```
(*This code relates to the paper:
  Henshaw JM, Fromhage L,
Jones AG. Sex roles and the evolution of parental care specialisation.

  It was written for Wolfram Mathematica version 11.2.*)

(*Set parameter values*)

(*Number of males and females contributing to each brood*)
nm = 1; nf = 1;
(*Minimum mortality rate*)
muMin = 0.01;
(*Species-specific factor affecting the rate of mate encounters*)
M = 1;
(*Sex ratio at maturation*)
r = 1;
(*Importance of care of types 1 and 2*)
alpha1 = 0.1; alpha2 = 0.1;
(*Benefit of specialisation*)
gamma = 1.0;

(*Set initial trait values for each sex*)

Tm = 0; Tf = 1.5;
qm = 0.4; qf = 0.6;
xm = 1.5; xf = 0.5;

(*Parameters used in numerical calculation of adaptive dynamic trajectories*)

(*Number of times to calculate selection gradient and
 let trait values evolve a small 'step' in its direction*)
steps = 50 000;
(*Record current trait values after multiples
 of this many steps. Must be a divisor of 'steps'*)
divisor = 100;
(*Steps are of length delta*|selection gradient|,
up to a maximum of 'maxstep'*)
delta = 0.01;
(*Maximum step size*)
maxstep = 0.01;

(*Define mortality and offspring survival functions*)

mu[x_] := muMin (1 + x^1.5);
S[tau1all_, tau2all_] := Exp[-alpha1 / tau1all] * Exp[-alpha2 / tau2all];

(*Initialise an empty array that will store the evoluionary
```

```
 trajectories and set its first element to the initial trait values*)

output = Array[{} &, steps / divisor + 1];
output[[1]] = {Tm, xm, qm, Tf, xf, qf};

(*This loop calculates the selection gradients in each sex,
then moves the population trait values a
 small step in the direction of these gradients*)

Do[

  (*Calculate population values for all variables*)

  (*Mortality rates*)
  mum = mu[xm];
  muf = mu[xf];

  (*Probability of surviving time-out*)
  pOm = Exp[-mu[xm] Tm];
  pOf = Exp[-mu[xf] Tf];

  (*Operational sex ratio rO*)
  A = M * (xm * xf) (r * nf (1 - pOf) - nm (1 - pOm));
  rO = r * muf / mum + A / (2 mum^2) (A + Sqrt[A^2 + 4 r * mum * muf]);

  (*Mating rates*)
  am = M * (xm * xf) nm * rO^(-1 / 2);
  af = M * (xm * xf) nf * rO^(1 / 2);

  (*Probability of surviving time-in*)
  pIm = am / (am + mum);
  pIf = af / (af + muf);

  (*An individual's effective contributions to care of each type*)
  tau1m = qm (1 - gamma * qm (1 - qm)) (1 - Exp[-mum * Tm]) / mum;
  tau1f = qf (1 - gamma * qf (1 - qf)) (1 - Exp[-muf * Tf]) / muf;
  tau2m = (1 - qm) (1 - gamma * qm (1 - qm)) (1 - Exp[-mum * Tm]) / mum;
  tau2f = (1 - qf) (1 - gamma * qf (1 - qf)) (1 - Exp[-muf * Tf]) / muf;

  (*Probability of offspring survival*)
  Spop = S[nm * tau1m + nf * tau1f, nm * tau2m + nf * tau2f];

  (*Fitness*)
  Wm = pIm / (1 - pOm * pIm) * nm / nf * Spop;
  Wf = pIf / (1 - pOf * pIf) * Spop;
```

```
(*Calculate mutant values for all variables*)

(*Mortality rates*)
mumMut = mu[xmMut];
mufMut = mu[xfMut];

(*Probability of surviving time-out*)
pOmMut = Exp[-mumMut * TmMut];
pOfMut = Exp[-mufMut * TfMut];

(*Mating rates*)
amMut = M * (xmMut * xf) nm * r0 ^ (-1 / 2);
afMut = M * (xfMut * xm) nf * r0 ^ (1 / 2);

(*Probability of surviving time-in*)
pImMut = amMut / (amMut + mumMut);
pIfMut = afMut / (afMut + mufMut);

(*An individual's effective contributions to care of each type*)
tau1mMut = qmMut (1 - gamma * qmMut (1 - qmMut)) (1 - Exp[-mumMut * TmMut]) / mumMut;
tau1fMut = qfMut (1 - gamma * qfMut (1 - qfMut)) (1 - Exp[-mufMut * TfMut]) / mufMut;
tau2mMut =
 (1 - qmMut) (1 - gamma * qmMut (1 - qmMut)) (1 - Exp[-mumMut * TmMut]) / mumMut;
tau2fMut = (1 - qfMut) (1 - gamma * qfMut (1 - qfMut))
   (1 - Exp[-mufMut * TfMut]) / mufMut;

(*Probability of offspring survival*)
SmMut =
 S[tau1mMut + (nm - 1) tau1m + nf * tau1f, tau2mMut + (nm - 1) tau2m + nf * tau2f];
SfMut = S[nm * tau1m + tau1fMut + (nf - 1) tau1f,
   nm * tau2m + tau2fMut + (nf - 1) tau2f];

(*Fitness*)
WmMut = pImMut / (1 - pOmMut * pImMut) * nm / nf * SmMut;
WfMut = pIfMut / (1 - pOfMut * pIfMut) * SfMut;


(*Calculate selection gradients
 for each of the three trait in males and females*)

SelGrad = {D[WmMut / Wm, TmMut] /. {TmMut → Tm, xmMut → xm, qmMut → qm},
   D[WmMut / Wm, xmMut] /. {TmMut → Tm, xmMut → xm, qmMut → qm},
   D[WmMut / Wm, qmMut] /. {TmMut → Tm, xmMut → xm, qmMut → qm},
   D[WfMut / Wf, TfMut] /. {TfMut → Tf, xfMut → xf, qfMut → qf},
   D[WfMut / Wf, xfMut] /. {TfMut → Tf, xfMut → xf, qfMut → qf},
```

```
  D[WfMut / Wf, qfMut] /. {TfMut → Tf, xfMut → xf, qfMut → qf}};

(*Set the selection gradient to zero in boundary cases to avoid impossible
 trait values (e.g. parental care duration less than zero)*)
If[SelGrad[[1]] < 0 && Tm == 0, SelGrad[[1]] = 0];
If[SelGrad[[2]] < 0 && xm == 0, SelGrad[[2]] = 0];
If[
 (SelGrad[[3]] < 0 && qm == 0) || (SelGrad[[3]] > 0 && qm == 1), SelGrad[[3]] = 0];
If[SelGrad[[4]] < 0 && Tf == 0, SelGrad[[4]] = 0];
If[SelGrad[[5]] < 0 && xf == 0, SelGrad[[5]] = 0];
If[
 (SelGrad[[6]] < 0 && qf == 0) || (SelGrad[[6]] > 0 && qf == 1), SelGrad[[6]] = 0];

(*If the selection gradient is larger than 'maxstep',
reduce it to 'maxstep' in size*)
If[delta * Norm[SelGrad] > maxstep,
 SelGrad = SelGrad / Norm[SelGrad] * maxstep / delta];

(*Round any trait values that
 exceed their bounaries back to the nearest boundary*)
{Tm, xm, qm, Tf, xf, qf} = Max[#, 0] & /@
  ({Tm, xm, qm, Tf, xf, qf} + delta * SelGrad);
{qm, qf} = Min[#, 1] & /@ {qm, qf};

(*If the step number is a multiple of 'divisor',
print the current strategy set and save it to 'output'*)

If[Divisible[step, divisor], Print[{Tm, xm, qm, Tf, xf, qf}]];

If[Divisible[step, divisor],
 output[[step / divisor + 1]] = {Tm, xm, qm, Tf, xf, qf};],

{step, 1, steps}];
```