Kimura\_NumIntegr\_Technical\_Doc

Dr. Royal Truman

Contents

[**Overview** 1](#_Toc211079088)

[Biological Context 1](#_Toc211079089)

[**Mathematical Foundation** 2](#_Toc211079090)

[**Program Architecture** 2](#_Toc211079091)

[Input/Output 2](#_Toc211079092)

[Main Function: kimura\_fixation\_probability\_numeric() 3](#_Toc211079093)

[Numerical Integration: \_kimura\_numeric\_integral\_shift() 4](#_Toc211079094)

[**Numerical Accuracy Considerations** 4](#_Toc211079095)

[When the Diffusion Approximation is Valid 4](#_Toc211079096)

[Convergence and Grid Resolution 5](#_Toc211079097)

[Method Selection Trade-offs 5](#_Toc211079098)

[Code Structure and Data Flow 5](#_Toc211079099)

[Usage Example 6](#_Toc211079100)

[Dependencies 6](#_Toc211079101)

[**Example dataset to test all calculation methods** 6](#_Toc211079102)

## **Overview**

Kimura\_NumIntegr.py is a Python implementation of Kimura's diffusion approximation for computing fixation probabilities in diploid Wright-Fisher populations under natural selection. **The equations refer to those used in the accompanying document *Kimura\_NumInteg.docx*** that derives the mathematics involved.

The program numerically solves the backward Kolmogorov equation (eqn. 15) to determine the probability that a beneficial or deleterious allele will eventually become fixed (reach 100% frequency) in a finite population.

### Biological Context

The program models a diploid population of effective size Ne with two alleles (or haplotypes) at a single locus: A (the focal allele) and a. The three possible diploid genotypes have different fitnesses that determine reproductive success:

* **WAA = 1 + s**: homozygote for allele A
* **WAa = 1 + hs**: heterozygote
* **Waa = 1**: homozygote for allele a (reference fitness)

**This it the most common convention for genotype fitness used in population genetics but is not what many earlier workers in the field used and some textbooks are still using. The results of the python program must be interpreted accordingly.**

Key biological parameters:

* **s** (selection coefficient): measures the fitness advantage (s > 0) or disadvantage (s < 0) of the AA genotype relative to aa
* **h** (dominance coefficient): determines how the heterozygote fitness compares to the homozygotes
  + h = 0.5: semidominant (additive) selection
  + h = 1: fully dominant beneficial allele
  + h = 0: fully recessive beneficial allele
* **p0** (initial allele frequency): the starting frequency of allele A in the population (0 ≤ p0 ≤ 1)
* **π(p0)** (fixation probability): the probability that allele A eventually fixes given initial frequency p0

## **Mathematical Foundation**

The program implements the diffusion approximation solution described in the accompanying mathematical derivation document. The key steps are:

**1. Diffusion approximation** (eqns. 2-3, 13): Under weak selection and large population size, the discrete Wright-Fisher process converges to a continuous diffusion process with drift M(p) and variance V(p):

* M(p) ≈ sp(1-p)[h + p(1-2h)] (eqn. 3)
* V(p) = p(1-p)/(2Ne) (eqn. 13)

**2. Backward Kolmogorov equation** (eqn. 15): The fixation probability π(p) satisfies:

0 = M(p)π'(p) + ½V(p)π''(p)

with boundary conditions π(0) = 0 and π(1) = 1.

**3. Integral solution** (eqn. 17): Using an integrating factor ψ(p) (eqn. 16), the solution takes the form:

π(p) = [∫0p ψ(y)dy] / [∫01 ψ(y)dy]

**4. Exponent form** (eqns. 18-22): For the genotype fitness model, the integrating factor becomes:

ψ(y) = exp[-Φ(y)]

where Φ(y) = Ay + By2, with:

* A = γh
* B = γ(1-2h)/2
* γ = 4Nes (scaled selection parameter, eqn. 19)

## **Program Architecture**

### Input/Output

**Input file format** (semicolon-separated, default: input\_data\_Kimura.txt):

Ne;s;h;p0

1000;0.01;0.5;0.1

10000;-0.001;0.0;0.05

**Output file format** (semicolon-separated, default: results\_Kimura\_NumIntegr.txt):

Ne;s;h;p0;Pfix;method

1000;0.01;0.5;0.1;0.523456;erf-closed

10000;-0.001;0.0;0.05;0.049123;numeric

The method column indicates which computational approach was used (explained below).

### Main Function: kimura\_fixation\_probability\_numeric()

This function computes π(p0) using one of three methods, automatically selected based on parameter values:

#### Method Selection Logic

**1. Boundary cases**

* If p0 ≤ 0: return π = 0 (allele already lost)
* If p0 ≥ 1: return π = 1 (allele already fixed)

**2. Near-neutral case** (abs(γ) < 1e-14)

* When selection is negligible (γ ≈ 0), the fixation probability equals the initial frequency
* Returns π = p0 with method "near-neutral"

**3. Linear exponent case** (abs(B) < 1e-30, corresponding to h ≈ 0.5)

When B ≈ 0, Φ(y) = Ay becomes linear, and the integrals have closed-form solutions (eqn. 23-24):

π(p0) = (1 - exp(-Ap0)) / (1 - exp(-A))

**Numerical stability**: The program uses math.expm1(x) which computes ex - 1 accurately for small |x| (eqn. 25). This prevents catastrophic cancellation when γ is small. The identity used is:

1 - exp(-x) = -expm1(-x)

Returns with method "h=0.5-closed".

**4. Quadratic exponent with B > 0** (eqn. 26-30)

When B > 0, the integrals can be expressed analytically using the error function erf(x):

By completing the square (eqn. 27) and substituting, the ratio simplifies to:

π(p0) = [erf(√B·p0 + A/(2√B)) - erf(A/(2√B))] / [erf(√B + A/(2√B)) - erf(A/(2√B))]

**Why this is stable**: The exponential prefactor exp(A2/4B) appears in both numerator and denominator and cancels exactly, preventing overflow when A2/4B is large.

Returns with method "erf-closed". If the denominator is too small (< 1e-18) or non-finite, falls back to numerical integration.

**5. Quadratic exponent with B < 0** (requires numerical integration)

When B < 0, the error function approach requires complex arguments (erfi), which is numerically unstable. The program falls back to robust numerical integration.

Returns with method "numeric".

### Numerical Integration: \_kimura\_numeric\_integral\_shift()

This function implements the trapezoid rule with an exponent shift trick for numerical stability (Section 6 of the mathematical document).

#### Algorithm Steps

**1. Grid construction**

Creates a uniform grid of GRID\_SIZE = 10,001 points on [0, 1]:

z = np.linspace(0.0, 1.0, grid\_size)

With 10,001 points, the spacing Δy ≈ 0.0001, giving trapezoid error O(Δy2) ≈ 10-8 (eqn. 34).

**2. Compute Φ(y) on grid**

phi = γ(h·z + 0.5(1-2h)z²)

**3. Exponent shift for stability** (eqn. 31)

The key numerical trick: instead of computing exp(-Φ(y)) directly (which can overflow or underflow), shift by the minimum:

Φ̃(y) = Φ(y) - Φmin

Then compute:

ψshift(y) = exp(-Φ̃(y)) = exp(-(Φ(y) - Φmin))

**Why this works**: The shift constant exp(-Φmin) appears in both integrals and cancels in the ratio π(p0) = numerator/denominator. By choosing Φmin = min(Φ(y)), the largest value of ψshift is exp(0) = 1, preventing overflow. The smallest values are exp(-(Φmax - Φmin)), which is bounded and prevents underflow in typical cases.

**4. Cumulative integration via trapezoid rule** (eqn. 32)

For each segment [zi, zi+1]:

* Area = 0.5 · (zi+1 - zi) · [ψshift(zi) + ψshift(zi+1)]
* Cumulative sum: C(zj) = Σ (areas up to j)

dz = np.diff(z)

mid = 0.5 \* (psi\_shift[:-1] + psi\_shift[1:])

areas = mid \* dz

cum = np.concatenate(([0.0], np.cumsum(areas)))

**5. Interpolation and ratio** (eqn. 33)

* Numerator: interpolate cum at p0 using np.interp(p0, z, cum)
* Denominator: cum[-1] (the full integral from 0 to 1)
* π(p0) = numerator / denominator

If denominator is non-positive or non-finite, returns nan indicating numerical failure.

## **Numerical Accuracy Considerations**

### When the Diffusion Approximation is Valid

The program assumes the diffusion limit (eqns. 2-3): changes per generation are small. This requires:

* Large Ne (typically Ne > 100)
* Weak to moderate selection (|γ| = |4Nes| not too large, typically < 100)

**Warning**: The predicted fixation probabilities are less reliable when:

* s is large (|s| > 0.1)
* Ne is small (Ne < 50)
* |γ| > 200 (very strong selection)

In these cases, discrete-generation finite-state methods (like Wright-Fisher Exact Solver, WFES) may be more accurate.

### Convergence and Grid Resolution

With GRID\_SIZE = 10,001:

* Step size Δy = 1/10,000 = 0.0001
* Trapezoid error: O(Δy2) ≈ 10-8

Empirical tests show that increasing grid size from 5,001 to 10,001 changes results by less than 10-8 for typical parameters, confirming convergence.

### Method Selection Trade-offs

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Speed** | **Accuracy** | **Applicability** |
| near-neutral | Instant | Exact | γ ≈ 0 |
| h=0.5-closed | Very fast | Machine precision | B ≈ 0 |
| erf-closed | Fast | Machine precision | B > 0 |
| numeric | Moderate | ~10-8 | B < 0 or fallback |

The program automatically selects the fastest applicable method while maintaining high accuracy.

## Code Structure and Data Flow

main()

├─> Read input CSV with pandas (Ne, s, h, p0)

├─> For each row:

│ └─> kimura\_fixation\_probability\_numeric(Ne, s, h, p0)

│ ├─> Check boundary cases

│ ├─> Compute γ = 4Nes, A = γh, B = γ(1-2h)/2

│ ├─> Select method based on B value:

│ │ ├─> B ≈ 0: closed form with expm1

│ │ ├─> B > 0: closed form with erf

│ │ └─> B < 0: numeric integration

│ │ └─> \_kimura\_numeric\_integral\_shift(Ne, s, h, p0)

│ │ ├─> Build grid on [0,1]

│ │ ├─> Compute Φ(y) = Ay + By²

│ │ ├─> Shift: Φ̃ = Φ - Φmin

│ │ ├─> Integrate ψshift = exp(-Φ̃) via trapezoid

│ │ └─> Return π(p0) = C(p0)/C(1)

│ └─> Return (π, method)

└─> Write output CSV with additional columns (Pfix, method)

## Usage Example

Constants are defined in the program for the input and output files:

INPUT\_FILE = "input\_data\_Kimura.txt"

OUTPUT\_FILE = "results\_Kimura\_NumIntegr.txt"

## Dependencies

* Python 3.6+
* NumPy (for array operations and numerical integration)
* pandas (for CSV I/O)
* Standard library: math, argparse, sys, typing

## **Example dataset to test all calculation methods**

**input\_data\_Kimura.txt**

|  |  |  |  |
| --- | --- | --- | --- |
| Ne | s | h | p0 |
| 1000 | 0.00000000000000001 | 0.2 | 0.6 |
| 1000 | 0 | 0.3 | 0.25 |
| 1000 | 0.01 | 0.5 | 0.1 |
| 10 | -0.3 | 0.5 | 0.05 |
| 10 | -0.3 | 0.5 | 0.05 |
| 1000000 | 10 | 0.1 | 0.01 |
| 1000000 | 5 | 0.1 | 0.001 |
| 1000 | 0.01 | 0.5 | 0 |
| 1000 | 0.01 | 0.5 | 1 |
| 1000 | 0.01 | 0.3 | 0.2 |
| 1000 | 0.002 | 0.25 | 0.3 |
| 1000 | 0.01 | 0.7 | 0.2 |
| 2000 | 0.01 | 0.8 | 0.4 |

This produces the following output:

**results\_Kimura\_NumIntegr.txt**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ne | s | h | p0 | Pfix | method |
| 1000 | 0 | 0.2 | 0.6 | 0.6 | near-neutral |
| 1000 | 0 | 0.3 | 0.25 | 0.25 | near-neutral |
| 1000 | 0.01 | 0.5 | 0.1 | 0.864665 | h=0.5-closed |
| 10 | -0.3 | 0.5 | 0.05 | 0.000869 | h=0.5-closed |
| 10 | -0.3 | 0.5 | 0.05 | 0.000869 | h=0.5-closed |
| 1000000 | 10 | 0.1 | 0.01 | 1 | numeric-erf-small-den |
| 1000000 | 5 | 0.1 | 0.001 | 1 | numeric-erf-small-den |
| 1000 | 0.01 | 0.5 | 0 | 0 | boundary |
| 1000 | 0.01 | 0.5 | 1 | 1 | boundary |
| 1000 | 0.01 | 0.3 | 0.2 | 0.946405 | erf-closed |
| 1000 | 0.002 | 0.25 | 0.3 | 0.660219 | erf-closed |
| 1000 | 0.01 | 0.7 | 0.2 | 0.994214 | numeric |
| 2000 | 0.01 | 0.8 | 0.4 | 1 | numeric |

See the accompanying mathematical derivation document ***Kimura\_NumInteg.docx*** for complete details on equations, their derivations, and literature.