Matlab script for crystal growth contour plots

```
%% Input: Crystal Growth Contour Plot

%Temperature in Kelvin
T = 300;

%number of atoms in first step after nucleation
m_i = 3;

%number of atoms in step fragment
m = 10;

%Facet index (do not include brackets. Write 111 or 100)
facet = 111;

%Chloride coverage (between 0 and 1)
Cl = 0.0;

%% Input: Activation Energies, Vib. Frequencies

%{111} With Chloride EAM
%Activation Energies in eV
E_LLc111 = 0.080;
E_LSc111 = 0.075;
E_LKc111 = 0.052;
E_SSc111 = 0.243;
E_SKc111 = 0.223;
E_KSc111 = 0.356;
E_SLc111 = 0.320;
E_KLc111 = 0.282;
E_USc111 = 0.203;
E_UKc111 = 0.142;
d_hkl_111 = 0.353;

%{111} Without Chloride EAM
%Activation Energies in eV
E_LLp111 = 0.102;
E_LSp111 = 0.096;
E_LKp111 = 0.067;
E_SSp111 = 0.311;
E_SKp111 = 0.286;
E_KSp111 = 0.456;
E_SLp111 = 0.410;
E_KLp111 = 0.420;
E_USp111 = 0.260;
E_UKp111 = 0.181;

%{100} With Chloride EAM
%Activation Energies in eV
E_LLc100 = 0.383;
E_LSc100 = 0.361;
E_LKc100 = 0.252;
E_SSc100 = 0.236;
E_SKc100 = 0.217;
E_KSc100 = 0.355;
```

```
E_SLc100 = 0.533;
E_KLc100 = 0.468;
E_USc100 = 0.413;
E_UKc100 = 0.288;
d_hkl_100 = 0.408;

%{100} Without Chloride EAM
%Activation Energies in eV
E_LLp100 = 0.490;
E_LSp100 = 0.462;
E_LKp100 = 0.322;
E_SSp100 = 0.302;
E_SKp100 = 0.278;
E_KSp100 = 0.455;
E_SLp100 = 0.682;
E_KLp100 = 0.698;
E_USp100 = 0.528;
E_UKp100 = 0.368;

%Vibrational Frequencies in 1/s
v_LL = 1.76E+12;
v_LS = 1.42E+12;
v_LK = 1.43E+12;
v_SS = 1.41E+12;
v_SK = 1.45E+12;
v_KS = 1.45E+12;
v_SL = 1.91E+12;
v_KL = 1.75E+12;
v_US = 1.42E+12;
v_UK = 1.43E+12;

%% Activation Energies

%Facet-dependent activation energies
if facet == 111
    E_LLc = E_LLc111;
    E_LSc = E_LSc111;
    E_LKc = E_LKc111;
    E_SSc = E_SSc111;
    E_SKc = E_SKc111;
    E_KSc = E_KSc111;
    E_SLc = E_SLc111;
    E_KLc = E_KLc111;
    E_USc = E_USc111;
    E_UKc = E_UKc111;
    E_LLp = E_LLp111;
    E_LSp = E_LSp111;
    E_LKp = E_LKp111;
    E_SSp = E_SSp111;
    E_SKp = E_SKp111;
    E_KSp = E_KSp111;
    E_SLp = E_SLp111;
    E_KLp = E_KLp111;
    E_USp = E_USp111;
    E_UKp = E_UKp111;
else
    E_LLc = E_LLc100;
    E_LSc = E_LSc100;
    E_LKc = E_LKc100;
    E_SSc = E_SSc100;
    E_SKc = E_SKc100;
    E_KSc = E_KSc100;
    E_SLc = E_SLc100;
    E_KLc = E_KLc100;
    E_USc = E_USc100;
    E_UKc = E_UKc100;
    E_LLp = E_LLp100;
    E_LSp = E_LSp100;
    E_LKp = E_LKp100;
    E_SSp = E_SSp100;
    E_SKp = E_SKp100;
    E_KSp = E_KSp100;
    E_SLp = E_SLp100;
    E_KLp = E_KLp100;
    E_USp = E_USp100;
    E_UKp = E_UKp100;
end

E_LL = Cl*E_LLc+(1-Cl)*E_LLp;
E_LS = Cl*E_LSc+(1-Cl)*E_LSp;
E_LK = Cl*E_LKc+(1-Cl)*E_LKp;
E_SS = Cl*E_SSc+(1-Cl)*E_SSp;
E_SK = Cl*E_SKc+(1-Cl)*E_SKp;
E_KS = Cl*E_KSc+(1-Cl)*E_KSp;
E_SL = Cl*E_SLc+(1-Cl)*E_SLp;
E_KL = Cl*E_KLc+(1-Cl)*E_KLp;
E_US = Cl*E_USc+(1-Cl)*E_USp;
E_UK = Cl*E_UKc+(1-Cl)*E_UKp;

%% Input: Contour Plot Display

interval = 350;

%a = min flux, b = max flux
a = 10^0;
b = 10^15;

%f = min Nuc. Rate, g = Nuc. Rate
f = 10^0;
g = 10^15;

%nc = number of contours.
nc = 25;

%% Data: Diffusion Rates and Physical Constants
format longG

%Boltzmann's Constant in eV/K
k_B = 8.6173324E-5;

%Thermal Energy
kT = k_B*T;
```

```
%Monolayer Height
if facet == 111
    d_hkl = d_hkl_111;
else
    d_hkl = d_hkl_100;
end

d_hkl;

%% Date: Diffusion Rates

preR_LL = v_LL*exp(-E_LL/kT);
preR_LS = v_LS*exp(-E_LS/kT);
preR_LK = v_LK*exp(-E_LK/kT);
preR_SS = v_SS*exp(-E_SS/kT);
preR_SK = v_SK*exp(-E_SK/kT);
preR_KS = v_KS*exp(-E_KS/kT);
preR_SL = v_SL*exp(-E_SL/kT);
preR_KL = v_KL*exp(-E_KL/kT);
preR_US = v_US*exp(-E_US/kT);
preR_UK = v_UK*exp(-E_UK/kT);

%Facet-dependent diffusion rates
if facet == 111
    pij_LL = (1/3)*preR_LL;
    pij_LS = preR_LS/(preR_LS+2*preR_LL);
    pij_LK = preR_LK/(preR_LK+2*preR_LL);
    pij_SS = (1/2);
    pij_SK = preR_SK/(preR_SK+preR_SS);
    pij_KS = 0;
    pij_SL = preR_SL/(preR_SL+2*preR_SS);
    pij_KL = 1;
    pij_US = preR_US/(preR_US+2*preR_LL);
    pij_UK = preR_UK/(preR_UK+2*preR_LL);
else
    pij_LL = (1/4)*preR_LL;
    pij_LS = preR_LS/(preR_LS+3*preR_LL);
    pij_LK = preR_LK/(preR_LK+preR_LS+2*preR_LL);
    pij_SS = (1/2);
    pij_SK = preR_SK/(preR_SK+preR_SS);
    pij_KS = 0;
    pij_SL = preR_SL/(preR_SL+2*preR_SS);
    pij_KL = 1;
    pij_US = preR_US/(preR_US+3*preR_LL);
    pij_UK = preR_UK/(preR_UK+3*preR_LL);
end

R_LL = preR_LL*pij_LL;
R_LS = preR_LS*pij_LS;
R_LK = preR_LK*pij_LK;
R_SS = preR_SS*pij_SS;
R_SK = preR_SK*pij_SK;
R_KS = preR_KS*pij_KS;
R_SL = preR_SL*pij_SL;
R_KL = preR_KL*pij_KL;
R_US = preR_US*pij_US;
R_UK = preR_UK*pij_UK;
```

```
%Flux is the flux of growth units to the step front in atoms nm^-1 s^-1

syms Flux

D_Au = 0.288;
flux_step = D_Au*Flux;

%E_T0 is the expectation time for growth units to arrive at the step front
E_T0 = flux_step^(-1);

%% Data: Arrival Time

loga = log10(a);
logb = log10(b);
logf = log10(f);
logg = log10(g);

FL = ones(1,interval+1);

for j = (1:interval+1)
    k = 10^(loga+(j-1)*(logb-loga)/interval);
    FL(j) = subs(flux_step,Flux,k);

end

MATFL = ones(m,interval+1);

for l = (1:m)
    MATFL(l,:) = l*FL;

end

MATFL;

ET0_L = 1./MATFL;
ET0_LU = ET0_L/2;

%% Data: Incorporation Generator

Markov_incorporation = ones(m,1);
MarkovLU_incorporation = ones(m,1);

for n = (3:m)

    ProbjL = 1/n;
    ProbjLU = 1/(2*n);

    timejL = 1/R_LS;
    timejU = 1/R_US;

    %Q is the infinitesimal generator
    pkA = ones(n-1);
    pkB = eye(n-1);
    pkC = tril(pkA,-2);
    pkD = triu(pkA,2);
    pkE = pkA-pkB-pkC-pkD;
    pkF = -R_SS*pkE;
    pkG = pkB*(2*R_SS);

    Q = pkF+pkG;
    Q(1,1) = R_SK+R_SS;
    Q(n-1,n-1) = R_SS;

    Qone = ones(n-1,1);

    E_partMarkov = Q\Qone;

    T1T2mark = ones(n-1,1);

    for q = (1:n-1)
        T1T2mark(q) = ProbjL*(timejL+E_partMarkov(q));
    end

    T1T2mark0 = ProbjL*(1/R_LK);

    T1T2_Markov = [T1T2mark0;T1T2mark];

    T1T2one = ones(1,n);

    T1T2L = T1T2one*T1T2_Markov;

    Markov_incorporation(n) = T1T2L;

    E_partMarkovLU = Q\Qone;

    T1T2markLU = ones(n-1,1);

    for h = (1:n-1)
        T1T2markLU(h) = ProbjLU*(timejL+timejU+2*E_partMarkovLU(h));
    end

    T1T2LUmark0 = ProbjLU*(1/R_LK+1/R_UK);

    T1T2LU_Markov = [T1T2LUmark0;T1T2markLU];

    T1T2LUone = ones(1,n);

    T1T2LU = T1T2LUone*T1T2LU_Markov;

    MarkovLU_incorporation(n) = T1T2LU;
end

Markov_incorporation(1) = 1/R_LK;

Markov_incorporation(2) = (1/2)*(1/R_LK)+(1/2)*((1/R_LS)+(1/R_SK));
```

```
MarkovLU_incorporation(1) = (1/2)*((1/R_LK)+(1/R_UK));

MarkovLU_incorporation(2) =
(1/4)*((1/R_LK)+(1/R_UK))+(1/4)*((1/R_LS)+2*(1/R_SK)+(1/R_US));

Markov_incorporation;
MarkovLU_incorporation;


%% Data: Expected Time to Complete Step by CTMC

Time_MarkovL = ones(m,interval+1);
Time_MarkovLU = ones(m,interval+1);

for v = (1:m)
    Time_MarkovL(v,:) = ET0_L(v,:)+Markov_incorporation(v);
    Time_MarkovLU(v,:) = ET0_LU(v,:)+MarkovLU_incorporation(v);
end


Time_MarkovL;
Time_MarkovLU;

CORXNL= ones(m,interval+1);
CORXNLU= ones(m,interval+1);

for v = (1:m)
    CORXNL(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovL(v,:));
    CORXNLU(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovLU(v,:));
end


T_markL = ones(m,interval+1);
T_markLU = ones(m,interval+1);

for v = (1:m)
    T_markL(v,:) = CORXNL(v,:).*Time_MarkovL(v,:);
    T_markLU(v,:) = CORXNLU(v,:).*Time_MarkovLU(v,:);
end

numL = numel(T_markL);

for d = (1:numL)
    if T_markL(d)<0
        T_markL(d)=NaN;
    end
    if T_markLU(d)<0
        T_markLU(d)=NaN;
    end
end

T_markL;
T_markLU;
```

```
Time_Markov_L = cumsum(T_markL);
Time_Markov_LU = cumsum(T_markLU);

%% Data: Expected Time for 1D Nucleation

T1DL = ET0_L + 1/R_LS;
T1DLU = ET0_LU + 0.5/R_LS + 0.5/R_US;

CTMCL = ones(1+m-m_i,interval+1);
CTMCLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_M1 = ones(p,interval+1);
    E_T_M2 = ones(p,interval+1);

    for i = (1:p)
        E_T_M1(i,:) = (1/p)*Time_Markov_L(max(i-1,p-i),:);
        E_T_M2(i,:) = (1/p)*Time_Markov_LU(max(i-1,p-i),:);
    end

    jumpinL = T1DL(p,:);
    jumpinLU = T1DLU(p,:);

    E_T_ML = sum(E_T_M1);
    E_T_MLU = sum(E_T_M2);

    EML = E_T_ML + jumpinL;
    EMLU = E_T_MLU + jumpinLU;

    CTMCL(1+p-m_i,:) = EML;
    CTMCLU(1+p-m_i,:) = EMLU;

end

CTMCL;
CTMCLU;

%% Plot: Log Expectation Time versus Step Length versus Log Flux


set(gcf,'Colormap',jet)

contourf(log10(FL),(m_i:m),log10(CTMCL),25)

set(gca,'FontSize',28)

title('Log(Step Time) vs. Log(Flux)','Fontsize',28)

xhandle=xlabel('Log(Flux)');
yhandle=ylabel('n');

set(xhandle,'Fontsize',28)

set(xhandle,'Fontname','Helvetica')
set(yhandle,'Fontsize',28)

set(yhandle,'Fontname','Helvetica')

colormap(jet)
 cb=colorbar;

set(cb,'fontsize',28, 'Fontname','Helvetica');

%% Data: Kink Propagation

kinkL = ones(1+m-m_i,interval+1);
kinkLU = ones(1+m-m_i,interval+1);
kinkarrival = ones(m,interval+1);
kinkarrivalLU = ones(m,interval+1);

for x = (1:m)
    kinkarrival(x,:) = 1./FL;
    kinkarrivalLU(x,:) = 1./(2*FL);
end

kinkcum = cumsum(kinkarrival);
kinkcumLU = cumsum(kinkarrivalLU);

for y = (1:m)
    kinkL(y,:) = y/R_LK+kinkcum(y,:);
    kinkLU(y,:) = y*(0.5/R_LK+0.5/R_UK)+kinkcumLU(y,:);
end
kinkL;
kinkLU;


KPL = ones(1+m-m_i,interval+1);
KPLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_KPL = ones(p,interval+1);
    E_T_KPLU = ones(p,interval+1);

    for i = (1:p)
        E_T_KPL(i,:) = (1/p)*kinkL(max(i-1,p-i),:);
        E_T_KPLU(i,:) = (1/p)*kinkLU(max(i-1,p-i),:);
    end

    KPL1 = sum(E_T_KPL);
    KPLU1 = sum(E_T_KPLU);

    tkL = T1DL(p,:);
    tkLU = T1DLU(p,:);

    KPL(1+p-m_i,:) = KPL1+tkL;
    KPLU(1+p-m_i,:) = KPLU1+tkLU;

end
```

```
KPL;
KPLU;

%% Data: Filters

Markov_Filter = ones(m,interval+1);
Markov_Filter_LU = ones(m,interval+1);
Kink_Filter = ones(m,interval+1);
Kink_Filter_LU = ones(m,interval+1);

for i = (1:interval+1)
    Markov_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Markov_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Markov_Filter(j)<=0
        Markov_Filter(j) = 0;
    else Markov_Filter(j) = 1;
    end
    if Markov_Filter_LU(j)<=0
        Markov_Filter_LU(j) = 0;
    else Markov_Filter_LU(j) = 1;
    end
end

for i = (1:interval+1)
    Kink_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Kink_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Kink_Filter(j)<=0
        Kink_Filter(j) = 0;
    else Kink_Filter(j) = 1;
    end
    if Kink_Filter_LU(j)<=0
        Kink_Filter_LU(j) = 0;
    else Kink_Filter_LU(j) = 1;
    end
end

Markov_Filter;
Non_Markov_Filter = 1-Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU = 1-Markov_Filter_LU;

%% Data: Time for Direct Step Adsorption

Direct_Step_L = ones(1+m-m_i,interval+1);
Direct_Step_LU = ones(1+m-m_i,interval+1);
premat_step_L = cumsum(ET0_L);
premat_step_LU = cumsum(ET0_LU);

for s = (m_i:m)
```

```
    Direct_Step_L(1+s-m_i,:) = premat_step_L(s,:)+s/R_LS;
    Direct_Step_LU(1+s-m_i,:) = premat_step_LU(s,:)+s/R_LS;
end

Direct_Step_L;
Direct_Step_LU;

%% Data: Minimum Time Direct Incorporation

MinDirectL = min(Direct_Step_L,KPL);
MinDirectLU = min(Direct_Step_LU,KPLU);

%% Collection of Important Matrices and Vectors

% Flux vector. 1 x int+1
FL;

% Expected Time for atom to arrive at step. n x int+1
ET0_L;
ET0_LU;

% Expected Time for 1D nucleation. n x int+1
T1DL;
T1DLU;

% Expected Time for atom to diffuse via CTMC to kink. n x 1
Markov_incorporation;
MarkovLU_incorporation;

% Growth Mechanism Filters
Markov_Filter;
Non_Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU;

% Expected Time to complete step via direct step incorporation. n x int+1
Direct_Step_L;
Direct_Step_LU;


% Expected Time to complete step via direct kink propagation. n x int+1
KPL;
KPLU;

% Expected Time to complete step via CTMC mechanism. n x int+1
CTMCL;
CTMCLU;

% (minimum) Expected Time for direct incorporation. n x int+1
MinDirectL;
MinDirectLU;

%% Data: Filtered Results

MF = ones(1+m-m_i,interval+1);
```

```
MFLU = ones(1+m-m_i,interval+1);

for f = (m_i:m)
    MF(1+f-m_i,:) = Markov_Filter(f,:);
end
MF;
nMF = 1-MF;

for g = (m_i:m)
    MFLU(1+g-m_i,:) = Markov_Filter_LU(g,:);
end
MFLU;
nMFLU = 1-MFLU;

Filtered_CTMCL = MF.*CTMCL;
Filtered_MINDIR = nMF.*MinDirectL;

Filtered_CTMCLU = MFLU.*CTMCLU;
Filtered_MINDIRU = nMFLU.*MinDirectLU;


% Expected times for step completion
TIME_L = Filtered_CTMCL+Filtered_MINDIR;
TIME_LU = Filtered_CTMCLU+Filtered_MINDIRU;

%Time for monolayer spread

T_Spread_L = sum(TIME_L)';
T_Spread_LU = sum(TIME_LU)';

%% Plot: Log Expectation Time versus Log Flux

plot(log10(FL),log10(T_Spread_L),'o')

set(gca,'FontSize',28)

title('Log(Step Time) vs. Log(Flux)','Fontsize',28)

xhandle=xlabel('Log(Flux)');
yhandle=ylabel('Log(Step Time)');

set(xhandle,'Fontsize',28)

set(xhandle,'Fontname','Helvetica')

set(yhandle,'Fontsize',28)

set(yhandle,'Fontname','Helvetica')

hold on

plot(log10(FL),log10(T_Spread_LU),'o','markeredgecolor','r')

hold off
```

```
legend('Lower Terrace Only','Lower and Upper
Terraces','location','southwest')

set(legend,'Fontsize', 16)


%% Data: Crystal Growth Contour Plot

NL = ones(1,interval+1);
NR = ones(1,interval+1);

for j = (1:interval+1)
    k = 10^(logf+(j-1)*(logg-logf)/interval);
    NR(j) = k;
    NL(j) = 1/k;

end

NR;
NL;

Markov_ContourL1 = ones(interval+1,interval+1);
Markov_ContourLU1 = ones(interval+1,interval+1);

for l = (1:interval+1)
    Markov_ContourL1(l,:) = NL+T_Spread_L(l);
    Markov_ContourLU1(l,:) = NL+T_Spread_LU(l);
end

Markov_ContourL1;
Markov_ContourLU1;

Markov_ContourL = Markov_ContourL1/d_hkl;
Markov_ContourLU = Markov_ContourLU1/d_hkl;

%% Plot: Crystal Growth Contour Plot

logy = log10(FL./D_Au);


% for unfilled contour plot use the following:
%contour(log10(NR),logy,log10(1./Markov_ContourL),nc,'linewidth',2)

% for filled contour plot use the following:
contourf(log10(NR),logy,log10(1./Markov_ContourL),nc)

set(gca,'FontSize',28)

title('Log(Rhkl)','Fontsize',28)

xhandle=xlabel('Log(Nucleation Rate)');
yhandle=ylabel('Log(Flux)');
```

```
set(xhandle,'Fontsize',28)

set(xhandle,'Fontname','Helvetica')

set(yhandle,'Fontsize',28)

set(yhandle,'Fontname','Helvetica')
set(gca,'XTick',(0:3:15),'FontSize',28)
set(gca,'YTick',(0:3:15),'FontSize',28)

colormap(jet)
 cb=colorbar;

set(cb,'fontsize',0.1, 'Fontname','Helvetica');

set(gca,'CLim',[0,8])
set(cb,'YTick',(0:1:8),'FontSize',28)
maxv = max(max(log10(1./Markov_ContourL)));
minv = min(min(log10(1./Markov_ContourL)));
range = (maxv-minv)/(nc+1);

v = (minv:range:maxv-range);

hold on

contour(log10(NR),logy,log10(1./Markov_ContourL),maxv-
range,'k','linewidth',3.5)

hold off

ratematrix = log10(1./Markov_ContourL);
size(ratematrix);

ndtrans1 = ones(interval+1,interval+1);
for a = (1:interval)
    ndtrans1(:,a) = ratematrix(:,a+1)-ratematrix(:,a);
end

ndtrans1(:,interval+1) = zeros(interval+1,1);

ndtrans2 = ones(interval+1,interval+1);
for a = (1:interval)
    ndtrans2(a,:) = ratematrix(a+1,:)-ratematrix(a,:);
end

ndtrans2(interval+1,:) = zeros(1,interval+1);

ndtransA = ndtrans1 - ndtrans2;
ndtransB = ndtrans2 - ndtrans1;
ndtransC = ndtransA.*ndtransB;

[M,I] = max(ndtransC);

I;
```

```
testmat = zeros(interval+1);

for b = (1:interval+1)
    testmat(I(b),b) = 1;
end

testmat;

testq = maxv-range;

maxmat = testmat.*ratematrix;

[row,col] = find(maxmat>=testq);

lrc = length(row);

maxmat2 = maxmat;

for t = (1:lrc)
    maxmat2(row(t),col(t)) = 0;
end

maxmat3 = maxmat2;

for s = (1:(interval+1)^2)
    if maxmat2(s) == 0;
        maxmat3(s) = NaN;
    end
end

maxmat3;

testline = max(maxmat3);

N = isnan(testline);

NN = 1-N;

LN = nnz(NN);

rrr = ones(LN,1);
ccc = ones(LN,1);

for r = (1:LN)
    [rrr(r),ccc(r)] = find(maxmat3 == testline(r));
end

rrr;
ccc;

xxx = log10(NR);

yyy = logy;
```

```
xvec = ones(1,LN);

for x = (1:LN)
    xvec(x) = xxx(x);
end

xvec;

yvec = ones(1,LN);

for y = (1:LN)
    yvec(y) = yyy(rrr(y));
end

yvec;

hold on

plot(xvec,yvec,'k','linewidth',3.5)

hold off

J = isnan(ratematrix(:,1));

JJ = 1-J;

LJ = nnz(JJ);

Jinit = interval+2 - LJ;

minline = ones(interval+1,1);

for j = (1:interval+1)
    minline(j) = yyy(Jinit);
end

hold on

plot(log10(NR),minline,'k','linewidth',3.5)

hold off

hold on


%% Plot: Experimental Rate Contour

% z = [log10(experimental growth rate),log10(experimental growth rate)]
z = [2,2];

%%
%contour(log10(NR),logy,log10(1./Markov_ContourL),z,'w','linewidth',3.5)
```

```
hold off

set(gcf,'Color','w')
```

Matlab script for diffusion-limited growth rate distribution

```
%% Input: Diffusion-Limited Simulation

%Temperature in Kelvin
T = 300;

%number of atoms in first step after nucleation
m_i = 3;

%number of atoms in final step of net monolayer
m = 10;

%Facet index (do not include brackets. Write 111 or 100)
facet = 111;

%Chloride coverage (between 0 and 1)
Cl = 0;

%% Monte Carlo
% Variables
mu=534950; % mean
sd=5349.50; % sigma

%% Input: Activation Energies, Vib. Frequencies

%{111} With Chloride EAM
%Activation Energies in eV
E_LLc111 = 0.080;
E_LSc111 = 0.075;
E_LKc111 = 0.052;
E_SSc111 = 0.243;
E_SKc111 = 0.223;
E_KSc111 = 0.356;
E_SLc111 = 0.320;
E_KLc111 = 0.282;
E_USc111 = 0.203;
E_UKc111 = 0.142;
d_hkl_111 = 0.353;

%{111} Without Chloride EAM
%Activation Energies in eV
E_LLp111 = 0.102;
E_LSp111 = 0.096;
E_LKp111 = 0.067;
E_SSp111 = 0.311;
E_SKp111 = 0.286;
E_KSp111 = 0.456;
E_SLp111 = 0.410;
```

```
E_KLp111 = 0.420;
E_USp111 = 0.260;
E_UKp111 = 0.181;

%{100} With Chloride EAM
%Activation Energies in eV
E_LLc100 = 0.383;
E_LSc100 = 0.361;
E_LKc100 = 0.252;
E_SSc100 = 0.236;
E_SKc100 = 0.217;
E_KSc100 = 0.355;
E_SLc100 = 0.533;
E_KLc100 = 0.468;
E_USc100 = 0.413;
E_UKc100 = 0.288;
d_hkl_100 = 0.408;

%{100} Without Chloride EAM
%Activation Energies in eV
E_LLp100 = 0.490;
E_LSp100 = 0.462;
E_LKp100 = 0.322;
E_SSp100 = 0.302;
E_SKp100 = 0.278;
E_KSp100 = 0.455;
E_SLp100 = 0.682;
E_KLp100 = 0.698;
E_USp100 = 0.528;
E_UKp100 = 0.368;

%Vibrational Frequencies in 1/s
v_LL = 1.76E+12;
v_LS = 1.42E+12;
v_LK = 1.43E+12;
v_SS = 1.41E+12;
v_SK = 1.45E+12;
v_KS = 1.45E+12;
v_SL = 1.91E+12;
v_KL = 1.75E+12;
v_US = 1.42E+12;
v_UK = 1.43E+12;

%% Activation energies

%Facet identity
if facet == 111
    E_LLc = E_LLc111;
    E_LSc = E_LSc111;
    E_LKc = E_LKc111;
    E_SSc = E_SSc111;
    E_SKc = E_SKc111;
    E_KSc = E_KSc111;
    E_SLc = E_SLc111;
    E_KLc = E_KLc111;
    E_USc = E_USc111;
    E_UKc = E_UKc111;
    E_LLp = E_LLp111;
    E_LSp = E_LSp111;
    E_LKp = E_LKp111;
    E_SSp = E_SSp111;
    E_SKp = E_SKp111;
    E_KSp = E_KSp111;
    E_SLp = E_SLp111;
    E_KLp = E_KLp111;
    E_USp = E_USp111;
    E_UKp = E_UKp111;
else
    E_LLc = E_LLc100;
    E_LSc = E_LSc100;
    E_LKc = E_LKc100;
    E_SSc = E_SSc100;
    E_SKc = E_SKc100;
    E_KSc = E_KSc100;
    E_SLc = E_SLc100;
    E_KLc = E_KLc100;
    E_USc = E_USc100;
    E_UKc = E_UKc100;
    E_LLp = E_LLp100;
    E_LSp = E_LSp100;
    E_LKp = E_LKp100;
    E_SSp = E_SSp100;
    E_SKp = E_SKp100;
    E_KSp = E_KSp100;
    E_SLp = E_SLp100;
    E_KLp = E_KLp100;
    E_USp = E_USp100;
    E_UKp = E_UKp100;
end

E_LL = Cl*E_LLc+(1-Cl)*E_LLp;
E_LS = Cl*E_LSc+(1-Cl)*E_LSp;
E_LK = Cl*E_LKc+(1-Cl)*E_LKp;
E_SS = Cl*E_SSc+(1-Cl)*E_SSp;
E_SK = Cl*E_SKc+(1-Cl)*E_SKp;
E_KS = Cl*E_KSc+(1-Cl)*E_KSp;
E_SL = Cl*E_SLc+(1-Cl)*E_SLp;
E_KL = Cl*E_KLc+(1-Cl)*E_KLp;
E_US = Cl*E_USc+(1-Cl)*E_USp;
E_UK = Cl*E_UKc+(1-Cl)*E_UKp;

%% Input: Growth Phase Diagram Display

interval = 400;

%a = min flux, b = max flux
a = mu-5*sd;
b = mu+5*sd;

%f = min Nuc. Rate, g = Nuc. Rate
f = 10^0;
g = 10^15;

%nc = number of contours.
```

```
nc = 25;

%% Data: Diffusion Rates and Physical Constants
format longG

%Boltzmann's Constant in eV/K
k_B = 8.6173324E-5;

%Thermal Energy
kT = k_B*T;

%Monolayer Height
if facet == 111
    d_hkl = d_hkl_111;
else
    d_hkl = d_hkl_100;
end

d_hkl;

%Diffusion Rates in Jumps/s
R_LL = v_LL*exp(-E_LL/kT);
R_LS = v_LS*exp(-E_LS/kT);
R_LK = v_LK*exp(-E_LK/kT);
R_SS = v_SS*exp(-E_SS/kT);
R_SK = v_SK*exp(-E_SK/kT);
R_KS = v_KS*exp(-E_KS/kT);
R_SL = v_SL*exp(-E_SL/kT);
R_KL = v_KL*exp(-E_KL/kT);
R_US = v_US*exp(-E_US/kT);
R_UK = v_UK*exp(-E_UK/kT);

%Flux is the flux of growth units to the step front in atoms nm^-1 s^-1

D_Au = 0.288;

%% Data: Arrival Time

%Construct grid.
loga = log10(a);
logb = log10(b);
logf = log10(f);
logg = log10(g);

FL=linspace(0.288*(mu-5*sd),0.288*(mu+5*sd),interval+1);

MATFL = ones(m,interval+1);

for l = (1:m)
    MATFL(l,:) = l*FL;

end

MATFL;
```

```
ET0_L = 1./MATFL;
ET0_LU = ET0_L/2;

%% Data: Incorporation Generator

Markov_incorporation = ones(m,1);
MarkovLU_incorporation = ones(m,1);

for n = (3:m)

    ProbjL = 1/n;
    ProbjLU = 1/(2*n);

    timejL = 1/R_LS;
    timejU = 1/R_US;

    %Q is the infinitesimal generator

    pkA = ones(n-1);
    pkB = eye(n-1);
    pkC = tril(pkA,-2);
    pkD = triu(pkA,2);
    pkE = pkA-pkB-pkC-pkD;
    pkF = -R_SS*pkE;
    pkG = pkB*(2*R_SS);

    Q = pkF+pkG;
    Q(1,1) = R_SK+R_SS;
    Q(n-1,n-1) = R_SS;

    Qone = ones(n-1,1);

    E_partMarkov = Q\Qone;

    T1T2mark = ones(n-1,1);

    for q = (1:n-1)
        T1T2mark(q) = ProbjL*(timejL+E_partMarkov(q));
    end

    T1T2mark0 = ProbjL*(1/R_LK);

    T1T2_Markov = [T1T2mark0;T1T2mark];

    T1T2one = ones(1,n);

    T1T2L = T1T2one*T1T2_Markov;

    Markov_incorporation(n) = T1T2L;

    E_partMarkovLU = Q\Qone;

    T1T2markLU = ones(n-1,1);
    for h = (1:n-1)
        T1T2markLU(h) = ProbjLU*(timejL+timejU+2*E_partMarkovLU(h));
    end

    T1T2LUmark0 = ProbjLU*(1/R_LK+1/R_UK);

    T1T2LU_Markov = [T1T2LUmark0;T1T2markLU];

    T1T2LUone = ones(1,n);

    T1T2LU = T1T2LUone*T1T2LU_Markov;

    MarkovLU_incorporation(n) = T1T2LU;
end

Markov_incorporation(1) = 1/R_LK;

Markov_incorporation(2) = (1/2)*(1/R_LK)+(1/2)*((1/R_LS)+(1/R_SK));

MarkovLU_incorporation(1) = (1/2)*((1/R_LK)+(1/R_UK));

MarkovLU_incorporation(2) =
(1/4)*((1/R_LK)+(1/R_UK))+(1/4)*((1/R_LS)+2*(1/R_SK)+(1/R_US));

Markov_incorporation;
MarkovLU_incorporation;

%% Data: Time to Complete Step by CTMC

Time_MarkovL = ones(m,interval+1);
Time_MarkovLU = ones(m,interval+1);

for v = (1:m)
    Time_MarkovL(v,:) = ET0_L(v,:)+Markov_incorporation(v);
    Time_MarkovLU(v,:) = ET0_LU(v,:)+MarkovLU_incorporation(v);
end


Time_MarkovL;
Time_MarkovLU;

CORXNL= ones(m,interval+1);
CORXNLU= ones(m,interval+1);

for v = (1:m)
    CORXNL(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovL(v,:));
    CORXNLU(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovLU(v,:));
end


T_markL = ones(m,interval+1);
T_markLU = ones(m,interval+1);

for v = (1:m)
```

```
    T_markL(v,:) = CORXNL(v,:).*Time_MarkovL(v,:);
    T_markLU(v,:) = CORXNLU(v,:).*Time_MarkovLU(v,:);
end

numL = numel(T_markL);

for d = (1:numL)
    if T_markL(d)<0
        T_markL(d)=NaN;
    end
    if T_markLU(d)<0
        T_markLU(d)=NaN;
    end
end

T_markL;
T_markLU;

Time_Markov_L = cumsum(T_markL);
Time_Markov_LU = cumsum(T_markLU);

%% Data: Expected Time for 1D Nucleation

T1DL = ET0_L + 1/R_LS;
T1DLU = ET0_LU + 0.5/R_LS + 0.5/R_US;

CTMCL = ones(1+m-m_i,interval+1);
CTMCLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_M1 = ones(p,interval+1);
    E_T_M2 = ones(p,interval+1);

    for i = (1:p)
        E_T_M1(i,:) = (1/p)*Time_Markov_L(max(i-1,p-i),:);
        E_T_M2(i,:) = (1/p)*Time_Markov_LU(max(i-1,p-i),:);
    end

    jumpinL = T1DL(p,:);
    jumpinLU = T1DLU(p,:);

    E_T_ML = sum(E_T_M1);
    E_T_MLU = sum(E_T_M2);

    EML = E_T_ML + jumpinL;
    EMLU = E_T_MLU + jumpinLU;

    CTMCL(1+p-m_i,:) = EML;
    CTMCLU(1+p-m_i,:) = EMLU;

end

CTMCL;
CTMCLU;
```

```
%% Data: Kink Propagation

kinkL = ones(1+m-m_i,interval+1);
kinkLU = ones(1+m-m_i,interval+1);
kinkarrival = ones(m,interval+1);
kinkarrivalLU = ones(m,interval+1);

for x = (1:m)
    kinkarrival(x,:) = 1./FL;
    kinkarrivalLU(x,:) = 1./(2*FL);
end

kinkcum = cumsum(kinkarrival);
kinkcumLU = cumsum(kinkarrivalLU);

for y = (1:m)
    kinkL(y,:) = y/R_LK+kinkcum(y,:);
    kinkLU(y,:) = y*(0.5/R_LK+0.5/R_UK)+kinkcumLU(y,:);
end
kinkL;
kinkLU;


KPL = ones(1+m-m_i,interval+1);
KPLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_KPL = ones(p,interval+1);
    E_T_KPLU = ones(p,interval+1);

    for i = (1:p)
        E_T_KPL(i,:) = (1/p)*kinkL(max(i-1,p-i),:);
        E_T_KPLU(i,:) = (1/p)*kinkLU(max(i-1,p-i),:);
    end

    KPL1 = sum(E_T_KPL);
    KPLU1 = sum(E_T_KPLU);

    tkL = T1DL(p,:);
    tkLU = T1DLU(p,:);

    KPL(1+p-m_i,:) = KPL1+tkL;
    KPLU(1+p-m_i,:) = KPLU1+tkLU;

end

KPL;
KPLU;

%% Data: Filters

Markov_Filter = ones(m,interval+1);
Markov_Filter_LU = ones(m,interval+1);
Kink_Filter = ones(m,interval+1);
Kink_Filter_LU = ones(m,interval+1);
```

```
for i = (1:interval+1)
    Markov_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Markov_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Markov_Filter(j)<=0
        Markov_Filter(j) = 0;
    else Markov_Filter(j) = 1;
    end
    if Markov_Filter_LU(j)<=0
        Markov_Filter_LU(j) = 0;
    else Markov_Filter_LU(j) = 1;
    end
end

for i = (1:interval+1)
    Kink_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Kink_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Kink_Filter(j)<=0
        Kink_Filter(j) = 0;
    else Kink_Filter(j) = 1;
    end
    if Kink_Filter_LU(j)<=0
        Kink_Filter_LU(j) = 0;
    else Kink_Filter_LU(j) = 1;
    end
end

Markov_Filter;
Non_Markov_Filter = 1-Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU = 1-Markov_Filter_LU;

%% Data: Time for direct step adsorption

Direct_Step_L = ones(1+m-m_i,interval+1);
Direct_Step_LU = ones(1+m-m_i,interval+1);
premat_step_L = cumsum(ET0_L);
premat_step_LU = cumsum(ET0_LU);

for s = (m_i:m)
    Direct_Step_L(1+s-m_i,:) = premat_step_L(s,:)+s/R_LS;
    Direct_Step_LU(1+s-m_i,:) = premat_step_LU(s,:)+s/R_LS;
end

Direct_Step_L;
Direct_Step_LU;

%% Data: Minimum Time Direct Incorporation

MinDirectL = min(Direct_Step_L,KPL);
```

```
MinDirectLU = min(Direct_Step_LU,KPLU);

%% Collection of Important Matrices and Vectors

% Flux vector. 1 x int+1
FL;

% Expected Time for atom to arrive at step. n x int+1
ET0_L;
ET0_LU;

% Expected Time for 1D nucleation. n x int+1
T1DL;
T1DLU;

% Expected Time for atom to diffuse via CTMC to kink. n x 1
Markov_incorporation;
MarkovLU_incorporation;

% Growth Mechanism Filters
Markov_Filter;
Non_Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU;

% Expected Time to complete step via direct step incorporation. n x int+1
Direct_Step_L;
Direct_Step_LU;


% Expected Time to complete step via direct kink propagation. n x int+1
KPL;
KPLU;

% Expected Time to complete step via CTMC mechanism. n x int+1
CTMCL;
CTMCLU;

% (minimum) Expected Time for direct incorporation. n x int+1
MinDirectL;
MinDirectLU;

%% Data: Filtered Results

MF = ones(1+m-m_i,interval+1);
MFLU = ones(1+m-m_i,interval+1);

for f = (m_i:m)
    MF(1+f-m_i,:) = Markov_Filter(f,:);
end
MF;
nMF = 1-MF;

for g = (m_i:m)
    MFLU(1+g-m_i,:) = Markov_Filter_LU(g,:);
```

```
end
MFLU;
nMFLU = 1-MFLU;

Filtered_CTMCL = MF.*CTMCL;
Filtered_MINDIR = nMF.*MinDirectL;

Filtered_CTMCLU = MFLU.*CTMCLU;
Filtered_MINDIRU = nMFLU.*MinDirectLU;


% Expected times for step completion
TIME_L = Filtered_CTMCL+Filtered_MINDIR;
TIME_LU = Filtered_CTMCLU+Filtered_MINDIRU;

%Time for monolayer spread

T_Spread_L = sum(TIME_L)';
T_Spread_LU = sum(TIME_LU)';

%% Data: Growth Phase Diagram

NL = ones(1,interval+1);
NR = ones(1,interval+1);

for j = (1:interval+1)
    k = 10^(logf+(j-1)*(logg-logf)/interval);
    NR(j) = k;
    NL(j) = 1/k;

end

NR;
NL;

Markov_ContourL1 = ones(interval+1,interval+1);
Markov_ContourLU1 = ones(interval+1,interval+1);

for l = (1:interval+1)
    Markov_ContourL1(l,:) = NL+T_Spread_L(l);
    Markov_ContourLU1(l,:) = NL+T_Spread_LU(l);
end

Markov_ContourL1;
Markov_ContourLU1;

Markov_ContourL = Markov_ContourL1/d_hkl;
Markov_ContourLU = Markov_ContourLU1/d_hkl;

RATE_L = 1./Markov_ContourL;
RATE_LU = 1./Markov_ContourLU;

%% Histograms

N=10^6; % # times
```

```
% Solve for the analytical sampling equation and run N times
x=zeros(length(N));
x2=zeros(length(N));
y=zeros(length(N));
y2=zeros(length(N));
for i=1:N
    r1=rand;
    r2=rand;
    r3=rand;
    r4=rand;
    x(i)=sqrt(-2*(0.288*sd)^2*log(r1))*sin(2*pi*r2);
    x(i)=x(i)+0.288*mu;
    %x2(i)=sqrt(-2*s2^2*log(r3))*sin(2*pi*r4);
    %x2(i)=x2(i)+mu2;
end

bin = histc(x,FL)';
%%
rcol = RATE_L(:,interval+1);

blue = find(RATE_L(:,interval+1)>=100);

numel(blue)

%%
red = sum(isnan(RATE_L(:,interval+1)));

lg = (min(blue)-1)-(max(red)+1)+1;
lb = max(blue)-min(blue)+1;

green = ones(lg,1);

for g = (max(red)+1:min(blue)-1)
    green(g-red)=rcol(g);
end

greenx = ones(lg,1);
for g = (max(red)+1:min(blue)-1)
    greenx(g-red)=bin(g);
end

sumg = sum(greenx);

percentgreen = sumg/1000000; %percent lateral growth

%%
green;

blu = ones(lb,1);

for b = (min(blue):max(blue))
    blu(b-(min(blue))+1)=rcol(b);
end
```

```
blu;

bing = ones(lg,1);

for g = (max(red)+1:min(blue)-1)
    bing(g-red)=bin(g);
end

bing;

binb = ones(lb,1);

for b = (min(blue):max(blue))
    binb(b-(min(blue))+1)=bin(b);
end

binb;
%%
mu3 = 10;
sd3 = 100;

intblu = blu(2)-blu(1);

maxred = min(green)-intblu;

minred = mu3-max(blu);

redel = 1+ceil((maxred-minred)/intblu);

reds = ones(redel,1);

for r = (1:redel)
    reds(r) = maxred-(r-1)*intblu;
end

redvals = flipud(reds);

xpdf = [redvals;green;blu];

ypdf = normpdf(xpdf,mu3,sd3);



datax = [green;red];

datay = [bing;binb];

negdatay = zeros(length(datay)-27,1);

for d = (1:length(datay)-31)
    negdatay(d) = datay(d+3);
end

negdatax = flipud(negdatay);
```

```
datastitch = [negdatax;bing;binb];

gaussEqn = 'a*exp(-((x-b)/c)^2)+d';

startPoints = [1 10 10 0.6];

gauss = fit(xpdf,datastitch,gaussEqn,'Start',startPoints);

figure(1)
bar(green,bing,1,'FaceColor',[0 0.85 0.15],'EdgeColor',[0 0.85 0.15])

xlim([-600 600])
ylim([0 12000])
hold on
bar(blu,binb,1,'FaceColor',[0 0.5 1],'EdgeColor',[0 0.5 1])
hold on
plot(gauss,'--r')
xlabel('Rhkl','FontSize',28)
ylabel('Frequency','FontSize',28)
set(gca,'FontSize',0.1)
set(gca,'XTick',(-600:200:600),'FontSize',28)
set(gca,'YTick',(0:2000:12000),'FontSize',28)
set(gcf,'color',[1 1 1])
hold off
hold off
```

Matlab script for diffusion-limited growth rate distribution

```
%% Input: Crystal Growth Contour Plot

%Temperature in Kelvin
T = 300;

%number of atoms in first step after nucleation
m_i = 3;

%number of atoms in step fragment
m = 10;

%Facet index (do not include brackets. Write 111 or 100)
facet = 111;

%Chloride coverage (between 0 and 1)
Cl = 0;

%% Monte Carlo
% Variables

mu2=28.5; % mean 2
sd2=0.285; % sigma 2

%% Input: Activation Energies, Vib. Frequencies
```

```
%{111} With Chloride EAM
%Activation Energies in eV
E_LLc111 = 0.080;
E_LSc111 = 0.075;
E_LKc111 = 0.052;
E_SSc111 = 0.243;
E_SKc111 = 0.223;
E_KSc111 = 0.356;
E_SLc111 = 0.320;
E_KLc111 = 0.282;
E_USc111 = 0.203;
E_UKc111 = 0.142;
d_hkl_111 = 0.353;

%{111} Without Chloride EAM
%Activation Energies in eV
E_LLp111 = 0.102;
E_LSp111 = 0.096;
E_LKp111 = 0.067;
E_SSp111 = 0.311;
E_SKp111 = 0.286;
E_KSp111 = 0.456;
E_SLp111 = 0.410;
E_KLp111 = 0.420;
E_USp111 = 0.260;
E_UKp111 = 0.181;

%{100} With Chloride EAM
%Activation Energies in eV
E_LLc100 = 0.383;
E_LSc100 = 0.361;
E_LKc100 = 0.252;
E_SSc100 = 0.236;
E_SKc100 = 0.217;
E_KSc100 = 0.355;
E_SLc100 = 0.533;
E_KLc100 = 0.468;
E_USc100 = 0.413;
E_UKc100 = 0.288;
d_hkl_100 = 0.408;

%{100} Without Chloride EAM
%Activation Energies in eV
E_LLp100 = 0.490;
E_LSp100 = 0.462;
E_LKp100 = 0.322;
E_SSp100 = 0.302;
E_SKp100 = 0.278;
E_KSp100 = 0.455;
E_SLp100 = 0.682;
E_KLp100 = 0.698;
E_USp100 = 0.528;
E_UKp100 = 0.368;

%Vibrational Frequencies in 1/s
v_LL = 1.76E+12;
```

```
v_LS = 1.42E+12;
v_LK = 1.43E+12;
v_SS = 1.41E+12;
v_SK = 1.45E+12;
v_KS = 1.45E+12;
v_SL = 1.91E+12;
v_KL = 1.75E+12;
v_US = 1.42E+12;
v_UK = 1.43E+12;

%% Activation energies

%Facet identity
if facet == 111
    E_LLc = E_LLc111;
    E_LSc = E_LSc111;
    E_LKc = E_LKc111;
    E_SSc = E_SSc111;
    E_SKc = E_SKc111;
    E_KSc = E_KSc111;
    E_SLc = E_SLc111;
    E_KLc = E_KLc111;
    E_USc = E_USc111;
    E_UKc = E_UKc111;
    E_LLp = E_LLp111;
    E_LSp = E_LSp111;
    E_LKp = E_LKp111;
    E_SSp = E_SSp111;
    E_SKp = E_SKp111;
    E_KSp = E_KSp111;
    E_SLp = E_SLp111;
    E_KLp = E_KLp111;
    E_USp = E_USp111;
    E_UKp = E_UKp111;
else
    E_LLc = E_LLc100;
    E_LSc = E_LSc100;
    E_LKc = E_LKc100;
    E_SSc = E_SSc100;
    E_SKc = E_SKc100;
    E_KSc = E_KSc100;
    E_SLc = E_SLc100;
    E_KLc = E_KLc100;
    E_USc = E_USc100;
    E_UKc = E_UKc100;
    E_LLp = E_LLp100;
    E_LSp = E_LSp100;
    E_LKp = E_LKp100;
    E_SSp = E_SSp100;
    E_SKp = E_SKp100;
    E_KSp = E_KSp100;
    E_SLp = E_SLp100;
    E_KLp = E_KLp100;
    E_USp = E_USp100;
    E_UKp = E_UKp100;
end

E_LL = Cl*E_LLc+(1-Cl)*E_LLp;
```

```
E_LS = Cl*E_LSc+(1-Cl)*E_LSp;
E_LK = Cl*E_LKc+(1-Cl)*E_LKp;
E_SS = Cl*E_SSc+(1-Cl)*E_SSp;
E_SK = Cl*E_SKc+(1-Cl)*E_SKp;
E_KS = Cl*E_KSc+(1-Cl)*E_KSp;
E_SL = Cl*E_SLc+(1-Cl)*E_SLp;
E_KL = Cl*E_KLc+(1-Cl)*E_KLp;
E_US = Cl*E_USc+(1-Cl)*E_USp;
E_UK = Cl*E_UKc+(1-Cl)*E_UKp;

%% Input: Growth Phase Diagram Display

interval = 350;

%a = min flux, b = max flux
a = 10^0;
b = 10^15;

%f = min Nuc. Rate, g = Nuc. Rate
f = mu-5*sd;
g = mu+5*sd;

%nc = number of contours.
nc = 25;


%% Data: Diffusion Rates and Physical Constants
format longG

%Boltzmann's Constant in eV/K
k_B = 8.6173324E-5;

%Thermal Energy
kT = k_B*T;

%Monolayer Height
if facet == 111
    d_hkl = d_hkl_111;
else
    d_hkl = d_hkl_100;
end

d_hkl;

%Diffusion Rates in Jumps/s
R_LL = v_LL*exp(-E_LL/kT);
R_LS = v_LS*exp(-E_LS/kT);
R_LK = v_LK*exp(-E_LK/kT);
R_SS = v_SS*exp(-E_SS/kT);
R_SK = v_SK*exp(-E_SK/kT);
R_KS = v_KS*exp(-E_KS/kT);
R_SL = v_SL*exp(-E_SL/kT);
R_KL = v_KL*exp(-E_KL/kT);
R_US = v_US*exp(-E_US/kT);
R_UK = v_UK*exp(-E_UK/kT);
```

```
%Flux is the flux of growth units to the step front in atoms nm^-1 s^-1

syms Flux

D_Au = 0.288;
flux_step = D_Au*Flux;

%E_T0 is the expectation time for growth units to arrive at the step front
E_T0 = flux_step^(-1);

%% Data: Arrival Time

%Construct grid.
loga = log10(a);
logb = log10(b);
logf = log10(f);
logg = log10(g);

FL = ones(1,interval+1);

for j = (1:interval+1)
    k = 10^(loga+(j-1)*(logb-loga)/interval);
    FL(j) = subs(flux_step,Flux,k);

end

MATFL = ones(m,interval+1);

for l = (1:m)
    MATFL(l,:) = l*FL;

end

MATFL;

ET0_L = 1./MATFL;
ET0_LU = ET0_L/2;

%% Data: Incorporation Generator

Markov_incorporation = ones(m,1);
MarkovLU_incorporation = ones(m,1);

for n = (3:m)

    ProbjL = 1/n;
    ProbjLU = 1/(2*n);

    timejL = 1/R_LS;
    timejU = 1/R_US;

    %Q is the infinitesimal generator

    pkA = ones(n-1);
    pkB = eye(n-1);
    pkC = tril(pkA,-2);
    pkD = triu(pkA,2);
    pkE = pkA-pkB-pkC-pkD;
    pkF = -R_SS*pkE;
    pkG = pkB*(2*R_SS);

    Q = pkF+pkG;
    Q(1,1) = R_SK+R_SS;
    Q(n-1,n-1) = R_SS;

    Qone = ones(n-1,1);

    E_partMarkov = Q\Qone;

    T1T2mark = ones(n-1,1);

    for q = (1:n-1)
        T1T2mark(q) = ProbjL*(timejL+E_partMarkov(q));
    end

    T1T2mark0 = ProbjL*(1/R_LK);

    T1T2_Markov = [T1T2mark0;T1T2mark];

    T1T2one = ones(1,n);

    T1T2L = T1T2one*T1T2_Markov;

    Markov_incorporation(n) = T1T2L;

    E_partMarkovLU = Q\Qone;

    T1T2markLU = ones(n-1,1);

    for h = (1:n-1)
        T1T2markLU(h) = ProbjLU*(timejL+timejU+2*E_partMarkovLU(h));
    end

    T1T2LUmark0 = ProbjLU*(1/R_LK+1/R_UK);

    T1T2LU_Markov = [T1T2LUmark0;T1T2markLU];

    T1T2LUone = ones(1,n);

    T1T2LU = T1T2LUone*T1T2LU_Markov;

    MarkovLU_incorporation(n) = T1T2LU;
end

Markov_incorporation(1) = 1/R_LK;

Markov_incorporation(2) = (1/2)*(1/R_LK)+(1/2)*((1/R_LS)+(1/R_SK));
```

```
MarkovLU_incorporation(1) = (1/2)*((1/R_LK)+(1/R_UK));

MarkovLU_incorporation(2) =
(1/4)*((1/R_LK)+(1/R_UK))+(1/4)*((1/R_LS)+2*(1/R_SK)+(1/R_US));

Markov_incorporation;
MarkovLU_incorporation;


%% Data: Time to Complete Step by CTMC

Time_MarkovL = ones(m,interval+1);
Time_MarkovLU = ones(m,interval+1);

for v = (1:m)
    Time_MarkovL(v,:) = ET0_L(v,:)+Markov_incorporation(v);
    Time_MarkovLU(v,:) = ET0_LU(v,:)+MarkovLU_incorporation(v);
end


Time_MarkovL;
Time_MarkovLU;

CORXNL= ones(m,interval+1);
CORXNLU= ones(m,interval+1);

for v = (1:m)
    CORXNL(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovL(v,:));
    CORXNLU(v,:) = (1/R_KL)./((1/R_KL)-Time_MarkovLU(v,:));
end


T_markL = ones(m,interval+1);
T_markLU = ones(m,interval+1);

for v = (1:m)
    T_markL(v,:) = CORXNL(v,:).*Time_MarkovL(v,:);
    T_markLU(v,:) = CORXNLU(v,:).*Time_MarkovLU(v,:);
end

numL = numel(T_markL);

for d = (1:numL)
    if T_markL(d)<0
        T_markL(d)=NaN;
    end
    if T_markLU(d)<0
        T_markLU(d)=NaN;
    end
end

T_markL;
T_markLU;

Time_Markov_L = cumsum(T_markL);
```

```
Time_Markov_LU = cumsum(T_markLU);

%% Data: Expected Time for 1D Nucleation

T1DL = ET0_L + 1/R_LS;
T1DLU = ET0_LU + 0.5/R_LS + 0.5/R_US;

CTMCL = ones(1+m-m_i,interval+1);
CTMCLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_M1 = ones(p,interval+1);
    E_T_M2 = ones(p,interval+1);

    for i = (1:p)
        E_T_M1(i,:) = (1/p)*Time_Markov_L(max(i-1,p-i),:);
        E_T_M2(i,:) = (1/p)*Time_Markov_LU(max(i-1,p-i),:);
    end

    jumpinL = T1DL(p,:);
    jumpinLU = T1DLU(p,:);

    E_T_ML = sum(E_T_M1);
    E_T_MLU = sum(E_T_M2);

    EML = E_T_ML + jumpinL;
    EMLU = E_T_MLU + jumpinLU;

    CTMCL(1+p-m_i,:) = EML;
    CTMCLU(1+p-m_i,:) = EMLU;

end

CTMCL;
CTMCLU;

%% Data: Kink Propagation

kinkL = ones(1+m-m_i,interval+1);
kinkLU = ones(1+m-m_i,interval+1);
kinkarrival = ones(m,interval+1);
kinkarrivalLU = ones(m,interval+1);

for x = (1:m)
    kinkarrival(x,:) = 1./FL;
    kinkarrivalLU(x,:) = 1./(2*FL);
end

kinkcum = cumsum(kinkarrival);
kinkcumLU = cumsum(kinkarrivalLU);

for y = (1:m)
    kinkL(y,:) = y/R_LK+kinkcum(y,:);
    kinkLU(y,:) = y*(0.5/R_LK+0.5/R_UK)+kinkcumLU(y,:);
end
```

```
kinkL;
kinkLU;


KPL = ones(1+m-m_i,interval+1);
KPLU = ones(1+m-m_i,interval+1);

for p = (m_i:m)
    E_T_KPL = ones(p,interval+1);
    E_T_KPLU = ones(p,interval+1);

    for i = (1:p)
        E_T_KPL(i,:) = (1/p)*kinkL(max(i-1,p-i),:);
        E_T_KPLU(i,:) = (1/p)*kinkLU(max(i-1,p-i),:);
    end

    KPL1 = sum(E_T_KPL);
    KPLU1 = sum(E_T_KPLU);

    tkL = T1DL(p,:);
    tkLU = T1DLU(p,:);

    KPL(1+p-m_i,:) = KPL1+tkL;
    KPLU(1+p-m_i,:) = KPLU1+tkLU;

end

KPL;
KPLU;

%% Data: Filters

Markov_Filter = ones(m,interval+1);
Markov_Filter_LU = ones(m,interval+1);
Kink_Filter = ones(m,interval+1);
Kink_Filter_LU = ones(m,interval+1);

for i = (1:interval+1)
    Markov_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Markov_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Markov_Filter(j)<=0
        Markov_Filter(j) = 0;
    else Markov_Filter(j) = 1;
    end
    if Markov_Filter_LU(j)<=0
        Markov_Filter_LU(j) = 0;
    else Markov_Filter_LU(j) = 1;
    end
end

for i = (1:interval+1)
    Kink_Filter(:,i) = T1DL(:,i)-Markov_incorporation;
    Kink_Filter_LU(:,i) = T1DLU(:,i)-MarkovLU_incorporation;
end

for j = (1:m*(interval+1))
    if Kink_Filter(j)<=0
        Kink_Filter(j) = 0;
    else Kink_Filter(j) = 1;
    end
    if Kink_Filter_LU(j)<=0
        Kink_Filter_LU(j) = 0;
    else Kink_Filter_LU(j) = 1;
    end
end

Markov_Filter;
Non_Markov_Filter = 1-Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU = 1-Markov_Filter_LU;

%% Data: Time for direct step adsorption

Direct_Step_L = ones(1+m-m_i,interval+1);
Direct_Step_LU = ones(1+m-m_i,interval+1);
premat_step_L = cumsum(ET0_L);
premat_step_LU = cumsum(ET0_LU);

for s = (m_i:m)
    Direct_Step_L(1+s-m_i,:) = premat_step_L(s,:)+s/R_LS;
    Direct_Step_LU(1+s-m_i,:) = premat_step_LU(s,:)+s/R_LS;
end

Direct_Step_L;
Direct_Step_LU;

%% Data: Minimum Time Direct Incorporation

MinDirectL = min(Direct_Step_L,KPL);
MinDirectLU = min(Direct_Step_LU,KPLU);

%% Collection of Important Matrices and Vectors

% Flux vector. 1 x int+1
FL;

% Expected Time for atom to arrive at step. n x int+1
ET0_L;
ET0_LU;

% Expected Time for 1D nucleation. n x int+1
T1DL;
T1DLU;

% Expected Time for atom to diffuse via CTMC to kink. n x 1
Markov_incorporation;
MarkovLU_incorporation;
```

```
% Growth Mechanism Filters
Markov_Filter;
Non_Markov_Filter;
Markov_Filter_LU;
Non_Markov_Filter_LU;

% Expected Time to complete step via direct step incorporation. n x int+1
Direct_Step_L;
Direct_Step_LU;


% Expected Time to complete step via direct kink propagation. n x int+1
KPL;
KPLU;

% Expected Time to complete step via CTMC mechanism. n x int+1
CTMCL;
CTMCLU;

% (minimum) Expected Time for direct incorporation. n x int+1
MinDirectL;
MinDirectLU;

%% Data: Filtered Results

MF = ones(1+m-m_i,interval+1);
MFLU = ones(1+m-m_i,interval+1);

for f = (m_i:m)
    MF(1+f-m_i,:) = Markov_Filter(f,:);
end
MF;
nMF = 1-MF;

for g = (m_i:m)
    MFLU(1+g-m_i,:) = Markov_Filter_LU(g,:);
end
MFLU;
nMFLU = 1-MFLU;

Filtered_CTMCL = MF.*CTMCL;
Filtered_MINDIR = nMF.*MinDirectL;

Filtered_CTMCLU = MFLU.*CTMCLU;
Filtered_MINDIRU = nMFLU.*MinDirectLU;


% Expected times for step completion
TIME_L = Filtered_CTMCL+Filtered_MINDIR;
TIME_LU = Filtered_CTMCLU+Filtered_MINDIRU;

%Time for monolayer spread

T_Spread_L = sum(TIME_L)';
```

```
T_Spread_LU = sum(TIME_LU)';

%% Plot: Growth Phase Diagram

NR =linspace(mu-5*sd,mu+5*sd,interval+1);
NL =1./NR;

Markov_ContourL1 = ones(interval+1,interval+1);
Markov_ContourLU1 = ones(interval+1,interval+1);

for l = (1:interval+1)
    Markov_ContourL1(l,:) = NL+T_Spread_L(l);
    Markov_ContourLU1(l,:) = NL+T_Spread_LU(l);
end

Markov_ContourL1;
Markov_ContourLU1;

Markov_ContourL = Markov_ContourL1/d_hkl;
Markov_ContourLU = Markov_ContourLU1/d_hkl;


%% Histograms

N=10^6; % # times

% Solve for the analytical sampling equation and run N times
x=zeros(length(N));
x2=zeros(length(N));
y=zeros(length(N));
y2=zeros(length(N));
for i=1:N
    r1=rand;
    r2=rand;
    r3=rand;
    r4=rand;
    x(i)=sqrt(-2*sd^2*log(r1))*sin(2*pi*r2);
    x(i)=x(i)+mu;
end

NR;
bin = histc(x,NR);

figure(1)
hist(x,linspace(mu-5*sd,mu+5*sd,interval+1))
xlabel('x')
ylabel('Frequency')
hold on

RATE_L = 1./Markov_ContourL;

rcol = RATE_L(interval+1,:);

sum(bin)
```

```
%% Plot: Nucleation-Limited Growth Rate Distribution

bing = zeros(1,length(rcol));

for b = (1:length(rcol))
    bing(b) = bin(b);
end

green = bing';
xpdf = rcol';

gaussEqn = 'a*exp(-((x-b)/c)^2)+d';

startPoints = [15000 10 0.1 0.1];

gauss = fit(xpdf,green,gaussEqn,'Start',startPoints);

bar(xpdf,green,1,'FaceColor',[0 0.85 0.15],'EdgeColor',[0 0.85 0.15])


xlim([5 15])
ylim([0 12000])
hold on
plot(gauss,'--r')
xlabel('Rhkl','FontSize',28)
ylabel('Frequency','FontSize',28)
set(gca,'FontSize',0.1)
set(gca,'XTick',(0:1:15),'FontSize',28)
set(gca,'YTick',(0:2000:12000),'FontSize',28)
set(gcf,'Color','w')
hold off
```