How to Study Evolution Using Scientific Python: Appendix

Yoav Ram

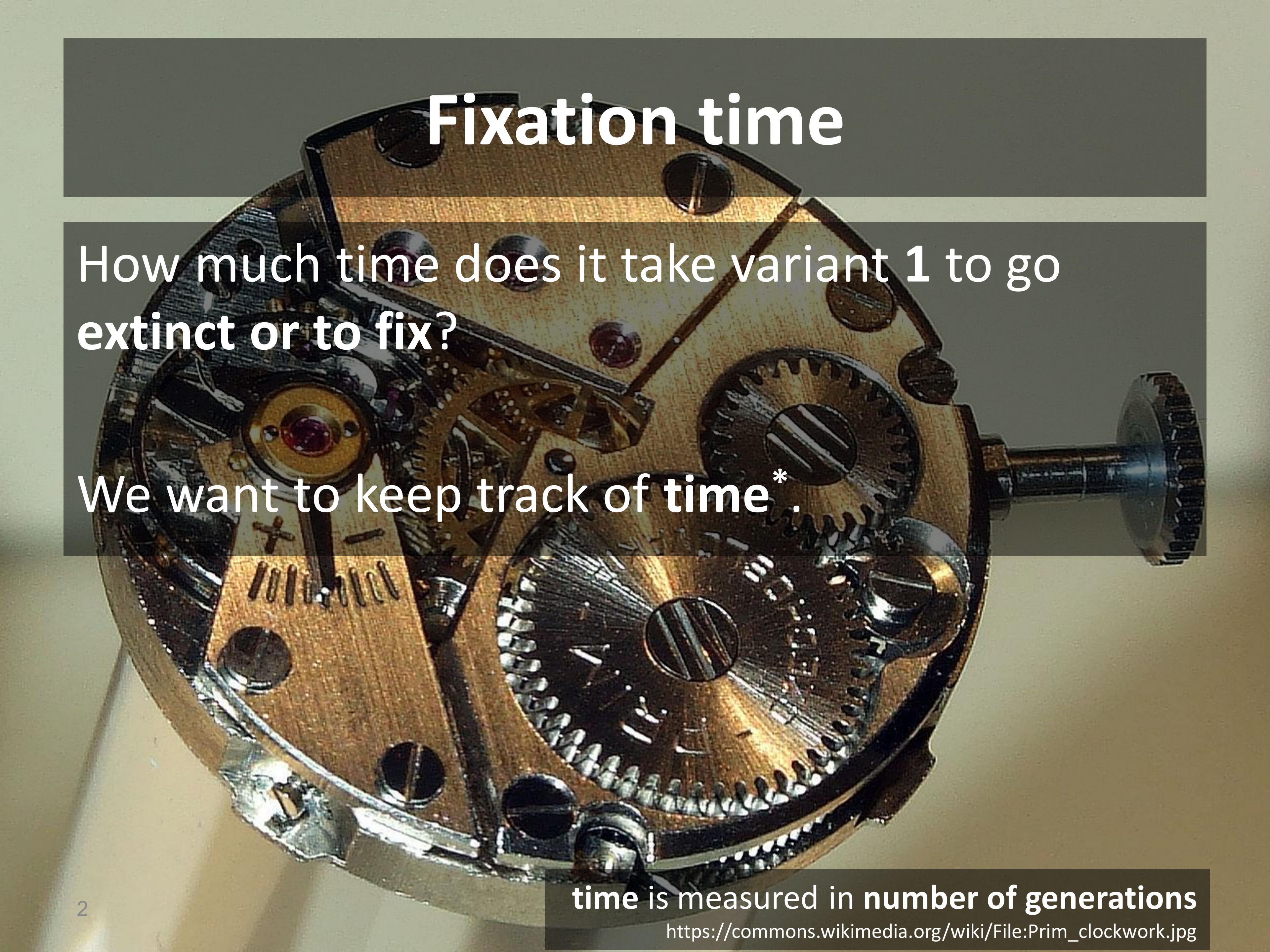
PyCon Israel 2016

Encyclopædia Britannica Online

Fixation time

How much time does it take variant **1** to go **extinct or to fix**?

We want to keep track of **time***.



* **time** is measured in **number of generations**

https://commons.wikimedia.org/wiki/File:Prim_clockwork.jpg

```python
def simulation(N, s, repetitions):
    n1 = np.ones(repetitions)
    T = np.empty_like(n1)
    update = (n1 > 0) & (n1 < N)
    t = 0

    while update.any():
        t += 1
        p = n1 * (1 + s) / (N + n1 * s)
        n1[update] = binomial(N, p[update])
        T[update] = t
        update = (n1 > 0) & (n1 < N)

    return n1 == N, T
```

`t` keeps track of time

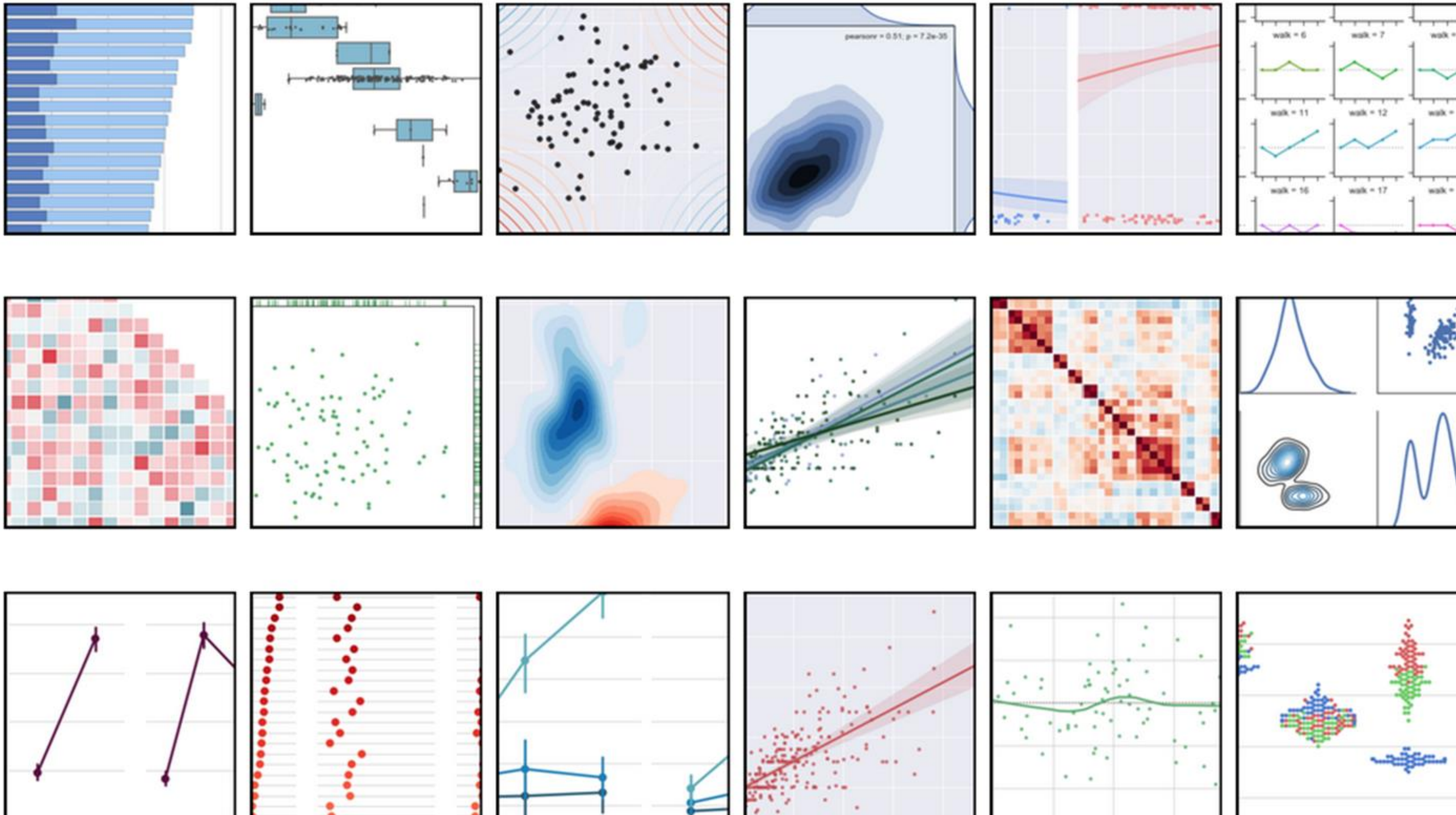
```python
def simulation(N, s, repetitions):
    n1 = np.ones(repetitions)
    T = np.empty_like(n1)
    update = (n1 > 0) & (n1 < N)
    t = 0

    while update.any():
        t += 1
        p = n1 * (1 + s) / (N + n1 * s)
        n1[update] = binomial(N, p[update])
        T[update] = t
        update = (n1 > 0) & (n1 < N)

    return n1 == N, T
```

T holds time for extinction/fixation

```python
def simulation(N, s, repetitions):
    n1 = np.ones(repetitions)
    T = np.empty_like(n1)
    update = (n1 > 0) & (n1 < N)
    t = 0

    while update.any():
        t += 1
        p = n1 * (1 + s) / (N + n1 * s)
        n1[update] = binomial(N, p[update])
        T[update] = t
        update = (n1 > 0) & (n1 < N)

    return n1 == N, T
```

Return both Booleans and times (T)

Statistical data visualization with Seaborn



Plot with Seaborn

```
from seaborn import distplot

fixations, times = simulation(…)

distplot(times[fixations])

distplot(times[~fixations])
```

Plot with Seaborn

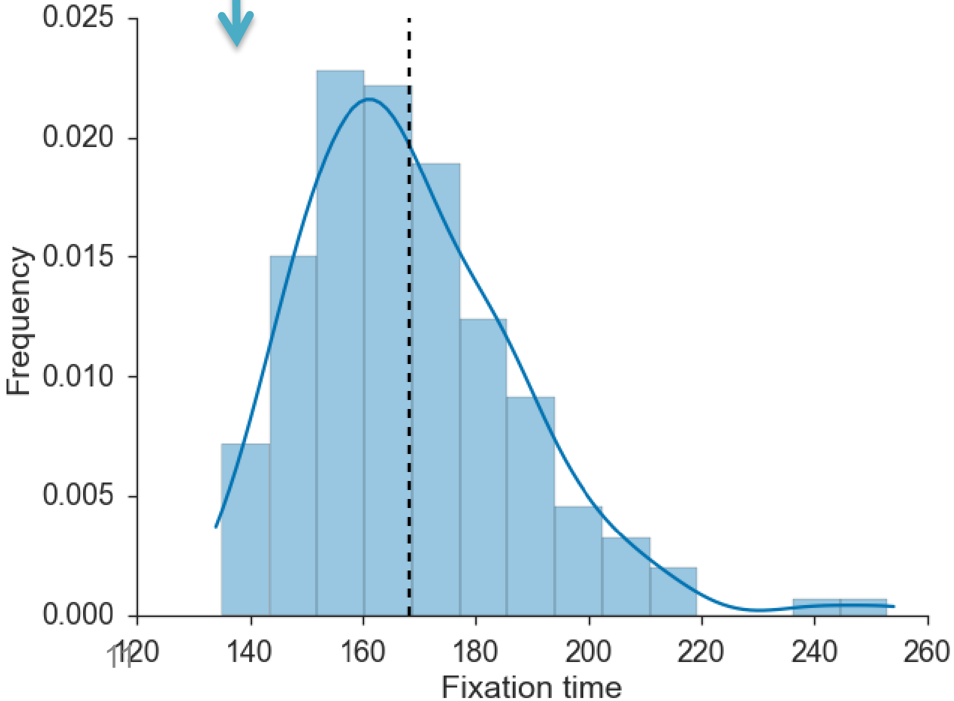
```
from seaborn import distplot

fixations, times = simulation(…)

distplot(times[fixations])

distplot(times[~fixations])
```

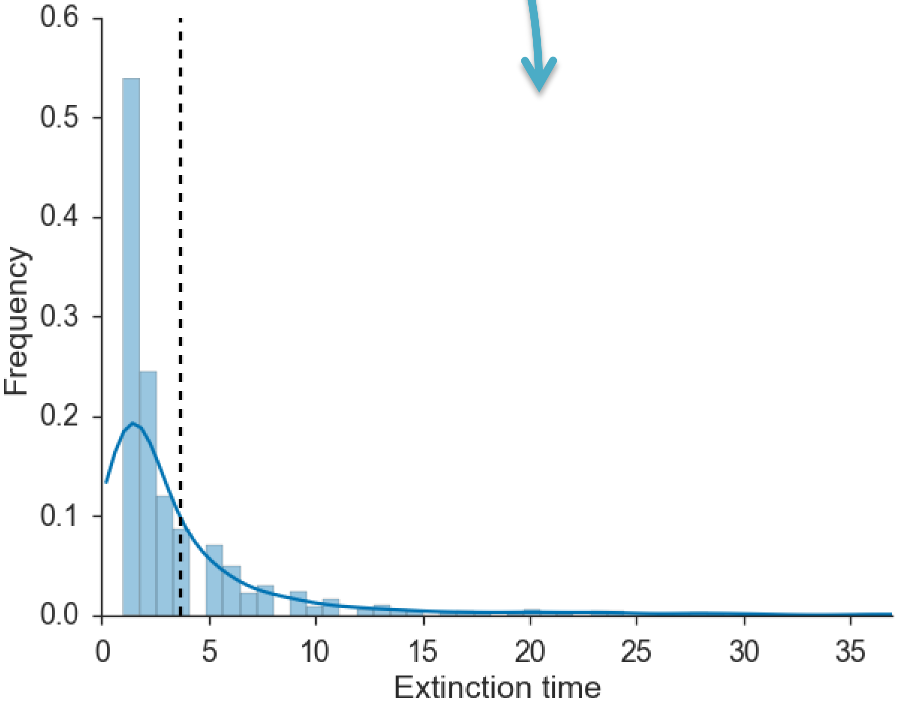
Plot with Seaborn

```
from seaborn import distplot

fixations, times = simulation(…)

distplot(times[fixations])

distplot(times[~fixations])
```

Plot with Seaborn

```
fixations, times = simulation(…)
distplot(times[fixations])
distplot(times[~fixations])
```





Diffusion equation approximation

$$I_1(x) = \frac{1 - e^{-2Nsx} - e^{-2Ns(1-x)} + e^{-2Ns}}{x(1-x)}$$

$$I_2(x) = \frac{(e^{2Nsx} - 1)(1 - e^{-2Ns(1-x)})}{x(1-x)}$$

$$J_1 = \frac{1}{s(1 - e^{-2Ns})} \int_x^1 I_1(y)dy$$

$$J_2 = \frac{1}{s(1 - e^{-2Ns})} \int_0^x I_2(y)dt$$

$$u = \frac{1 - e^{-2Nsx}}{1 - e^{-2Ns}}$$

$$\boxed{T_{fix} = J1 + \frac{1-u}{u} J_2}$$

Motoo Kimura
1924-1994
Japan & USA

Diffusion equation approximation

$$I_1(x) = \frac{1 - e^{-2Nsx} - e^{-2Ns(1-x)} + e^{-2Ns}}{x(1-x)}$$

$$I_2(x) = \frac{(e^{2Nsx} - 1)(1 - e^{-2Ns(1-x)})}{x(1-x)}$$

$$J_1 = \frac{1}{s(1 - e^{-2Ns})} \int_x^1 I_1(y)dy$$

$$J_2 = \frac{1}{s(1 - e^{-2Ns})} \int_0^x I_2(y)dt$$

$$u = \frac{1 - e^{-2Nsx}}{1 - e^{-2Ns}}$$

$$\boxed{T_{fix} = J1 + \frac{1-u}{u} J_2}$$

Requires integration…

Motoo Kimura
1924-1994
Japan & USA

```python
from functools import partial
from scipy.integrate import quad


def integral(f, N, s, a, b):
  f = partial(f, N, s)
  return quad(f, a, b)[0]
```

`integral` **will calculate** $\int_a^b f(N, s, x)dx$

```
from functools import partial
from scipy.integrate import quad


def integral(f, N, s, a, b):
    f = partial(f, N, s)
    return quad(f, a, b)[0]
```

`partial` **freezes** `N` **and** `s`
in `f(N, s, x)` **to create** `f(x)`

```python
from functools import partial
from scipy.integrate import quad


def integral(f, N, s, a, b):
  f = partial(f, N, s)
  return quad(f, a, b)[0]
```

SciPy's quad **computes a definite integral** $\int_a^b f(x)dx$

(using a technique from the Fortran library QUADPACK)

```python
def I1(N, s, x):
    ...

def I2(N, s, x):
    ...
```

$$I_1(x) = \frac{1 - e^{-2Nsx} - e^{-2Ns(1-x)} + e^{-2Ns}}{x(1-x)}$$

$$I_2(x) = \frac{(e^{2Nsx} - 1)(1 - e^{-2Ns(1-x)})}{x(1-x)}$$

$$J_1 = \frac{1}{s(1 - e^{-2Ns})} \int_x^1 I_1(y)dy$$

$$J_2 = \frac{1}{s(1 - e^{-2Ns})} \int_0^x I_2(y)dt$$

$$u = \frac{1 - e^{-2Nsx}}{1 - e^{-2Ns}}$$

$$T_{fix} = J1 + \frac{1-u}{u} J_2$$

`I1` and `I2` are defined according to the equations

```python
@np.vectorize
def T_kimura(N, s):
    x = 1.0 / N
    J1 = -1.0 / (s * expm1(-2 * N * s)) *
            integral(I1, N, s, x, 1)
    J2 = -1.0 / (s * expm1(-2 * N *s)) *
            integral(I2, N, s, 0, x)
    u = expm1(-2 * N * s * x) /
        expm1(-2 * N * s)

    return J1 + ((1 - u) / u) * J2
```

`T_kimura` **is the fixation time given a single copy of variant** 1**: frequency** `x=1/N`

```python
@np.vectorize
def T_kimura(N, s):
    x = 1.0 / N
    J1 = -1.0 / (s * expm1(-2 * N * s)) *
              integral(I1, N, s, x, 1)
    J2 = -1.0 / (s * expm1(-2 * N *s)) *
              integral(I2, N, s, 0, x)
    u = expm1(-2 * N * s * x) /
        expm1(-2 * N * s)

    return J1 + ((1 - u) / u) * J2
```

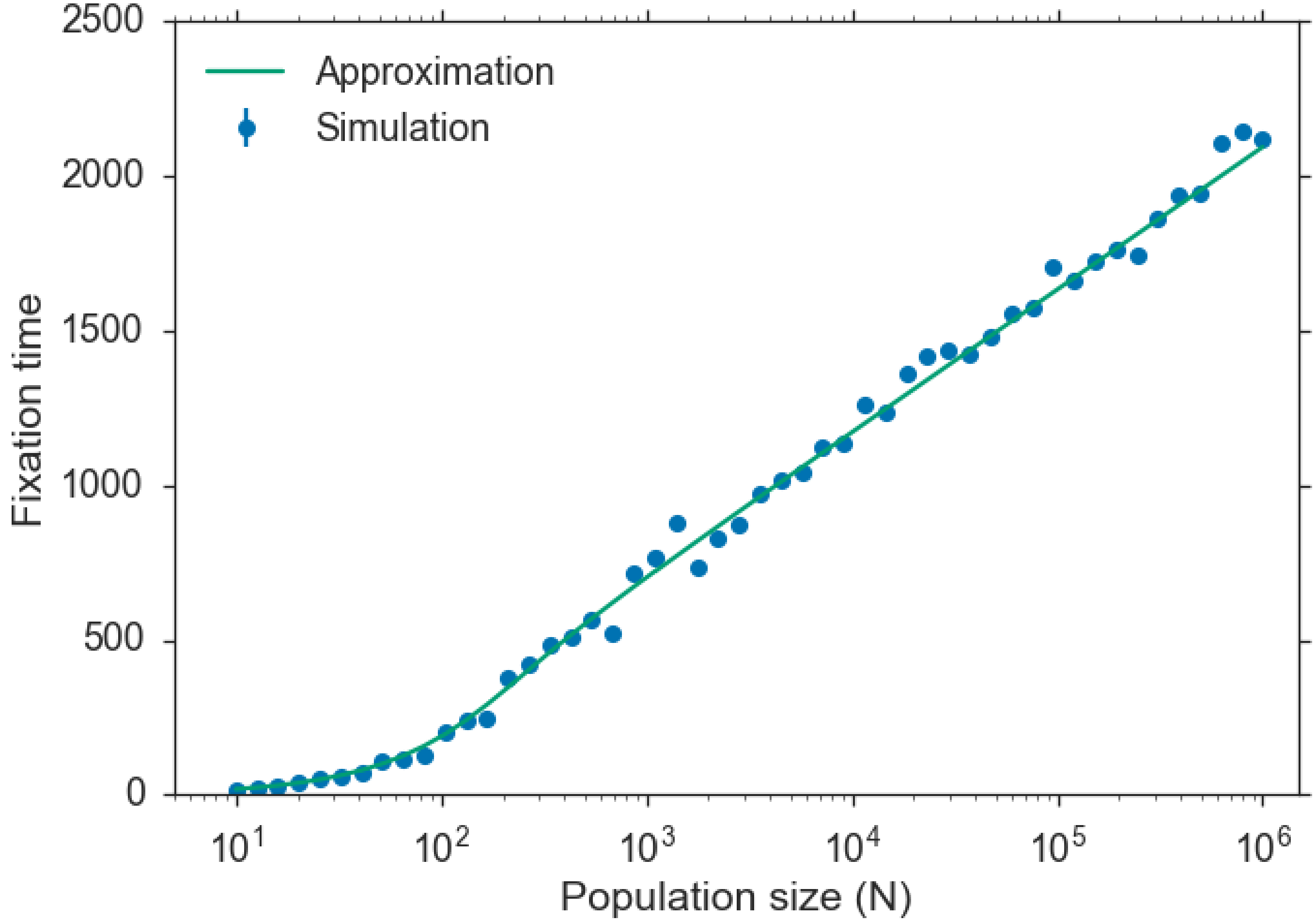
`J1` and `J2` are calculated using integrals of `I1` and `I2`

```python
@np.vectorize
def T_kimura(N, s):
  x = 1.0 / N
  J1 = -1.0 / (s * expm1(-2 * N * s)) *
          integral(I1, N, s, x, 1)
  J2 = -1.0 / (s * expm1(-2 * N *s)) *
          integral(I2, N, s, 0, x)
  u = expm1(-2 * N * s * x) /
      expm1(-2 * N * s)

  return J1 + ((1 - u) / u) * J2
```

T_{fix} **is the return value**

```python
@np.vectorize
def T_kimura(N, s):
    x = 1.0 / N
    J1 = -1.0 / (s * expm1(-2 * N * s)) *
            integral(I1, N, s, x, 1)
    J2 = -1.0 / (s * expm1(-2 * N *s)) *
            integral(I2, N, s, 0, x)
    u = expm1(-2 * N * s * x) /
        expm1(-2 * N * s)

    return J1 + ((1 - u) / u) * J2
```

`np.vectorize` **creates a function that takes a sequence and returns an array - x2 faster**


2500
2000
1500
1000
500
0
Approximation
Simulation
Fixation time
10^1
10^2
10^3
10^4
10^5
10^6
Population size (N)

Presentation, Jupyter notebook, and more at

github.com/yoavram/PyConIL2016

yoav@yoavram.com
@yoavram
github.com/yoavram
www.yoavram.com
python.yoavram.com