CMCpy: Genetic Code-Message Coevolution Models in Python

Peter J. Becich¹, Brian P. Stark², Harish S. Bhat³ and David H. Ardell¹,⁴
¹Center for Computational Biology, University of California, Merced, CA. ²Academic Program in Computer Science and Engineering, University of California, Merced, CA. ³Applied Mathematics Unit, University of California, Merced, CA. ⁴Program in Quantitative and Systems Biology, University of California, Merced, CA. Corresponding author emails: dardell@ucmerced.edu; hbhat@ucmerced.edu

Abstract: Code-message coevolution (CMC) models represent coevolution of a genetic code and a population of protein-coding genes (“messages”). Formally, CMC models are sets of quasispecies coupled together for fitness through a shared genetic code. Although CMC models display plausible explanations for the origin of multiple genetic code traits by natural selection, useful modern implementations of CMC models are not currently available. To meet this need we present CMCpy, an object-oriented Python API and command-line executable front-end that can reproduce all published results of CMC models. CMCpy implements multiple solvers for leading eigenpairs of quasispecies models. We also present novel analytical results that extend and generalize applications of perturbation theory to quasispecies models and pioneer the application of a homotopy method for quasispecies with non-unique maximally fit genotypes. Our results therefore facilitate the computational and analytical study of a variety of evolutionary systems. CMCpy is free open-source software available from http://pypi.python.org/pypi/CMCpy/.

Keywords: quasispecies, CUDA, genetic code, homotopy, perturbative method

Evolutionary Bioinformatics 2013:9 111–125
doi: 10.4137/EBO.S11169

This article is available from http://www.la-press.com.

© the author(s), publisher and licensee Libertas Academica Ltd.

This is an open access article. Unrestricted non-commercial use is permitted provided the original work is properly cited.
**Introduction**

Code-Message Coevolution (CMC) models were introduced to facilitate the study of co-evolutionary systems in which a large population of individuals share an evolvable genetic code coupled to many and/or long protein-coding genes ("messages"). Initial study of these models demonstrated that base mutations in protein-coding genes can significantly influence the fitness of genetic codes. More detailed study of CMC models has yielded formal demonstrations for how natural selection likely contributed to the origins of codon redundancy and non-random patterns of amino acid assignments in genetic codes. CMC models have been extended to study the effects of population structure and gene sharing on coevolving systems of codes and messages.

In their original formulations, CMC models are deterministic evolutionary genetic models that couple together sets of large populations of asexually reproducing genotypes evolving under mutation and natural selection. One such population is called a “quasispecies”, a concept reviewed recently both generally and specifically in connection to population genetics theory. In CMC models, each of multiple quasispecies represents a large population of codons evolving to meet the same physicochemical requirements of a “site-type” in proteins under translation by a given genetic code. The coupling of multiple quasispecies through a genetic code occurs through reuse of the same codon type (or allele) in different site-types. CMC models follow deterministic trajectories that alternate between equilibration of messages to an established genetic code by mutation and selection, and locally adaptive “gradient ascent” hill-climbing of the genetic code through single codon assignments and reassignments. This “quasisstatic” dynamic continues until no single codon reassignment yields higher fitness with the current message population. Published CMC models always converge to a stable local fitness optimum—a process called “code freezing.”

Because CMC models compound quasispecies models, analytical solutions to quasispecies models are relevant to their study. Previous applications of perturbation theory for approximate solutions to quasispecies models assume uniqueness of maximally fit genotypes and small perturbation parameters (small mutation rates). In this work, we present two analytical methods that relax these assumptions. The first is a perturbative method that clarifies, extends and generalizes the perturbative approach to quasispecies models. Our presentation provides a derivation to arbitrary order, which relaxes any restriction on the magnitude of the perturbation parameter. The second method is the first published application of homotopy methods to quasispecies models, which handles cases with non-unique maximally fit genotypes. Numerical tests show that, for two example problems, both algorithms converge. Notably, the error of the perturbative method decreases exponentially in the number of iterations, suggesting its potential to outperform the power method. Both approaches are flexible and can be extended to a variety of quasispecies mutation models and fitness schemes. Also, both methods have been formulated so as to facilitate future connections between quasispecies models and established results in eigenvalue perturbation theory and homotopy methods.

In addition to these analytical results, we present a new code-base implementing CMC models. The dissemination and further study of CMC models has been sorely hindered by the lack of a modern code-base implementing them. The original C/C++ CMC code-base has been rendered obsolete by evolution of compilers and language standards from the time of its writing in the late 1990s. This has created a need for reimplementing the original models in an easy-to-use and program, powerful, and efficient, high-level scripting language like Python.

Here we present CMCpy, a free open-source code-base that implements CMC models in an easy to use object oriented Python API. The code-base comes with a front-end command-line executable called that can drive the exploration of a variety of CMC models and reproduce published results.

**Implementation**

CMCpy was developed in Python 2.7. Figure 1 shows the organization of classes in CMCpy. A rich class hierarchy allows convenient specification of a wide-range of CMC models with very few lines of Python code.

An element not shown in Figure 1 is that the ArdellSellaEvolver class is an abstract base class for a variety of subclasses implementing different strategies to solve dominant eigenpairs. The default solver relies on the Numpy eig() function for its speed and efficiency.
accuracy, but for reasons of flexibility, verification, and to experiment with different methods, we include a legacy central processing unit (CPU)-based power method, a multicore CPU-based power method implementation and an experimental graphics processing unit (GPU) implementation of the power method that relies on pyCUDA.¹⁷

The pyCUDA solver is implemented in a CUDA C kernel with supportive Python code. The Python code uses NumPy for simple operations, to cast data to types acceptable by the CUDA platform, and to reshape matrices to one-dimensional data frames, for use by CUDA C. The CUDA C implementation of the power method approximately solves the eigenvector of each site-type, in parallel. Each site-type is assigned to one virtual “block” of the GPU, each of which corresponds to a physical processor when the number of blocks is below a hardware limit. This organization of the parallel workload was chosen to stay below CUDA specification limits, access GPU memory efficiently, and avoid excess complexity. Ultimately, the CUDA power method implementation is faster than the CPU power method implementation. Figure 2 shows system clock execution times of various methods on a 3.0 GHz Core 2 Duo with an Nvidia GeForce 460 GTX GPU running Ubuntu 12.04. The benchmark script in R that generated this figure is provided as supplementary data.

CMCpy comes with a front-end command-line executable in Python called cmc. This executable provides users, including non-programmers, the capability to reproduce (at least qualitatively) all of the published results on CMC models¹⁻⁴ as well as run individual and batch simulations of other models and parameter spaces. In Table 1 we list command-line options to the cmc executable and their corresponding model parameters.

A variety of observables and statistics are implemented including the Normalized Encoded Range² for one-dimensional amino acid/site-type spaces. Although CMC models are deterministic, exact quantitative differences may arise in results with CMCpy depending on floating point representation differences by platform and differences in convergence thresholds using power method-based eigensolvers.

It may be useful to restate the assumptions of the “Ardell-Sella” models currently implemented and available in CMCpy. These include the following:

1. The numbers and/or lengths of protein-coding genes, or “messages,” translated by a common genetic code, are large with respect to every possible site-type in proteins.
2. For every possible site-type in proteins there corresponds a uniquely most fit amino acid.
3. The machineries to decode codons, and to associate any codon to any amino acid, pre-exist the evolution of the genetic code.
4. Fitness contributions of amino acids across sites are independent and multiplicative.
5. Fitness contributions of different amino acids within the same site are independent and additive.
6. Bases in messages mutate independently of one another.

Figure 1. Overview of class hierarchy and containment relationships in CMCpy.
7. Messages are haploid and asexually reproducing.
8. Genetic codes evolve much slower than messages, through discrete and independent assignments or reassignments of amino acids to codons.

Analytical Methods for Quasispecies Solutions
In this section we develop two different analytical methods to solve for the equilibrium growth rate and genotype distributions for a wide range of quasispecies models. The Matlab/Octave code used to implement these solutions are provided as supplementary data. A version of the homotopy method is also implemented in CMCpy for ring models. Full implementations of both methods will be incorporated into CMCpy at a later date.

Perturbative method: quasispecies with unique fittest genotype
We start with a ring mutation model discussed in prior work.¹ Let μ denote the $N \times N$ mutation matrix.

Table 1. Options to the cmc executable and corresponding model parameters.

| Option | Long version | Description                                      | Default |
|--------|--------------|--------------------------------------------------|---------|
| -a     | --numaas     | Number of amino-acids/site-types (AA/ST)         | 10      |
| -d     | --numdims    | Dimensionality of AA/ST space                    | 1       |
| -s     | --seed       | Seed for random initialization of AA/ST coordinates | 42      |
| -t     | --numtrials  | Num. trials with reinitialized AA/ST spaces      | 1       |
| -c     | --numcodons  | Num. codons in a “double ring” model             | N/A     |
| -b     | --numbases   | Alphabet size for word-based codon model         | 4       |
| -p     | --numpositions | Length of codons for word-based codon model    | N/A     |
| -m     | --mu         | Base (word models) or codon mutation rate        | 0.1     |
| -k     | --kappa      | Transition/transversion mutation bias ratio       | 1       |
| -f     | --phi        | Missense tolerance parameter                     | 0.25    |
| -r     | --misreading | Misreading parameters                            | N/A     |
\[
\mu = \begin{pmatrix}
1 - 2\mu & \mu & \mu \\
\mu & 1 - 2\mu & \mu \\
\mu & \mu & 1 - 2\mu \\
\mu & \mu & \mu
\end{pmatrix},
\] (1)

where blank spaces should be interpreted as zeros. Throughout this derivation and the following one, we assume \(0 \leq \mu < 1\) and \(N > 1\). Let \(w\) denote the \(N \times N\) fitness matrix

\[
w = \begin{pmatrix}
w_1 \\
\vdots \\
w_N
\end{pmatrix},
\]

We assume that \(w_1\) is the unique maximum of the finite set \(\{w_1, \ldots, w_N\}\). We also assume \(w_i > 0\) for each \(i\), consistent with prior work.\(^1\)

Our goal in this derivation is to determine the leading eigenpair, consisting of the largest eigenvalue together with its corresponding eigenvector, of the matrix

\[
\tilde{Q}(\mu) = \mu w.
\] (2)

We write \(\tilde{Q}(\mu)\) to emphasize the fact that since \(\mu\) is a function of \(\mu\), so is \(\tilde{Q}\). The diagonal matrix \(w\) is constant with respect to \(\mu\). Next we define

\[
Q(\mu) = w^{1/2} \tilde{Q}(\mu) w^{-1/2} = \mu w^{1/2} w^{1/2}.
\] (3)

Since the matrices \(Q\) and \(\tilde{Q}\) are related by a similarity transformation, their eigenpairs are closely related. One can check that

\[
Qv = \lambda v \quad \text{if and only if} \quad \tilde{Q} w^{-1/2} v = \lambda w^{-1/2} v \quad (4)
\]

We focus our attention on \(Q\) because it is symmetric, i.e., \(Q(\mu)^T = Q(\mu)\) for all \(\mu\).

For \(Q(\mu)\), the leading eigenpair \((\lambda(\mu), v(\mu))\) must satisfy

\[
Q(\mu)v(\mu) = \lambda(\mu)v(\mu). \quad (5)
\]

When \(\mu = 0\), the matrix \(\mu\) reduces to the \(N \times N\) identity matrix. Therefore, \(Q(0) = w\), and the leading eigenpair of \(Q(0)\) is given by

\[
\lambda(0) = w_i \quad \text{and} \quad v(0) = e_i. \quad (6)
\]

Here \(e_i\) denotes the \(j\)-th basis vector in \(N\)-dimensional space, i.e., the vector with all zeros except for 1 in the \(j\)-th slot.

**Guiding principle**

Since \(Q(\mu)\) is symmetric for all real \(\mu\), standard theoretical results in eigenvalue perturbation theory\(^{14}\) guarantee that both the eigenvalue \(\lambda(\mu)\) and eigenvector \(v(\mu)\) are analytic functions of \(\mu\). This means that there exists \(M > 0\) such that for \(\mu \in (-M, M)\), the following power series expansions converge:

\[
\lambda(\mu) = \sum_{j=0}^{\infty} \frac{\lambda_j \mu^j}{j!} \quad (7a)
\]

\[
v(\mu) = \sum_{j=0}^{\infty} \frac{v_j \mu^j}{j!} \quad (7b)
\]

In this notation, (6) can be written as \(\lambda_0 = w_i\) and \(v_0 = e_i\). Note also that we have arranged the coefficients so that

\[
\frac{d}{d \mu} \lambda(\mu) = \lambda_j \quad (8a)
\]

\[
\frac{d}{d \mu} v(\mu) = v_j. \quad (8b)
\]

Our strategy now will be to derive from (5) a recursive set of equations for the coefficients \(\lambda_j, v_j\). Once we have these coefficients for \(j = 0, 1, \ldots, J\), we have an approximation to the leading eigenpair.

Let \(I\) denote the \(n \times n\) identity matrix. Then\(\mu = I + \mu s\), where

\[
s = \begin{pmatrix}
-2 & 1 & 1 \\
1 & -2 & 1 \\
1 & \ddots & \ddots \\
& \ddots & -2 & 1 \\
1 & 1 & -2
\end{pmatrix}.
\]
This implies that

\[
\frac{d^j}{d\mu^j} \mathcal{M} = \begin{cases} s & j = 1 \\ 0 & j \geq 2. \end{cases}
\]

By (3), we have

\[
\frac{d^j}{d\mu^j} Q(\mu) = \begin{cases} w^{1/2}s^{1/2} & j = 1 \\ 0 & j \geq 2. \end{cases}
\]

Perturbative solution: part I

Armed with the above facts, we differentiate (5) with respect to \( \mu \) on both sides:

\[
Q'(\mu)\nu(\mu) + Q(\mu)\nu'(\mu) = \lambda'(\mu)\nu(\mu) + \lambda(\mu)\nu'(\mu).
\]

We then set \( \mu = 0 \) and use (8), (3), (9), and (6) to obtain

\[
w^{1/2}s^{1/2}e_i + \nu_i = \lambda_i e_i + w_i v_i.
\]

Multiplying this equation on the left by the row vector \( e_i^T \):

\[
e_i^T w^{1/2}s^{1/2} e_i + e_i^T \nu_i = \lambda_i e_i + w_i e_i^T v_i.
\]

Note that \( e_i^T w = w_i e_i^T \), which implies \( e_i^T \nu_i = w_i e_i^T v_i \). Using this equality in (11), we have

\[
e_i^T w^{1/2}s^{1/2} e_i = \lambda_i.
\]

This shows that if we already know \( (\lambda_{0i}, v_i) \), we can determine \( \lambda_i \). To determine \( v_i \), we return to (10) except now we treat \( \lambda_i \) as known. Rearranging the equation, we have

\[
w - w_i I \nu_i = [\lambda_i I - w_i^{1/2}s^{1/2}]e_i.
\]

We substitute our definition of \( w \) on the left-hand side to obtain

\[
\begin{pmatrix} 0 \\ w_2 - w_i \\ w_3 - w_i \\ \vdots \\ w_N - w_i \end{pmatrix}
\nu_i = \left[ \lambda_i I - w_i^{1/2}s^{1/2} \right] e_i.
\]

Suppose that \( v_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(N)}) \). We set \( v_i^{(1)} = 0 \). For the remaining components, we solve the above matrix-vector system to obtain

\[
v_i^{(k)} = \frac{1}{w_k - w_i} e_i^T \left[ \lambda_i I - w_i^{1/2}s^{1/2} \right] e_i,
\]

for \( k = 2, 3, \ldots, N \).

We have completed the loop, showing how to proceed from the zeroth-order eigenpair \( (\lambda_0, v_i) \) to the first-order eigenpair \( (\lambda_i, v_i) \).

In the next section, we show how to iterate this procedure to generate the \( j \)-th order eigenpair \( (\lambda_j, v_j) \) from the previously obtained eigenpairs.

Perturbative solution: part II

We return to (5) and take \( j \) derivatives with respect to \( \mu \) on both sides. Using the general Leibniz rule, we have

\[
\sum_{m=0}^{j} \binom{j}{m} \frac{d^m}{d\mu^m} Q(\mu) \frac{d^{j-m}}{d\mu^{j-m}} v(\mu) = \sum_{m=0}^{j} \binom{j}{m} \frac{d^m}{d\mu^m} \lambda(\mu) \frac{d^{j-m}}{d\mu^{j-m}} v(\mu).
\]

After differentiating, we set \( \mu = 0 \) and use (8) to obtain

\[
\sum_{m=0}^{j} \binom{j}{m} \frac{d^m}{d\mu^m} Q(\mu) v_{j-m} = \sum_{m=0}^{j} \binom{j}{m} \lambda_m v_{j-m}.
\]

Applying (3) and (9) yields

\[
wv_j + jw^{1/2}s^{1/2}v_{j-1} = \sum_{m=0}^{j} \binom{j}{m} \lambda_m v_{j-m}.
\]

We peel off the \( m = 0 \) term from the right-hand side:

\[
wv_j + jw^{1/2}s^{1/2}v_{j-1} = \lambda_0 v_j + \sum_{m=1}^{j} \binom{j}{m} \lambda_m v_{j-m}.
\]

Multiplying on the left by \( e_i^T \) and using \( e_i^T w v_j = w_i e_i^T v_j = \lambda_i e_i^T v_j \), we see that the first term on the left-hand side cancels the first term on the right-hand side. We are left with
\[ j e_i^T w^{1/2} w^{1/2} v_{j-1} = \sum_{m=1}^{j-1} (j \lambda_m e_i^T v_{j-m}) \]

Now on the right-hand side, we peel off the \( m = j \) term—note that this the only term in which \( \lambda_j \) appears. Hence

\[ j e_i^T w^{1/2} w^{1/2} v_{j-1} = \sum_{m=1}^{j-1} (j \lambda_m e_i^T v_{j-m} + \lambda_j e_i^T v_0). \]

We use (6) and \( e_i^T e_i = 1 \), and we solve for \( \lambda_j \):

\[ \lambda_j = j e_i^T w^{1/2} w^{1/2} v_{j-1} - \sum_{m=1}^{j-1} (j \lambda_m e_i^T v_{j-m}). \quad (13) \]

We have therefore shown that if we already know \( (\lambda_n, v_n) \) for \( m = 0, 1, \ldots, j - 1 \), we can solve for \( \lambda_j \). Now, using this \( \lambda_j \), we can solve for \( v_j \) in the same way as before. We go back to (12), isolate all terms involving \( v_j \), and apply (6) to derive

\[ [w - w_j I] v_j = -j w^{1/2} w^{1/2} v_{j-1} + \sum_{m=1}^{j-1} (j \lambda_m v_{j-m}) \]

The right-hand side is clearly valid only for \( j \geq 1 \). The matrix on the left-hand side is the same one that appeared earlier:

\[
\begin{pmatrix}
0 & w_2 - w_1 & \cdots & w_N - w_1 \\
0 & w_3 - w_1 & \cdots & w_N - w_1 \\
\vdots & \vdots & \ddots & \vdots \\
0 & w_{j+2} - w_1 & \cdots & w_N - w_1
\end{pmatrix}
\]

We again set \( v^{(i)} = 0 \). For the remaining components, we have

\[ v^{(j)} = \frac{1}{w_k - w_1} \left( -j w^{1/2} w^{1/2} v_{j-1} + \sum_{m=1}^{j} (j \lambda_m v_{j-m}) \right) \]

for \( k = 2, 3, \ldots, N \).

Algorithmic improvements
Equations (13) and (14) complete the step of deriving both \( (\lambda_j, v_j) \) using only the previously derived eigenpairs \( (\lambda_m, v_m) \) for \( m = 0, 1, \ldots, j - 1 \), giving us a recursive solution procedure. Once we have determined \( (\lambda_n, v_n) \) for \( j = 0, 1, \ldots, J \), we can use these coefficients in (7) and thereby obtain approximations to the leading eigenpair \( (\lambda(\mu), v(\mu)) \). Hence we view (13) and (14) as an algorithm for computing the leading eigenpair.

Turning to the numerical implementation, we now describe two improvements to the algorithm given by (13) and (14).

First, we note that in (14), we always set \( v^{(i)} = 0 \). This implies that \( e_i^T v_j = 0 \) for all \( j \geq 1 \). This means that all terms under the summation symbol in (13) vanish, yielding the simplified update formula:

\[ \lambda_j = j e_i^T w^{1/2} w^{1/2} v_{j-1}. \quad (15) \]

Second, we note that (14) contains a binomial coefficient that becomes prohibitively large to compute for large \( j \). A natural question is whether these large coefficients are compensated by the inverse factors of \( j! \) in (7). To quantify this, we define

\[ \hat{\lambda}_j = \frac{\lambda_j}{j!} \quad \text{and} \quad \hat{v}_j = \frac{v_j}{j!}. \quad (16) \]

We then substitute \( \lambda_j = j! \hat{\lambda}_j \) and \( v_j = j! \hat{v}_j \) in (14) and derive

\[ v^{(j)} = \frac{1}{w_k - w_1} \left( -j(j-1)! w^{1/2} w^{1/2} v_{j-1} + \sum_{m=1}^{j} (j - m)! \hat{\lambda}_m \hat{v}_{j-m} \right) \]

Dividing through by \( j! \) and using (16), we derive, again for \( j \geq 1 \),

\[ \hat{v}^{(j)} = \frac{1}{w_k - w_1} \left( -w^{1/2} w^{1/2} \hat{v}_{j-1} + \sum_{m=1}^{j} \hat{\lambda}_m \hat{v}_{j-m} \right). \quad (17) \]
Applying the same substitutions in (15), we derive
\[ \lambda_j = e_i^T w^{1/2} s w^{1/2} v_{j-1} \] (18)

Using (16) in (7), we obtain
\[ \lambda(\mu) = \sum_{j=0}^{\infty} \hat{\lambda}_j \mu^j \quad \text{and} \quad v(\mu) = \sum_{j=0}^{\infty} \hat{v}_j \mu^j. \] (19)

Examining the equations in the \( \hat{\lambda} \) and \( \hat{v} \) variables, we see that all large binomial coefficients and factorials have disappeared. For this reason, in our Octave implementation of the perturbative method, we use the recursive system given by (18) and (17), together with the summation formula (19). Finally, applying (4), our answer for the leading eigenpair of the matrix \( \hat{Q}(\mu) \) defined by (2) is \((\lambda(\mu), w^{-1/2}v(\mu))\).

**Example**

Let us give an example of the perturbative method in practice. We set \( N = 5 \), \( \mu = 0.01 \), and \( w \) equal to a diagonal matrix whose entries along the diagonal are \((\phi, \phi^2, \phi^3, \phi^4, \phi^5)\) where \( d = 0.2 \) and \( \phi = 0.32768 \).

Let \( J \) denote the total number of iterations we run the perturbative method. Starting from \( \hat{\lambda}_0 = w_1 \) and \( \hat{v}_0 = e_1 \) as in (6), we iterate using (18) and (17) from \( j = 1 \) up to \( j = J \). We then evaluate the solution using (19) truncated at \( j = J \), giving us an approximate solution that we denote \((\lambda'(\mu), w^{-1/2}v'(\mu))\).

For an \( N \)-dimensional vector \( x = (x_1, x_2, ..., x_N) \), let \( \|x\|_\infty = \max_{1 \leq i \leq N} |x_i| \), the infinity-norm of \( x \), with respect to which the error of the approximate solution after \( J \) iterations is
\[ \text{error'} = \|w(w^{-1/2}v'(\mu)) - \lambda'(\mu)(w^{-1/2}v'(\mu))\|_\infty. \]

In Figure 3, we plot (in circles) the \( \log_{10} \) of the error as a function of the number of iterations \( J \), and (in solid black) the least-squares line of best fit to the data. From \( J = 1 \) to \( J = 14 \), the \( \log_{10} \) errors show a strongly linear trend, confirmed by the \( R^2 = 0.9958 \) value for the regression line. The slope of the line is approximately \(-0.9944\), implying
\[ \text{error'} \propto 10^{-0.9944J}. \]

Machine epsilon in Octave is approximately \( 2.2204 \times 10^{-16} \), and the error after \( J = 14 \) iterations is \( 2.2590 \times 10^{-16} \). Therefore, for this particular example, Figure 3 shows that the perturbative method converges exponentially to a solution with error on the order of machine epsilon.

**Extension to base/codon/word mutation models**

Consider now a matrix \( \mu_B \) defined as follows:
\[
\mu_B = \begin{pmatrix}
1 - \mu & k \mu & \mu & \mu \\
k \mu & 1 - \mu & \mu & \mu \\
k \mu & k \mu & 1 - \mu & k \mu \\
k \mu & k \mu & k \mu & 1 - \mu
\end{pmatrix},
\]

with constant parameters \( 0 < \mu < 1 \) and \( k \geq 1 \), and the indexing of both rows and columns corresponding to bases in the ordered set \( B = (A, G, C, T) \). This matrix was employed in prior work\(^{11}\) and corresponds to the Kimura two-parameter base mutation model.\(^{18}\) When \( k = 1 \), \( \mu_B \) represents the Jukes-Cantor mutation model.\(^{19}\)

Matrix \( \mu_B \) shares with matrix \( \mu \) the properties of linearity in parameter \( \mu \), and reduction to the identity
matrix when $\mu = 0$. The methods of this section therefore apply to $\mu_B$ directly.

More biologically realistic CMC models\textsuperscript{2–4} employ codon mutation models. Let $C = B^p$, the $p$-th Cartesian product of the set $B$. A codon $c \in C$ is a string of bases $b_1, b_2, \ldots, b_p$ of pre-specified length $p$ with $b_i \in B$.

The codon mutation models studied by Ardell and Sella assume independence of mutation of bases within codons and that all bases mutate according to the same model of evolution $\mu_B$. With these assumptions, mutation from any codon $c_1 \in C$ to another codon $c_2 \in C$ is represented by a matrix $\mu_c$ that is the $p$-th Kronecker power of a matrix $\mu_B$ as follows:

$$\mu_c = \bigotimes_{i=1}^{p} \mu_B,$$

where ($\bigotimes$) is the Kronecker product, and codons are indexed in both rows and columns in lexicographic order.

If $\lambda$ is the leading eigenvalue $\mu_B$, then $\lambda_c = \lambda^p$ is the leading eigenvalue of $\mu_c$. Similarly, if $v$ is the eigenvector corresponding to the leading eigenvalue of $\mu_B$, then $v_c = \bigotimes_{i=1}^{p} v$ is the eigenvector corresponding to the leading eigenvalue of $\mu_c$.

Therefore, the methods of this section allow calculation of the leading eigenpairs of matrices $\mu_c$ as Kronecker powers of leading eigenpairs of $\mu_B$.

**Homotopy method: quasispecies with multiple most fit genotypes**

We now give a second method for finding the leading eigenpair of the matrix $Q$ defined in (3). This method, which we call the homotopy method, is motivated by the desire to handle a fitness matrix $w$ that does not have a unique maximal element along its diagonal. The homotopy method produces accurate approximations of the leading eigenpair for such problems.

**Problem formulation**

Our goal is still to find the leading eigenpair of $\tilde{Q}$ defined in (2). As before, we will instead focus our attention on the symmetric matrix $Q$ defined by (3).

We define the matrix-valued function

$$F(\varepsilon) = \mu + \varepsilon (w^{1/2} \mu w^{1/2} - \mu).$$

Note that $F(0) = \mu$ and $F(1) = Q$. The function $F$ smoothly deforms $\mu$ into $Q$—such a function is often called a homotopy in the mathematical literature.\textsuperscript{16} Note that for all $\varepsilon \in [0, 1]$, the leading eigenpair $(\lambda(\varepsilon), v(\varepsilon))$ of $F$ must satisfy

$$F(\varepsilon)v(\varepsilon) = \lambda(\varepsilon)v(\varepsilon).$$

(21)

The basic idea behind the homotopy method is to use $F$ to form a bridge between $\mu$, a matrix whose leading eigenpair we already know, and $Q$, a matrix whose leading eigenpair we seek.

When $\varepsilon = 0$, (21) reduces to $\mu v(0) = \lambda(0)v(0)$. By the results provided in supplementary materials, we know that the leading eigenvalue of $F(0) = \mu$ is 1 with corresponding eigenvector $\tilde{1} = (1, 1, \ldots, 1)^T$, the column vector of $N$ ones. This implies that

$$\lambda(0) = 1 \quad \text{and} \quad v(0) = \tilde{1}.$$  \hspace{1cm} (22)

When $\varepsilon = 1$, (21) reduces to $Q v(1) = \lambda(1)v(1)$. Thus the question is how we can use our knowledge of $(\lambda(0), v(0))$ and the function $F(\varepsilon)$ to derive $(\lambda(1), v(1))$, the leading eigenpair of $Q$.

Unlike the perturbative solution, at no point will we assume that the entries of $w$ have a unique maximum. To make the derivation easier to read, we define

$$P = w^{1/2} \mu w^{1/2} - \mu,$$

so that $F(\varepsilon) = \mu + \varepsilon P$ and

$$F'(\varepsilon) = \frac{d}{d\varepsilon} F(\varepsilon) = P.$$  \hspace{1cm} (24)

**Homotopy solution**

We differentiate both sides of (21) once with respect to $\varepsilon$ and obtain

$$F'(\varepsilon)v(\varepsilon) + F(\varepsilon)v'(\varepsilon) = \lambda'(\varepsilon)v(\varepsilon) + \lambda(\varepsilon)v'(\varepsilon).$$

(25)

Since $F(\varepsilon)$ is symmetric for all $\varepsilon$, we see that the transposition of (21) can be written $v(\varepsilon)^TF(\varepsilon) = \lambda(\varepsilon) v(\varepsilon)^T$. Thus, after multiplying (25) through on the left by $v(\varepsilon)^T$, the second term on the left-hand side cancels the second term on the right-hand side, leaving
\[ \hat{\lambda}'(\varepsilon) = \frac{\nu(\varepsilon)^T P \nu(\varepsilon)}{\nu(\varepsilon)^T \nu(\varepsilon)}. \]  

(26)

Now let us substitute (26) back into (25). Solving for \( \nu'(\varepsilon) \), we have

\[ \nu'(\varepsilon) = -[F(\varepsilon) - \hat{\lambda}(\varepsilon)I]^{-1} F'(\varepsilon) \nu(\varepsilon). \]  

(27)

We now recognize (26) and (27) as a system of ordinary differential equations (ODEs) with \( \varepsilon \) playing the role of a time-like independent variable:

\[ \frac{d}{d\varepsilon} \left( \hat{\lambda}(\varepsilon) \right) = \left( \begin{array}{c} \nu(\varepsilon)^T P \nu(\varepsilon) \\ \nu(\varepsilon)^T \nu(\varepsilon) \\ -[\mu + P - \hat{\lambda}(\varepsilon)I]^{-1} P \nu(\varepsilon) \end{array} \right) \]  

(28)

We also recognize (22) as the initial conditions for this system of ODEs. Let us now describe an elementary algorithm for solving this system:

1. Set \( \lambda = 1 \) and \( \nu = \tilde{1} \). Fix an integer number of steps \( n_{\text{steps}} \) and then set \( \Delta \varepsilon = 1/n_{\text{steps}} \). Also set \( \varepsilon = 0 \), initially.

2. While \( \varepsilon < 1 \):
   a. Compute \( \lambda' \) using (26), i.e., \( \lambda' = (\nu^T P \nu)/(\nu^T \nu) \).
   b. Set \( \lambda \leftarrow \lambda + (\Delta \varepsilon) \lambda' \).
   c. Using this updated \( \lambda' \), compute \( \nu' \) using (27), i.e., \( \nu' = -[\mu + \varepsilon P - \lambda I]^{-1} P \nu \).
   d. Set \( \varepsilon \leftarrow \varepsilon + (\Delta \varepsilon) \nu' \).
   e. Set \( \varepsilon \leftarrow \varepsilon + (\Delta \varepsilon) \).

The algorithm will terminate in \( n_{\text{steps}} \) steps, yielding an approximation to the leading eigenpair of \( \hat{Q} \) that is stored in \( \lambda \) and \( \nu \).

To obtain the leading eigenvector of \( \hat{Q} \), we compute \( \tilde{\nu} = w^{-1/2} \nu \). The leading eigenpair of \( \hat{Q} \) is then \((\tilde{\lambda}, \tilde{\nu})\).

Example

We now give test results for the homotopy method applied to a particular problem. We set \( N = 8 \), \( \mu = 0.1 \), and \( w \) equal to a diagonal matrix whose entries along the diagonal are \((a, b, b, b, b, b, b, b)^T \) with \( a = 0.63631836 \) and \( b = 0.73306514 \). We also set the parameter, \( \varphi = 1 \).

We repeatedly run the algorithm given above for different values of \( n_{\text{steps}} \); specifically, we take \( n_{\text{steps}} = 10^j \), where \( j = 1, 2, 3, 4, 5, 6 \). For each value of \( n_{\text{steps}} \), we compute an approximation to the leading eigenpair, which we denote by \((\lambda', \nu')\). We then evaluate the residual error of this approximation using

\[ \text{error}^i = \| \mu w^{-1/2} \nu' - \lambda' (w^{-1/2} \nu') \|_\infty. \]

In Figure 4, we plot (in circles) the \( \log_{10} \) of the error as a function of the \( \log_{10} \) of the number of steps. We also plot (in solid black) the least-squares line of best fit to the data; for this line, \( R^2 = 0.9899 \) and the slope is approximately \(-1.0115 \), implying

\[ \text{error}^i \propto n_{\text{steps}}^{-1.0115} = n_{\text{steps}}^{-1} = \Delta \varepsilon. \]

Note that with \( n_{\text{steps}} = 10^3 \), the error is approximately \( 2 \times 10^{-5} \). This level of error is acceptable if we seek to use the homotopy method to reproduce, for example, Figure 4 in a previously published paper.\(^4\) Hence we use this value of \( n_{\text{steps}} \) as the default value in the CMCpy implementation of the homotopy method.

Further note that when \( n_{\text{steps}} = 10^6 \), even the elementary algorithm described above to solve the system of nonlinear ODEs (28) is capable of producing a residual error of approximately \( 2 \times 10^{-8} \).

For this example, the homotopy method displays convergence that is linear in \( \Delta \varepsilon \)—this relatively slow rate can be improved dramatically by using more
sophisticated methods to solve the system of nonlinear ODEs (28), an issue we leave for future work.

**Extension to base/codon/word mutation models**

In order to apply the homotopy method to models where the mutation matrix is given by $\mu_B$ as defined earlier, there is one requirement: we must be sure that $\mu_B$ has a unique maximal eigenvalue of 1. For the $\mu_B$ matrix, it turns out that we can explicitly derive all eigenvectors and eigenvalues, for general values of both $\mu$ and $k$. With the constraints $0 < \mu < 1$ and $k \geq 1$, the derivation that we give in supplementary materials below proves that the $\mu_B$ matrices have a unique largest eigenvalue of 1. Matrices $\mu_B$ therefore fulfill the minimum requirements for applicability of the methods of this section, after which the leading eigenpair for corresponding matrices $\mu_C$ may be calculated using Kronecker powers as before.

**Future outlook**

We succeeded in programming a GPU-based power method implementation that is faster than its CPU analogue; however, the performance gain that we obtained with it was not as great as we had hoped. Furthermore, even though our implementation is correct, we could not completely eliminate divergence in evolutionary trajectories in power method implementations arising from differences in machine number representations and precision across platforms. We believe that this arises from deviations in the way double precision floating point numbers are represented, computed on, and rounded in CUDA compute capability 1.3. Perhaps utilization of the cuBLAS library in the future would make performance closer to Numpy and better conform to IEEE standards.

Furthermore, while the power method converges linearly, our new perturbation method provides exponential convergence with the same accuracy. However, this method cannot handle the case of non-unique most fit genotypes that occurs in CMC models at their initialized state. On the other hand, our new homotopy method does handle this case, yet as currently implemented it also converges linearly, rather slower than the power method (results not shown). Incorporation of more sophisticated methods to solve systems of nonlinear ordinary differential equations should dramatically improve performance of our new homotopy method application. We leave further development of both methods and their implementation in CMCpy for future work; perhaps other CPU or GPU implementations of them will compete with Numpy. More generally, our analytical results greatly expand the domain of quasispecies models that can be accurately solved using analytical approaches, particularly multi-site models with biologically realistic mutation parameters.

A variety of open problems remain concerning CMC models and in the field of the evolution of the genetic code. Current models of the genetic code have not yet integrated a theory for the origin of translation *per se*. We believe that extensions to CMC models will better address such fundamental questions and hope that CMCpy and our analytical solutions to quasispecies models will play a role in that work.

**Acknowledgements**

The authors would like to thank members of the Ardell Lab, Jason Davis and Suzanne Sindi, for their comments and suggestions on this work, as well as Prof. Masakatsu Watanabe and Prof. Michael Colvin for their leadership in creating and running the Undergraduate Research in Computational Biology (URCB) program at UC Merced, without which the collaboration of undergraduate Peter Becich would not have been possible.

**Funding**

DHA acknowledges the UC Merced division Graduate Research Council for a UC Faculty Research and Chancellor’s awards that supported undergraduate BPS, as well as the NSF-funded program in Undergraduate Research in Computational Biology at UC Merced (DBI-1040962), which supported and trained PJB and paid publication costs.

**Competing Interests**

Author(s) disclose no potential conflicts of interest.

**Author Contributions**

Conceived CMCpy project: DHA. Wrote software: DHA, PJB, BPS. Analyzed data: DHA, HSB, PJB, BPS. Analytical results: HSB, with minor contributions from DHA. Wrote first draft of the manuscript:
Disclosures and Ethics
As a requirement of publication author(s) have provided to the publisher signed confirmation of compliance with legal and ethical obligations including but not limited to the following: authorship and contributorship, conflicts of interest, privacy and confidentiality and (where applicable) protection of human and animal research subjects. The authors have read and confirmed their agreement with the ICMJE authorship and conflict of interest criteria. The authors have also confirmed that this article is unique and not under consideration or published in any other publication, and that they have permission from rights holders to reproduce any copyrighted material. Any disclosures are made in this section. The external blind peer reviewers report no conflicts of interest.

References
1. Sella G, Ardell DH. The impact of message mutation on the fitness of a genetic code. J Mol Evol. 2002;54(5):638–51.
2. Ardell DH, Sella G. On the evolution of redundancy in genetic codes. J Mol Evol. 2001;53(4–5):269–81.
3. Ardell DH, Sella G. No accident: genetic codes freeze in error-correcting patterns of the standard genetic code. Philos Trans R Soc Lond B Biol Sci. 2002;357(1427):1625–42.
4. Sella G, Ardell DH. The coevolution of genes and genetic codes: Crick’s frozen accident revisited. J Mol Evol. 2006;63(3):297–313.
5. Vetsigian K, Woese C, Goldenfeld N. Collective evolution and the genetic code. Proc Natl Acad Sci U S A. 2006;103(28):10696–701.
6. Eigen M. Molecular self-organization and the early stages of evolution. Q Rev Biophys. 1971;4(2):149–212.
7. Bull JJ, Meyers LA, Lachmann M. Quasispecies made simple. PLoS Comput Biol. 2005;1(6):e61.
8. Higgs PG. Error thresholds and stationary mutant distributions in multi-locus diploid genetics models. Genet Res. 1994;63:63–78.
9. Wilke CO. Quasispecies theory in the context of population genetics. BMC Evol Biol. 2005;5:44.
10. Swetina J, Schuster P. Self-replication with errors. A model for polynucleotide replication. Biophys Chem. 1982;16(4):329–45.
11. Eigen M, McCaskill J, Schuster P. Molecular quasi-species. J Phys Chem. 1988;92:6881–91.
12. Hiroshi F. Application of Eigen’s evolution model to infinite population genetic algorithms with selection and mutation. Complex Systems. 1996;10:345–66.
13. Rellich F. Perturbation Theory of Eigenvalue Problems. New York: New York University, Courant Institute of Mathematical Sciences; 1954.
14. Tosio K. Perturbation Theory for Linear Operators. Berlin: Springer-Verlag; 1995.
15. Baumgartel H. Analytic Perturbation Theory for Matrices and Operators. Basel: Birkhäuser Verlag; 1985.
16. Chu MT. A simple application of the homotopy method to symmetric eigenvalue problems. Linear Algebra Appl. 1984;59:85–90.
17. Klöckner A, Pinto N, Lee Y, Catanzaro B, Ivanov P, Fasih A. PyCUDA and PyOpenCL: a script-based approach to GPU run-time code generation. Parallel Comput. 2012;38:157–74.
18. Kimura M. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J Mol Evol. 1980;16(2):111–20.
19. Jukes T, Cantor C. Evolution of protein molecules. In: Munro H, editor. Mammalian Protein Metabolism, III. New York: Academic Press; 1969:21–132.
20. Higgs PG. A four-column theory for the origin of the genetic code: tracing the evolutionary pathways that gave rise to an optimized code. Biol Direct. 2009;4:16.
21. Koonin EV, Novozhilov AS. Origin and evolution of the genetic code: the universal enigma. J Exp Biol. 2009;212(9):99–111.
22. Thust S. A colorful origin for the genetic code: information theory, statistical mechanics and the emergence of molecular codes. Phys Life Rev. 2010;7(3):362–76.
23. Philip GK, Freeland SJ. Did evolution select a nonrandom “alphabet” of amino acids? Astrobiology. 2011;11(3):235–40.
24. Caporaso JG, Knight R. New insight into the diversity of life’s building blocks: evenness, not variance. Astrobiology. 2011;11(3):197–8.
Supplementary Materials

**Supplementary 1: Derivation of complete eigendecomposition of circulant \( \mu \) matrices**

The matrix \( \mu \) is circulant, an important property because the eigenvalues and eigenvectors of any circulant matrix can be written in closed form. Let \( i = \sqrt{-1} \) and define for \( j = 0, 1, \ldots, N - 1 \) the roots of unity

\[
\omega_j = \exp \left( \frac{2\pi ij}{N} \right).
\]

Then the \( N \) eigenvalues of \( \mu \) are, for \( j = 0, 1, \ldots, N - 1 \), given by

\[
s_j = (1 - 2\mu) + \mu [\omega_j + \omega_j^{-1}]
= (1 - 2\mu) + \mu \left[ \exp \left( \frac{2\pi ij}{N} \right) + \exp \left( \frac{2\pi i(N-1)}{N} \right) \right]
= (1 - 2\mu) + 2\mu \cos \left( \frac{2\pi ij}{N} \right).
\]

(29)

Let us now show that the maximum eigenvalue is always equal to 1 and that this eigenvalue is unique. Consider the function

\[
f(x) = (1 - 2\mu) + 2\mu \cos \left( \frac{2\pi x}{N} \right)
\]

where \( x \) is a real variable satisfying \(- (N - 1) \leq x \leq N - 1\). Then using the methods of calculus, we find that the only \( x \in \{- (N - 1), N - 1\}\) that satisfies \( f'(x) = 0 \) and \( f''(x) < 0 \) is \( x = 0 \). At the boundary, we have

\[
f(\pm (N - 1)) = (1 - 2\mu) + 2\mu \cos (2\pi (N - 1)/N)
= (1 - 2\mu) + 2\mu \cos (2\pi/N)
= 1 + 2\mu [\cos (2\pi/N) - 1] < f(0),
\]

as long as \( \mu > 0 \) and \( N > 1 \). Hence \( x = 0 \) is the unique global maximum of \( f(x) \) on the interval \( x \in \{- (N - 1), N - 1\}\). If we restrict \( x \) to integer values in \([0, N - 1]\), then \( x = 0 \) will still be the unique global maximum of \( f(x) \). Therefore, the unique maximum eigenvalue \( s_j \) occurs at \( j = 0 \) and is always equal to \( s_0 = 1 \).

### Eigenvectors

Let \( v^* \) denote \( v^T \), the conjugate transpose of the vector \( v \). The eigenvector of \( \mu \) that corresponds to the eigenvalue \( s_j \) is

\[
y_j = \frac{1}{\sqrt{N}} (1, \omega_j, \omega_j^2, \ldots, \omega_j^{N-1})^T,
\]

(30)

written here as a column vector. The \( 1/\sqrt{N} \) factor is included to normalize the eigenvalue so that \( y_j^*y_j = 1 \). Note that the eigenvector corresponding to the eigenvalue \( s_0 = 1 \) is particularly simple:

\[
y_0 = \frac{1}{\sqrt{N}} (1, 1, \ldots, 1)^T = \frac{1}{\sqrt{N}} \mathbf{1}.
\]

Since we have explicit expressions for the eigenvalues (29) and eigenvectors (30), we can write down a useful outer product representation of \( \mu \):

\[
\mu = \sum_{j=0}^{N-1} y_j y_j^*.
\]

(31)

Let us now show that (30), as defined, is an orthonormal system of eigenvectors. We take the Hermitian inner product of an eigenvector \( y_j \) with an eigenvector \( y_k \), assuming \( j \neq k \):

\[
y_j^*y_k = \frac{1}{N} \sum_{l=0}^{N-1} (\bar{\omega}_l)^j \omega_l^k
= \frac{1}{N} \sum_{l=0}^{N-1} \exp \left( \frac{-2\pi ilj}{N} \right) \exp \left( \frac{2\pi ikl}{N} \right)
= \frac{1}{N} \sum_{l=0}^{N-1} \exp \left( \frac{2\pi i(k-j)}{N} \right)
= \frac{1}{N} 1 - \exp(2\pi i(k-j))/N
= 0.
\]

If instead we have \( k = j \), then we obtain from the third line above:

\[
y_j^*y_j = \frac{1}{N} \sum_{l=0}^{N-1} l' = 1.
\]

Hence \( \{ y_j \}_{j=0}^{N-1} \) is an orthonormal system of eigenvectors.
Supplementary 2: Derivation of complete eigendecomposition of $\mu_B$ matrices

It is clear that each row of $\mu_B$ sums to 1. This, by itself, implies that $v_1 = (1, 1, 1, 1)^T$ is an eigenvector of $\mu_B$ with corresponding eigenvalue $s_1 = 1.$

In what follows, we use the fact that

$$
\mu_B = a \begin{pmatrix} 1 - \mu & k\mu & \mu & \mu \\ k\mu & k + 2 & 1 - \mu & k + 2 \\ k\mu & k + 2 & k\mu & k + 2 \\ k + 2 & k + 2 & k + 2 & 1 - \mu \end{pmatrix} + b \begin{pmatrix} \mu & \mu & \mu & \mu \\ k\mu & k + 2 & 1 - \mu & k + 2 \\ k\mu & k + 2 & k\mu & k + 2 \\ k + 2 & k + 2 & k + 2 & 1 - \mu \end{pmatrix} + c \begin{pmatrix} \mu & \mu & \mu & \mu \\ k\mu & k + 2 & 1 - \mu & k + 2 \\ k\mu & k + 2 & k\mu & k + 2 \\ k + 2 & k + 2 & k + 2 & 1 - \mu \end{pmatrix} + d \begin{pmatrix} \mu & \mu & \mu & \mu \\ k\mu & k + 2 & 1 - \mu & k + 2 \\ k\mu & k + 2 & k\mu & k + 2 \\ k + 2 & k + 2 & k + 2 & 1 - \mu \end{pmatrix}.
$$

(32)

The four column vectors that appear on the right-hand side are the four columns of $\mu_B.$

It is clear that if we subtract the fourth column of $\mu_B$ from the third column of $\mu_B,$ then the first two entries cancel, while the last two entries are exact opposites of one another:

$$
\begin{pmatrix} \mu \\ \kappa + 2 \\ \mu \\ \kappa + 2 \\ 1 - \mu \\ \kappa \mu \\ (\kappa + 2) \end{pmatrix} - \begin{pmatrix} \mu \\ \kappa + 2 \\ \mu \\ \kappa + 2 \\ 1 - \mu \\ \kappa \mu \\ (\kappa + 2) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.
$$

By fact (32), we can rewrite the left-hand side and thereby derive:

$$
\mu_B \begin{pmatrix} 0 \\ 1 \\ 0 \\ -1 \end{pmatrix} = (1 - \mu - \kappa \mu / (\kappa + 2)) \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.
$$

Hence we have shown that $v_2 = (0, 1, 1, -1)^T$ is an eigenvector of $\mu_B$ with corresponding eigenvalue $s_2 = 1 - \mu - \kappa \mu / (\kappa + 2).$

In a very similar way, we can see that

$$
\begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} = (1 - \mu - \kappa \mu / (\kappa + 2)) \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix},
$$

showing that $v_3 = (1, -1, 0, 0)^T$ is an eigenvector of $\mu_B$ with corresponding eigenvalue $s_3 = 1 - \mu - \kappa \mu / (\kappa + 2).$

Note that $s_2 = s_3.$ Since $v_2$ and $v_3$ are clearly orthogonal, we see that $\{v_2, v_3\}$ span a two-dimensional eigenspace, and that the geometric multiplicity of $s_2$ equals its algebraic multiplicity. This is consistent with the fact that $\mu_B$ is, by definition, a real symmetric matrix and therefore, by the spectral theorem, must possess four real eigenvalues together with an orthonormal basis of eigenvectors.

As we have found three orthogonal eigenvectors $\{v_1, v_2, v_3\},$ we appeal to orthogonality to find $v_4.$ Let $v_4 = (1, x, y, z)^T.$ Orthogonality with $v_2$ and $v_3$ respectively imply $1 - x = 0$ and $y - z = 0,$ so $v_4 = (1, 1, y, y)^T.$ Now, orthogonality with $v_1$ implies that $2 + 2y = 0,$ so $y = -1,$ and we have $v_4 = (1, 1, -1, -1)^T.$ Multiplying $\mu_B$ by $v_4,$ we see that

$$
\mu_B \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 - \mu + \kappa \mu / (\kappa + 2) \\ 1 - \mu + \kappa \mu / (\kappa + 2) \\ 1 - \mu + \kappa \mu / (\kappa + 2) \\ 1 - \mu + \kappa \mu / (\kappa + 2) \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ -1 \end{pmatrix},
$$

confirming that $v_4$ is an eigenvector with corresponding eigenvalue

$$
s_4 = 1 - \mu + \frac{\kappa \mu}{\kappa + 2} = 1 - \frac{4\mu}{\kappa + 2}.
$$

Normalizing the eigenvectors, we obtain the set
$y_1 = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad y_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}, \quad y_3 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix},

\quad y_4 = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix},$

where $(y_j)^* y_j = 1$ for $j = 1, 2, 3, 4$, and each eigenvector $y_j$ corresponds to the eigenvalues $s_j$ given above. Indeed, examining the expressions of the eigenvalues $s_j$, and using the fact that $0 < \mu < 1$ and $k \geq 1$, it is clear that $s_1 = 1$ is the unique maximal eigenvalue of $\mu_B$.

Supplementary 3: Benchmark script in R to generate Figure 2.

Supplementary 4: Matlab/Octave implementations of the perturbative and homotopy methods presented in this manuscript.