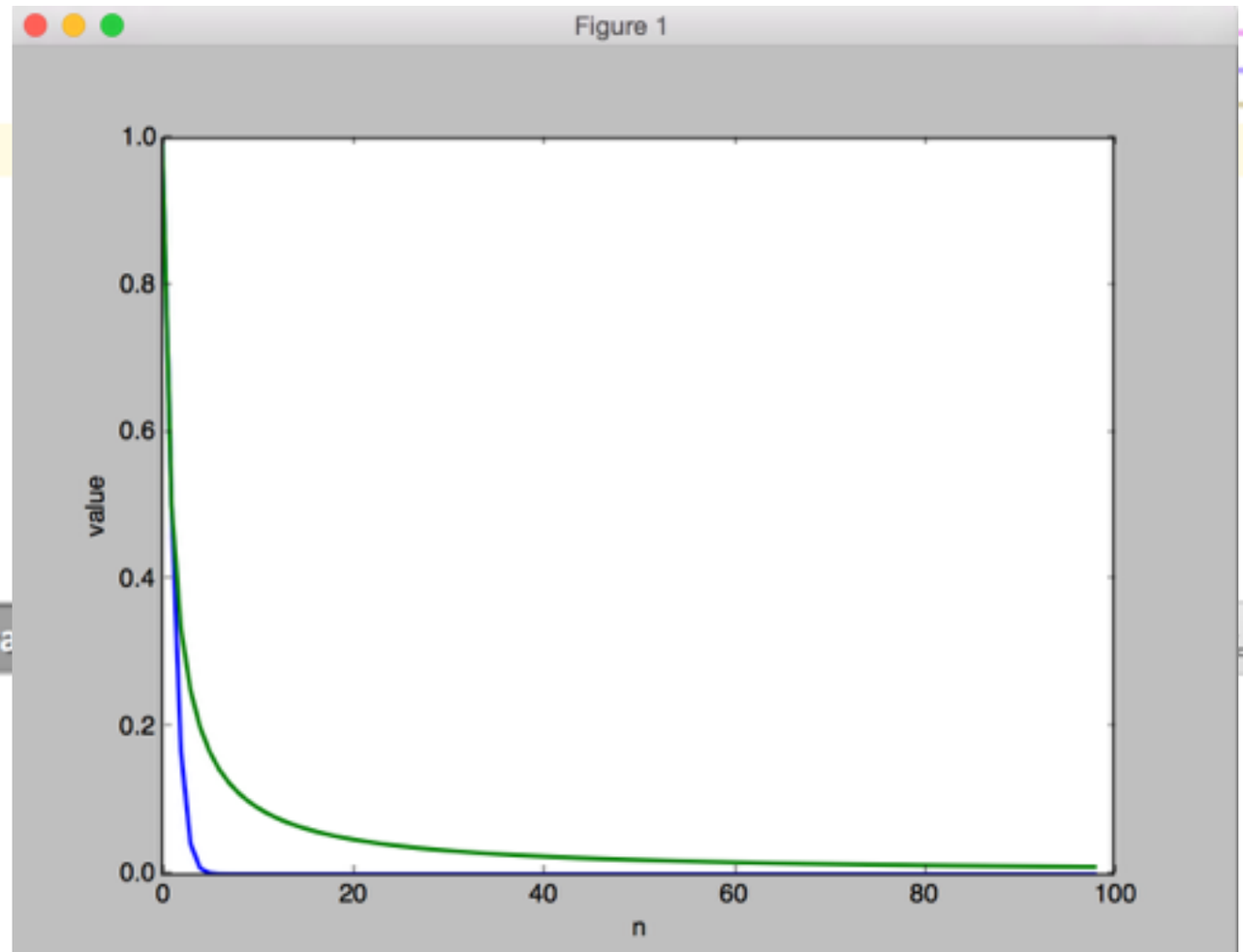
Super Linear Convergence Rate in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1  import matplotlib.pyplot as plt
2
3  conv = []
4  const = []
5  e = 1.0
6  for n in range(1,100):
7      c = 1/n
8      e = c * e
9      conv.append(e)
10     const.append(c)
11 plt.plot(conv, linewidth=2)
12 plt.plot(const, linewidth=2)
13 plt.ylabel('value')
14 plt.xlabel('n')
15 plt.show()
```

Seems faster than linear case….
but hard to tell apart just by looking



Run: scratch_3 | scratch_3 | scratch_3 | scra

4: Run | Python Console

New course available: Challenge 1 (today 12:25) 7:12 LF UTF-8

Quadratic Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

$$e_{n+1} = 2\, e_n^2$$

Plot Quadratic Convergence Rate in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

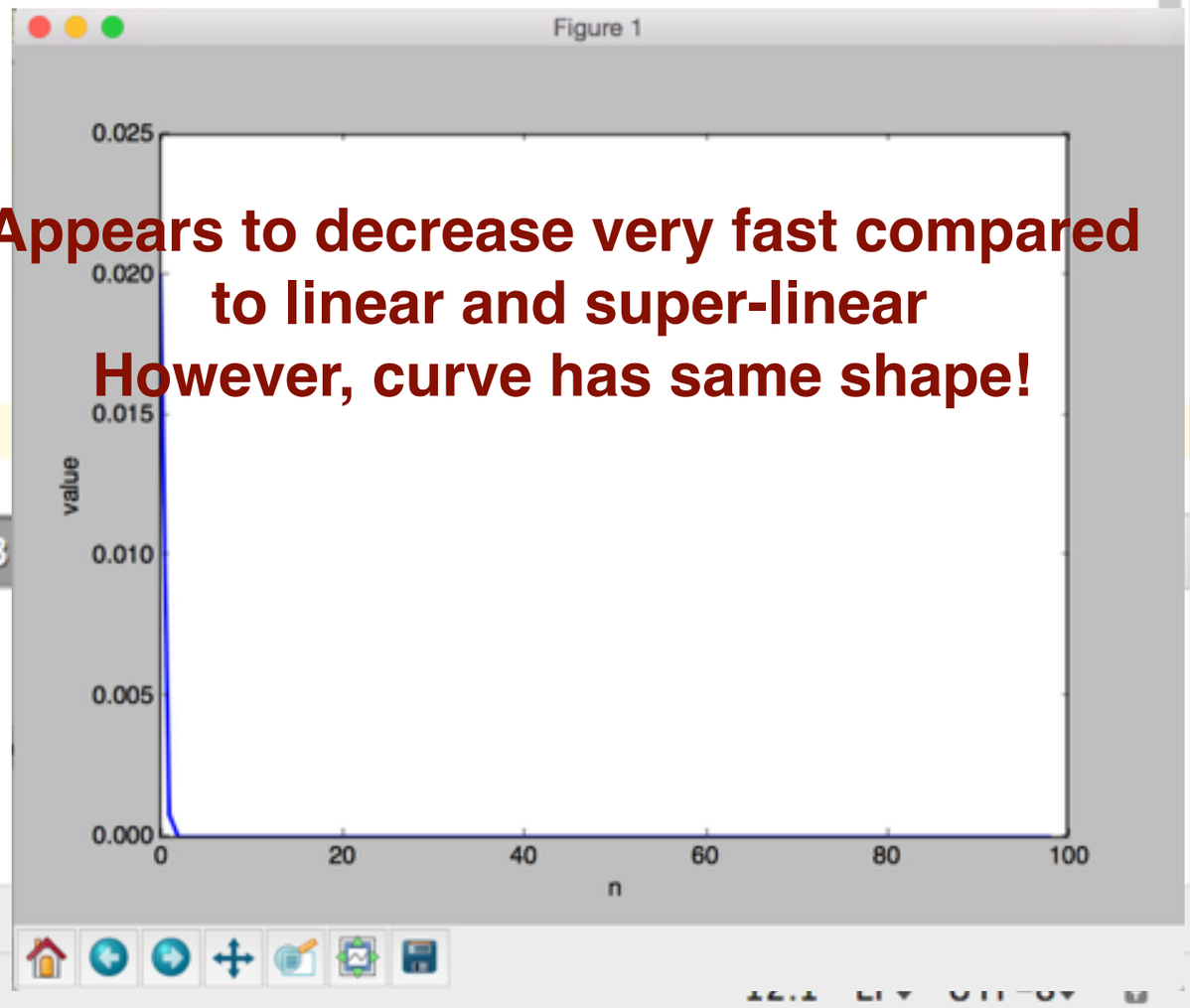
string_formatting.py × | scratch_1.py × | scratch_3.py × | scratch_2.py ×

```
1  import matplotlib.pyplot as plt
2
3  conv = []
4  e = 0.1
5  for n in range(1,100):
6      e = 2 * e**2
7      conv.append(e)
8  plt.plot(conv, linewidth=2)
9  plt.ylabel('value')
10 plt.xlabel('n')
11 plt.show()
12
```

Run: scratch_3 | scratch_3 | scratch_3 | scratch_3

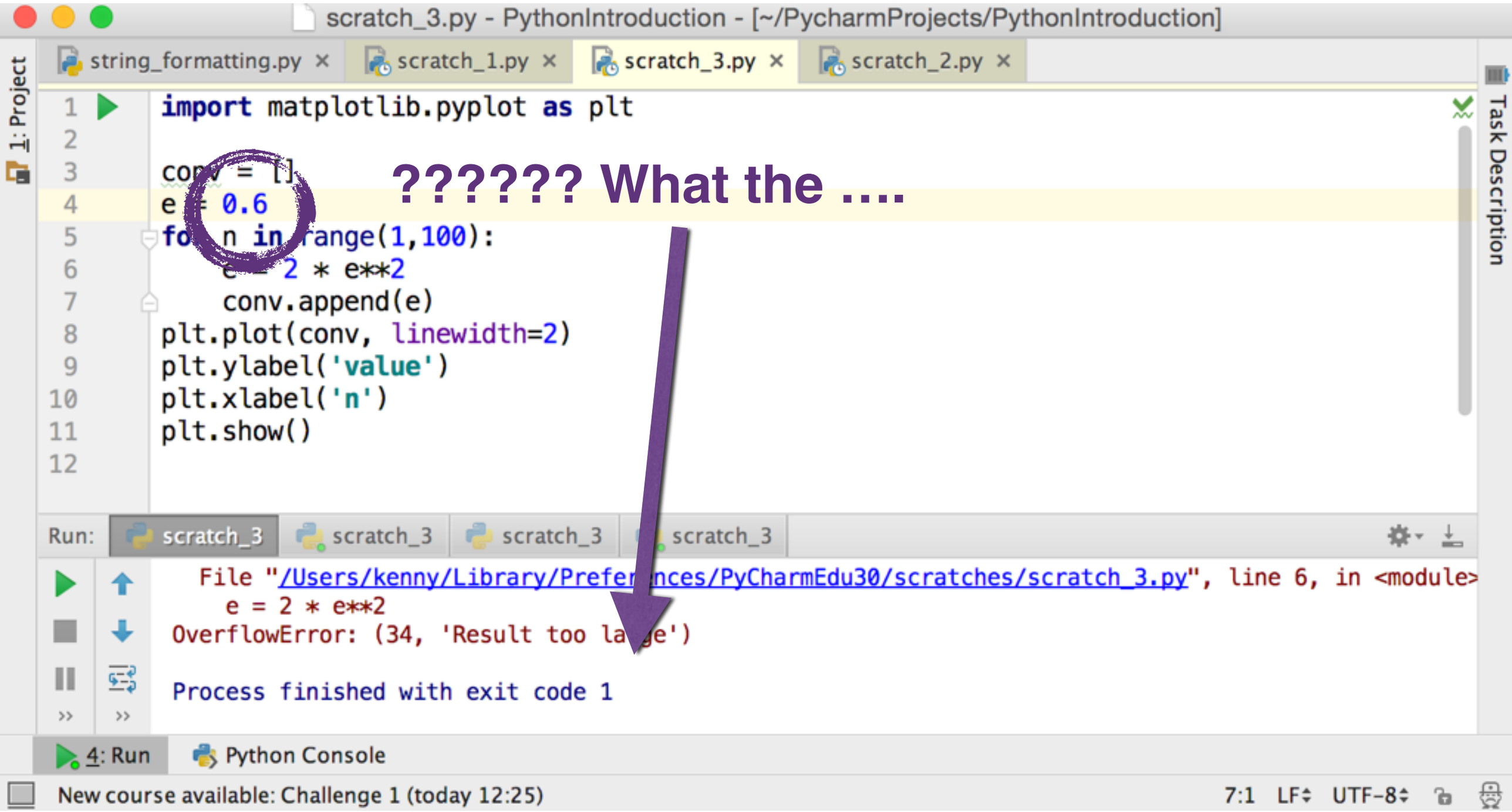
4: Run | Python Console

New course available: Challenge 1 (today 12:25)



Appears to decrease very fast compared to linear and super-linear
However, curve has same shape!

Plot Quadratic Convergence Rate in Python



Plot Quadratic Convergence Rate in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py | scratch_1.py | scratch_3.py | scratch_2.py

```
1  import matplotlib.pyplot as plt
2
3  conv = []
4  e = 0.5
5  for n in range(1,100):
6      e = 2 * e**2
7      conv.append(e)
8  plt.plot(conv, linewidth=2)
9  plt.ylabel('value')
10 plt.xlabel('n')
11 plt.show()
12
```

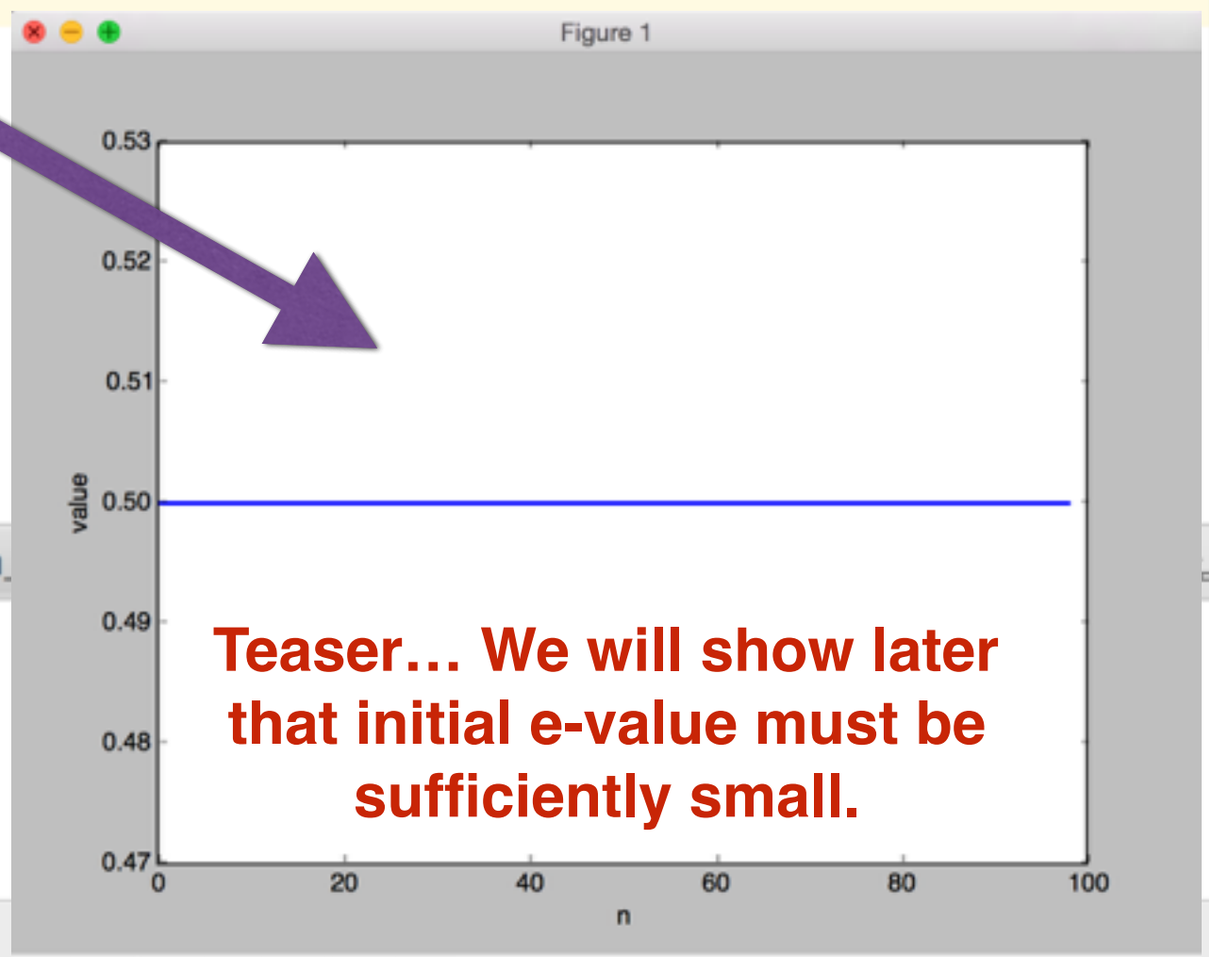
???? Huba ....

Run: scratch_3 | scratch_3 | scratch_3 | scratch_

```
Process finished with exit code 0
```

4: Run | Python Console

New course available: Challenge 1 (today 12:25)



Teaser… We will show later that initial e-value must be sufficiently small.

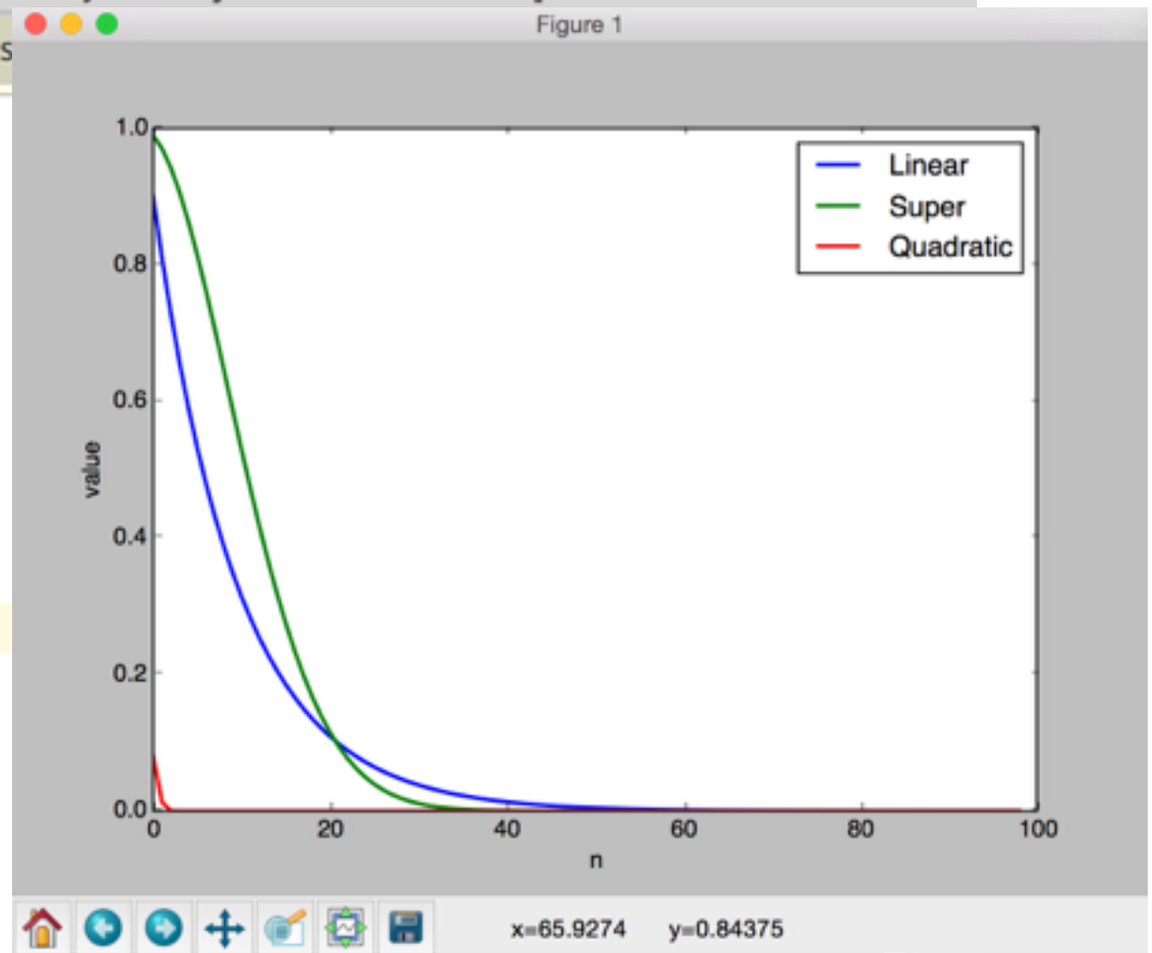
Can you tell what plots are what rates? Only by looking at the plots?

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × scratch_1.py × scratch_3.py ×

1: Project

```
1  import matplotlib.pyplot as plt
2
3  conv1 = []
4  conv2 = []
5  conv3 = []
6  e1 = 1.0
7  e2 = 1.0
8  e3 = 0.2
9  for n in range(1,100):
10     e1 = 0.9 * e1
11     e2 = 100/(n+100) * e2
12     e3 = 2 * e3**2
13     conv1.append(e1)
14     conv2.append(e2)
15     conv3.append(e3)
16 plt.plot(conv1, linewidth=2, label='Linear')
17 plt.plot(conv2, linewidth=2, label='Super')
18 plt.plot(conv3, linewidth=2, label='Quadratic')
19 plt.legend()
20 plt.ylabel('value')
21 plt.xlabel('n')
22 plt.show()
```



Run: scratch_3 scratch_3 scratch_3 scratch_3

4: Run Python Console

New course available: Challenge 1 (today 12:25) 11:26 LF UTF-8

We wish to be able to classify the convergence rate “easily”

- It will allow us to verify if our implementations are working as theory tell us we should expect them to work (Experimentally verification of convergence theorems)

- Notice the “proof” of a certain rate is only half of the story… The other is whether one code delivers on this too!!!

Linear Convergence Rate Again

$$
\begin{aligned}
e_1 &= c\, e_0 \\
e_2 &= c^2\, e_0 \\
e_3 &= c^3\, e_0 \\
&\vdots \\
e_n &= c^n\, e_0
\end{aligned}
$$

Taking the logarithm

$$
\begin{aligned}
\log e_n &= \log\left(c^n\, e_0\right) \\
\log e_n &= \log c^n + \log e_0 \\
\underbrace{\log e_n}_{y} &= \underbrace{(\log c)}_{a}\underbrace{n}_{x} + \underbrace{\log e_0}_{b}
\end{aligned}
$$

We have a straight-line in a log plot

Quadratic Convergence Rate Again

$$e_1 = C\, e_0^2$$

$$e_2 = C\, e_1^2 = C^3\, e_0^4$$

$$\vdots$$

$$e_n = C\, e_{n-1}^2 = C^{2^n-1}\, e_0^{2^n}$$

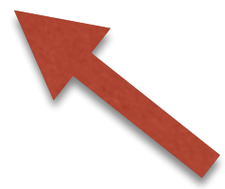
Taking logarithm

$$\log e_n = \log\left(C^{2^n-1}\, e_0^{2^n}\right)$$

$$\log e_n = \log\left(C^{2^n-1}\right) + \log\left(e_0^{2^n}\right)$$

$$\log e_n = (2^n - 1)\, \log C + 2^n\, \log e_0$$

$$\log e_n = 2^n \log\left(C e_0\right) - \log C$$



Important! C must be positive

Quadratic Convergence Rate Again

So

$$\log e_n = \log\left(C^{2^n-1}\, e_0^{2^n}\right)$$

$$\log e_n = \log\left(C^{2^n-1}\right) + \log\left(e_0^{2^n}\right)$$

$$\log e_n = (2^n - 1)\, \log C + 2^n\, \log e_0$$

$$\log e_n = 2^n \log\left(C e_0\right) - \log C$$

We must have

$$e_n \to 0 \;\Rightarrow\; log(e_n) \to -\infty$$

Only possible if

$$\log\left(C e_0\right) < 0 \quad \text{or} \quad e_0 < \frac{1}{C}$$

Important! Initial error has to be small enough!

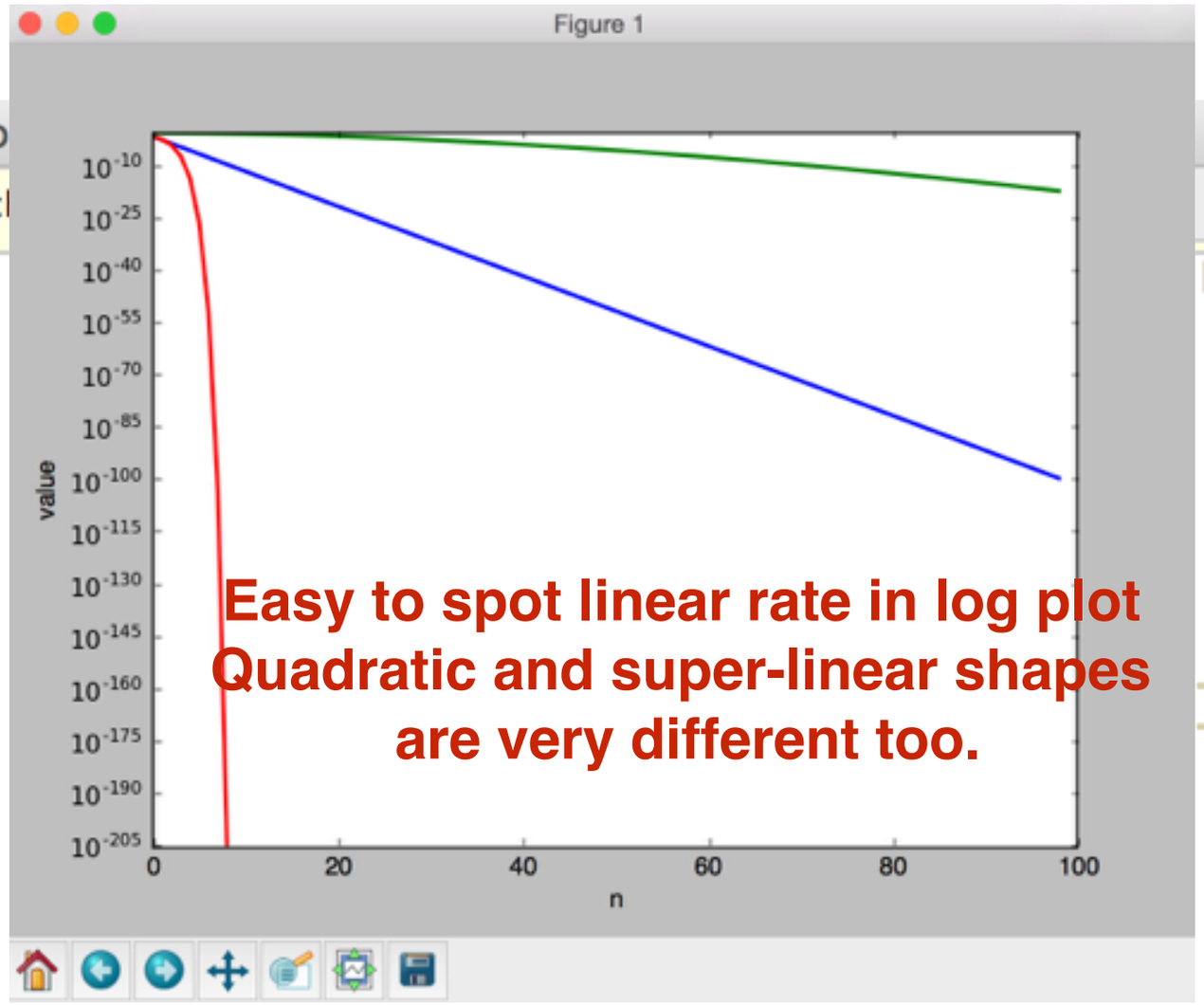
Log-Scale Shows Linear Convergence Rate as a straight line

scratch_3.py - PythonIntroductio

string_formatting.py × scratch_1.py × scratcl

1: Project

```
1  import matplotlib.pyplot as plt
2
3  conv1 = []
4  conv2 = []
5  conv3 = []
6  e1 = 1.0
7  e2 = 1.0
8  e3 = 0.2
9  for n in range(1,100):
10     e1 = 0.1 * e1
11     e2 = 100/(n+100) * e2
12     e3 = 2 * e3**2
13     conv1.append(e1)
14     conv2.append(e2)
15     conv3.append(e3)
16 plt.semilogy(conv1, linewidth=2, label='Linear')
17 plt.semilogy(conv2, linewidth=2, label='Super')
18 plt.semilogy(conv3, linewidth=2, label='Quadratic')
19 plt.ylabel('value')
20 plt.xlabel('n')
21 plt.show()
22
```



Task Description

Can we get straight lines for quadratic convergence rate?

We have

$$\log e_n = 2^n \log(Ce_0) - \log C$$

Observe

$$e_n \to 0 \quad \Rightarrow \quad \log e_n \to -\infty$$

Means $\log(\log e_n)$ is undefined

If we flip sign then

$$\log(-\log e_n) = \log(-2^n \log Ce_n + \log(\log C))$$

This will never be affine function in n

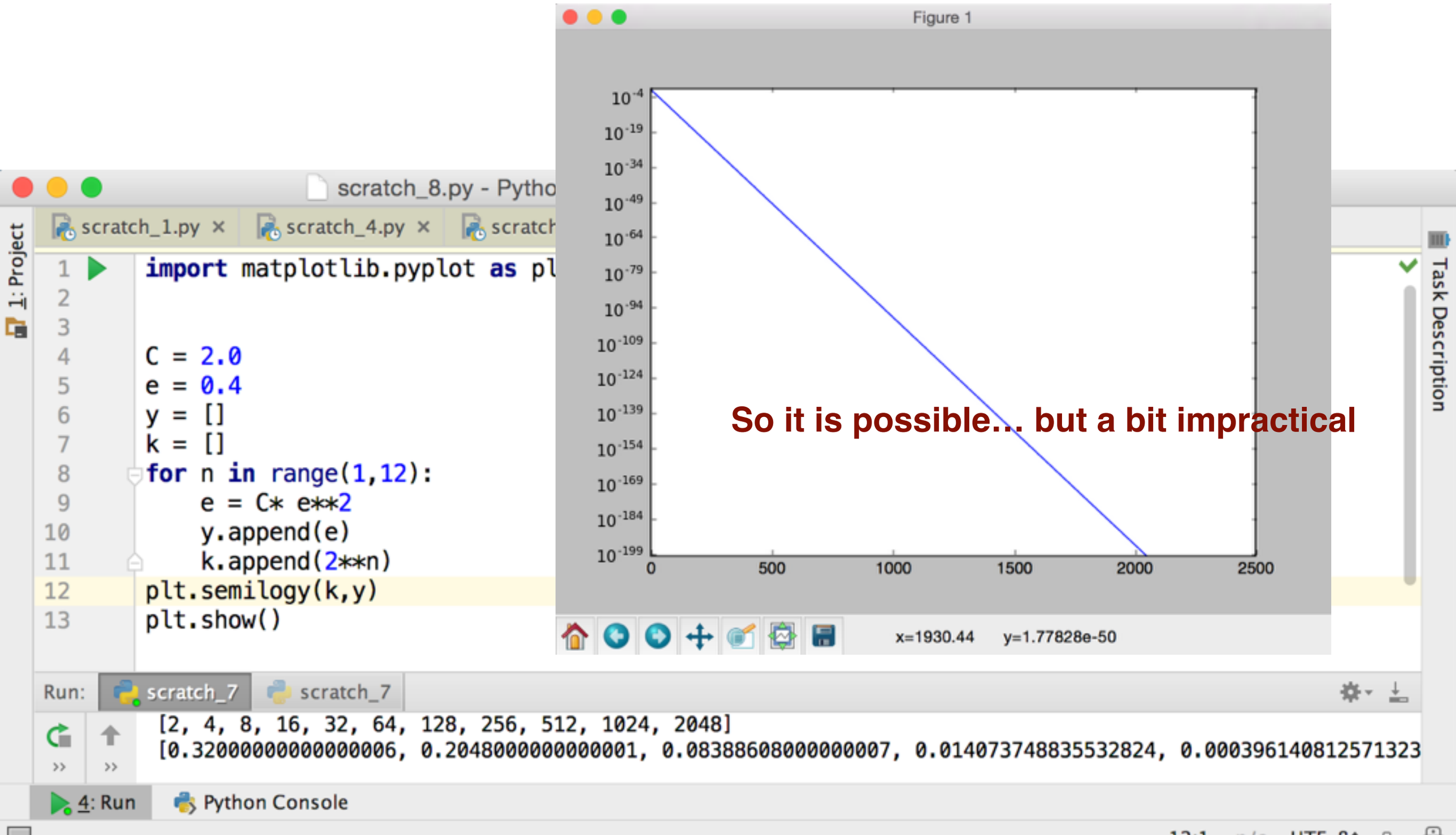
Can we get straight lines for quadratic convergence rate?

Let us make a substitution

$$\log e_n = \underbrace{2^n}_{\equiv k} \log C e_n - \log C$$

If we plot log error as a function of k then we have a affine function

Quadratic convergence rate as a straight line in Python



Big O definition for Sequences

We write

$$x_n = \mathcal{O}(y_n) \qquad n \to \infty$$
$$x_n = \mathcal{O}(y_n) \qquad n \to 0$$
$$x_n = \mathcal{O}(y_n) \qquad n \to n^*$$

This means there exist $C > 0$ and $N > 0$ such that

$$|x_n| \leq C|y_n| \quad n \geq N$$
$$|x_n| \leq C|y_n| \quad n \leq N$$
$$|x_n| \leq C|y_n| \quad |n^* - n| \leq N$$

Big O Example

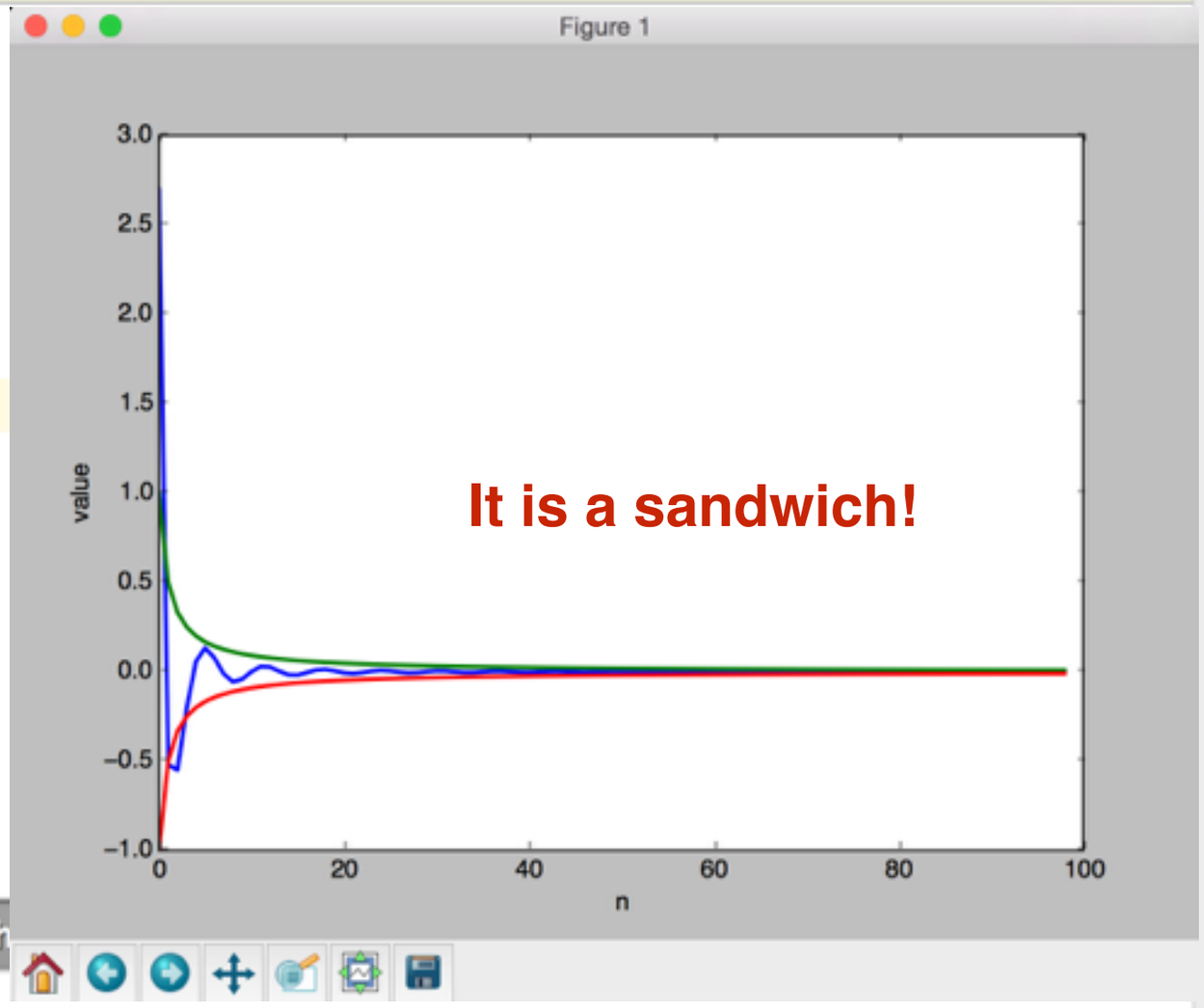
$$\frac{5\cos(n)}{n^2} = \mathcal{O}\left(\frac{1}{n}\right) \qquad n \to \infty$$

Big O Example in Python

scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string_formatting.py × scratch_1.py × scratch_3.py × scratch_2.py ×

```
1  import matplotlib.pyplot as plt
2  from math import cos
3
4  f = []
5  gp = []
6  gm = []
7  for n in range(1,100):
8      f.append(  5*cos(n) / n**2 )
9      gp.append(  1 / n )
10     gm.append(  -1 / n )
11 plt.plot(f, linewidth=2)
12 plt.plot(gp, linewidth=2)
13 plt.plot(gm, linewidth=2)
14 plt.ylabel('value')
15 plt.xlabel('n')
16 plt.show()
17
```



Run: scratch_3 scratch_3 scratch_3 scratch

4: Run Python Console

New course available: Challenge 1 (today 12:25) 8:17 LF UTF-8

Example Using Big “O” to tell how cost scales

Given $\vec{x}, \vec{y} \in \mathbf{R}^n$ what is the computational cost of

$$\text{cost}(\vec{x} \cdot \vec{y}) = \mathcal{O}\,(?)$$

Answer

$$\text{cost}(\vec{x} \cdot \vec{y}) = \mathcal{O}\,(n)$$

Given $A \in \mathbf{R}^{m \times n}$ what is the computational cost of

$$\text{cost}(A\,\vec{x}) = \mathcal{O}\,(?)$$

The n is called the problem size and big O tells how computations scale when n grows

Limit Cases Often Encountered when Analysing Errors and Costs

	Functions	Sequences
Infinity	$\lim_{x \to \infty} f(x) = c$	$\lim_{n \to \infty} x_n = c$
Value	$\lim_{x \to x^*} f(x) = c$	$\lim_{n \to 0} x_n = c$

What is a Limit?

We write

$$\lim_{x \to x^*} f(x) = c$$

If and only if

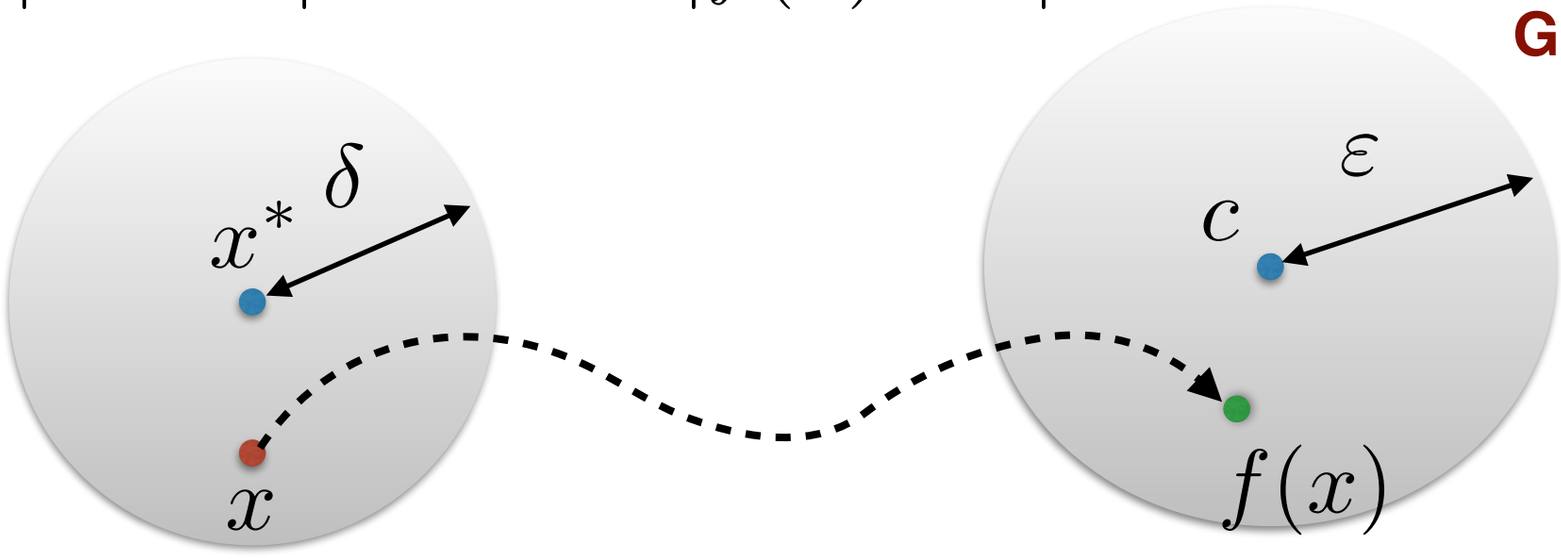
$$\forall \varepsilon > 0, \ \exists \delta > 0$$

such that

$$|x - x^*| < \delta \quad \Rightarrow |f(x) - c| < \varepsilon$$

Given any ε

Got to prove δ exists!



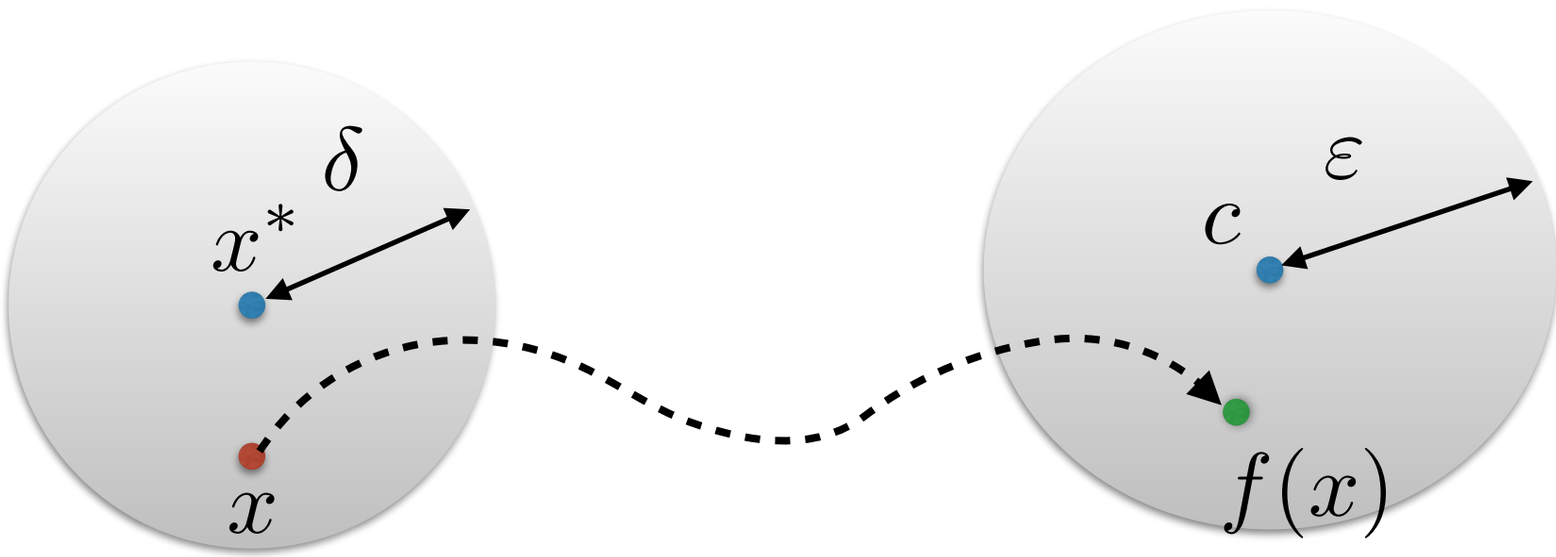
Geometric Interpretation of Limit

$$\lim_{x \to x^*} f(x) = c$$

Means

$$x \in \mathcal{B}_\delta(x^*) \;\Rightarrow\; f(x) \in \mathcal{B}_\varepsilon(c)$$

Got to prove δexists!



Given any ε

Limit Example

Prove for some fixed, positive integer n,

$$\lim_{x \to 0} x^n = 0$$

Observe $x^* = c = 0$ Pick any $\varepsilon > 0$

$$|x^n - c| = |x^n| = |x|^n < \varepsilon$$

$$|x - x^*| = |x - 0| = |x| < \sqrt[n]{\varepsilon}$$

$$\delta = \sqrt[n]{\varepsilon}$$

We proved that δ exists

Little o definition for sequences

We write

$$x_n = \mathbf{o}\left(y_n\right) \qquad n \to \infty$$
$$x_n = \mathbf{o}\left(y_n\right) \qquad n \to 0$$
$$x_n = \mathbf{o}\left(y_n\right) \qquad n \to n^*$$

This means

$$\lim_{n\to\infty} \left(\frac{x_n}{y_n}\right) = 0$$
$$\lim_{n\to 0} \left(\frac{x_n}{y_n}\right) = 0$$
$$\lim_{n\to n^*} \left(\frac{x_n}{y_n}\right) = 0$$

Interpretation x's goes more rapid to zero than y's