**Bertels et al., Discovering complete quasispecies in bacterial genomes**

**Supplementary Information**

**Defining functions and importing data**

**Functions and definitions**

Function that calculates equilibrium frequencies for a given set of fitness values and a given mutation matrix:

```mathematica
CalculateEquilibriumFrequencies[fitness_, mutmatrix_] :=
    Module[{},
        W := Transpose[mutmatrix*fitness];
        ESys := Eigensystem[W];
        sorted = Sort[ESys];
        Return[sorted[[2, 1]]/Total[sorted[[2, 1]]]]
    ];
```

Functions that calculate the mutation matrix.
(*calculate mutation rate for a single entry in the mutation matrix*)

entry[i_, j_, u_, L_] := Module[{backMutation, forwardMutation, maintain},
   backMutation = (i - 1) * u/3; (* non-zero back mutation rate*)
   forwardMutation = u * L - (i - 1) * u/3; (* forward mutation rate with non-zero back mutation rate*)
   maintain = 1 - backMutation - forwardMutation;
   Return[If[i == j, maintain,
      If[i == j + 1, backMutation, If[i == j - 1, forwardMutation, 0]]]]];

makeMutMatrix[u_, L_, mutclasses_] := Module[{matrix},
   matrix = Table[entry[i, j, u, l], {i, 1, mutclasses}, {j, 1, mutclasses}];
   (*matrix[[mutclasses, mutclasses]] = 1 - matrix[[mutclasses, mutclasses - 1]];*)
   matrix[[mutclasses, mutclasses - 1]] = 0; (*no back mutations for the last mutation class*)
   matrix[[mutclasses, mutclasses]] = 1; (*once a sequence reaches the last mutation class it will stay there*)
   Return[matrix];
]

Function that calculates the fitness values from the equilibrium frequencies with a given mutation matrix.
CalculateFitness[frequencies_, mutmatrix_, mutclasses_, name_, u_] :=
Module[{x, f, farray, q, ϕ, equilibriumEq, eqs, s, min, fitnessTable, i, j},
  x = frequencies; (*frequencies of the different mutation classes*)
  fmutclasses - 1 = 1; (*set the frequency of the lowest mutation class to 1*)
  farray = Table[f_{i - 1}, {i, 1, mutclasses}];
  q = mutmatrix; (*the mutation matrix is called q*)
  ϕ[x_, f_] := Σ_{i = 1}^{mutclasses} (x[i] f[i]); (*ϕ for no back mutations*)
  equilibriumEq[i_] := -ϕ[x, farray] x[i] + Σ_{j = 1}^{mutclasses} (x[j] farray[j] q[j, i]);
  eqs = Table[0 == equilibriumEq[i], {i, 1, mutclasses}];
  (*Solve quasispecies model for fitness*)
  s = NSolve[eqs, farray[1 ;; mutclasses - 1]];
  (*if there are fitness values below one; scale these to one*)
  min = Min[Table[s[[1, j, 2]], {j, 1, mutclasses - 1}]], 1];
  fitnessTable = Table[s[[1, j, 2]]/min, {j, 1, mutclasses - 1}];
  fitnessTable = Append[fitnessTable, 1/min];
  (*add mutation rate to output*)
  fitnessTable = Append[fitnessTable, u ];
  (*add observed frequencies to output*)
  For[j = 1, j <= mutclasses, j++,
    fitnessTable = Append[fitnessTable, x[j] // N];
  ];
  fitnessTable = Append[fitnessTable, name];
  Return[fitnessTable];
];

Function that processes the input data (i.e. sequence frequencies) and varies the frequencies of the last mutation class by "var".
ProcessData[realFreqs_, var_] := Module[{inner, realTab, names, seqLength, temp},

j = 0;
i = 0;
realTab = {};
inner = {};
names = {};
seqLength = {};

While[i < Length[realFreqs],
  i++;
  j = 0;
  inner = {realFreqs[[i + j]][[3]]};

  While[
    i + j < Length[realFreqs] && realFreqs[[i + j]][[1]] == realFreqs[[i + j + 1]][[1]],
    j++;
    inner = Append[inner, realFreqs[[i + j]][[3]]];
  ];

  (*if there is not enough data add 1s to REPIN population count, CAUTION: THIS WILL SIGNIFICANTLY ALTER THE DATA, WE ONLY ADDED THIS FOR CONSISTENCY REASONS It may be better to reduce the number of mutation classes if this is an issue.*)
  While[Length[inner] < mutclasses, inner = Append[inner, 1]];]

  (*vary the last mutation class by "var" This demonstrates the effect small variations in the last mutation class have on the inferred fitness values.*)
  For[k = -var, k <= var, k++,
    temp = inner;
    temp[[Length[temp]]] = temp[[Length[temp]]] + k;
    If[temp[[Length[temp]]] > 0,
      realTab = Append[realTab, temp];
      names = Append[names, realFreqs[[i]][[1]]];
      seqLength = Append[seqLength, realFreqs[[i]][[4]]];
    ];
  ];

i = i + j;

];

names = StringReplace[names, "_largestCluster" -> ""];

Return[{realTab, seqLength, names}];}
Number of sequence classes is given by mutclasses

```math
mutclasses = 6;
(* var=0 is the default,
   which will only infer fitness values for the provided dataset;
   larger values of var will add fitness inferences to the results
   where all integer values up to var are added or subtracted. *)
var = 0;
```

Following Wielgoss et al. 2011 the mutation rate in E. coli is set to \( u = 8.9 \times 10^{-11} \);

**Importing data**

Next we import the sequence frequencies from the real data. The format of the imported data is in the form of a table with three columns, each referring to
- 1. Organism name
- 2. Sequence Class
- 3. Sequence Frequency
realFreqs = Import[NotebookDirectory[] <> "File_S9.txt", "Table"];
data = ProcessData[realFreqs, var];
realTab = data[[1]];
names = data[[3]];(*organism names*)
seqLength = data[[2]];
(*calculate relative frequencies from absolute frequencies*)
freqs = Table[realTab[[i]]/Total[realTab[[i]]], {i, 1, Length[realTab]}];

i = 0;
newf = {};
While[i < Length[freqs],
  i++;
  j = 0;
  inner = {};
  While[j < mutclasses - 1,
    j++;
    inner = Append[inner, freqs[[i, j]]];
  ];
  (*if there are more entries than mutation classes then add
  all remaining entries and put them in the last mutation class*)
  last = Sum[freqs[[i, j]], {j, mutclasses, Length[freqs[[i]]]}];
  inner = Append[inner, last];
  newf = Append[newf, inner];
]
freqs = newf;

---

Model

Solving for the equilibrium frequencies of the mutation classes.

The fitness of the mutation classes is given by the fitnessTable.
outputUnscaled = {};  
For[i = 1, i ≤ Length[freqs], i++,  
    L = seqLength[i]];  
(* sequence length, which is different for different data sets *)  
q = makeMutMatrix[u, L, mutclasses];  
(* the mutation matrix also differs due to different sequence lengths *)

(* equilibrium frequencies x *)  
    x = Table[freqs[i, j], {j, 1, Length[freqs[i]]}];

fitnessTable = CalculateFitness[x // N, q, mutclasses, names[i], u];  
fitnessTable = Append[fitnessTable, seqLength[i]]];  
outputUnscaled = Append[outputUnscaled, fitnessTable];

Export[NotebookDirectory[] <>  
    "quasispecies_fitness_unscaled_" <> ToString[mutclasses] <>  
    ",_" <> ToString[var] <> ".txt", outputUnscaled, "Table"];

Exporting raw output and scaled with mutation rate

What does scaling do?

If we want to determine the fitness values that give us comparable results but for a higher mutation rate then we need to do the following:

1. Calculate the number of generations one time step at the new mutation rate corresponds to under the old mutation rate.

(* scale results for a mutation rate of 10^{-4} *)
newU = 10^{-4};  
gens = newU / u;  
(* calculate new fitness values *)

2. Scale up each fitness value by the number of generations we calculated above and calculate the new equilibrium frequencies for the new mutation rate.
output = outputUnscaled;
For[i = 1, i <= Length[freqs], i++,
    fitness = {};
    For[j = 1, j <= mutclasses, j++,
        newFit = output[[i]][[j]] \[\text{\textasciitilde}\] gens;
        output[[i]][[j]] = newFit;
        fitness = Append[fitness, newFit];
    ];
    output[[i]][[mutclasses + 1]] = newU \ // N;
(*calculate new equilibrium frequencies for the new fitness values*)
eqfreqs = CalculateEquilibriumFrequencies[
    fitness, makeMutMatrix[newU, seqLength[[i]], mutclasses]];

For[j = 1, j <= mutclasses, j++,
    k = mutclasses + 1;
    output[[i]][[k + j]] = eqfreqs[[j]];]}
(*output the scaled fitness values and equilibrium frequencies*)
Export[NotebookDirectory[] <> "quasispecies_fitness_" <>
    ToString[mutclasses] <> "_" <> ToString[var] <> ".txt", output, "Table"];

**Calculate error thresholds**
For each species we will calculate the fitness value, at which the frequency of the master sequence drops below 1% by depressing the whole fitness landscape in increments of $10^{-12}$. 
error = outputUnscaled;
For[i = 1, i ≤ Length[freqs], i++],

(* equilibrium frequencies *)

q = makeMutMatrix[u, seqLength[[i]], mutclasses];
f = Table[outputUnscaled[[i]][[j]], {j, 1, mutclasses}];
start = IntegerPart[(f[[1]] - 1) * 10^12];
threshold = 0.01;

For[j = 1, j < start, j++,
fint = Table[IntegerPart[(f[[i]] - 1) * 10^12], {i, 1, mutclasses}];
For[k = 2, k ≤ mutclasses, k++,
  If[fint[[k]] > 0, f[[k]] = 1 + (fint[[k]] - 1) * 10^-12];
];
f[[1]] = 1 + (start - j) 10^-12;

effreqs = CalculateEquilibriumFrequencies[f, q];
If[effreqs[[1]] < threshold,
j = start;
];
]

For[j = 1, j < Length[f], j++,
  error[[i]][[j]] = f[[j]] // N;
];
For[j = 1, j <= mutclasses, j++,
k = mutclasses + 1;
  error[[i]][k + j] = effreqs[[j]];
];
]

Export[NotebookDirectory[] <> "error_threshold_" <>
  ToString[mutclasses] <> "_" <> ToString[var] <> ".txt", error, "Table"];