```
SetDirectory @NotebookDirectory [];
```

Generating Function Functions

Functions to obtain the generating function via recursion. We use addition to represent coalesent states of ancestral lineages. This works because addition is associative and allows simplification of the branch indices as much as possible along the way. That means we either pass a fully-labelled set {{a},{b},{c}} and obtain the full generating function for branches of type {a+b} and {c}. If we pass {{1},{1},{1}} we obtain i-Ton labelled branch types, e.g., {2}, {1}.

The machinery for the recursion (below) will provide the generating function for a fully labelled sub-set, which we examine for a sample of n=2 lineages. However, the remaining functions are used to simplify and automate analysis of the i-Ton labelled branches and are defined for a sample of n>2 lineages.

Neutral Coalescent

Helper Functions

In[ ]:=
```
TakeAway::usage =
  "TakeAway[U,V]  gives the set U, minus elements V. Note that this is not
    the same as Complement[U,V] if U contains repeated elements. ";
```

In[ ]:=
```
TakeAway[U_List, {}] := U;
(*TakeAway[U_List,{{}}]:=If[MemberQ[U,{{}}],TakeAway[U,{{}}],U];*)
TakeAway[U_List, {v_}] :=
  Drop[U, Cases[Position[U, v], {_Integer}][[1]] /; Complement[{v}, U] == {};
TakeAway[U_List, V_List] :=
  TakeAway[TakeAway[U, Take[V, 1]], Drop[V, 1]] /; Complement[V, U] == {};
TakeAway[U_List, W__List, V_List] := TakeAway[TakeAway[U, W], V];
```

In[ ]:=
```
GetVars::usage =
  "GetVars[expr,ψ] lists the configurations of genes in all terms of the form
    ψ[ω,genes,options], GetVars[{expr,...},ψ] lists all the variables.";
```

In[ ]:=
```
GetVars[expr_, GF_] := Cases[Variables[GF], expr[_]];
```

In[ ]:=
```
(*functions to convert the {a+b+c} notation for the i-
 Ton branch recursion to the {a,b,c} notation for the fully-
 labelled topology as described in the manuscript*)
Listify[yl_List] := Block[{x = yl[[1]]},
  If[Length[x] == 0, {x}, x[[0]] = List; x]
 ];
ConvertNotation[expr_, ω_] := expr /. ω[x_] :> ω[Listify[x]];
```

Recursion

In[ ]:=
```
MakeCoalEvents[xl_List] := Module[
   {myTupleRates = Tally[Subsets[Sort[xl], {2}]],
    yl},
   yl = {#[[2]], Append[TakeAway[xl, #[[1]]], List[Total[Flatten[#[[1]]]]]} & /@ myTupleRates
  ];

GFn[ω_, xl_List] :=
  Module[{coalList = MakeCoalEvents[xl], λ = Binomial[Length[xl], 2]},
   If[Length[xl] == 1, 1,
    1/(λ + Total[ω[#] & /@ xl]) * Total[(# [[1]] * GFn[ω, #[[2]]]) & /@ coalList]
   ]
  ];
```

Starlike Approximation

Recursion

Note that we retain P[i,n] unevaluated as the probability that i out of n lineages escape the sweep. This can help for analysis as well as for simplification steps. Substituting P-> PknToAlpha will lead to evaluation as a function of alpha, the scaled distance from the sweep center: α = r/s Log[2ns], where r is the rate of recombination between the neutral and selected site.

In[ ]:=
```
Pe[k_, n_, alpha_] := Module[
   {Pesc = 1 - Exp[-alpha]},
   Binomial[n, k] * Pesc^k * (1 - Pesc)^(n - k);
   (*If[k==0,(1 - Sum[P[j,n],{j,1,n}]),
    P[k,n]]*)
```

```
    P[k, n]
  ];

PknToAlpha[k_, n_] := Module[
    {Pesc = 1 - Exp[-α]},
    Binomial[n, k] * Pesc^k (1 - Pesc)^(n - k)
  ];

MakeStarlikeEvents[xl_List, delta_, alpha_] := Module[
  {myTupleRates, n = Length[xl], numTuples,
    yl, returnList = {}
  },
  For[k = 0, k < n, k++,
    myTupleRates = Tally[Subsets[Sort[xl], {n - k}]];
    numTuples = Total[#[[2]] & /@ myTupleRates];
    yl = {delta * Pe[k, n, alpha] * #[[2]] / numTuples,
        Append[TakeAway[xl, #[[1]]], List[Total[Flatten[#[[1]]]]]} & /@ myTupleRates;
    returnList = Join[returnList, yl];
  ];
  (*for k = n separate to avoid adding empty {} lineage to branch set*)
  yl = {{delta * Pe[n, n, alpha] * 1, xl}};
  returnList = Join[returnList, yl];
  returnList = Flatten[returnList, 0];
  Return[returnList]
 ]
GFstar[alpha_, ω_, xl_List, dl_List] := Module[
   {coalList = MakeCoalEvents[xl],
    starList = MakeStarlikeEvents[xl, dl[[1]], alpha], allEvents, totalRates},
   totalRates = Total[#[[1]] & /@ coalList] + Total[#[[1]] & /@ starList];
   If[Length[xl] == 1, 1,
    1/(totalRates + Total[ω[#] & /@ xl]) * (
      Total[(#[[1]] * GFstar[alpha, ω, #[[2]], dl]) & /@ coalList] +
       Total[(#[[1]] * GFn[ω, #[[2]]]) & /@ starList]
     )
   ]
  ];
```

Simplification Functions

In[ ]:=

```
subAllPeeToOne[n_] := Table[
    Sum[P[i, k], {i, 0, k}] → 1,
```

```
    {k, 1, n}];
subAllPeeToEpsilon [n_] := Table[
    Sum[ϵ_ P[i, k], {i, 0, k}] → ϵ,
    {k, 1, n}];
subPeeToOneMinus [n_] := Block[
    {part,
     res = List[],
     allVals, currentVals
    },
    For[k = 2, k ≤ n, k++,
     allVals = Table[ii, {ii, 0, k}];
     For[x = 0, x ≤ k, x++,
      currentVals = DeleteCases[allVals, x];
      part = Sum[P[i, k], {i, currentVals}] → 1 - P[x, k];
      res = Append[res, part];
     ];
    ];
    res
  ];
subPeeToEpsilonMinus [n_] := Block[
    {part,
     res = List[],
     allVals, currentVals
    },
    For[k = 2, k ≤ n, k++,
     allVals = Table[ii, {ii, 0, k}];
     For[x = 0, x ≤ k, x++,
      currentVals = DeleteCases[allVals, x];
      part = Sum[ϵ_ P[i, k], {i, currentVals}] → ϵ (1 - P[x, k]);
      res = Append[res, part];
     ];
    ];
    res
  ];

SubSimplifyPeeRulesv2 [expr_, n_] := Block[
    {thisExp = ReleaseHold[expr],
     nextExp,
     subIntPos = i_Integer P[s__] :> HoldForm[(i - 1) P[s]] + P[s] /; i > 1,
     subIntNeg = i_Integer P[s__] :> HoldForm[(i + 1) P[s]] - P[s] /; i < -1,
     a},

    While[True,
```

```
    nextExp = thisExp //. subIntPos //. subIntNeg;
    nextExp = thisExp //. subIntPos //. subIntNeg;
    nextExp = nextExp //. subAllPeeToOne[n] //. subAllPeeToEpsilon[n] //.
        subPeeToOneMinus[n] //. subPeeToEpsilonMinus[n];
    nextExp = nextExp //. subAllPeeToOne[n] //. subAllPeeToEpsilon[n] //.
        subPeeToOneMinus[n] //. subPeeToEpsilonMinus[n];
    nextExp = ReleaseHold[nextExp];
    a = thisExp - nextExp;

    If[a == 0, Break[], Print["no"]];
    thisExp = nextExp

   ];
   thisExp = thisExp;
   (*//.subPeeToOneMinus[n]//.subPeeToEpsilonMinus[n];*)
   Return[thisExp]
  ];
```

functions to get means, pdfs, cdfs for marginals T1, T2, T3,...

In[ ]:=

```
GetGfAsList[n_] := Block[{doIfTrue, doIfFalse, sample = Table[{1}, {i, 1, n}]},
   doIfFalse[] := Block[{}, Print["n <= 2"]; Return[Return["False"]]];
   doIfTrue[] := Block[
     {gf, peeTable, subPSums, deltaPeeTable, subDeltaPeeSums},

     gf = Expand[GFstar[α, ω, sample, {δ}]];
     gf[[0]] = List;
     gf = SubSimplifyPeeRulesv2[#, n] & /@ gf // Parallelize;
     gf
    ];
   If[n > 2, doIfTrue[], doIfFalse[]]
  ];
```

In[ ]:=
```
MakeMeanList[gfDelta_List, n_, ω_, T_, δ_, Ta_] := Block[
  {
   means = {},
   xs,
   subTable,
   steps
  },
  steps[pathGF_, ii_] := Block[{pgf},
    pgf = (-1) D[pathGF, ω[{ii}]];
    pgf = pgf /. subTable;
    pgf =  InverseLaplaceTransform [pgf / δ, δ, Ta];
    pgf
   ];
  subTable = ω[{#}] → 0 & /@ Table[j, {j, 1, n}]; (*all sub to zero here*)
  For[i = 1, i < n, i++,
   xs = steps[#, i] & /@ gfDelta // Parallelize;
   means = Append[means, xs];
  ];
  means = Total[#] & /@ means;
  Return[means]
 ]
```

In[ ]:=
```
MakePDFList[gfDelta_List , n_, ω_, T_, δ_, Ta_] := Block[
  {
   marginals = {},
   xs,
   subTable ,
   steps
  },

  steps[pathGF_ , ii_, sTab_] := Block[{pgf},
    pgf = pathGF /. sTab ;
    pgf =  InverseLaplaceTransform [pgf / δ, δ, Ta];
    pgf = InverseLaplaceTransform [pgf, ω[{ii}], T]
   ];

  For[i = 1, i < n, i++,
   subTable  = DeleteCases [Table[j , {j , 1, n}], i];
   subTable  = ω[{#}] → 0 & /@ subTable ;

   xs = steps[#, i, subTable] & /@ gfDelta  // Parallelize ;

   marginals  = Append[marginals , xs];
  ];
  marginals  = Total[#] & /@ marginals ;
  Return[marginals]
 ]
```

In[ ]:=
```
MakeCDFList[gfDelta_List , n_, ω_, T_, δ_, Ta_] := Block[
  {
   marginals = {},
   xs,
   subTable ,
   steps
  },

  steps[pathGF_ , ii_, sTab_] := Block[{pgf},
    pgf = pathGF /. sTab;
    pgf =  InverseLaplaceTransform [pgf / δ, δ, Ta];
    pgf = InverseLaplaceTransform [pgf / ω[{ii}], ω[{ii}], T]

   ];

  For[i = 1, i < n, i++,
   subTable  = DeleteCases [Table[j , {j , 1, n}], i];
   subTable  = ω[{#}] → 0 & /@ subTable // Parallelize ;

   xs = steps[#, i, subTable] & /@ gfDelta  // Parallelize ;
   marginals  = Append[marginals , xs];
  ];
  marginals  = Total[#] & /@ marginals ;
  Return[marginals]
 ]
```

In[ ]:=
```
(*This takes the list of marginal pdfs
 and cdfs. For each marginal (1-ton, 2-ton etc.),
it determines the discontinuities (heavisides and diracdeltas)
 present. From these, we want to determine the size of the
 point masses.  I was previously checking every discontinuity ,
just in case. That's too expensive. And it's dumb.  DiracDelta terms
 tell us there's a pointmass. So now I'm only looking at those.

  The return is a list of
 {i-ton type, {discontinuities }, {point masses}}
     The point masses are described
 as {T-Td,val} where T_d is the discontinuitiy
 (was easy, could be modified to show T→ Td instead. If it returns {T,val},
  the pointmass is located at 0.
  *)
```

```
MakeALim[exp_Symbol , T_] := {T → 0};
MakeALim[exp_Plus , T_] := {T → T - exp};

FromTheLeft[exp_ , lim_] :=
  If[Cases[Values[lim], 0] ≠ {}, 0, Limit[exp, lim, Direction → "FromBelow"]];
FromTheRight[exp_ , lim_] := Limit[exp, lim, Direction → "FromAbove"];

GetJumps[spL_List , scL_List , T_] := Block[
  {ret, thisRes , disconts , diracDisconts , limits , leftLims , rightLims , jumpSizes},
  ret = {};
  For[i = 1, i ≤ Length[spL], i++,
   thisRes = {i};
   (*Print[i];*)
   disconts =
    DeleteDuplicates[Cases[spL[[i]], HeavisideTheta[_] | DiracDelta[_], Infinity]];
   thisRes = thisRes ~Append~ disconts ;
   diracDisconts = Cases[disconts , DiracDelta[_], Infinity];
   (*Print[diracDisconts ];*)
   If[Length[diracDisconts ] == 0, diracDisconts = {diracDisconts }, None];
   (*Print[diracDisconts ];*)
   diracDisconts = diracDisconts /. {DiracDelta[ζ_] → ζ} ;
   (*Print[diracDisconts ];*)
   diracDisconts = DeleteDuplicates[diracDisconts ];
   limits = MakeALim[#, T] & /@ diracDisconts ;
   (*Print[limits];*)
   leftLims = FromTheLeft[scL[[i]], #] & /@ limits ;
   rightLims = FromTheRight[scL[[i]], #] & /@ limits ;
   (*Print[leftLims];
   Print[rightLims];*)
   jumpSizes = MapThread[
     FullSimplify[(#1 - #2) /. HeavisideTheta[_] → 0] &, {rightLims , leftLims}];
   jumpSizes = FullSimplify[SubSimplifyPeeRulesv2[#, 4]] & /@ jumpSizes ;
   thisRes = Append[thisRes , MapThread[{#1, #2} &, {diracDisconts , jumpSizes}]];
   (*thisRes = MapThread[{{i,#1},#2}&,{diracDisconts ,jumpSizes}];*)
   (*ret = Append[ret,thisRes];*)
   ret = Append[ret, thisRes];
  ];
  ret
 ]

MakePiecewise[expression_ , t_, assums_] :=
  Module[{expr, disconts , left, right, intervals , assumptions , piecewise},
   (*disconts = DeleteDuplicates[
```

```
      Cases[expr,HeavisideTheta[_]|DiracDelta[_],Infinity]];*)
  expr = Assuming[assums, FullSimplify[expression]];
  (*Print[expr];*)
  disconts =
   DeleteDuplicates[Cases[expr, HeavisideTheta[_] | DiracDelta[_], Infinity]];
  (*Print[disconts];*)
  If[disconts == {}, Return[{{expr, 0 ≤ t < Infinity}}]];
  disconts = (t - # /. p_[x_] :> x) & /@ disconts;
  disconts = DeleteDuplicates[{0} ~Join~ disconts ~Join~ {Infinity}] // Sort;
  left = disconts[[1 ;; -2]];
  right = disconts[[2 ;; -1]];
  assumptions = MapThread[#1 ≤ t < #2 &, {left, right}];
  (*Print[assumptions];*)
  piecewise = Assuming[assums && #,
    FullSimplify[expr /. HeavisideTheta → UnitStep]] & /@ assumptions ;
  piecewise = SubSimplifyPeeRulesv2[Expand[#], 4] & /@ piecewise // FullSimplify;
  MapThread[{#2, assums && #1} &, {assumptions, piecewise}]
 ];
```

Instant Yule Approximation

Partition probabilities

In[ ]:=

```
PFi[n_, i_] := ∏_{j=1}^{n-1} (i - j)/(i + j) (*i number of lineages in big yule tree,
n lineages in sample*)
pnotrec[f_, r_, s_, Ne_, M_] := e^(-(r/s * Σ_{i=f}^{M} 1/i)) (*probability of no recombination *)
m[s_, r_, n_] := Max[1, r/s * Σ_{i=1}^{n-1} 1/i] (*correction factor*)


(*E=size of early recombinant family,
L = number of late recombining singletons, n=sample size,
n-E-L=number of lineages that did not escape*)
Clear[PELN];
PELN[E_, L_, n_, s_, r_, Ne_, M_] := PELN[E, L, n, s, r, Ne, M] = Module[
    {pfisf, pnorec, prec, ExpectPf, correctionF = m[s, r, n]},
    (*probability P(F=f), subtract P(F≤i+1)-P(F≤i) = PFi[i+1]-PFi[i]*)
    pfisf = Differences[PFi[n, #] & /@ Range[n - 1, M]];
```

```
pnorec = pnotrec[#, r, s, Ne, M] & /@ Range[n, M];
(*pnorec=p[#,r,s,Ne]&/@Range[n,M];*)
prec = 1 - pnorec;
ExpectPf = Total[pnorec^(n-L) * prec^L * pfisf];
    If[E ≥ 2, ExpectPf * (r*n)/(s*correctionF) * 1/(E (E-1))
  ((n-1) * Binomial[n-2, E-2] * Boole[L+E == n] + Binomial[n-1, L]),
 If[E == 1, ExpectPf * (r*n)/(s*correctionF) *
   (Boole[L+1 == n] + Binomial[n-1, L] * Sum[1/i, {i, 2, n-1}] + Sum[Binomial[n-S, L-S+1]/(S-1), {S, 2, n}]),
  If[correctionF > 1, ExpectPf * (Max[0, Binomial[n, L](1-correctionF)] +
      (r*n)/(s*correctionF) (1/n * Boole[L == n] + Binomial[n-1, L-1] * Sum[1/i, {i, 2, n-1}] +
        Sum[Binomial[n-S, n-L] * 1/(S (S-1)), {S, 2, n}])),
   ExpectPf * (Binomial[n, L] (1 - (r*n)/s (Sum[1/i, {i, 1, n-1}] - L/n Sum[1/i, {i, 2, n-1}])) +
      (r*n)/s (1/n * Boole[L == n] + Sum[Binomial[n-S, n-L] * 1/(S (S-1)), {S, 2, n}]))]]]
];
```

Helper Functions For GF recrusion for yule approximation

In[ ]:=
```
(*gives the number of ways to divide n into 3 buckets, including empty bucket*)
partitions[X_] := Catenate[Permutations /@ IntegerPartitions[X, {3}, Range[0, X]]];
(*given the types of partitions, make all combinations for our sample,
and calculate probability of this configuration. returns
 (partition, probability, sample_configuration following coalescence). Hold
 is used to avoid having to specify all parameters
 prior to taking inverse laplace transforms.*)
part3[a_List, p_List, s_, r_, Ne_, M_] :=
  Module[{f, sym, prob, result}, Attributes[f] = Orderless;
   (*prob=PELN[p[[1]],p[[2]],Total[p],s,r, Ne,M];*)
   (*prob=Peln[p[[1]],p[[2]],Total[p]];*)
   prob = Peln[p[[1]], p[[2]], Total[p], s, r, Ne, M];
   sym = Unique["x", Temporary] & /@ p;
   result = ReplaceList[f @@ a,
     f @@ MapThread[Pattern[#, Repeated[_, {#2}]] &, {sym, p}] → {p, prob, List /@ sym}];
   Return[result]
  ];

FlatList[L_List] := {Flatten[L[[1]]], L[[2]], Flatten[L[[3]]]}
(*FlatList collapses {E,L,no_escape}
 configuration into lineages remaining to coalesce*)
FlatListImpr[L_List] :=
 Sort[DeleteCases[Join[{Flatten[L[[1]]]}, L[[2]], {Flatten[L[[3]]]}], {}]]

sumelms[a_List] := Module[{E, nonrec},
   E = If[Length[a[[3, 1]]] == 0, {}, Total[a[[3, 1]]]];
   nonrec = If[Length[a[[3, 3]]] == 0, {}, Total[a[[3, 3]]]];
   {a[[1]], a[[2]], List[E, Sort[a[[3, 2]]], nonrec]};
```

Recursion

In[ ]:=

```
MakeYulelikeEvents[xl_List, delta_, s_, r_, Ne_, M_] := Module[
  {allFamilies, Counts, CountsAssociation, countFamilies, n = Length[xl]
  },
  allFamilies = Flatten[part3[xl, #, s, r, Ne, M] & /@ partitions[n], 1];
  allFamilies = sumelms /@ allFamilies;
  Counts = Tally[allFamilies[[All, 1]]];
  CountsAssociation = AssociationThread[Counts[[All, 1]], Counts[[All, 2]]];
  allFamilies =
   List[delta * #[[2]] / CountsAssociation[#[[1]]], FlatListImpr[#[[3]]]] & /@ allFamilies;
  countFamilies = Tally[allFamilies];
  allFamilies = List[#[[1, 1]] * #[[2]], #[[1, 2]]] & /@ countFamilies;
  Return[allFamilies]
 ];

GFYule[s_, r_, Ne_, M_, ω_, xl_List, dl_List] := Module[
  {coalList = MakeCoalEvents[xl],
   YuleList = MakeYulelikeEvents[xl, dl[[1]], s, r, Ne, M], allEvents, totalRates},
  totalRates = Total[#[[1]] & /@ coalList] + Total[#[[1]] & /@ YuleList];
  If[Length[xl] == 1, 1,
   1 / (totalRates + Total[ω[#] & /@ xl]) * (
     Total[(#[[1]] * GFYule[s, r, Ne, M, ω, #[[2]], dl]) & /@ coalList] +
      Total[(#[[1]] * GFn[ω, #[[2]]]) & /@ YuleList]
    )
  ]
 ];
```

Marginal PDFs and CDFs

In[ ]:=
```
MakeMargeList[gfDelta_List , sample_ , ω_ , δ_] := Module[
  {n = Length[sample],
   marginals = {},
   xs,
   subTable
  },
  For[i = 1, i < n, i++,
   subTable  = DeleteCases[Table[j , {j , 1, n}], i];
   subTable  = ω[{#}] → 0 & /@ subTable // Parallelize ;
   xs = # /. subTable  & /@ gfDelta // Parallelize ;
   xs = InverseLaplaceTransform [1 / δ #, δ, Ta] & /@ xs // Parallelize ;
   xs = InverseLaplaceTransform [#, ω[{i}], t] & /@ xs // Parallelize ;
   marginals  = Append[marginals , xs];
  ];
  Return[marginals]
 ]
```

In[ ]:=
```
MakeCumuList[gfDelta_List , sample_ , ω_ , δ_] := Module[
  {n = Length[sample],
   marginals = {},
   xs,
   subTable
  },
  For[i = 1, i < n, i++,
   subTable  = DeleteCases[Table[j , {j , 1, n}], i];
   subTable  = ω[{#}] → 0 & /@ subTable // Parallelize ;
   xs = # /. subTable  & /@ gfDelta // Parallelize ;
   xs = InverseLaplaceTransform [1 / δ #, δ, Ta] & /@ xs // Parallelize ;
   xs = InverseLaplaceTransform [# / ω[{i}], ω[{i}], t] & /@ xs // Parallelize ;
   marginals  = Append[marginals , xs];
  ];
  Return[marginals]
 ]
```

```
(*these substitution tables are helpful for making the results
 more redable when keeping the P[i,k] unevaluated as mathematica
 doesn't know yet that they sum to one*)(* summing over all P[i,k]
 appears in results and here we substitute that this is equal to 1*)
(* similarly Sum[ δ P[i,k] ] appears in results and
 here we substitute that this is equal to δ *)
```

Functions for Data

In[ ]:=
```
MakeDataArray[Ts_, dataFile_, shape_, datNe_] := Module[
  {
   data = BinaryReadList[dataFile, "Real32"], reshape, reshapeTranspose, margTable
  },
  reshape = ArrayReshape[data, shape];
  reshapeTranspose = Transpose[reshape];
  margTable = Table[
    {(j - 1),
     Table[#[[i]] / (2 * datNe) & /@ reshapeTranspose[[j]], {i, 1, 3, 1}]}
    , {j, 1, Length[reshapeTranspose]}]
 ]
```

Results

n=2

full label

In[ ]:=
```
sample = {{a}, {b}}
```

Out[ ]=
```
{{a}, {b}}
```

In[ ]:=
```
neuGF = ConvertNotation[GFn[ω, sample], ω]
(*note that convert notation is not actually needed in the n=2 case*)
```

Out[ ]=
$$\frac{1}{1 + \omega[\{a\}] + \omega[\{b\}]}$$

```
In[ ]:= gfSelDelta = GFstar[α, ω, sample, {δ}] // Expand
        (*expanded out, each term of sum is a unique topology*)
```

$$\text{Out[ ]=}\ \frac{1}{1+\delta\,P[0,\,2]+\delta\,P[1,\,2]+\delta\,P[2,\,2]+\omega[\{a\}]+\omega[\{b\}]}+\frac{\delta\,P[0,\,2]}{1+\delta\,P[0,\,2]+\delta\,P[1,\,2]+\delta\,P[2,\,2]+\omega[\{a\}]+\omega[\{b\}]}+\frac{\delta\,P[1,\,2]}{(1+\omega[\{a\}]+\omega[\{b\}])\,(1+\delta\,P[0,\,2]+\delta\,P[1,\,2]+\delta\,P[2,\,2]+\omega[\{a\}]+\omega[\{b\}])}+\frac{\delta\,P[2,\,2]}{(1+\omega[\{a\}]+\omega[\{b\}])\,(1+\delta\,P[0,\,2]+\delta\,P[1,\,2]+\delta\,P[2,\,2]+\omega[\{a\}]+\omega[\{b\}])}$$

```
In[ ]:= gfSelDelta = gfSelDelta //. subAllPeeToEpsilon[2];
        gfSelDeltaList = gfSelDelta;
        gfSelDeltaList[[0]] = List;
        gfSelDeltaList
        (*Simplification by summing P[i,k] terms. obtain the gf list-wise by topology.*)
```

$$\text{Out[ ]=}\ \left\{\frac{1}{1+\delta+\omega[\{a\}]+\omega[\{b\}]},\ \frac{\delta\,P[0,\,2]}{1+\delta+\omega[\{a\}]+\omega[\{b\}]},\ \frac{\delta\,P[1,\,2]}{(1+\omega[\{a\}]+\omega[\{b\}])\,(1+\delta+\omega[\{a\}]+\omega[\{b\}])},\ \frac{\delta\,P[2,\,2]}{(1+\omega[\{a\}]+\omega[\{b\}])\,(1+\delta+\omega[\{a\}]+\omega[\{b\}])}\right\}$$

```
In[ ]:= gfSelTaList = InverseLaplaceTransform[#/δ, δ, Ta] &/@ gfSelDeltaList;
        (*take inverse laplace transform separately for each toplogy*)
        gfSelTaList
```

$$\text{Out[ ]=}\ \left\{\frac{e^{-Ta\,(1+\omega[\{a\}]+\omega[\{b\}])}\left(-1+e^{Ta\,(1+\omega[\{a\}]+\omega[\{b\}])}\right)}{1+\omega[\{a\}]+\omega[\{b\}]},\ e^{-Ta\,(1+\omega[\{a\}]+\omega[\{b\}])}\,P[0,\,2],\ \frac{e^{-Ta\,(1+\omega[\{a\}]+\omega[\{b\}])}\,P[1,\,2]}{1+\omega[\{a\}]+\omega[\{b\}]},\ \frac{e^{-Ta\,(1+\omega[\{a\}]+\omega[\{b\}])}\,P[2,\,2]}{1+\omega[\{a\}]+\omega[\{b\}]}\right\}$$

```
In[ ]:= (*path probabilities*)
        gfSelTaList /. ω[_] -> 0 // Simplify
```

$$\text{Out[ ]=}\ \left\{1-e^{-Ta},\ e^{-Ta}\,P[0,\,2],\ e^{-Ta}\,P[1,\,2],\ e^{-Ta}\,P[2,\,2]\right\}$$

time to the most recent common ancestor.

Substitution to get i-ton labeled genelogy, here resulting in the singleton branch lengths. We also halve ω to get Tmrca, rather than the singleton branch length, in order to have a dual measure for genetic diversity.

```
In[ ]:= tmrcaNeuGF = neuGF //. {ω[{a}] → ω / 2, ω[{b}] → ω / 2}
        tmrcaSelGFDelta = GFstar[α, ω, sample, {δ}] /. {ω[{a}] → ω / 2, ω[{b}] → ω / 2}
        tmrcaSelGFTa = InverseLaplaceTransform [tmrcaSelGFDelta / δ, δ, Ta] //. subAllPeeToOne [2]
```

$$\text{Out[ ]=}\ \frac{1}{1+\omega}$$

$$\text{Out[ ]=}\ \frac{1+\delta\, P[0,\ 2]+\frac{\delta\, P[1,2]}{1+\omega}+\frac{\delta\, P[2,2]}{1+\omega}}{1+\omega+\delta\, P[0,\ 2]+\delta\, P[1,\ 2]+\delta\, P[2,\ 2]}$$

$$\text{Out[ ]=}\ \frac{1+e^{Ta\,(-1-\omega)}\,\omega\, P[0,\ 2]}{1+\omega}$$

expected time to most recent common ancestor

```
In[ ]:= neuMeanTmrca = Limit[(-1) D[tmrcaNeuGF , ω], {ω → 0}]
```

Out[ ]= 1

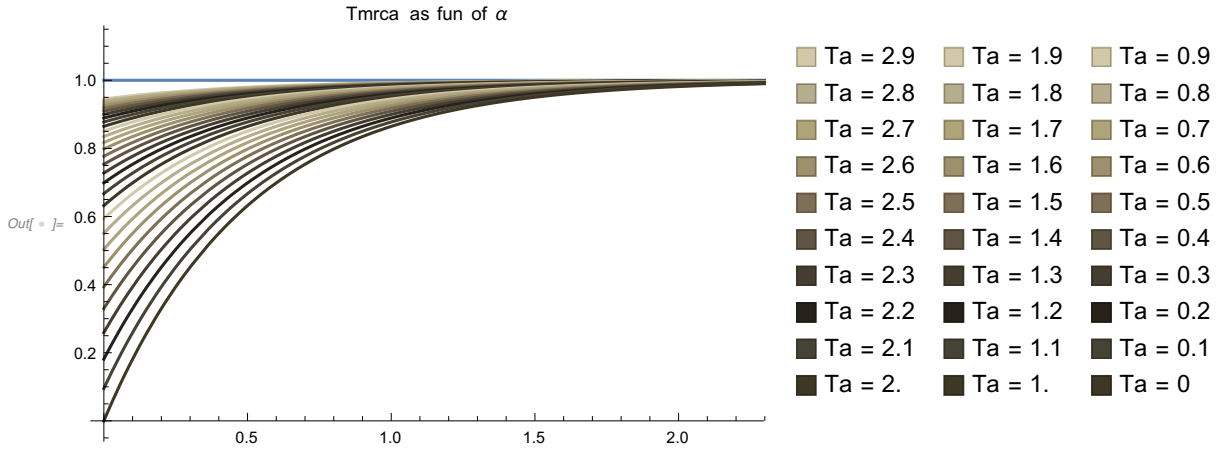
```
In[ ]:= selMeanTmrca = Limit[(-1) D[tmrcaSelGFTa , ω], {ω → 0}]
        selMeanTmrca = selMeanTmrca /. P → PknToAlpha
```

Out[ ]= $1-e^{-Ta}\, P[0,\ 2]$

Out[ ]= $1-e^{-Ta-2\,\alpha}$

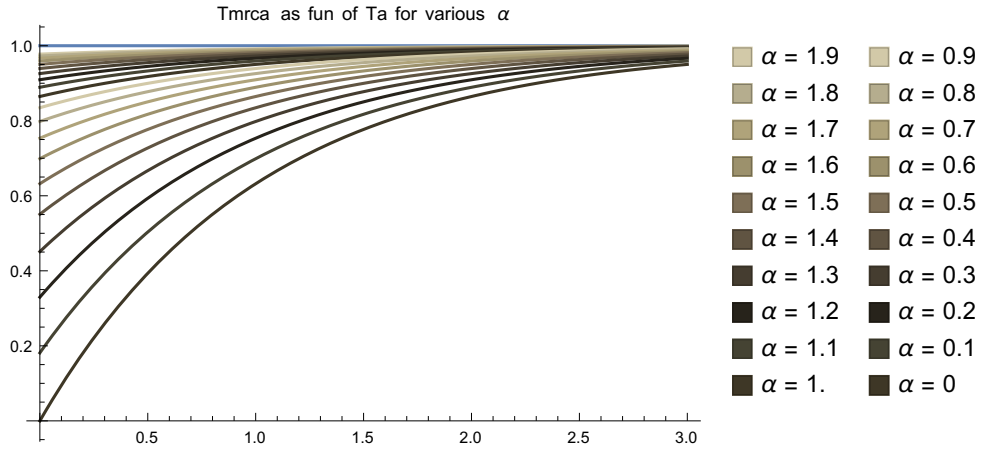
In[ ]:=
```
Module[{
  T1List = {0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5,
    1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9},
  PlotList,
  LegendItems
 },
 PlotList = selMeanTmrca /. {α → α, Ta → #} & /@ T1List // Reverse;
 LegendItems = ToString /@ T1List // Reverse;
 LegendItems = "Ta = " <> # & /@ LegendItems;
 p01 = Show[
   Plot[neuMeanTmrca, {α, 0, 6}],
   Plot[PlotList, {α, 0, 6}, PlotRange → All, PlotStyle → ColorData[49],
    PlotLegends → SwatchLegend[LegendItems]
   ],
   PlotLabel → "Tmrca as fun of α",
   PlotRange → {{0, 2.25}, {0, 1.1}}
  ];
 p01
]
```

Out[ ]=



In[ ]:=
```
Module[{
  alphaList = {0, .1, .2, .3, .4, .5, .6, .7,
    .8, .9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9},
  PlotList,
  LegendItems
 },
 PlotList = selMeanTmrca /. {α → #, Ta → Ta} & /@ alphaList // Reverse;
 LegendItems = ToString /@ alphaList // Reverse;
 LegendItems = "α = " <> # & /@ LegendItems;
 p02 = Show[
   Plot[neuMeanTmrca, {Ta, 0, 3}],
   Plot[PlotList, {Ta, 0, 3}, PlotRange → All, PlotStyle → ColorData[49],
    PlotLegends → SwatchLegend[LegendItems]
   ],
   PlotLabel → "Tmrca as fun of Ta for various α",
   PlotRange → All
  ];
 p02
]
```

Out[ ]=



distribution of time to most recent common ancestor

In[ ]:=
```
neuPDFTmrca = InverseLaplaceTransform[tmrcaNeuGF, ω, t]
```

Out[ ]= e^{-t}

In[ ]:=
```
neuCDFTmrca = InverseLaplaceTransform[tmrcaNeuGF / ω, ω, t]
```

Out[ ]= $1 - e^{-t}$

```
In[ ]:= selPDFTmrca = InverseLaplaceTransform[tmrcaSelGFTa, ω, t]
```

```
Out[ ]= e^-t + e^-Ta (-e^-t+Ta + DiracDelta[t - Ta]) HeavisideTheta[t - Ta] P[0, 2]
```

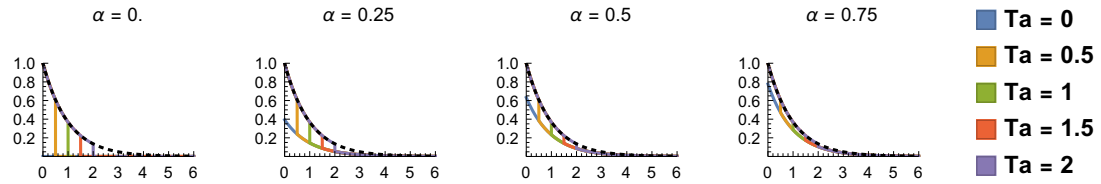
```
In[ ]:= selCDFTmrca = InverseLaplaceTransform[tmrcaSelGFTa / ω, ω, t]
```

```
Out[ ]= e^-t (-1 + e^t + HeavisideTheta[t - Ta] P[0, 2])
```

```
In[ ]:= selPDFTmrca = selPDFTmrca /. P → PknToAlpha ;
(*substituting to get as function of α for plotting*)
selCDFTmrca = selCDFTmrca /. P → PknToAlpha ;
```

PDF

```
In[ ]:=
selPDFTmrca
With[{alphaList = Table[0.25 * i, {i, 0, 3}], TaList = {0, .5, 1, 1.5, 2} // Reverse},
  LegendItems = ToString /@ TaList // Reverse ;
  LegendItems = "Ta = " <> # & /@ LegendItems ;
  plt[a_] := Module[{
      PlotList
    },
    PlotList = selPDFTmrca /. {α → a, Ta → #} & /@ TaList // Reverse ;
    Show[
     Plot[PlotList, {t, 0, 6}, PlotRange → {{0, 6}, {0, 1}}, Exclusions → None,
      (*PlotLegends →SwatchLegend[LegendItems],*)PlotLabel → "α = " <> ToString[a]
     ],
     Plot[neuPDFTmrca /. {t → T}, {t, 0, 6}, PlotStyle → {Black, Dotted}],
     PlotRange → All
    ]
   ];
  rowOfPlots = plt /@ alphaList ;
  p1 = Legended[
    GraphicsRow[rowOfPlots, ImageSize → 800, Spacings → 0],
    SwatchLegend[ColorData[97, "ColorList"], LegendItems]
   ];
 ];
p1
```

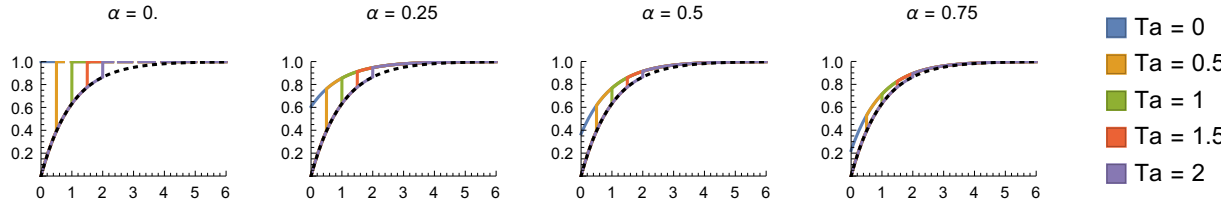


Out[ ]= $e^{-t} + e^{-Ta-2\alpha}\left(-e^{-t+Ta} + \text{DiracDelta}[t - Ta]\right)\text{HeavisideTheta}[t - Ta]$

CDF

In[ ]:=
```
selCDFTmrca
With[{alphaList = Table[0.25 * i, {i, 0, 3}], TaList = {0, .5, 1, 1.5, 2} // Reverse},
  LegendItems = ToString /@ TaList // Reverse;
  LegendItems = "Ta = " <> # & /@ LegendItems;
  plt[a_] := Module[{
     PlotList
    },
    PlotList = selCDFTmrca /. {α → a, Ta → #} & /@ TaList // Reverse;
    Show[
     Plot[PlotList, {t, 0, 6}, PlotRange → {{0, 6}, {0, 1}}, Exclusions → None,
      PlotLabel → "α = " <> ToString[a]
     ],
     Plot[neuCDFTmrca /. {t → T}, {t, 0, 6}, PlotStyle → {Black, Dotted}],
     PlotRange → All,
     Exclusions -> None
    ]
   ];
  rowOfPlots = plt /@ alphaList;
  p1 = Legended[
    GraphicsRow[rowOfPlots, ImageSize → 800, Spacings → 0],
    SwatchLegend[ColorData[97, "ColorList"], LegendItems]
   ];
 ];
p1
```

Out[ ]= $e^{-t}\left(-1 + e^{t} + e^{-2\alpha}\,\text{HeavisideTheta}[t - Ta]\right)$

Out[ ]=


Determining size of the pointmass at t = Ta:

```
In[ ]:= lower = Limit[selCDFTmrca , {t → Ta}, Direction → "FromBelow "]
        upper = Limit[selCDFTmrca , {t → Ta}, Direction -> "FromAbove "]
        pointMassSize = upper - lower // FullSimplify
```

Out[ ]= $1 - e^{-Ta}$

Out[ ]= $e^{-Ta} \left(-1 + e^{Ta} + P[0, 2]\right)$

Out[ ]= $e^{-Ta} P[0, 2]$

n = 3 iton labeled section

Note that the MakeMeanList-type functions now require two dummy variables, which will be needed for the adaptive introgression section

```
In[ ]:= n = 3;
```

neutral and starlike GF

```
In[ ]:= neugfList = GFn[ω, Table[{1}, {i, 1, n}]] // Expand ; neugfList[[0]] = List;
```

```
In[ ]:= nmList = MakeMeanList[neugfList , n, ω, t, δ, Ta];
        neuMeans = Function[{α, Ta}, Evaluate[# /. P → PknToAlpha ]] & /@ nmList ;
        npList = MakePDFList[neugfList , n, ω, t, δ, Ta];
        neuPdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ npList ;
        ncList = MakeCDFList[neugfList , n, ω, t, δ, Ta];
        neuCdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ ncList ;
```

```
In[ ]:= nmList
        npList // FullSimplify
        ncList // FullSimplify
```

Out[ ]= {2, 1}

Out[ ]= $\left\{6 e^{-t} - \frac{3}{4} e^{-3t/2} (8 + 3t),\ \frac{1}{5} e^{-3t} \left(9 + e^{5t/2}\right)\right\}$

Out[ ]= $\left\{1 - 6 e^{-t} + e^{-3t/2} \left(5 + \frac{3t}{2}\right),\ 1 - \frac{1}{5} e^{-3t} \left(3 + 2 e^{5t/2}\right)\right\}$

```
In[ ]:= selgfDeltaList = GetGfAsList[n];
        selgfTsList = InverseLaplaceTransform [1 / δ #, δ, Ts] & /@ selgfDeltaList ;
```

```
In[ • ]:= smList = MakeMeanList[selgfDeltaList , n, ω, t, δ, Ta];
          selMeans = Function[{α, Ta}, Evaluate[# /. P → PknToAlpha ]] & /@ smList;
          spList = MakePDFList[selgfDeltaList , n, ω, t, δ, Ta];
          selPdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ spList;
          scList = MakeCDFList[selgfDeltaList , n, ω, t, δ, Ta];
          selCdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ scList;
```

PDFs for neutral case piecewise, first is singletons, second is doubletons, third is tripletons.

```
In[ • ]:= npListPieceWise = {MakePiecewise[#, t, Ta == 0], MakePiecewise[#, t, Ta > 0]} & /@ npList;
          Print @ Piecewise[Flatten[#, 1]] & /@ npListPieceWise
```

$$\begin{cases} 6\,e^{-t} - \frac{3}{4}\,e^{-3\,t/2}\,(8 + 3\,t) & 0 \le t < \infty \\ 0 & \text{True} \end{cases}$$

$$\begin{cases} \frac{1}{5}\,e^{-3\,t}\left(9 + e^{5\,t/2}\right) & 0 \le t < \infty \\ 0 & \text{True} \end{cases}$$

```
Out[ • ]= {Null, Null}
```

PDFs for selective case, ordered by 1-Ton, 2-Ton, 3-Ton. In each, separate the case where Ta=0 from Ta>0.

```
In[ • ]:= spListPieceWise = {MakePiecewise[#, t, Ta == 0], MakePiecewise[#, t, Ta > 0]} & /@ spList;
          Print @ Piecewise[Flatten[#, 1]] & /@ spListPieceWise // FullSimplify
```

$$\begin{cases} \text{DiracDelta}[t]\,P[0, 3] + e^{-t}\,(P[1, 3] + t\,(P[2, 3] + P[3, 3])) \\ e^{-t}\,t \\ \frac{1}{2}\,e^{-t}\,(2\,t + 3\,(1 - t + Ta)\,P[0, 2]) \\ e^{-t}\,\text{DiracDelta}[t - 3\,Ta]\,P[0, 3] + e^{-t}\,(P[1, 3] + 3\,Ta\,(P[1, 2] + P[2, 2] - P[2, 3] - P[3, 3]) + t\,(P[2, 3] + P[3, 3])) \\ 0 \end{cases}$$

$$\begin{cases} e^{-t}\,(1 - P[0, 3]) + \text{DiracDelta}[t]\,P[0, 3] & Ta == 0 \;\&\&\; 0 \le t < \infty \\ e^{-t-3\,Ta}\left(e^{3\,Ta} + P[0, 2] + 2\,e^{3\,t}\,P[0, 2] - P[0, 3]\right) + e^{-3\,Ta}\,\text{DiracDelta}[t]\,P[0, 3] & Ta > 0 \;\&\&\; 0 \le t < Ta \\ e^{-t-3\,Ta}\left(-e^{3\,Ta}\,(-1 + P[0, 2]) + P[0, 2] - P[0, 3]\right) & Ta > 0 \;\&\&\; Ta \le t < \infty \\ 0 & \text{True} \end{cases}$$

```
Out[ • ]= {Null, Null}
```

```
In[ ]:= GetJumps[npList, ncList, t][[3]] (*note that when there are no discontinuitites,
        it prints nonsense back. here, only the 3-Ton branches have a discontinuitiy *)
```

$$\left\{\left\{1,\ \{\},\ \left\{\left\{\{\},\ \text{Limit}\left[1-6\,e^{-t}+e^{-3\,t/2}\left(5+\frac{3\,t}{2}\right),\ \text{MakeALim}[\{\},\ t],\ \text{Direction}\to\text{FromAbove}\right]-\right.\right.\right.\right.$$
$$\left.\left.\left.\left.\text{Limit}\left[1-6\,e^{-t}+e^{-3\,t/2}\left(5+\frac{3\,t}{2}\right),\ \text{MakeALim}[\{\},\ t],\ \text{Direction}\to\text{FromBelow}\right]\right\}\right\}\right\},$$
$$\left\{2,\ \{\},\ \left\{\left\{\{\},\ \text{Limit}\left[1-\frac{1}{5}\,e^{-3\,t}\left(3+2\,e^{5\,t/2}\right),\ \text{MakeALim}[\{\},\ t],\ \text{Direction}\to\text{FromAbove}\right]-\right.\right.\right.\right.$$
$$\left.\left.\left.\left.\text{Limit}\left[1-\frac{1}{5}\,e^{-3\,t}\left(3+2\,e^{5\,t/2}\right),\ \text{MakeALim}[\{\},\ t],\ \text{Direction}\to\text{FromBelow}\right]\right\}\right\}\right\}[[3]]$$

```
In[ ]:= Print[#] & /@ GetJumps[spList, scList, t]
```

```
{1, {HeavisideTheta [t - 3 Ta], HeavisideTheta [t - Ta], DiracDelta [t - 3 Ta]}, {{t - 3 Ta, e^(-3 Ta) P[0, 3]}}}
{2, {HeavisideTheta [t - Ta], DiracDelta [t]}, {{t, e^(-3 Ta) P[0, 3]}}}
```

```
Out[ ]= {Null, Null}
```

n = 4 iton labeled section

Note that the MakeMeanList-type functions now require two dummy variables, which will be needed for the adaptive introgression section

```
In[ ]:= n = 4;
```

neutral and starlike GF

```
In[ ]:= neugfList = GFn[ω, Table[{1}, {i, 1, n}]] // Expand; neugfList[[0]] = List;
```

```
In[ ]:= nmList = MakeMeanList[neugfList, n, ω, t, δ, Ta];
        neuMeans = Function[{α, Ta}, Evaluate[# /. P → PknToAlpha]] & /@ nmList;
        npList = MakePDFList[neugfList, n, ω, t, δ, Ta];
        neuPdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha]] & /@ npList;
        ncList = MakeCDFList[neugfList, n, ω, t, δ, Ta];
        neuCdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha]] & /@ ncList;
```

```
In[ ]:= nmList
        npList // FullSimplify
        ncList // FullSimplify
```

$$\text{Out[ ]= } \left\{2,\ 1,\ \frac{2}{3}\right\}$$

$$\text{Out[ ]= } \left\{6\,e^{-t}-\frac{3}{4}\,e^{-3\,t/2}\,(8+3\,t),\ \frac{1}{5}\,e^{-3\,t}\left(9+e^{5\,t/2}\right),\ \frac{2\,e^{-t}}{3}+\frac{\text{DiracDelta}[t]}{3}\right\}$$

$$\text{Out[ ]= } \left\{1-6\,e^{-t}+e^{-3\,t/2}\left(5+\frac{3\,t}{2}\right),\ 1-\frac{1}{5}\,e^{-3\,t}\left(3+2\,e^{5\,t/2}\right),\ 1-\frac{2\,e^{-t}}{3}\right\}$$

```
In[ ]:= selgfDeltaList = GetGfAsList[n];
        selgfTsList = InverseLaplaceTransform[1 / δ #, δ, Ts] & /@ selgfDeltaList ;
```

```
In[ ]:= smList = MakeMeanList[selgfDeltaList , n, ω, t, δ, Ta];
        selMeans = Function[{α, Ta}, Evaluate[# /. P → PknToAlpha ]] & /@ smList ;
        spList = MakePDFList[selgfDeltaList , n, ω, t, δ, Ta];
        selPdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ spList ;
        scList = MakeCDFList[selgfDeltaList , n, ω, t, δ, Ta];
        selCdfs = Function[{α, Ta, t}, Evaluate[# /. P → PknToAlpha ]] & /@ scList ;
```

PDFs for neutral case piecewise, first is singletons, second is doubletons, third is tripletons.

```
In[ ]:= npListPieceWise = {MakePiecewise[#, t, Ta == 0], MakePiecewise[#, t, Ta > 0]} & /@ npList ;
        Print @ Piecewise[Flatten[#, 1]] & /@ npListPieceWise
```

$$\begin{cases} 6\,e^{-t}-\frac{3}{4}\,e^{-3\,t/2}\,(8+3\,t) & 0\le t<\infty \\ 0 & \text{True} \end{cases}$$

$$\begin{cases} \frac{1}{5}\,e^{-3\,t}\left(9+e^{5\,t/2}\right) & 0\le t<\infty \\ 0 & \text{True} \end{cases}$$

$$\begin{cases} \frac{2\,e^{-t}}{3}+\frac{\text{DiracDelta}[t]}{3} & (\text{Ta}==0\ \&\&\ 0\le t<\infty)\ ||\ (\text{Ta}>0\ \&\&\ 0\le t<\infty) \\ 0 & \text{True} \end{cases}$$

```
Out[ ]= {Null, Null, Null}
```

PDFs for selective case, ordered by 1-Ton, 2-Ton, 3-Ton. In each, separate the case where Ta=0 from Ta>0.

```
In[ ]:= spListPieceWise = {MakePiecewise[#, t, Ta == 0], MakePiecewise[#, t, Ta > 0]} & /@ spList ;
        Print @ Piecewise[Flatten[#, 1]] & /@ spListPieceWise
```

```
Out[ ]= $Aborted
```

```
Out[ ]= spListPieceWise
```

In[ ]:=
```
GetJumps[npList, ncList, t][[3]](*note that when there are no discontinuitites ,
it prints nonsense back. here, only the 3-Ton branches have a discontinuitiy *)
```

Values : The argument MakeALim [{}, t] is not a valid Association or a list of rules .

Limit : Limit specification MakeALim [{}, t] is not of the form x –> x0 .

Limit : Limit specification MakeALim [{}, t] is not of the form x –> x0 .

Values : The argument MakeALim [{}, t] is not a valid Association or a list of rules .

Limit : Limit specification MakeALim [{}, t] is not of the form x –> x0 .

General : Further output of Limit ::lim will be suppressed during this calculation .

Out[ ]= $\left\{3, \{\text{DiracDelta}[t]\}, \left\{\left\{t, \frac{1}{3}\right\}\right\}\right\}$

Here we obtain the location of the discontinuities and point masses needed in order to re-define the marginal branch length distribution functions as piece-wise continuous functions. Below we relate these to their context in the selective sweep model and to the main-text results about the effect of the sweep on the shape of genealogies in the surrounding genome.

In[ ]:=
```
Print[#] & /@ GetJumps[spList, scList, t]
```

$\{1, \{\text{HeavisideTheta}[t - 4\,Ta], \text{HeavisideTheta}[t - 2\,Ta], \text{HeavisideTheta}[t - Ta], \text{DiracDelta}[t - 4\,Ta]\}, \{\{t - 4\,Ta, e^{-6\,Ta}\,P[0, 4]\}\}\}$

$\{2, \{\text{HeavisideTheta}[t - Ta], \text{HeavisideTheta}[t - 2\,Ta], \text{DiracDelta}[t]\}, \{\{t, e^{-6\,Ta}\,(P[0, 4] + P[1, 4])\}\}\}$

$\left\{3, \{\text{DiracDelta}[t], \text{HeavisideTheta}[t - Ta]\}, \left\{\left\{t, \frac{1}{3}e^{-6\,Ta}\,(e^{6\,Ta} - 4\,P[0, 3] + 4\,e^{3\,Ta}\,P[0, 3] + 2\,P[0, 4] - P[1, 4])\right\}\right\}\right\}$

Out[ ]= {Null, Null, Null}

There are three discontinuities in the PDF of singleton branches: removable discontinuities at $t_1 = T_a$ and $t_1=2 T_a$ and a jump discontinuity at $t_1 = 4 T_a$ with a point mass of size $e^{-6 T_a} P_{0,4}$, corresponding to the case where the first event is the simultaneous coalescence of all lineages during the sweep. The PDF of t_2 contains two removable discontinuities, one at $t_2= T_a$ and one at $t_2 = 2 T_a$ and one point mass at $t_2 = 0$ of size $e^{-6 T_a}(P_{0,4} + P_{1,4})$. This point-mass is again caused the by multiple-merger coalescence events that give rise to a topology devoid of doubleton branches. For t_3, we observed a removable discontinuity at $t_3 = T_a$ and, as for the neutral case, a point-mass when $t_3 = 0$. The size of this point mass, given below, equals $P_{sym} + P_{star}$ (see above), and so is no longer a simple function of the $P_{i,k}$ under the star-like approximation, as P_{sym} may result from different combinations of neutral coalescence events and sweep induced coalescence, whereas $P_{star} =P_{0,4}$ corresponds to the probability that all lineages coalesce in a sweep-induced multiple merger event:

%\begin{equation*}

% \frac{1}{3}(1 + e^{-3T_a}4P_{0,3} - e^{-6T_a}(4 P_{0,3} + P_{1,4} - 2 P_{0,4})).

%\end{equation*}%

Instant Yule GF

```
In[ ]:= sample = Table[{1}, {i, 1, 4}];
        substitute[allpees_List] :=
          Total[δ * Peln[#[[1]], #[[2]], Total[#], s, r, Ne, M] & /@ allpees] → δ;
        allFs = Table[partitions[i], {i, 2, Length[sample]}];
        substituteL = substitute /@ allFs
```

```
Out[ ]= {δ Peln[0, 0, 2, s, r, Ne, M] + δ Peln[0, 1, 2, s, r, Ne, M] + δ Peln[0, 2, 2, s, r, Ne, M] +
           δ Peln[1, 0, 2, s, r, Ne, M] + δ Peln[1, 1, 2, s, r, Ne, M] + δ Peln[2, 0, 2, s, r, Ne, M] → δ,
         δ Peln[0, 0, 3, s, r, Ne, M] + δ Peln[0, 1, 3, s, r, Ne, M] + δ Peln[0, 2, 3, s, r, Ne, M] +
           δ Peln[0, 3, 3, s, r, Ne, M] + δ Peln[1, 0, 3, s, r, Ne, M] +
           δ Peln[1, 1, 3, s, r, Ne, M] + δ Peln[1, 2, 3, s, r, Ne, M] + δ Peln[2, 0, 3, s, r, Ne, M] +
           δ Peln[2, 1, 3, s, r, Ne, M] + δ Peln[3, 0, 3, s, r, Ne, M] → δ,
         δ Peln[0, 0, 4, s, r, Ne, M] + δ Peln[0, 1, 4, s, r, Ne, M] + δ Peln[0, 2, 4, s, r, Ne, M] +
           δ Peln[0, 3, 4, s, r, Ne, M] + δ Peln[0, 4, 4, s, r, Ne, M] + δ Peln[1, 0, 4, s, r, Ne, M] +
           δ Peln[1, 1, 4, s, r, Ne, M] + δ Peln[1, 2, 4, s, r, Ne, M] + δ Peln[1, 3, 4, s, r, Ne, M] +
           δ Peln[2, 0, 4, s, r, Ne, M] + δ Peln[2, 1, 4, s, r, Ne, M] + δ Peln[2, 2, 4, s, r, Ne, M] +
           δ Peln[3, 0, 4, s, r, Ne, M] + δ Peln[3, 1, 4, s, r, Ne, M] + δ Peln[4, 0, 4, s, r, Ne, M] → δ}
```

```
In[ ]:= yulegfDeltaList = Expand @GFYule[s, r, Ne, M, ω, sample, {δ}];
        yulegfDeltaList[[0]] = List;
        yulegfDeltaList = yulegfDeltaList //. substituteL;
```

```
In[ ]:= yulePDFS = MakeMargeList[yulegfDeltaList, sample, ω, δ];
        yulePDFS = Total[#] & /@ yulePDFS;
```

```
In[ ]:= yuleCDFS = MakeCumuList[yulegfDeltaList, sample, ω, δ];
        yuleCDFS = Total[#] & /@ yuleCDFS;
```

```
In[ ]:= # /. {s → .05, Ne → 10 000, r → .005, Ta → .1, M → Floor[2 * 10 000 * .05]} /. Peln → PELN & /@
         yuleCDFS(*substitution to get numerical
           values. then can evaluate by changing Peln to PELN*)
```

```
Out[ ]= {0. + 0.0774806 HeavisideTheta[-0.4 + t] + 0.154509 (1 - e^(0.4 - t)) HeavisideTheta[-0.4 + t] +
           0.201528 (1/3 - 1/3 e^(-3/2 (-0.4 + t))) HeavisideTheta[-0.4 + t] +
           0.403056 (1/3 - e^(0.4 - t) + 2/3 e^(-3/2 (-0.4 + t))) HeavisideTheta[-0.4 + t] +
           0.0330252 e^(-0.6 - 3/2 (-0.4 + t)) (-2 + 2 e^(3/2 (-0.4 + t)) - 3 (-0.4 + t)) HeavisideTheta[-0.4 + t] +
           0.132101 e^(-0.6 - 3/2 (-0.4 + t)) (8 - 9 e^(1/2 (-0.4 + t)) + e^(3/2 (-0.4 + t)) + 3 (-0.4 + t)) HeavisideTheta[-0.4 + t] +
           0.919853 (-(1/3 + 2/3 e^(0.6 - 3 t/2) - e^(0.4 - t))) HeavisideTheta[-0.4 + t] +
```

$$1.34986\left(\frac{1}{3}+\frac{2}{3}e^{0.3-\frac{3t}{2}}-e^{0.2-t}\right)\text{HeavisideTheta}[-0.2+t]\Big)+0.1996\,e^{-\frac{3}{2}(0.4+t)}$$
$$\Big(3e^{0.5+\frac{t}{4}}\left(4-5e^{t/4}+e^{5t/4}\right)+2\left(9.11059-6e^{0.5+\frac{t}{4}}+e^{3t/2}\right)\text{HeavisideTheta}[-0.4+t]-$$
$$6.74929\left(-1.10517+e^{t/2}\right)^2\left(2.21034+e^{t/2}\right)\text{HeavisideTheta}[-0.2+t]\Big)+$$
$$0.662891\,e^{-\frac{3}{2}(0.4+t)}\left(\left(1.82212-e^{3t/2}\right)\text{HeavisideTheta}[-0.4+t]-\right.$$
$$\left.1.34986\left(1.34986-e^{3t/2}\right)\text{HeavisideTheta}[-0.2+t]\right)+$$
$$\frac{1}{30}e^{-\frac{3}{2}(0.4+t)}\Big(1.64872\left(-72e^{t/4}+90e^{t/2}-18e^{3t/2}+10e^{0.1+\frac{3t}{2}}-5.52585\,(2+3t)\right)+$$
$$\left(72e^{0.5+\frac{t}{4}}-2e^{3t/2}+9.11059\,(-12.8-3t)\right)\text{HeavisideTheta}[-0.4+t]-$$
$$13.4986\left(9e^{0.2+\frac{t}{2}}-e^{3t/2}+1.34986\,(-7.4-3t)\right)\text{HeavisideTheta}[-0.2+t]\Big)+$$
$$0.510967\,e^{-\frac{3}{2}(0.4+t)}\Big(-\left(-9e^{0.4+\frac{t}{2}}+e^{3t/2}+1.82212\,(6.8+3t)\right)\text{HeavisideTheta}[-0.4+t]-$$
$$1.34986\left(9e^{0.2+\frac{t}{2}}-e^{3t/2}+1.34986\,(-7.4-3t)\right)\text{HeavisideTheta}[-0.2+t]\Big)+$$
$$0.127742\,e^{-\frac{3}{2}(0.4+t)}\left(\left(-2e^{3t/2}+1.82212\,(0.8+3t)\right)\text{HeavisideTheta}[-0.4+t]+\right.$$
$$\left.1.34986\left(2e^{3t/2}+1.34986\,(-1.4-3t)\right)\text{HeavisideTheta}[-0.2+t]\right)+$$
$$0.136549\,e^{-2(0.3+t)}\Big(\left(-20e^{0.6+\frac{t}{2}}+2e^{2t}+18e^{\frac{1}{3}(2.+t)}\right)\text{HeavisideTheta}[-0.4+t]+$$
$$\left(-30.2063+20e^{0.6+\frac{t}{2}}-5e^{0.3+2t}\right)\text{HeavisideTheta}[-0.2+t]+$$
$$4.94616\left(6.10701+e^{2t}-6e^{\frac{1}{3}(0.5+t)}\right)\text{HeavisideTheta}[-0.1+t]\Big)+$$
$$0.262652\,e^{-2(0.3+t)}\Big(\left(40e^{0.6+\frac{t}{2}}+2e^{2t}-15e^{0.4+t}-27e^{\frac{1}{3}(2.+t)}\right)\text{HeavisideTheta}[-0.4+t]+$$
$$6.74929\left(1.10517-e^{t/2}\right)^3\left(3.31551+e^{t/2}\right)\text{HeavisideTheta}[-0.2+t]+$$
$$4.94616\left(-6.10701+e^{2t}-5e^{0.1+t}+9e^{\frac{1}{3}(0.5+t)}\right)\text{HeavisideTheta}[-0.1+t]\Big)+$$
$$\frac{2}{15}e^{-2(0.3+t)}\Big(\left(-e^{2t}+81e^{\frac{1}{3}(2.+t)}+5e^{0.6+\frac{t}{2}}(-17.2+3t)\right)\text{HeavisideTheta}[-0.4+t]+$$
$$6.74929\left(-13.4264+e^{2t}+e^{0.3+\frac{t}{2}}(9.2-6t)\right)\text{HeavisideTheta}[-0.2+t]+$$
$$1.64872\Big(5e^{0.1+\frac{t}{2}}\left(8-9e^{t/2}+e^{3t/2}+3t\right)+$$
$$9\left(6.10701-e^{2t}+5e^{0.1+t}-9e^{\frac{1}{3}(0.5+t)}\right)\text{HeavisideTheta}[-0.1+t]\Big)\Big),$$
$$0.206163+0.620551\left(\frac{1}{3}-\frac{e^{-3t}}{3}\right)+0.0258269\left(1-e^{-t/2}\right)+0.310276\left(\frac{1}{3}+\frac{e^{-3t}}{15}-\frac{2e^{-t/2}}{5}\right)+$$
$$0.0374699\Big(2-12e^{5t/2}+10e^{3t}+3\left(1.64872+4e^{5t/2}-5e^{0.1+2t}\right)\text{HeavisideTheta}[-0.2+t]-$$
$$5\left(-1.10517+e^{t}\right)^2\left(1.10517+2e^{t}\right)\text{HeavisideTheta}[-0.1+t]\Big)+$$
$$0.517111\left(-1+e^{3t}+\left(1.34986-e^{3t}\right)\text{HeavisideTheta}[-0.1+t]\right)+$$
$$0.255484\,e^{-3(0.2+t)}\left(\left(-1+e^{3t}\right)^2-\left(-1.34986+e^{3t}\right)^2\text{HeavisideTheta}[-0.1+t]\right)+$$
$$0.0399067\,e^{-0.6-\frac{t}{2}}\left(6-7e^{t/2}+e^{7t/2}+\left(-8.51441+7e^{0.3+\frac{t}{2}}-e^{7t/2}\right)\text{HeavisideTheta}[-0.1+t]\right)+$$

$$\frac{2}{15} e^{-3(0.2+t)} \left(-9.11059 - e^{3t} + e^{6t} + 9 e^{0.5+t} - 9 e^{0.5+3t} + 5 e^{0.6+3t} + \left(9.11059 - e^{6t} + 5 e^{3(0.1+t)} - 9 e^{0.5+t}\right) \text{HeavisideTheta}[-0.1 + t]\right) + 0.399201\, e^{-2(0.3+t)} \left(-4.94616 + 2 e^{2t} - 2 e^{5t} + 3 e^{0.5+2t} + \left(4.94616 + 2 e^{5t} - 5 e^{0.3+2t}\right) \text{HeavisideTheta}[-0.1 + t]\right) + 0.0187608\, e^{-0.6-\frac{t}{2}} \left(2\left(-5 + 7 e^{t/2} - 7 e^{3t} + 5 e^{7t/2}\right) + 7\left(-3.64424 + 3 e^{0.5+\frac{t}{2}} + 2 e^{3t} - 3 e^{0.1+\frac{5t}{2}}\right) \text{HeavisideTheta}[-0.2 + t] + \left(34.0576 - 35 e^{0.3+\frac{t}{2}} - 10 e^{7t/2} + 21 e^{0.1+\frac{5t}{2}}\right) \text{HeavisideTheta}[-0.1 + t]\right) + 0.00364977\, e^{-3(0.2+t)} \left(-7 + 72 e^{5t/2} - 70 e^{3t} + 5 e^{6t} + \left(12.7548 - 5 e^{6t} + 70 e^{3(0.1+t)} - 72 e^{0.35+\frac{5t}{2}}\right) \text{HeavisideTheta}[-0.1 + t]\right) + \frac{1}{15} e^{-3(0.2+t)} \left(1.82212 - e^{3t} + 6 e^{11t/2} - 5 e^{6t} - 6 e^{0.6+\frac{5t}{2}} + 5 e^{0.6+3t} + \left(-6 e^{11t/2} + 6 e^{0.6+\frac{5t}{2}} - 9 e^{0.5+3t} + 9 e^{0.1+5t}\right) \text{HeavisideTheta}[-0.2 + t] + \left(-1.82212 + 5 e^{6t} + 5 e^{3(0.1+t)} - 9 e^{0.1+5t}\right) \text{HeavisideTheta}[-0.1 + t]\right),$$

$$0.379588 + 0.566741\left(1 - e^{-t}\right) + \frac{2}{15} e^{-0.6-t} \left(-9.11059 - e^{t} + e^{6t} - 5 e^{3(0.1+t)} + 5 e^{0.3+t} + 5 e^{0.6+t} + \left(9.11059 - e^{6t} + 5 e^{3(0.1+t)} - 9 e^{0.5+t}\right) \text{HeavisideTheta}[-0.1 + t]\right) + 0.0875505\, e^{-0.6-t} \left(8.49859 + 6 e^{t} - e^{6t} + 5 e^{3(0.1+t)} - 15 e^{0.3+t} + \left(-9.11059 + e^{6t} - 5 e^{3(0.1+t)} + 9 e^{0.5+t}\right) \text{HeavisideTheta}[-0.1 + t]\right) + 0.0749398 \left(-4.74929 - 2 e^{5t} + 5 e^{0.3+2t} + \left(4.94616 + 2 e^{5t} - 5 e^{0.3+2t}\right) \text{HeavisideTheta}[-0.1 + t]\right)\Big\}$$

i-Ton marginals: starlike vs yule approximation

In[ ]:=
```
recpersite = 1. * 10^-7;
winDist = 1000;
datNe = 10 000;
sweepTimes = {0, .1, .25, .5, 1.0, 1.5, 2.0} * datNe * 2
classicDataShape = {10 000, 250, 3}
```

Out[ ]= {0, 2000., 5000., 10 000., 20 000., 30 000., 40 000.}

Out[ ]= {10 000, 250, 3}

In[ ]:=
```
localPath = SetDirectory[NotebookDirectory[]];
pathToSims = localPath <> "/simulations/classic_marginals/";
classicSweepDataFiles = {"np_s4_0.dat",
   "np_s4_0_1.dat",
   "np_s4_0_25.dat",
   "np_s4_0_5.dat",
   "np_s4_1.dat",
   "np_s4_1_5.dat",
   "np_s4_2.dat"
  };
classicSweepDataFiles = pathToSims <> # & /@ classicSweepDataFiles
```

Out[ ]=
```
{/home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _0.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _0_1.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _0_25.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _0_5.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _1.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _1_5.dat,
 /home/derek/Dropbox/projects/selection/ms_notebooks/simulations/classic_marginals/np_s4
   _2.dat}
```

In[ ]:=
```
classicSweepData = MapThread[MakeDataArray[#1, #2, classicDataShape, datNe] &,
   {sweepTimes, classicSweepDataFiles}];
```

MapThread : Object classicSweepDatFiles at position {2, 2} in MapThread [MakeDataArray [#1, #2, classicDataShape , datNe] &, {{0, 2000. , 5000. , 10000. , 20000. , 30000. , 40000. }, classicSweepDatFiles }] has only 0 of required 1 dimensions .

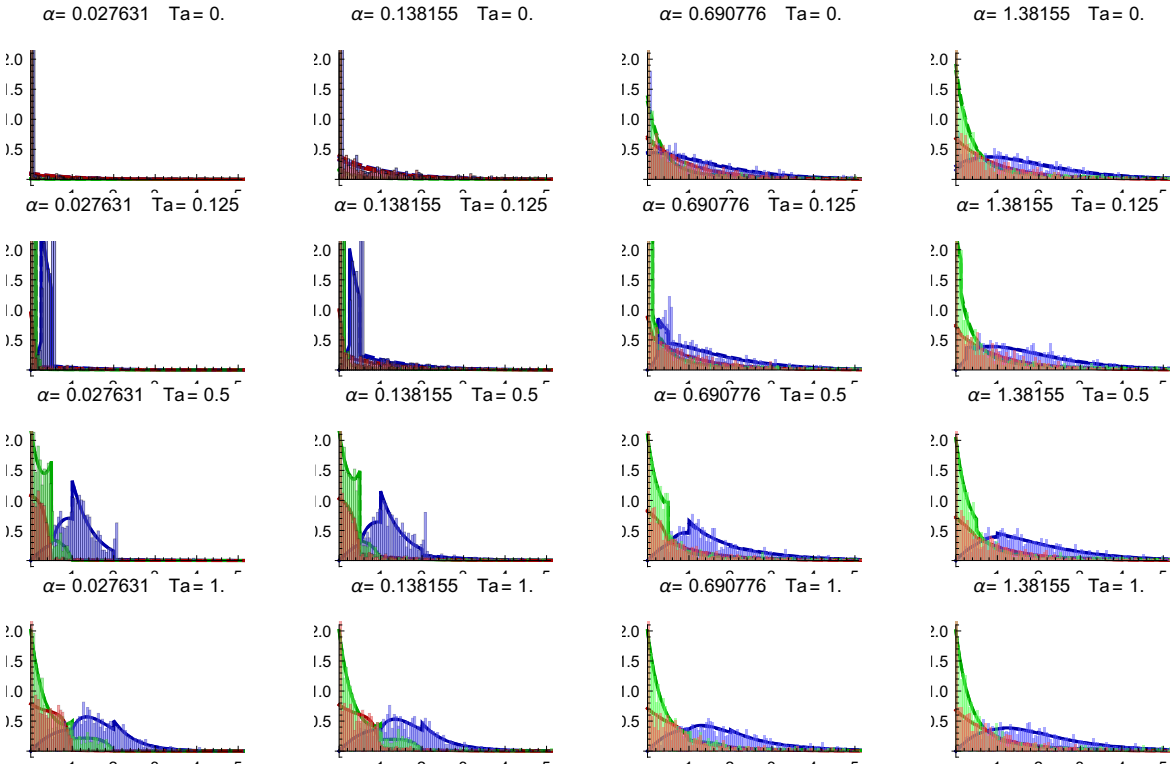
The starlike approximation suffices for classic hard sweeps. We see little difference between the predictions of the starlike (dashed) and yule (solid) predictions in the figures below, if only a slight improvement of the fit of the CDFs using the more-complicated Yule approximations. In contrast, the computation cost increases dramatically when using the Yule approximation.

```
In[ ]:= plt0[idxT_, idxr_, ss_, nn_] := Block[
          {rec = recpersite * idxr * winDist,
           M = Floor[2 nn 2 ss], Td = sweepTimes[[idxT]] / (2 * datNe), α},
          α = rec / ss Log[2 nn ss];
          Show[
           Plot[Evaluate[yulePDFS /. {s → ss, r → rec, Ne → nn, M → M, Ta → Td} /. Peln → PELN],
            {t, 0, 6}, PlotStyle → Evaluate[Darker[#] & /@ {Blue, Green, Red}],
            Exclusions → None, PlotRange → Full],
           Plot[Evaluate[#[α, Td, t] & /@ selPdfs], {t, 0, 6}, PlotRange → Full, PlotStyle →
             Evaluate[{Darker[#], Dashed} & /@ {Blue, Green, Red}], Exclusions → None],
           Histogram[classicSweepData [[idxT]][[idxr]][[2]], {.05}, "PDF",
            ChartStyle → Evaluate[Lighter[#] & /@ {Blue, Green, Red}]
           ],
           PlotLabel → "α= " <> ToString[α] <> "  Ta= " <> ToString[Td],
           PlotRange → {{0, 5}, {0, 2}}
          ]
         ]

In[ ]:= plt1[idxT_, idxr_, ss_, nn_] := Block[
          {rec = recpersite * idxr * winDist,
           M = Floor[2 nn 2 ss], Td = sweepTimes[[idxT]] / (2 * datNe), α},
          α = rec / ss Log[2 nn ss];
          Show[
           Plot[Evaluate[yuleCDFS /. {s → ss, r → rec, Ne → nn, M → M, Ta → Td} /. Peln → PELN],
            {t, 0, 6}, PlotStyle → Evaluate[Darker[#] & /@ {Blue, Green, Red}],
            Exclusions → None, PlotRange → Full],
           Plot[Evaluate[#[α, Td, t] & /@ selCdfs], {t, 0, 6}, PlotRange → Full, PlotStyle →
             Evaluate[{Darker[#], Dashed} & /@ {Blue, Green, Red}], Exclusions → None],
           Histogram[classicSweepData [[idxT]][[idxr]][[2]], {.05}, "CDF",
            ChartStyle → Evaluate[Lighter[#] & /@ {Blue, Green, Red}]
           ],
           PlotLabel → "α= " <> ToString[α] <> "  Ta= " <> ToString[Td],
           PlotRange → {{0, 5}, {0, 1.01}}
          ]
         ]
```

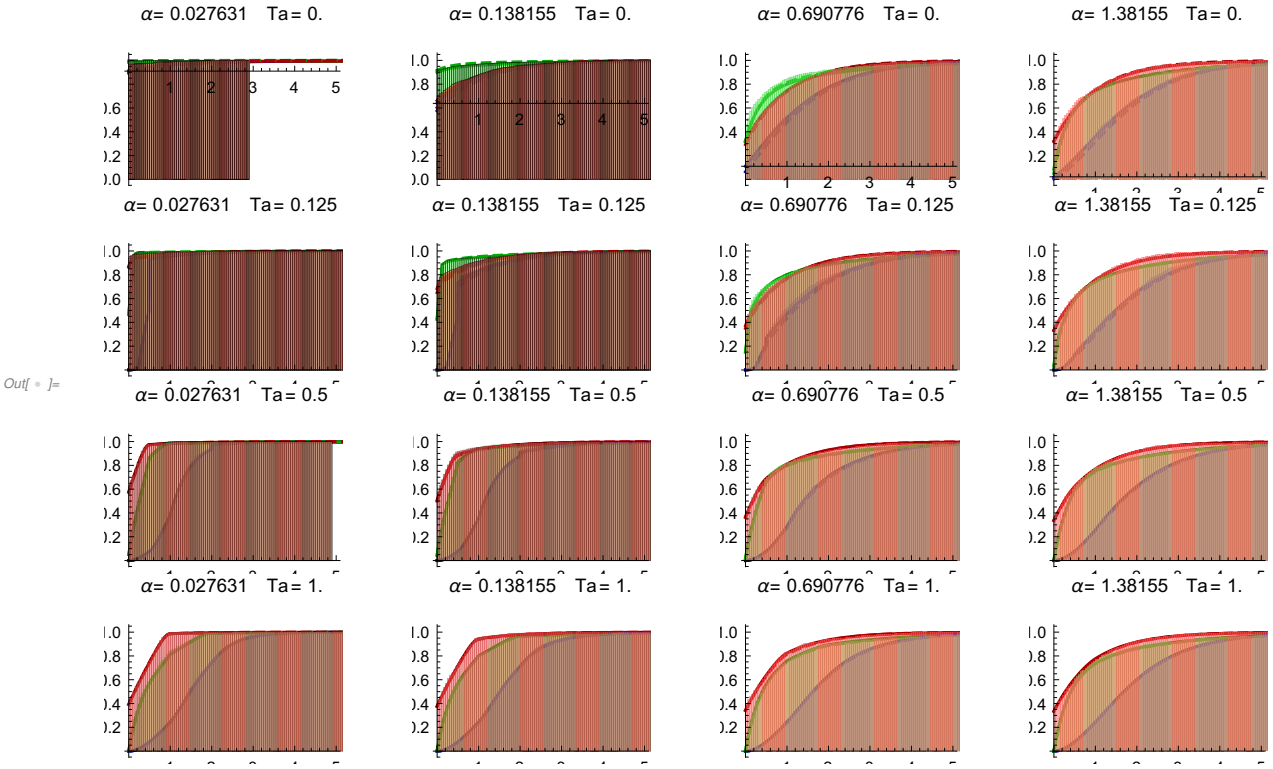
In[ ]:=
```
GraphicsGrid[
 Table[
  Table[
   plt0[idxT, idxr, .05, 10000], {idxr, {2, 10, 50, 100}}], {idxT, {1, 3, 5, 7}}],
 ImageSize → Full, Spacings → 0
]
```

Out[ ]=



In[ ]:=
```
GraphicsGrid[
 Table[
  Table[
   plt1[idxT, idxr, .05, 10000], {idxr, {2, 10, 50, 100}}], {idxT, {1, 3, 5, 7}}],
 ImageSize → Full, Spacings → 0
]
```

Out[ ]=



Site Frequency Spectrum, n = 9

```
(*n=9;
gfList = Block[{sample = Table[{1},{i,1,n}],gfDeltaEpsilon},
   gfDeltaEpsilon =GFstar[α,ω,sample,δ] //Expand;
   gfDeltaEpsilon[[0]] = List;
   gfDeltaEpsilon = SubSimplifyPeeRulesv2 [#,n]&/@gfDeltaEpsilon //Parallelize;
   gfDeltaEpsilon];

meansList = MakeMeanList[gfList,n,ω,t,δ,Ta];
DumpSave["iTonBL_9_gfList_meanList.mx",{gfStarList,meansList}]*)
```

In[ ]:=
```
SetDirectory[NotebookDirectory[]];
(*DumpSave["iTonBL_9_gfList_meanList.mx",{gfStarList,meansList}]*)
Get["iTonBL_9_gfList_meanList.mx"];
meanItonFunList = Function[{α, Ta}, Evaluate[# //. P → PknToAlpha]] & /@ meanList;
sfs[α_, Ta_] := Block[{a = #[α, Ta] & /@ meanItonFunList}, a / Total[a]]
```

In[ ]:=
```
talist = {0.00000001, .1, .25, .5, 1, 1.5, 2};
alphaList = {0, 0.25, 0.5};
legendItems = "Ta= " <> ToString[#] & /@ {0, .1, .25, .5, 1, 1.5, 2};
titleItems = "α= " <> ToString[#] & /@ {0, .25, .75}

(*plt1 = getSFS[expectedBLs,0,#]&/@tslist;
plt2 = getSFS[expectedBLs,0.25,#]&/@tslist;
plt3 = getSFS[expectedBLs,0.75,#]&/@tslist;*)
plt1 = sfs[0, #] & /@ talist;
plt2 = sfs[0.25, #] & /@ talist;
plt3 = sfs[0.75, #] & /@ talist;
```

Out[ ]=
```
{α= 0, α= 0.25, α= 0.75}
```

In[ ]:=
```
pltFull = GraphicsRow[
    MapThread[
     ListPlot[#1, Joined → True, PlotRange → {{0, 9}, All},
       Frame → {{True, False}, {True, False}}, FrameLabel → {"iTon class", "prob"},
       FrameStyle → Directive[Thick, Black, Bold, 12],
       LabelStyle → Directive[12, Bold, Black], PlotStyle → Directive[12, Bold, Thick],
       (*PlotLegends→SwatchLegend[legendItems],*)
       PlotLabel → #2] &, {{plt1, plt2, plt3}, titleItems}
    ],
    Spacings → 0, ImageSize → Full
  ];
(*Export["~/Dropbox/projects/selection/figs_for_ms/n=9_sfs.pdf",pltFull];*)
pltFull
```

Out[ ]=