Supplemental Text

Relationship Between Growth Rates and Fixation Times

The relationship between fixation of a mutated strain is related to the difference in starting and mutated growth rates. Fixation is defined to be when the mutated strain represents 95% of the total cell count. However, the percentage used does not affect the relationship.

$$\mu_1 = growth\ rate\ of\ starting\ strain$$
$$\mu_2 = growth\ rate\ of\ mutated\ strain$$
$$C_1 = initial\ cell\ count\ of\ starting\ strain$$
$$C_2 = initital\ cell\ count\ of\ mutated\ strain$$
$$t_{fix} = time\ to\ fixation$$
$$0.95 = \frac{C_2 e^{\mu_2 t_{fix}}}{C_1 e^{\mu_1 t_{fix}}}$$
$$C_2 = 1$$
$$0.95C_1 = \frac{e^{\mu_2 t_{fix}}}{e^{\mu_1 t_{fix}}}$$
$$0.95C_1 = e^{(\mu_2 - \mu_1)t_{fix}}$$
$$ln(0.95C_1) = (\mu_2 - \mu_1)t_{fix}$$
$$\frac{ln(0.95C_1)}{\mu_2 - \mu_1} = t_{fix}$$

There the fixation time is inversely related to the difference in growth rates.

Supplementary Data Files

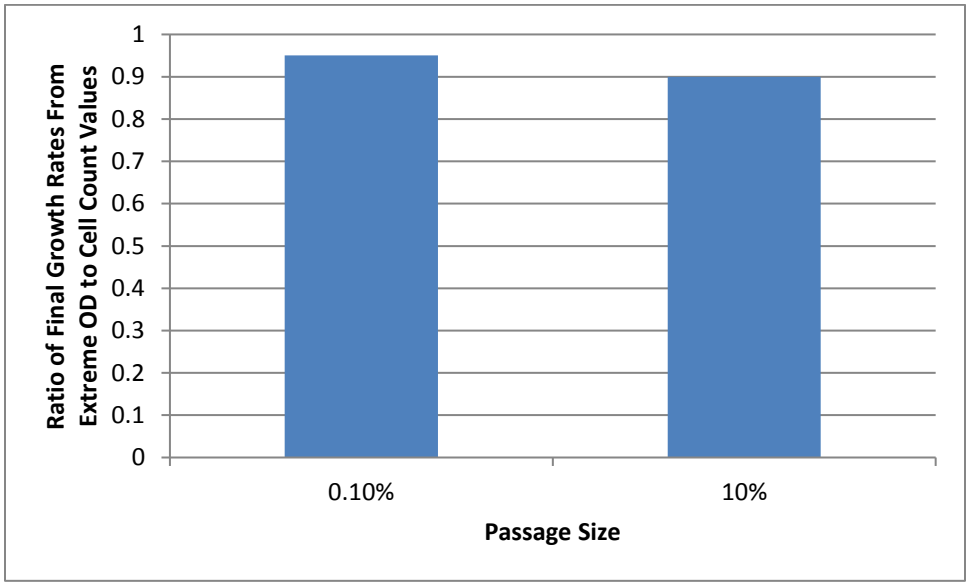
Supplementary Data File 1 – Fitness Data from ALE experiment

Table S1

Adaptive Evolution Summary

Experiment #	Batches	Doublings per batch	Cumulative Cell Divisions
1	208	2.3	6.35E+12
2	182	2.3	5.64E+12
3	176	2.3	3.76E+12
4	195	2.3	5.58E+12
5	216	2.3	6.57E+12
6	187	2.3	5.67E+12
7	106	4.6	3.54E+12
8	104	4.6	3.37E+12
9	105	4.6	3.53E+12
10	106	4.6	3.53E+12
11	109	4.6	3.60E+12
12	114	4.6	3.81E+12
13	69	6.9	2.36E+12
14	68	6.9	2.36E+12
15	71	6.9	2.28E+12
16	70	6.9	2.42E+12
17	70	6.9	2.43E+12
18	70	6.9	2.42E+12
19	34	9.2	1.30E+12
20	33	9.2	1.15E+12
21	39	9.2	1.38E+12
22	44	9.2	1.38E+12
23	41	9.2	1.69E+12
24	42	9.2	1.49E+12
25	36	11.5	1.29E+12
26	25	11.5	8.98E+11
27	34	11.5	1.31E+12
28	34	11.5	1.24E+12
29	31	11.5	1.31E+12
30	33	11.5	1.17E+12

Figure S1



Sensitivity analysis of dry weight per cell values – Using extreme values of OD to cell count, Identical simulation were run. The difference in final growth rates observed was never more than 10% off of the original value.

ALEsim Source Code

```
*****************
* ALEsim\ALE2.m *
*****************



classdef ALE2 < handle


    properties
        flasks
        maxTime
        Volume
        inocVolume
        OD
        num_flasks
        od
        inoc_volume
        volume
        inoc_cells
        growth_rate
        props
        time
        max_growth
        time_stop
        folder_path
    end
    properties (Hidden)
```

```
    end

    methods

        function obj = ALE2(Volume, inocVolume, OD, inocCells,
growthRate, props, time, maxGrowth, samples, sDev, varargin)
```

```
if ~isempty(varargin)
    title = varargin{1};
else
    title = 'ALE_sim';
end

if samples > 1
    growthRate(samples) = 0;
    inocCells(samples) = 0;
    for i = 2:1:samples
        growthRate(i) = growthRate(1) +
sDev*sqrt(12*10)*((rand+rand+rand+rand+rand+rand+rand+rand+rand+rand)/
10-.5);
```

```
                inocCells(i) = inocCells(1)*rand/100;
                props{i} = props{1};
            end
        end


        obj.maxTime = time;
        obj.volume = Volume;
        obj.inoc_volume = inocVolume;
        obj.od = OD;
        obj.inoc_cells = inocCells;
        obj.growth_rate = growthRate;
        obj.props = props;
        obj.time = time;
        obj.max_growth = maxGrowth;


        obj.num_flasks = max([length(obj.volume)
length(obj.inoc_volume) length(obj.od)]);


        if obj.num_flasks ==1
            if obj.time <= 0
                error('Time must be greater than 0.')
            end

            obj.time_stop = true;
```

```
            t = log(obj.volume/obj.inoc_volume)/obj.max_growth;
            obj.num_flasks = round(1.1*(obj.time/t+1));
        else
            obj.time_stop = false;
        end



        if length(obj.volume) == 1
            obj.volume = repmat(obj.volume,1,obj.num_flasks);
        end
        if length(obj.inoc_volume) == 1
            obj.inoc_volume =
repmat(obj.inoc_volume,1,obj.num_flasks);
        end
        if length(obj.od) == 1
            obj.od = repmat(obj.od,1,obj.num_flasks);
        end



        if length(obj.volume) ~= obj.num_flasks
            error('obj.volume array size does not match number of
flasks')
        end
        if length(obj.inoc_volume) ~= obj.num_flasks
            error('obj.inoc_volume array size does not match
number of flasks')
        end
```

```
            if length(obj.od) ~= obj.num_flasks
                error('obj.od array size does not match number of
flasks')
            end



            if obj.volume(1) < 0
                vol = ALE_Volume(1, obj.volume, obj.inoc_volume,
obj.od, obj.inoc_cells, obj.growth_rate, obj.props, obj.time);
            else
                vol = obj.volume(1);
            end



            if obj.inoc_volume(1) < 0
                iVol = ALE_obj.inoc_volume(1, obj.volume,
obj.inoc_volume, obj.od, obj.inoc_cells, obj.growth_rate, obj.props,
obj.time);
            else
                iVol = obj.inoc_volume(1);
            end



            if obj.od(1) < 0
                od = ALE_OD(1, obj.volume, obj.inoc_volume, obj.od,
obj.inoc_cells, obj.growth_rate, obj.props, obj.time);
            else
                od = obj.od(1);
            end
```

```
            obj.flasks{1} = flask(vol, iVol, obj.inoc_cells,
obj.growth_rate, obj.props, obj.max_growth, od);


            obj.run()

        end


        function run(obj)


            i = 2;

            while i <= obj.num_flasks









                if i > length(obj.volume)

                    obj.volume(i) = obj.volume(i-1);

                end

                if obj.volume(i) < 0

                    vol = ALE_Volume(i, obj.volume, obj.inoc_volume,
obj.od, obj.inoc_cells, obj.growth_rate, obj.props, obj.time);

                else

                    vol = obj.volume(i);

                end
```

```
                if i > length(obj.inoc_volume)

                    obj.inoc_volume(i) = obj.inoc_volume(i-1);

                end

                if obj.inoc_volume(i) < 0

                    iVol = ALE_obj.inoc_volume(i, obj.volume,
obj.inoc_volume, obj.od, obj.inoc_cells, obj.growth_rate, obj.props,
obj.time);

                else

                    iVol = obj.inoc_volume(i);

                end




                if i > length(obj.od)

                    obj.od(i) = obj.od(i-1);

                end

                if obj.od(i) < 0

                    od = ALE_OD(i, obj.volume, obj.inoc_volume,
obj.od, obj.inoc_cells, obj.growth_rate, obj.props, obj.time);

                else

                    od = obj.od(i);

                end




                arr = inoculum(obj.flasks{i-1}, iVol);
```

```
                obj.flasks{i} = flask(vol, iVol, arr{1}, arr{2},
arr{3}, obj.max_growth, od);


                i=i+1;



                if obj.time_stop
                    if totalTime(obj) < obj.time
                        obj.num_flasks = i+1;
                    else
                        obj.num_flasks = i-1;
                    end
                end

            end







        end


        function time = totalTime(obj)



            time = 0;
            flask = obj.flasks;
```

```
            for i = 1:1:length(flask)
                time = time + flask{i}.FinalTime;
            end
        end
    end
end
**************************
* ALEsim\apparentGrowth.m *
**************************



function apparentGrowth(evos, index)

    A = evos{index};
    flaskNum = 1:1:length(A.flasks);
    growth = flaskNum;
    for i = 1:1:length(flaskNum)
        growth(i) = A.flasks{i}.apparentGrowth;
    end

    figure
    plot(flaskNum,growth, '.')
    title('Apparent Growth rate per Flask')
    xlabel('Flask Number')
    ylabel('Apparent Growth Rate (1/hr)')
```

```
end
**********************
* ALEsim\avgGrowth.m *
**********************



function varargout = avgGrowth(evos)




    maxFlaskNum = 0;
    maxTime = 0;
    for i=1:1:length(evos)
        if length(evos{i}.flasks) > maxFlaskNum
            maxFlaskNum = length(evos{i}.flasks);
        end


        if totalTime(evos{i}) > maxTime
            maxTime = totalTime(evos{i});
        end
    end



    t = 0:maxTime/100:maxTime;


    avgG = zeros(1, length(t));
```

```
for i=1:1:length(t)

    avgTemp = zeros(1, length(evos));
    for ii=1:1:length(evos)
        try
        avgTemp(ii) = growthAtTime(evos{ii}, t(i));
        catch

        end
    end


    for n=length(avgTemp):-1:1
        if avgTemp(n) <= 0
            avgTemp(n) = [];
        end
    end

    avgG(i) = mean(avgTemp);

    str = [num2str(i-1) '% Completed'];
    fprintf(str);
end

varargout{1} = t;
varargout{2} = avgG;
```

```
    plot(t,avgG)
    title('Average Growth Rate per time')
    xlabel('Time (hr)')
    ylabel('growthRate (1/hr)')




end

function growth = growthAtTime(evo, t)



    tot = 0;
    i=1;
    loop = true;
    while loop && i < length(evo.flasks)
        tot = tot + evo.flasks{i}.FinalTime;
        if t < tot
            growth = evo.flasks{i}.apparentGrowth;
            loop = false;
        end
        i=i+1;
```

```
    end

    if t >= tot
        growth = -1;
    end

end
************************
* ALEsim\avgGrowthCCD.m *
************************

function varargout = avgGrowthCCD(evos)

    maxFlaskNum = 0;
    maxCCD = 0;
    for i=1:1:length(evos)
        if length(evos{i}.flasks) > maxFlaskNum
            maxFlaskNum = length(evos{i}.flasks);
        end

        if totalTime(evos{i}) > maxCCD
            maxCCD = totalCCD(evos{i});
        end
    end
```

```
CCD = 0:maxCCD/100:maxCCD;

avgG = zeros(1, length(CCD));

for i=1:1:length(CCD)

    avgTemp = zeros(1, length(evos));
    for ii=1:1:length(evos)
        try
        avgTemp(ii) = growthAtCCD(evos{ii}, CCD(i));
        catch

        end
    end


    for n=length(avgTemp):-1:1
        if avgTemp(n) <= 0
            avgTemp(n) = [];
        end
    end

    avgG(i) = mean(avgTemp);

    str = [num2str(i-1) '% Completed'];
```

```
        fprintf(str);
    end

    varargout{1} = CCD;
    varargout{2} = avgG;


    plot(CCD,avgG)
    title('Average Growth Rate per time')
    xlabel('Time (hr)')
    ylabel('Cumulative Cell Divisions ()')




end

function growth = growthAtCCD(evo, CCD)



    tot = 0;
    i=1;
    loop = true;
    while loop && i < length(evo.flasks)
```

```
        tot = tot + sum(evo.flasks{i}.FinalCells) -
sum(evo.flasks{i}.InitCells);

        if CCD < tot

            growth = evo.flasks{i}.apparentGrowth;

            loop = false;

        end

        i=i+1;

    end


    if CCD >= tot

        growth = -1;

    end


end




function total_CCD = totalCCD(evo)

    total_CCD = 0;

    for f = 1:1:length(evo.flasks)

        total_CCD = total_CCD + sum(evo.flasks{f}.FinalCells) -
sum(evo.flasks{f}.InitCells);

    end

end

**************************

* ALEsim\avgGrowthFlask.m *

**************************
```

```
function avgGrowthFlask(evos)




    minFlaskNum = length(evos{1}.flasks);
    for i=1:1:length(evos)
        if length(evos{i}.flasks) < minFlaskNum
            minFlaskNum = length(evos{i}.flasks);
        end
    end

    flask = 1:1:minFlaskNum;



    growths = zeros(length(evos), minFlaskNum);
    for i=1:1:length(evos)
        for n=1:1:minFlaskNum
            growths(i,n) = evos{i}.flasks{n}.apparentGrowth;
        end
    end



    sDev = zeros(1,minFlaskNum);
    avGrowth = sDev;
    for n = 1:1:minFlaskNum
        sDev(n) = std(growths(:,n));
```

```
        avGrowth(n) = mean(growths(:,n));
    end









    plot(flask,avGrowth,'*')
    title('Average Growth Rate per flask')
    xlabel('Flask
    ylabel('growthRate (1/hr)')
    hold off




end
********************
* ALEsim\binopdf.m *
********************


function y = binopdf(x,n,p)
```

```
if nargin < 3,
    error('stats:binopdf:TooFewInputs','Requires three input
arguments');
end


[errorcode x n p] = distchck(3,x,n,p);


if errorcode > 0
    error('stats:binopdf:InputSizeMismatch',...
          'Requires non-scalar arguments to match in size.');
end




if isa(x,'single') || isa(n,'single') || isa(p,'single')
   y = zeros(size(x),'single');
else
```

```
   y = zeros(size(x));
end



if ~isfloat(x)
   x = double(x);
end
if ~isfloat(n)
   n = double(n);
end
k = find(x >= 0  &  x == round(x)  &  x <= n);
if any(k)

   t = (p(k)==0);
   if any(t)
      kt = k(t);
      y(kt) = (x(kt)==0);
      k(t) = [];
   end
   t = (p(k)==1);
   if any(t)
      kt = k(t);
      y(kt) = (x(kt)==n(kt));
      k(t) = [];
   end
end
```

```
if any(k)
   nk = gammaln(n(k) + 1) - gammaln(x(k) + 1) - gammaln(n(k) - x(k) +
1);
   lny = nk + x(k).*log( p(k)) + (n(k) - x(k)).*log1p(-p(k));
   y(k) = exp(lny);
end

k1 = find(n < 0 | p < 0 | p > 1 | round(n) ~= n);
if any(k1)
   y(k1) = NaN;
end
************************
* ALEsim\cell2mutate.m *
************************



function cell = cell2mutate(cells)
    cells = cells/sum(cells);
    cell = randIndex(cells);
    if cell == 0
        error('cells array does not sum to 1')
    end
end


function index = randIndex(prob)
```

```
    rNum = rand;
    index = 1;



    if length(prob) == 1
        index = 1;



    else
        sums = cumsum(prob);
        for i=1:1:length(prob)-1
            if rNum <= sums(length(prob)-i+1)
                index = length(prob)-i+1;
            end
        end
    end
end
**********************
* ALEsim\distchck.m *
**********************
```

```
function [errorcode,out1,out2,out3,out4] =
distchck(nparms,arg1,arg2,arg3,arg4)









errorcode = 0;


if nparms == 1
    out1 = arg1;
    return;
end


if nparms == 2
    [r1 c1] = size(arg1);
    [r2 c2] = size(arg2);
    scalararg1 = (prod(size(arg1)) == 1);
    scalararg2 = (prod(size(arg2)) == 1);
    if ~scalararg1 & ~scalararg2
        if r1 ~= r2 | c1 ~= c2
            errorcode = 1;
            return;
        end
    end
    if scalararg1
```

```
        out1 = arg1(ones(r2,1),ones(c2,1));
    else
        out1 = arg1;
    end
    if scalararg2
        out2 = arg2(ones(r1,1),ones(c1,1));
    else
        out2 = arg2;
    end
end

if nparms == 3
    [r1 c1] = size(arg1);
    [r2 c2] = size(arg2);
    [r3 c3] = size(arg3);
    scalararg1 = (prod(size(arg1)) == 1);
    scalararg2 = (prod(size(arg2)) == 1);
    scalararg3 = (prod(size(arg3)) == 1);

    if ~scalararg1 & ~scalararg2
        if r1 ~= r2 | c1 ~= c2
            errorcode = 1;
            return;
        end
    end

    if ~scalararg1 & ~scalararg3
```

```
    if r1 ~= r3 | c1 ~= c3
        errorcode = 1;
        return;
    end
end

if ~scalararg3 & ~scalararg2
    if r3 ~= r2 | c3 ~= c2
        errorcode = 1;
        return;
    end
end

if ~scalararg1
  out1 = arg1;
end
if ~scalararg2
  out2 = arg2;
end
if ~scalararg3
  out3 = arg3;
end
rows = max([r1 r2 r3]);
columns = max([c1 c2 c3]);

if scalararg1
    out1 = arg1(ones(rows,1),ones(columns,1));
```

```
    end
   if scalararg2
        out2 = arg2(ones(rows,1),ones(columns,1));
   end
   if scalararg3
       out3 = arg3(ones(rows,1),ones(columns,1));
   end
     out4 =[];

end

if nparms == 4
    [r1 c1] = size(arg1);
    [r2 c2] = size(arg2);
    [r3 c3] = size(arg3);
    [r4 c4] = size(arg4);
    scalararg1 = (prod(size(arg1)) == 1);
    scalararg2 = (prod(size(arg2)) == 1);
    scalararg3 = (prod(size(arg3)) == 1);
    scalararg4 = (prod(size(arg4)) == 1);

    if ~scalararg1 & ~scalararg2
        if r1 ~= r2 | c1 ~= c2
            errorcode = 1;
            return;
        end
    end
```

```
if ~scalararg1 & ~scalararg3
    if r1 ~= r3 | c1 ~= c3
        errorcode = 1;
        return;
    end
end

if ~scalararg1 & ~scalararg4
    if r1 ~= r4 | c1 ~= c4
        errorcode = 1;
        return;
    end
end

if ~scalararg3 & ~scalararg2
    if r3 ~= r2 | c3 ~= c2
        errorcode = 1;
        return;
    end
end

if ~scalararg4 & ~scalararg2
    if r4 ~= r2 | c4 ~= c2
        errorcode = 1;
        return;
    end
```

```
 end

 if ~scalararg3 & ~scalararg4
     if r3 ~= r4 | c3 ~= c4
         errorcode = 1;
         return;
     end
 end



 if ~scalararg1
    out1 = arg1;
 end
 if ~scalararg2
    out2 = arg2;
 end
 if ~scalararg3
   out3 = arg3;
 end
 if ~scalararg4
   out4 = arg4;
 end

rows = max([r1 r2 r3 r4]);
columns = max([c1 c2 c3 c4]);
 if scalararg1
    out1 = arg1(ones(rows,1),ones(columns,1));
```

```
    end
    if scalararg2
        out2 = arg2(ones(rows,1),ones(columns,1));
    end
    if scalararg3
        out3 = arg3(ones(rows,1),ones(columns,1));
    end
    if scalararg4
        out4 = arg4(ones(rows,1),ones(columns,1));
    end
end
********************
* example_script.m *
********************




clear classes
clear java
```

```
Vol = 0.250;

iVol = 10e-6;

inoc = 1.8e+07;

growth = .28;

props{1} = 0;

samples = 0;
sDev = .1;

OD = 1.2;

maxGrowth = 1.0;

numEvos = 100;
```

```
    time = 24*50;



    title = 'ALEsim_example';




addpath('ALEsim')

mkdir(fullfile(pwd, title))

fprintf([datestr(now) ''])

tic

evos = {};

parfor ii=1:numEvos

    evos{ii} = ALE2(Vol, iVol, OD, inoc, growth, props, time,
maxGrowth, samples, sDev, title);

    e_temp = evos{ii};

    parsave(fullfile(fullfile(pwd, title), [title '_split_'
num2str(ii) '.mat']),e_temp)

end




fprintf([datestr(now) ''])

t = toc;

str = ['Elapsed time is ' num2str(t) ' seconds'];

fprintf(str)

str = ['Time per ALE is ' num2str(t/numEvos) ' seconds'];
```

```
fprintf(str)


save(fullfile(fullfile(pwd, title), [title datestr(now,30)
'.mat']),'evos')

******************

* ALEsim\flask.m *

******************



classdef flask < handle









    properties

        Volume

        InitCells

        GrowthRate

        CellProps

        FinalCells

        FinalTime

        OD
```

```
        mutNum
        apparentGrowth
        maxGrowth
    end
    properties (Hidden)
        wInit
        OD2Cells=1551724137931.03;
        InitTime
        cellsAtOD
    end
    methods
        function obj = flask(Volume, InnoculumVolume, Innoculum,
Growth, Props, maxGrowth, OD)

            obj.Volume = Volume+InnoculumVolume;
            obj.InitCells = Innoculum;
            obj.GrowthRate = Growth;
            obj.CellProps = Props;
            obj.FinalTime = 0;
            obj.wInit = Innoculum;
            obj.mutNum = 0;
            obj.maxGrowth = maxGrowth;
            obj.OD = OD;
            obj.cellsAtOD = obj.OD2Cells*obj.Volume*obj.OD;


            Div = mutateDivisions();
```

```
            mutateTime = zeroExp(obj.wInit, obj.GrowthRate,
(sum(obj.wInit)+Div));



            timeTilPassage = zeroExp(obj.wInit, obj.GrowthRate,
obj.cellsAtOD);



            while mutateTime < timeTilPassage


                obj.wInit = grow(obj.wInit, obj.GrowthRate,
mutateTime);
                obj.FinalTime = obj.FinalTime + mutateTime;



                Cell = cell2mutate(obj.wInit);



                arr = mutateCells(obj.GrowthRate(Cell),
obj.CellProps{Cell}, obj.maxGrowth);
                growth = arr{1};
                props = arr{2};



                obj.wInit = [obj.wInit 1];
```

```
                obj.GrowthRate = [obj.GrowthRate growth];

                obj.CellProps{length(obj.GrowthRate)} = props;




                    Div = mutateDivisions();



                    mutateTime = zeroExp(obj.wInit, obj.GrowthRate,
(sum(obj.wInit)+Div));



                    timeTilPassage = zeroExp(obj.wInit,
obj.GrowthRate, obj.cellsAtOD);

            end



            obj.FinalCells = grow(obj.wInit, obj.GrowthRate,
timeTilPassage);



            obj.FinalTime = obj.FinalTime + timeTilPassage;



            obj.mutNum = length(obj.FinalCells) -
length(obj.InitCells);
```

```
            obj.apparentGrowth =
log(sum(obj.FinalCells)/sum(obj.InitCells))/obj.FinalTime;
        end
        function arr = inoculum(obj, iVol)









            totalCells = sum(obj.FinalCells/obj.Volume*iVol);



            prob = obj.FinalCells/sum(obj.FinalCells);



            expected = prob.*totalCells;




            cells = expected;
            for i=1:1:length(cells)
                cells(i) = normrnd(expected(i), sqrt(expected(i)));
            end
            cells = round(cells);
            growth = obj.GrowthRate;
            props = obj.CellProps;
```

```
            i = 1;
            while i <= length(cells)
                if cells(i) <= 0
                    cells(i) = [];
                    growth(i) = [];
                    props(i) = [];
                    i=i-1;
                end
                i=i+1;
            end


            arr{1} = cells;
            arr{2} = growth;
            arr{3} = props;
        end
    end
end
*****************
* ALEsim\grow.m *
*****************


function cells = grow(C, u, t)
```

```
    cells = C.*exp(u*t);
end
************************
* ALEsim\mutateCells.m *
************************



function arr = mutateCells(growth, props, maxGrowth)









    growth = growth + normrnd(0.1,0.1);

    if growth > maxGrowth
        growth = maxGrowth;
    end

    arr{1} = growth;
    arr{2} = props;
end
****************************
* ALEsim\mutateDivisions.m *
****************************
```

```
function div = mutateDivisions()


    prob = 10^-8.8;


    Dmax = 1/prob*12;
    D = round(1:Dmax/100:Dmax);
    pdf = binopdf(1, D, prob);



    div = randpdf(pdf, D, [1,1]);
end
********************
* ALEsim\normrnd.m *
********************


function r = normrnd(mu,sigma,m,n);
```

```
if nargin < 2,
    error('Requires at least two input arguments.');
end

if nargin == 2
    [errorcode rows columns] = rndcheck(2,2,mu,sigma);
end

if nargin == 3
    [errorcode rows columns] = rndcheck(3,2,mu,sigma,m);
end

if nargin == 4
    [errorcode rows columns] = rndcheck(4,2,mu,sigma,m,n);
end

if errorcode > 0
    error('Size information is inconsistent.');
end



r = zeros(rows, columns);

r = randn(rows,columns) .* sigma + mu;
```

```
if any(any(sigma <= 0));
    if prod(size(sigma) == 1)
        tmp = NaN;
        r = tmp(ones(rows,columns));
    else
        k = find(sigma <= 0);
        tmp = NaN;
        r(k) = tmp(ones(size(k)));
    end
end


function [errorcode, rows, columns] =
rndcheck(nargs,nparms,arg1,arg2,arg3,arg4,arg5)









sizeinfo = nargs - nparms;
errorcode = 0;


if nparms == 3
    [r1 c1] = size(arg1);
    [r2 c2] = size(arg2);
    [r3 c3] = size(arg3);
```

```
end

if nparms == 2
    [r1 c1] = size(arg1);
    [r2 c2] = size(arg2);
end

if sizeinfo == 0
    if nparms == 1
        [rows columns] = size(arg1);
    end

    if nparms == 2
        scalararg1 = (prod(size(arg1)) == 1);
        scalararg2 = (prod(size(arg2)) == 1);
        if ~scalararg1 & ~scalararg2
            if r1 ~= r2 | c1 ~= c2
                errorcode = 1;
                return;
            end
        end
        if ~scalararg1
            [rows columns] = size(arg1);
        elseif ~scalararg2
            [rows columns] = size(arg2);
        else
            [rows columns] = size(arg1);
```

```
    end
end

if nparms == 3
    scalararg1 = (prod(size(arg1)) == 1);
    scalararg2 = (prod(size(arg2)) == 1);
    scalararg3 = (prod(size(arg3)) == 1);

    if ~scalararg1 & ~scalararg2
        if r1 ~= r2 | c1 ~= c2
            errorcode = 1;
            return;
        end
    end

    if ~scalararg1 & ~scalararg3
        if r1 ~= r3 | c1 ~= c3
            errorcode = 1;
            return;
        end
    end

    if ~scalararg3 & ~scalararg2
        if r3 ~= r2 | c3 ~= c2
            errorcode = 1;
            return;
        end
```

```
        end
            if ~scalararg1
                [rows columns] = size(arg1);
            elseif ~scalararg2
            [rows columns] = size(arg2);
            else
                [rows columns] = size(arg3);
            end
    end
end


if sizeinfo == 1
    scalararg1 = (prod(size(arg1)) == 1);
    if nparms == 1
        if prod(size(arg2)) ~= 2
            errorcode = 2;
            return;
        end
        if  ~scalararg1 & arg2 ~= size(arg1)
            errorcode = 3;
            return;
        end
        if (arg2(1) < 0 | arg2(2) < 0 | arg2(1) ~= round(arg2(1)) |
arg2(2) ~= round(arg2(2))),
            errorcode = 4;
            return;
        end
```

```
        rows    = arg2(1);
        columns = arg2(2);
    end


    if nparms == 2
        if prod(size(arg3)) ~= 2
            errorcode = 2;
            return;
        end
        scalararg2 = (prod(size(arg2)) == 1);
        if ~scalararg1 & ~scalararg2
            if r1 ~= r2 | c1 ~= c2
                errorcode = 1;
                return;
            end
        end
        if (arg3(1) < 0 | arg3(2) < 0 | arg3(1) ~= round(arg3(1)) |
arg3(2) ~= round(arg3(2))),
            errorcode = 4;
            return;
        end
        if ~scalararg1
            if any(arg3 ~= size(arg1))
                errorcode = 3;
                return;
            end
            [rows columns] = size(arg1);
```

```
        elseif ~scalararg2
            if any(arg3 ~= size(arg2))
                errorcode = 3;
                return;
            end
            [rows columns] = size(arg2);
        else
            rows    = arg3(1);
            columns = arg3(2);
        end
    end


    if nparms == 3
        if prod(size(arg4)) ~= 2
            errorcode = 2;
            return;
        end
        scalararg1 = (prod(size(arg1)) == 1);
        scalararg2 = (prod(size(arg2)) == 1);
        scalararg3 = (prod(size(arg3)) == 1);


        if (arg4(1) < 0 | arg4(2) < 0 | arg4(1) ~= round(arg4(1)) |
arg4(2) ~= round(arg4(2))),
            errorcode = 4;
            return;
        end
```

```
if ~scalararg1 & ~scalararg2
    if r1 ~= r2 | c1 ~= c2
        errorcode = 1;
        return;
    end
end

if ~scalararg1 & ~scalararg3
    if r1 ~= r3 | c1 ~= c3
        errorcode = 1;
        return;
    end
end

if ~scalararg3 & ~scalararg2
    if r3 ~= r2 | c3 ~= c2
        errorcode = 1;
        return;
    end
end
if ~scalararg1
    if any(arg4 ~= size(arg1))
        errorcode = 3;
        return;
    end
    [rows columns] = size(arg1);
elseif ~scalararg2
```

```
            if any(arg4 ~= size(arg2))
                errorcode = 3;
                return;
            end
            [rows columns] = size(arg2);
        elseif ~scalararg3
            if any(arg4 ~= size(arg3))
                errorcode = 3;
                return;
            end
            [rows columns] = size(arg3);
        else
            rows    = arg4(1);
            columns = arg4(2);
        end
    end
end

if sizeinfo == 2
    if nparms == 1
        scalararg1 = (prod(size(arg1)) == 1);
        if ~scalararg1
            [rows columns] = size(arg1);
            if rows ~= arg2 | columns ~= arg3
                errorcode = 3;
                return;
            end
```

```
        end
    if (arg2 < 0 | arg3 < 0 | arg2 ~= round(arg2) | arg3 ~=
round(arg3)),
        errorcode = 4;
        return;
    end
        rows = arg2;
        columns = arg3;
    end


    if nparms == 2
        scalararg1 = (prod(size(arg1)) == 1);
        scalararg2 = (prod(size(arg2)) == 1);
        if ~scalararg1 & ~scalararg2
            if r1 ~= r2 | c1 ~= c2
                errorcode = 1;
                return;
            end
        end
        if ~scalararg1
            [rows columns] = size(arg1);
            if rows ~= arg3 | columns ~= arg4
                errorcode = 3;
                return;
            end
        elseif ~scalararg2
            [rows columns] = size(arg2);
```

```
            if rows ~= arg3 | columns ~= arg4
                errorcode = 3;
                return;
            end
        else
            if (arg3 < 0 | arg4 < 0 | arg3 ~= round(arg3) | arg4 ~=
round(arg4)),
                errorcode = 4;
                return;
            end
            rows = arg3;
            columns = arg4;
        end
    end

    if nparms == 3
        scalararg1 = (prod(size(arg1)) == 1);
        scalararg2 = (prod(size(arg2)) == 1);
        scalararg3 = (prod(size(arg3)) == 1);

        if ~scalararg1 & ~scalararg2
            if r1 ~= r2 | c1 ~= c2
                errorcode = 1;
                return;
            end
        end
```

```
if ~scalararg1 & ~scalararg3
    if r1 ~= r3 | c1 ~= c3
        errorcode = 1;
        return;
    end
end

if ~scalararg3 & ~scalararg2
    if r3 ~= r2 | c3 ~= c2
        errorcode = 1;
        return;
    end
end

if ~scalararg1
    [rows columns] = size(arg1);
    if rows ~= arg4 | columns ~= arg5
        errorcode = 3;
        return;
    end
elseif ~scalararg2
    [rows columns] = size(arg2);
    if rows ~= arg4 | columns ~= arg5
        errorcode = 3;
        return;
    end
elseif ~scalararg3
```

```
            [rows columns] = size(arg3);
            if rows ~= arg4 | columns ~= arg5
                errorcode = 3;
                return;
            end
        else
            if (arg4 < 0 | arg5 < 0 | arg4 ~= round(arg4) | arg5 ~=
round(arg5)),
                errorcode = 4;
                return;
            end
            rows    = arg4;
            columns = arg5;
        end
    end
end
********************
* ALEsim\parsave.m *
********************



function parsave(fname,data)


var_name=genvarname(inputname(2));
eval([var_name '=data'])


try
```

```
    save(fname,var_name,'-append')
catch
    save(fname,var_name)
end
********************
* ALEsim\randpdf.m *
********************



function x=randpdf(p,px,dim)
```

```
error(nargchk(3, 3, nargin))


px=px(:);
p=p(:)./trapz(px,p(:));



pxi=[linspace(min(px),max(px),10000)]';
pi=interp1(px,p,pxi,'linear');


cdfp = cumtrapz(pxi,pi);


ind=[true; not(diff(cdfp)==0)];


cdfp=cdfp(ind);
pi=pi(ind);
pxi=pxi(ind);


uniformDistNum=rand(dim);
```

```
userDistNum=interp1(cdfp,pxi,uniformDistNum(:)',linear');



if nargout==0
    subplot(3,4,[1 2 5 6])
    [n,xout]=hist(userDistNum,50);
    n=n./sum(n)./(xout(2)-xout(1));
    bar(xout,n)
    hold on
    plot(pxi, pi./trapz(pxi,pi),'r')
    hold off
    legend('pdf from generated numbers','input pdf')

    subplot(3,4,[3 4 7 8])
    plot(pxi, cdfp,'g')
    ylim([0 1])
    legend('cdf from input pdf')

    subplot(3,4,[9:12])
    plot(userDistNum)
    legend('generated numbers')
else
    x=reshape(userDistNum,dim);
end
********************
```

```
* ALEsim\zeroExp.m *
********************


function [guess val] = zeroExp(C, u, OD)




guess = 20;
while func(C, u, OD, guess) < 0
    guess = guess + 50;
end
    err = 1000000;
    val = func(C, u, OD, guess);
    while abs(val) > err
        guess = guess - val/dfunc(C, u, guess);
        val = func(C, u, OD, guess);
    end
end

function val = func(C, u, OD, t)
    val = (sum(C.*exp(u.*t))-OD);
end

function val = dfunc(C, u, t)
    val = sum(C.*u.*exp(u.*t));
```

```
end
```