Supplementary material

Title: Bet-hedging via polyandry. A comment on 'Mating portfolios: bet-hedging, sexual selection and female multiple mating'

Authors: Jonathan M. Henshaw and Luke Holman

Here we present the computer code for the statistical analysis, written for Wolfram Mathematica v.9. The code appears in regular typeface, with comments in bold.

```
(* The code is structured as follows;

Section 1 is for user input of parameters;

Section 2 is to input and process data from the original experiment;

Section 3 replicates the original method of Garcia-Gonzalez et al. (2015) to
calculate the mean and confidence intervals for the difference in geometric mean
fitness between mating treatments;

Section 4 calculates the mean and central ranges for Δ using both Gillespie's
measure and the geometric mean fitness, following our new method;

Section 5 calculates the distributions of Δ for both fitness measures under the
null hypothesis, and uses these to calculate approximate p-values for the mean
Δ -values that were calculated in Section 4 *)


(* Section 1: User input *)

(* Choose the experiment. Options are 1 and 2 *)

experiment = 1;

(* Choose the fitness measure and environment. Options are;
1 = offspring viability in environment A;
2 = offspring viability in environment B;
3 = offspring viability in environment A+B;
4 = fertilisation rates *)

environment = 1;

(* Set the significance level *)

alpha = 0.05;

(* Set the population size for calculating Gillespie's measure *)

PopSize = 12;
```

(* Set the number of females and generations in the original dataset *)

NumFemalesExp = 12;
NumGenerationsExp = 3;

(* Set the number of females and generations for simulation of the geometric mean *)

NumFemalesSim = 12;
NumGenerationsSim = 3;

(* SamplesPerRun sets the number of permutations of generations for the purposes of the original method of Garcia-Gonzalez et al. (2015), and also the number of bootstrap resamples for our new method;

NumRuns sets the number of times bootstrap resampling is repeated in order to estimate the null distribution and calculate p-values *)

SamplesPerRun = 10000;
NumRuns = 10000;

(* Section 2: Processing data from the original experiment *)

(* Input data from the original experiment. Data for offspring viability and fertilisation rates are taken from columns Q and D respectively of the data supplement to Garcia-Gonzalez et al. (2015) *)

If[experiment == 1,

OffspringViability = {.3333333330, .70, .750, .4333333330, .6166666670, .3166666670, .7333333330, .7916666670, .851063830, .40, .6666666670, .297872340, .1666666670, .350, .241379310, .350, .550, .3559322030, .30, .3913043480, .360, .450, 0, .2653061220, .1333333330, .2166666670, .0166666670, .1166666670, .1166666670, .1166666670, .150, .1666666670, .0851063830, .2553191490, .1333333330, .0666666670, 0, .250, .0833333330, .10, .2166666670, .1166666670, .1025641030, .420, .4333333330, .1538461540, .3148148150, .250, .450, .1333333330, .2666666670, .150, .0833333330, .10, .3833333330, .0833333330, .1666666670, .10, .0166666670, .0833333330, .250, .650, 0, .3833333330, .6333333330, .250, .550, .5384615380, .4583333330, .3166666670, .2884615380, .43750, .350, .370370370, .40, .450, .370370370, .7166666670, .50, .3333333330, .350, .450, .3666666670, .30, .360, .5333333330, .6666666670, .5306122450, .5833333330, .6666666670, .3250, .3921568630, .6923076920, .40, .3333333330, .280, .40, .8235294120, .250, .2307692310, .4444444440, .5555555560, .1666666670, .520, .3333333330, .4545454550, .30, .2173913040, .150, .350, .1333333330, .1333333330, .3833333330, .2333333330, .3333333330, .2333333330, .3518518520, .30, .20, .20, .150, .1764705880, .3396226420, .3166666670, .1176470590, .2830188680, .4651162790, .20, .1666666670, .5263157890, .2727272730, .2439024390, .2666666670, .0333333330, .2333333330,

.1666666670, .1666666670, .2166666670, .2592592590, .180, .0833333330, .080,
.120, .316666667};

FertilisationRates =
N[{79, 94, 93, 94, 97, 92, 15, 8, 16, 90, 95, 26, 95, 4, 10, 22, 4, 89, 92,
99, 98, 98, 96, 85, 10, 11, 6, 99, 9, 20, 87, 2, 96, 7, 14, 97, 17, 25,
22, 82, 57, 68, 82, 96, 96, 31, 67, 39, 75, 17, 2, 83, 88, 30, 6, 4, 9,
79, 82, 88, 19, 18, 30, 9, 18, 4, 81, 4, 20, 29, 13, 77, 59, 22, 41, 35,
23, 26, 90, 90, 86, 24, 19, 21, 10, 36, 14, 82, 23, 88, 51, 44, 40, 61,
72, 65, 1, 2, 1, 59, 85, 51, 68, 47, 6, 1, 53, 32, 20, 13, 18, 4, 9, 13,
28, 29, 15, 24, 74, 56, 8, 29, 64, 21, 47, 14, 16, 5, 13, 61, 60, 66, 2,
0, 0, 15, 11, 0, 53, 25, 1, 2, 74, 2, 4, 2, 4, 5, 5, 1, 0, 2, 3, 2, 4, 2,
6, 15, 14, 4, 8, 0, 75, 62, 75, 84, 92, 78, 90, 100, 95, 41, 10, 68, 87,
76, 92, 89, 14, 14, 5, 9, 8, 14, 8, 9, 5, 4, 6, 7, 10, 0, 11, 9, 7, 6,
15, 3, 79, 70, 68, 22, 42, 32, 82, 77, 53, 50, 0, 1, 20, 85, 4, 77, 69,
14}];,

OffspringViability = {.1142857140, .3666666670, .0333333330, .2058823530,
.4333333330, .2166666670, .1166666670, .30, .10, .40, .2666666670, .2333333330,
.3333333330, .1833333330, .3225806450, .20, .1333333330, .781250, .2333333330,
.250, .3166666670, .4333333330, .2833333330, .2666666670, .2972972970,
.5166666670, .3833333330, .0277777780, .1166666670, .40, .250, .20, .20,
.1833333330, .2333333330, .1333333330, .150, .0333333330, 0, .150, .1166666670,
0, .050, .0166666670, .0166666670, .150, .0333333330, .1333333330,
.1666666670, .1166666670, .1166666670, .30, .1833333330, .4333333330,
.2333333330, .2666666670, .2666666670, .1333333330, .450, .80, .40, .3333333330,
.4333333330, .7833333330, .5833333330, .60, .5833333330, .4166666670,
.0666666670, .650, .6333333330, .650, .250, .3333333330, .3166666670, .10, .30,
.4333333330, .3666666670, .2833333330, .40, .1166666670, .30, .5166666670,
.0666666670, .1166666670, .2833333330, .150, .1333333330, .1833333330,
.0833333330, .2333333330, .1666666670, .1333333330, .2833333330, .150,
.0166666670, .0333333330, .050, .050, .1666666670, .0666666670, .050,
.0833333330, .1166666670, .1166666670, .1666666670, .1166666670, .20, .350,
.2833333330, .2166666670, .20, .350, .40, .2666666670, .2833333330, .1833333330,
.2333333330, .250, .1833333330, .2166666670, .1333333330, .050, .1666666670,
.1166666670, .150, .20, .20, .20, .1833333330, .1166666670, .350, .20, .050, .40, .50,
.30, .2166666670, .1166666670, .1333333330, .50, .4166666670, .266666667};

FertilisationRates =
N[{8, 12, 8, 25, 77, 77, 9, 24, 21, 48, 59, 23, 52, 47, 53, 25, 28, 23, 16,
29, 12, 13, 23, 9, 12, 7, 1, 78, 52, 48, 23, 13, 19, 74, 62, 72, 3, 5,
5, 91, 87, 86, 91, 88, 90, 97, 94, 89, 92, 89, 89, 77, 95, 99, 30, 35,
35, 40, 47, 45, 87, 64, 58, 38, 35, 42, 53, 38, 43, 46, 47, 46, 32, 52,
35, 74, 67, 66, 72, 60, 74, 72, 72, 71, 81, 77, 84, 85, 87, 74, 43, 27,
29, 35, 37, 39, 22, 24, 20, 55, 65, 44, 48, 44, 42, 33, 34, 33, 93, 78,
60, 97, 96, 90, 61, 55, 52, 81, 75, 90, 91, 82, 84, 78, 71, 78, 44, 36,
44, 97, 98, 99, 100, 99, 97, 98, 98, 98, 97, 98, 99, 99, 98, (99 + 98)/2,
58, 46, 60, 36, 44, 44, 95, 96, 95, 61, 42, 59, 58, 73, 50, 77, 72, 62,
32, 39, 46, 51, 28, 39, 47, 41, 39, 44, 46, 55, 60, 48, 49, 42, 39, 44,
100, 98, 98, 99, 99, 100, 53, 59, 47, 100, 100, 100, 99, 99, 99, 96, 97,

```
    97, 100, 100, 100, 88, 96, 87, 100, 100, 100, 100, 98, (100 + 98)/2, 96,
    100, 100, 99, 100, (99 + 100)/2}];];
```

(* From here onwards the code should run without user input *)

(* Separate the offspring viability data into arrays for each mating treatment (monandry M or polyandry P) and environment (A or B). Offspring viability in environment A+B is defined as the average viability across environments A and B;

Then separate the fertilisation rates data into arrays for each mating treatment;

The results are tables of size 12 x 3 *)

```
MA = Table[OffspringViability[[i ;; i + 2]], {i, 1, 133, 12}];
MB = Table[OffspringViability[[i ;; i + 2]], {i, 4, 136, 12}];
MAplusB = Mean[{MA, MB}];

PA = Table[OffspringViability[[i ;; i + 2]], {i, 7, 139, 12}];
PB = Table[OffspringViability[[i ;; i + 2]], {i, 10, 142, 12}];
PAplusB = Mean[{PA, PB}];

MFertRates =
  Table[(Total[#]/300) & /@ {FertilisationRates[[18 i + 1 ;; 18 i + 3]],
     FertilisationRates[[18 i + 4 ;; 18 i + 6]],
     FertilisationRates[[18 i + 7 ;; 18 i + 9]]}, {i, 0, 11}];

PFertRates =
  Table[(Total[#]/300) & /@ {FertilisationRates[[18 i + 10 ;; 18 i + 12]],
     FertilisationRates[[18 i + 13 ;; 18 i + 15]],
     FertilisationRates[[18 i + 16 ;; 18 i + 18]]}, {i, 0, 11}];
```

(* Define the geometric mean of a set of length n *)

```
GeoMean[set_, n_] := (Times @@ set)^(1/n);
```

(* Choose which dataset to work with *)

```
{Mdata, Pdata} =
  Piecewise[{{{MA, PA}, environment == 1}, {{MB, PB},
     environment == 2}, {{MAplusB, PAplusB},
     environment == 3}, {{MFertRates, PFertRates}, environment == 4}}];
```

(* Print which dataset is being used *)

```
Print["
 Experiment ", experiment]
If[environment == 4, Print["Fertilisation rates"],
 Print["Offspring viability in environment",
  Piecewise[{{" A", environment == 1}, {" B", environment == 2}, {" A+B",
```

```
    environment == 3}}]]]
```

(* Section 3: Replicating the original method *)

(* Generate tables MPermuted and PPermuted for the monandrous and polyandrous mating treatments respectively. Each table consists of SamplePerRuns subtables,which we refer to as 'samples'. Each sample is the same as the original data set except that generations are randomly permuted for each female;

Note that the same permutations are applied to each mating treatment in order to maintain the pairings of female/generation from the original experiment *)

```
PermuteTable =
  Table[RandomPermutation[NumGenerationsExp, NumFemalesExp], {sample, 1,
    SamplesPerRun}];

MPermuted =
  Table[Permute[Mdata[[female]], PermuteTable[[sample, female]]], {female, 1,
    NumFemalesExp}, {sample, 1, SamplesPerRun}];

PPermuted =
  Table[Permute[Pdata[[female]], PermuteTable[[sample, female]]], {female, 1,
    NumFemalesExp}, {sample, 1, SamplesPerRun}];
```

(* Create a vector of the differences in geometric mean fitness between mating treatments across samples *)

```
DeltaVectorOld = (GeoMean[#, NumGenerationsExp] &) /@
    Mean[PPermuted] - (GeoMean[#, NumGenerationsExp] &) /@ Mean[MPermuted];
```

(* Print mean and central range for Δ under the original method *)

```
Print["
  Mean Δ using original method of Garcia-Gonzalez et al.: ",
  N[100*Mean[DeltaVectorOld]], "
  ", Round[100*(1 - alpha), .1],
  "% central range for Δ using original method: ",
  N[100*Quantile[DeltaVectorOld, {alpha/2, 1 - alpha/2}]]];
```

(* Section 4: Bootstrapping from the original data to generate new means and central ranges for Δ using both Gillespie's measure and the difference in geometric mean fitness *)

(* Randomly generate a table of dimensions SamplesPerRun x NumFemalesSim x NumGenerationsSim. Each entry is a randomly chosen integer from {1,...,NumFemalesExp*NumGenerationsExp} *)

```
ResampleTable =
  RandomChoice[
   Array[# &, NumFemalesExp*NumGenerationsExp], {SamplesPerRun,
NumFemalesSim,
    NumGenerationsSim}];
```

(* Use ResampleTable to resample from the original datasets with replacement;

This is done by creating two new tables of dimensions SamplesPerRun x NumFemalesSim x NumGenerationsSim. The (i,j,k)th entry in each table is given by the Nth entry in the relevant (flattened) dataset, where N is the (i,j,k)th entry of ResampleTable;

Note that the same resample table is applied to both mating treatments, so that the pairings of female/generation from the experiment are maintained *)

```
MResample =
  Table[Flatten[Mdata][[ResampleTable[[sample, female, gen]]]], {sample, 1,
    SamplesPerRun}, {female, 1, NumFemalesSim}, {gen, 1, NumGenerationsSim}];

PResample =
  Table[Flatten[Pdata][[ResampleTable[[sample, female, gen]]]], {sample, 1,
    SamplesPerRun}, {female, 1, NumFemalesSim}, {gen, 1, NumGenerationsSim}];
```

(* Create a vector of Gillespie's measure across samples *)

```
DeltaVectorGillespie = ((Mean[Flatten[#]] -
       1/PopSize*Variance[Flatten[#]]) & /@
     PResample) - ((Mean[Flatten[#]] - 1/PopSize*Variance[Flatten[#]]) & /@
     MResample);
```

(* Print the mean and central range for Δ using Gillespie's measure *)

```
Print["
  Mean Δ using Gillespie's measure: ", N[100*Mean[DeltaVectorGillespie]]];

Print[Round[100*(1 - alpha), 1],
  "% central range for Δ using Gillespie's measure: ",
  N[100*Quantile[DeltaVectorGillespie, {alpha/2, 1 - alpha/2}]]];
```

(* Create a vector of the differences in geometric fitness between mating treatments across samples *)

```
DeltaVectorGeo = (GeoMean[Mean[#], NumGenerationsSim] & /@
     PResample) - (GeoMean[Mean[#], NumGenerationsSim] & /@ MResample);
```

(* Print the mean and central range for Δ using geometric mean fitness *)

```
Print["
  Mean Δ using geometric mean fitness: ", N[100*Mean[DeltaVectorGeo]]];
```

```
Print[Round[100*(1 - alpha), 1],
  "% central range for Δ using geometric mean fitness: ",
  N[100*Quantile[DeltaVectorGeo, {alpha/2, 1 - alpha/2}]]];


(* Section 5: Simulating the distribution of Δ under the null hypothesis *)

(* Create an empty vector of mean Δ-values for Gillespie's measure and for
geometric mean fitness *)

MeanDeltaVectorGillespieNull = ConstantArray[0, NumRuns];
MeanDeltaVectorGeoNull = ConstantArray[0, NumRuns];

(* Create a table of dimensions NumRuns x NumFemales x
NumGenerationsSim. Each entry is either 0 or 1 with equal probability *)

SwitchTable =
  RandomChoice[{0, 1}, {NumRuns, NumFemalesExp, NumGenerationsExp}];

(* Use SwitchTable to create two new 'mating treatment' tables. Each contains
NumRuns subtables, where each subtable is a random combination of the
original mating treatments *)

MdataTable = (#*Mdata + (1 - #) Pdata) & /@ SwitchTable;
PdataTable = ((1 - #) Mdata + #*Pdata) & /@ SwitchTable;

(* Repeat the following loop for each subtable of MdataTable and PdataTable in
order to estimate the null distribution of Δ-values *)

Do[

  (* Progress update *)

  If[Divisible[run, 100], Print[Round[100*run/NumRuns, 1], "% complete"]];

  (* Apply the same procedure as in Section 4 to one subtable of MdataTable and
PdataTable *)

  ResampleTable =
   RandomChoice[
    Array[# &, NumFemalesExp*NumGenerationsExp], {SamplesPerRun,
     NumFemalesSim, NumGenerationsSim}];

  MResample =
   Table[Flatten[MdataTable[[run]]][[
     ResampleTable[[sample, female, gen]]]], {sample, 1,
     SamplesPerRun}, {female, 1, NumFemalesSim}, {gen, 1,
     NumGenerationsSim}];
```

```
 PResample =
  Table[Flatten[PdataTable[[run]]][[
    ResampleTable[[sample, female, gen]]]], {sample, 1,
    SamplesPerRun}, {female, 1, NumFemalesSim}, {gen, 1,
    NumGenerationsSim}];
```

(* Calculate the mean of Gillespie's measure and of the difference in geometric mean fitness across all samples. Store the results in MeanDeltaVectorGillespieNull and MeanDeltaVectorGeoNull respectively *)

```
 MeanDeltaVectorGillespieNull[[run]] =
  Mean[((Mean[Flatten[#]] - 1/PopSize*Variance[Flatten[#]]) & /@
     PResample) - ((Mean[Flatten[#]] - 1/PopSize*Variance[Flatten[#]]) & /@
     MResample)];

 MeanDeltaVectorGeoNull[[run]] =
  Mean[(GeoMean[Mean[#], NumGenerationsSim] & /@
     PResample) - (GeoMean[Mean[#], NumGenerationsSim] & /@ MResample)];


 , {run, 1, NumRuns}];
```

(* Calulate approximate p-values by taking the proportion of mean Δ-values under the null hypothesis that exceed the mean Δ-value calculated in Section 4 *)

```
pvalueGillespie =
 Total[(Boole[Abs[#] > Abs[Mean[DeltaVectorGillespie]]]) & /@
    MeanDeltaVectorGillespieNull]/Length[MeanDeltaVectorGillespieNull];

pvalueGeo =
 Total[(Boole[Abs[#] > Abs[Mean[DeltaVectorGeo]]]) & /@
    MeanDeltaVectorGeoNull]/Length[MeanDeltaVectorGeoNull];
```

(* Print approximate p-value for Δ using each method *)

```
Print["
 Approximate p-value using Gillespie's measure: ", N[pvalueGillespie]]

Print["
 Approximate p-value using geometric mean fitness: ", N[pvalueGeo]]
```