**MAIN.M**

%%%% define output folder

outfolder..

%%%%%%%%%%     DEFINE CASE     %%%%%%%%%%

%%1-define surface (e.g., for a sinusoidal)

Ax=0.1;  %oscilation amplitude in x ;

fx=1.0;  %oscilation frequency in x

...

surfxmin=-5.0\*pi; %surface x-min

surfxmax=5.0\*pi;  %surface x-max

...

%%2-define particles

%trajectories

phi=0.0;  %phi = angle wrt x-axis 0 < phi < pi/2

dlt=pi/3.0; %delta = angle wrt -z axis, (pointing to surface); 0<delta<pi/2

th=pi-dlt ; %theta =angle wrt +z axis ; pi/2 < theta < pi

%launching area

initxmin=-2.0\*pi; %x-min of initializing ('launching') particles

initxmax=2.0\*pi;  %x-max of initializing particles

...

%number of 'particles'

NP=100000;

nsteps=2000; %resolution: NP/nsteps = npoints = number of surface grids

%%3-define materials (for Eckstein's fit formula)

Tg='W'; %target material

Pr='N'; %projectile

E0=100.0; %impact energy

%%load erosion and reflection parameters

Ecksteinfolder=...

ErosParamFile=...

ReflNParamFile=...

ReflEParamFile=...

%%some constants:

aB=0.529177; % in [A]

ec2=14.399651; % in [eV A]

%Lindh. screening length

aL=(9\*pi^2/128)^(1/3)\*aB\*(Zpr^(2/3)+Ztg^(2/3))^(-1/2);

%reduced energies; to be placed inside loop if not monoenergetic

eps\_L=E0\*(aL/(Zpr\*Ztg\*ec2))\*(Mtg/(Mtg+Mpr));

%coefficients for the cosine-like distribution of emitted particles' angle

%f=r1\*(cos(x))^n1+r2\*(cos(x)^n2;

r1=1.5; r2=-1.0; ...

%%%%%%%%%%     RUN CASE     %%%%%%%%%%

%create output file and save input data

sfile = ...;

filename=sfile;

save: Ax, fx, Ay, fy, NP, nsteps, phi, th, surfxmin, surfxmax, ... , Tg, Pr

run('zs\_zp\_intersect'):

Calculate intersection of surface and trajectory, and local impact angle (in function part\_surf\_local\_angle)

run('impact\_output\_and\_distributions'):

Calculate distributions; get matrixes ready to output and plot

run('cellarea'):

calculate area of each grid cell, as in a rough surface, a uniform grid formed as (xmax-xmin)/npoints will lead to different areas

run('localflux'):

scale to obtain flux reaching each area; local\_flux=Ncounts/cell\_area;

run('plot\_incoming\_impacts')

plotting incoming particles output: surface morphology, impact angle and flux

run('erosion')

Erosion of the target by the projectiles for the local conditions (flux and angles) calculated above, and using the Eckstein formula

run('reflection\_RN')

Projectile (particle) reflection for the local conditions (flux and angles) calculated above, and using the Eckstein formula

run('reflection\_RE')

Projectile energy reflection for the local conditions (flux and angles) calculated above, and using the Eckstein formula

run('particle\_emission')

particles reflected (species=projectile, E=energy reflection coeff. & angle=near specular reflection) and sputtered (species=target, E=Thompson distribution & angle=under/over-cosine) are emitted from the surface in straight lines. the function that finds the possible intersection of emitted particles with the surface

run('plot\_sputtering\_and\_reflection')

plotting of reflection and sputtering output

movefile('\*.png','outfile/')

**APPENDIX I: DESCRIPTION / ALGORITHM OF MODULES**

SURFACE DESCRIPTION FUNCTIONS:

surface function:

zs(x,y,Ax,fx,Ay,fy)= Ax\*sin(x/fx)+Ay\*sin(y/fy);

- - - - - - - - - - - - -

Surface derivative:

dzs(x,y,Ax,fx,Ay,fy)= (Ax/fx)\*cos(x/f) + (Ay/fy)\*cos(y/f);

- - - - - - - - - - - - -

Surface derivative wrt x:

dzsx(x,Ax,fx)= (Ax/fx)\*cos(x/fx);

- - - - - - - - - - - - -

Surface derivative wrt y:

dzsy(y,Ay,fy)= (Ay/fy)\*cos(y/fy);

PARTICLE TRAJECTORY:

trajectory wrt x-coord:

zpx(x,x0,z0,th,phi)=z0+(x-x0)/(tan(th)\*cos(phi));

- - - - - - - - - - - - -

trajectory wrt y-coord:

zpy(y,y0,z0,th,phi)= z0+(y-y0)/(tan(th)\*sin(phi));

--------------- 'zs\_zp\_intersect'---------------

'Calculate intersection of surface and trajectory ';

%%------------- LOOP ON PARTICLES:

for p = 1:NP

    %%z0=1.5\*max(zs)

    [x,fxmin]=fminbnd(@(x) zs(x,x,Ax,fx,0,fy),surfxmin,surfxmax);

    [y,fymin]=fminbnd(@(y) zs(y,y,0,fx,Ay,fy),surfymin,surfymax);

    z0=-1.5\*min(fxmin,fymin);

    %%different options for (x0,y0)

    %%a) random initial position (x0,y0), between (initmin,initmax)

    x0=0.5\*(initxmax-initxmin)\*(1-2\*rand); y0=...

    %%b) uniform grid of (x0,y0)

    %%NOT IMPLEMENTED YET

%%save particle's values as:    first index (p) = particle index

    %%component 1:3 = impact point; 4 = angle wrt surface normal

    %%so global (input) values of the particle are:

    partglobal(p,1)=x0; ...(p,2)=y0; ...(p,3)=z0; ...(p,4)=dlt\*180/pi;

    %%and local values of the particle are:

    %%given an analytical surface and particle trajectory, find the point and angle of intersection wrt the surface normal

    partlocal(p,:) = part\_surf\_local\_angle(p,x0,y0,z0,phi,th,Ax,fx,Ay,fy);

end

%%------------- LOOP DONE

save: 'partlocal'

- - - - - - - - - - - - -

--------------- part\_surf\_local\_angle ---------------

part\_surf\_local\_angle(p,x0,y0,z0,phi,th,Ax,fx,Ay,fy)

%%given an analytical surface and particle trajectory, find the intersection

%%point and angle wrt the surface normal

%normalized velocity components = particle trajectory's angle

%vx=cos(phi)\*sin(th); vy=sin(phi)\*sin(th); vz=cos(th);

%%zp=z0+vz\*t -> t=(z-z0)/cos(th); xp=x0+vx\*t ; yp=y0+vy\*t

%%xp = x0+ cos(phi)\*sin(th)\*((z-z0)/(cos(th)))=x0+(z-z0)\*(cos(phi)\*tan(th));

%%yp = y0+ sin(phi)\*sin(th)\*((z-z0)/(cos(th)))=y0+(z-z0)\*(sin(phi)\*tan(th));

%%look for intersecting point: zp=zs

%%a) if the surface is given analytically:

%%the solution will be along zp trajectory -> write x as a function of y: x=x0+(ys-y0)\*tan(phi)  and use it to pass only one variable to zs function

 if (...) %check for all kind of cases (trajectory aligned or perpendicular to ridgeline, surface variation in 1D...)

general case:

 else

 ys=fzero(@(y) zpy(y,y0,z0,th,phi)-zs(x0+(y-y0)/tan(phi),y,Ax,fx,Ay,fy), y0);

xs=x0+(ys-y0)\*tan(phi);

q=zs(xs,ys,Ax,fx,Ay,fy);

 end

%%calculate (x,y) at the intersecting point (in some cases needed)

xs=x0+(q-z0)\*cos(phi)\*tan(th);

ys=y0+(q-z0)\*sin(phi)\*tan(th);

%%find normal to the surface at intersection:

%a) if the surface is given analytically -> derivate analytically: dzs

%ns=[-dzs/dx,-dzs/dy,1];

nsx=-dzsx(xs,Ax,fx); nsy=-dzsy(ys,Ay,fy); nsz=1.0; ns=[nsx,nsy,nsz];

%np=[d(zp)/dx,d(zp)/dy,-1] direction opposite to incoming particles'

npx=-cos(phi)\*sin(th); npy=-sin(phi)\*sin(th); npz=-cos(th); np=[npx,npy,npz];

%%b) if the surface is not given analytically:

%%some development in initial versions, but not available yet

%%and angle wrt particles trajectory

cosa=dot(np,ns)/(norm(np)\*norm(ns));

locang=acos(cosa);

output(1)=xs; output(2)=ys; output(3)=q; output(4)=locang; %angle in rad

--------------- 'impact\_output\_and\_distributions'---------------

'distributions and output:'

%%derive distributions: #particles/unit cell and particle-surface angle

%%as well as cumulative distributions of both 2D plots

%%surface-traject intersection point

%%and local angle (that between surface and traject) at [xa,ya], in rad

xa=transpose(partlocal(:,1)); ya=transpose(partlocal(:,2));

za=transpose(partlocal(:,3)); pangle=transpose(partlocal(:,4));

run('surfacegrid'): create a gridded version of the surface

save: xg, yg, zg, xa, ya, za, pangle

run('histogram\_and\_distr'): calculate the histogram and distribution of impact points

profncountsx=...; profncountsy=...

save: profncountsx,profncountsy

run('angle\_profile\_ave\_and\_distr'): calculate the histogram and distribution of impact angle

- - - - - - - - - - - - -

--------------- surfacegrid---------------

create a gridded version of the surface

%%xg, yg, zg = gridded version of the surface;

xg=linspace(surfxmin,surfxmax,npoints);

yg=linspace(surfymin,surfymax,npoints);

zg(i,j)=zs(xg(i),yg(j),Ax,fx,Ay,fy);

- - - - - - - - - - - - -

--------------- histogram\_and\_distr---------------

%histrogram: ‘bin impact points’

di=(surfxmax-surfxmin)/(npoints-1); dj=(surfymax-surfymin)/(npoints-1);

i=floor((xa(p)-surfxmin)/di)+1;

j=floor((ya(p)-surfymin)/dj)+1;

   [check for possible particles that impacted outside the surface area...]

Ncounts(i,j)=Ncounts(i,j)+1;

save:'Ncounts'

%%find the actual distribution:

            ncint=Ncounts(i,j);

            inc(ncint)=ncint;

            ncdistr(ncint)=ncdistr(ncint)+1;

save: 'ncdistr'

- - - - - - - - - - - - -

--------------- angle\_profile\_ave\_and\_distr---------------

%angle profile: search for points along y-axis, around x=xprofile

xprofile=pi/4.0;

if (xa(p)<(xprofile+pi/4.0) && xa(p)>(xprofile-pi/4.0))

   ip=ip+1;

   yangleprof(ip)=ya(p);

   angleprofy(ip)=pangle(p); %in degrees

end

yprofile=pi/4.0;

... angleprofx(jp)=pangle(p); %in degrees;

save: xprofile, yprofile, yangleprof, angleprofy, xangleprof, angleprofx

%%cumulative distributions:

%angle

aint=floor((pangle(k)\*180/pi))+1; %ai in [1:pi] or [1:pi/2]

adistr(aint)=adistr(aint)+1;

save: 'ai','aint','adistr';

%average angle of each cell

cumulangle(i,j)=cumulangle(i,j)+pangle(p); %in rad;

avecellangle(i,j)=cumulangle(i,j)/Ncounts(i,j);

save: 'avecellangle';

--------------- cellarea---------------

%%for rough surfaces, a uniform grid formed as (xmax-xmin)/npoints will

%%lead to different areas for each cell: cellarea(i,j)

%a) if the surface is given analytically -> derivate analytically: dzs

%ns=[-dzsx(xs,Ax,fx),-dzsy(ys,Ay,fy),1];

%%xg, yg, zg = gridded version of the surface;

dx=xg(2)-xg(1);

dy=yg(2)-yg(1);

run('surfacegrid')

xi=xg(i)+0.5\*dx; yj=yg(j)+0.5\*dy; zk(i,j)=zs(xi,yj,Ax,fx,Ay,fy);

nsxg(i,j)=-dzsx(xi,Ax,fx); nsyg(i,j)=-dzsy(yj,Ay,fy);

sangles = surface\_n\_angles(nsxg(i,j),nsyg(i,j),nszg(i,j));

sg\_theta(i,j)=sangles(1); sg\_phi(i,j)=sangles(2);

normal\_cell\_area(i,j)=1/(cos(sg\_theta(i,j)\*sin(sg\_phi(i,j)))\*cos(sg\_theta(i,j)\*cos(sg\_phi(i,j)))); normal\_cell\_area(i,j)=dx\*dy;

cell\_area(i,j)=dx\*dy/normal\_cell\_area(i,j);

save: 'nsxg', 'nsyg', 'sg\_theta', 'sg\_phi', 'cell\_area', 'normal\_cell\_area'

--------------- localflux---------------

%flux for a flat surface: (# particles / (initial area)

Flux0=NP/((surfxmax-surfxmin)\*(surfymax-surfymin));

%local\_flux=Ncounts/cell\_area;

local\_flux(i,j)=Ncounts(i,j)/cell\_area(i,j);

normal\_local\_flux(i,j)=local\_flux(i,j)/Flux0;

save:'Flux0', 'local\_flux', 'normal\_local\_flux'

--------------- plot\_incoming\_impacts---------------

'plotting incoming particles output...';

%due to the way 3D surface plots auto-read axes from matrices, need to

%transpose each [x,y] matrix. Do within plot to not double variable names

%1-background: contour plot

'      ... 1: surface';

'      ...   a: morphology';

surf(xg,yg,transpose(zg));

title('1a: surface morphology')

xlabel('x')

ylabel('y')

zlabel('surface height')

print('1a\_morphology','-dpng')

%1b-cell area

'      ...   b: normalized cell area (wrt flat surf)';

surf(xg,yg,transpose(normal\_cell\_area));

caxis([1,max(max(normal\_cell\_area))]);

title('1b: cell area')

xlabel('x')

ylabel('y')

zlabel('area/S0')

print('1b\_cellarea','-dpng')

%1c-cells theta angle

'      ...   c: theta angle of cells normal ';

surf(xg,yg,transpose(sg\_theta\*180/pi)); %in degrees

caxis([0.0,90.0]);

title('1c: theta of cells normal')

xlabel('x')

ylabel('y')

zlabel('cell theta')

print('1c\_cellntheta','-dpng')

%1d-cells phi angle

'      ...   d: phi angle of cells normal ';

surf(xg,yg,transpose(sg\_phi\*180/pi)); %in degrees

caxis([-180,180]);

title('1d: phi of cells normal')

xlabel('x')

ylabel('y')

zlabel('cell phi')

print('1d\_cellnphi','-dpng')

%2-angle

S32='      ... 2: angle';

S32a='      ...   a: at each impact point';

scatter(xa,ya,[],pangle\*180/pi);

caxis([0,90]); %180?

title('2a: angle at each impact point')

xlabel('x')

ylabel('y')

zlabel('angle')

print('2a\_impactangle','-dpng')

%2b-average impact angle of each cell

'      ...   b: average impact angle of each cell';

surf(xg,yg,transpose(avecellangle\*180/pi));

caxis([0,90]);

title('2b:average impact angle of each cell')

xlabel('x')

ylabel('y')

zlabel('ave angle')

print('2b\_aveimpactangle','-dpng')

%2c - angle profile (at x=5+/-0.5)

'      ...   c: profile along x (at y=... +/-... )’

scatter(xangleprof,angleprofx\*180/pi)

axis ([surfxmin surfxmax 0 90]); %180?

title('2c: profile of the angle along x')

xlabel('x')

ylabel('angle profile')

print('2c\_angleprofilex','-dpng')

%2d - angle profile (at y=5+/-0.5)

'      ...   d: profile along y (at x=... +/-... )'];

scatter(yangleprof,angleprofy\*180/pi)

axis ([surfymin surfymax 0 90]);    %180?

title('2d: profile of the angle along y')

xlabel('y')

ylabel('angle profile')

print('2d\_angleprofiley','-dpng')

%2e-cumulative distribution of angles

'      ...   e: cumulative distribution of angles';

plot(ai,adistr);

title('2e: cumulative distribution of angles')

xlabel('number of cells w/ loc angle')

ylabel('local angle')

print('2e\_anglehistogram','-dpng')

%3-flux

'      ... 3: flux';

'      ...   a: number of impacts per cell';

surf(xg,yg,transpose(Ncounts));

caxis([0,max(max(Ncounts))]);

title('3a: number of impacts per unit cell')

xlabel('x')

ylabel('y')

zlabel('# impacts')

print('3a\_Nimpact','-dpng')

%3b-local flux normalized

'      ...   b: normalized local flux';

surf(xg,yg,transpose(normal\_local\_flux));

caxis([0,max(max(normal\_local\_flux))]);

title('3b: normalized local flux (wrt flux at z0)')

xlabel('x')

ylabel('y')

zlabel('flux/flux0')

print('3b\_localflux','-dpng')

%3c: histogram along x (integrated over y)

'      ...   c: profile of number of impacts along x (integrated along y, normalized)';

plot(xg,profncountsx)

title('3c: profile of number of impacts along x')

xlabel('x')

ylabel('#impacts')

print('3c\_Nimpactprofilex','-dpng')

%3d: histogram along y (integrated over x)

'      ...   d: profile of number of impacts along y (integrated along x, normalized)';

plot(yg,profncountsy)

title('3d: profile of number of impacts along y')

xlabel('y')

ylabel('#impacts')

print('3d\_Nimpactprofiley','-dpng')

%3e-cumulative distribution of histogram

'      ...   e: cumulative distribution of histogram';

plot(inc, ncdistr);

title('3e: cumulative distribution of histogram')

xlabel('number of cells w/ #impacts')

ylabel('#impacts')

print('3e\_Nimpacthistogram','-dpng')

' done!'

--------------- erosion---------------

%%Erosion of Tg(target) by Pr(projectile)

%%use distributions of #particles/unit cell and particle-surface angle

%%to calculate the local erosion using the Eckstein formula

%%eros(local)/eros(0) = Y\_loc\*Flux(local)/Y\_flat\*Flux(ave)

%%and the average flux (i.e. that for a flat surface) = #traj/Area

%0- load parametrization for Ecksteins fit-formula now in main file

w\_eps=eps\_L+0.1728\*(eps\_L)^(0.5)+0.008\*(eps\_L)^(0.1504);

sn=0.5\*log(1+1.2288\*eps\_L)/w\_eps;

Y\_e=qeros\*sn\*((E0/Eth-1)^(mu\_eros))/((lambda/w\_eps)+(E0/Eth-1)^(mu\_eros));

Save: 'Y\_e'

%%1-initialize

%2-Calculate erosion of each cell

alpha0=pi-acos(sqrt(1/(1+E0/Esp)));

if (Ncounts(i,j)>0)

  %a-local angle: locang=avecellangle(i,j);

  %b-Calculate the erosion yield for the local angle and energy

  relang=locang/alpha0;

  term1=(cos((relang\*pi/2)^c))^(-f); term2=exp(b\*(1-1/cos((relang\*pi/2)^c)));

tf1=isreal(term1);        tf2=isreal(term2);

  Y\_angle(i,j)=term1\*term2;

Y\_loc(i,j)=Y\_angle(i,j)\*Y\_e;

  %c-Local erosion: #particles eroded, for Ncounts / Flux hitting the cell

  cellNeros(i,j)=Y\_loc(i,j)\*Ncounts(i,j);

  cellFeros(i,j)=Y\_loc(i,j)\*local\_flux(i,j);

end

save:'Y\_angle', 'Y\_loc', 'cellNeros', 'cellFeros';

--------------- reflection\_RE---------------

'Energy reflection:';

%energy reflection yield using the Eckstein formula, for average local angle

%1-load parametrization for Ecksteins fit-formula in main file

%2-Calculate the energy reflection yield for E0 energy

tf=strcmp(Pr,Tg); %compare target and projectile components

if %NOT self-bombardment

    RE\_0=b1\*(eps\_L)^a2/(1+a3\*(eps\_L)^b4);

else

    RE\_0=exp(a1\*(eps\_L)^a2)/(1+exp(a3\*(eps\_L)^a4));

end

%tanh works better at small angles; atan at large angles

%NOTE: atan argument (d3\*locang+d4) should be near zero not to diverge ->

%as d3>0 and d4<0 in tabulated values -> use -d4 in our implementation

if (locang<pi/4)

    RE\_angle(i,j)=d1+d2\*tanh(d3\*locang-d4);

else

   RE\_angle(i,j)=d1+d2\*atan(d3\*locang-d4);

end

RE\_loc(i,j)=RE\_angle(i,j)\*RE\_0;

Save:'RE\_0', 'RE\_angle', 'RE\_loc';

--------------- reflection\_RN---------------

'particle reflection:'

%%Reflection of Pr(projectile) from Tg(target), using the Eckstein formula

%%refl(local)/refl(0) = RN\_loc\*Flux(local)/RN\_flat\*Flux(ave)

%1-load parametrization for Ecksteins fit-formula in main file

%2-Calculate the reflection yield for E0 energy

tf=strcmp(Pr,Tg); %compare target and projectile components

if %NOT self-bombardment

  RN\_0=b1\*(eps\_L)^b2/(1+b3\*(eps\_L)^b4);

else

  RN\_0=exp(b1\*(eps\_L)^b2)/(1+exp(b3\*(eps\_L)^b4));

end

%tanh works better at small angles; atan at large angles

%NOTE: atan argument (c3\*locang+c4) should be near zero not to diverge ->

%as c3>0 and c4<0 in tabulated values -> use -c4 in our implementation

if (locang<pi/4)

   RN\_angle(i,j)=c1+c2\*tanh(c3\*locang-c4);

else

   RN\_angle(i,j)=c1+c2\*atan(c3\*locang-c4);

end

RN\_loc(i,j)=RN\_angle(i,j)\*RN\_0;

%c-Total reflection from cell: Nparticles and flux reflected

cellNrefl(i,j)=RN\_loc(i,j)\*Ncounts(i,j);

cellFrefl(i,j)=RN\_loc(i,j)\*local\_flux(i,j);

save'RN\_0','RN\_angle','RN\_loc','cellNrefl','cellFrefl';

--------------- particle\_emission---------------

'---- calling particle emission -----'

%particle emission:

%2 populations: reflected and sputtered particles

%a) sputtered are emitted with E=Thompson distribution &

%   angle=under/over-cosine (to test sensitivity)

%b) reflected are emitted with E=energy reflection coeff. & angle=near

%   specular reflection

%particles travel in straight lines, so we can use functions built for

%launching initial particles here.

re-deposition: 0:  YES, particle re-deposited in final position')

            1-6: NO; Final position = projection of trajectory in initial z0

Possible exit flags for emitted particle’s trajectory not crossing surface:

1:  (x0,y0)=(xs,yx) (doesn’t cross surface at any other point)

ERR:     2: err in z-direction  3: err in x-direction  4: err in y-direction

5: invalid exitflag    6:  fzero(Exitflag)=6  7: invalid theta  '

%the function that finds the intersection with the surface is similar to

%that for the initial particles, but here we need also need to look for

%particles travelling in +z, which might hit the surface, or not

nSppart %number of particles emitted by sputtering

SpEout % energy of particles emitted through sputtering

SpThout %theta angle of particles emitted by sputtering

SpThout\_loc %theta particles emitted by sput., wrt local surf normal

aveSpThout %average SpThout

aveSpThout\_loc %average SpThout\_loc

SpPhiout% idem for phi angle of particles emitted by sputtering

SpPhiout\_loc, aveSpPhiout, aveSpPhiout\_loc

sput\_redepos\_p, sput\_redepos %characteristics of redeposited particles

%idem for particles emitted through reflection

nRpart, REout, ..., refl\_redepos\_p,...

NredepSp %where sputtered particles are redeposited

NredepR %where reflected particles are redeposited

%%%%%%%% LOOP over CELLS %%%%%%%%

%number of particles emitted from (i,j) cell that are redeposited

NSp, NR

%%%%%%%%%%%%%%%% 2-Sputtering %%%%%%%%%%%%%%%%

nSppart(i,j)=Y\_loc(i,j)\*Ncounts(i,j);

(take the integer part; if reminder >0.5, round up; else down.

%number of particles emitted from this cell were re-deposited

NSp, NR

% energy and angle of particles emitted through sputtering, wrt local normal to surf: all reflected particles are emitted with same energy, REout(i,j)

SpEout(i,j,1:nEmittSp) = thompson\_distr(E0, Esb, Mtg, Mpr, nEmittSp, npoints)

SpThout\_loc(i,j,1:nEmittSp) = cosn\_distrib(r1, n1, r2, n2, nEmittSp, npoints)

%For each sputtered particle

     %define characteristics, wrt global coordinate system

SpThout(i,j,p)=sg\_theta(i,j)+SpThout\_loc(i,j,p);

SpPhiout(i,j,p)=sg\_phi(i,j)+SpPhiout\_loc(i,j,p);

%intersection of sputtered particle with surface

sput\_redepos\_p=emitted\_part\_surf\_intersec(xi,yj,zk(i,j),SpPhiout(i,j),SpThout(i,j,p),Ax,fx,Ay,fy,z0);

sput\_redepos(i,j,p,1:4)=sput\_redepos\_p(1:4);   % store the data

% sorting redep positions

xr=sput\_redepos\_p(1);   yr=sput\_redepos\_p(2);

zr=sput\_redepos\_p(3); kr=sput\_redepos\_p(4);

if (kr==0)

         ir=floor((xr-surfxmin)/di)+1;

        ...check if ir out of surface boundary...

         jr=floor((yr-surfymin)/dj)+1;

         ...check if jr out of surface boundary...

  NredepSp(ir,jr)=NredepSp(ir,jr)+1;

NSp=NSp+1;

end

...take averages ...

        %%%%%%%%%%%%%%%% 3-Reflection  %%%%%%%%%%%%%%%%

nRpart(i,j)=RN\_loc(i,j)\*Ncounts(i,j);

% all reflected particles are emitted with same energy, REout(i,j)

% angle of particles emitted through reflection, wrt local

% normal to surf:

RThout\_loc(i,j,1:nEmittR) = cosn\_distrib(r1, n1, r2, n2, nEmittR, npoints);

%reflected energy and phi angle have a fixed value for each

%cell, not distributions; define wrt global coordinate system

REout(i,j)=E0\*RE\_loc(i,j)/RN\_loc(i,j);

Eout=REout(i,j); %not needed so far

  %specular direction

if (sign(phi)==sign(sg\_phi(i,j))) RPhiout\_loc(i,j)=sg\_phi(i,j)-phi;

else RPhiout\_loc(i,j)=pi+phi-sg\_phi(i,j); end

RPhiout(i,j) = sg\_phi(i,j)-RPhiout\_loc(i,j);

Normalize phi range

%For each reflected particle

%theta angle, wrt global coordinate system

RThout(i,j,p)=sg\_theta(i,j)+RThout\_loc(i,j,p);

%intersection of reflected particle with surface

refl\_redepos\_p=emitted\_part\_surf\_intersec(xi,yj,zk(i,j),RPhiout(i,j),RThout(i,j,p),Ax,fx,Ay,fy,z0);

refl\_redepos(i,j,p,1:4)=refl\_redepos\_p(1:4); % store values

% sorting redep positions

xr=refl\_redepos\_p(1); yr=refl\_redepos\_p(2);

zr=refl\_redepos\_p(3);kr=refl\_redepos\_p(4);

if (kr==0)

          ir=floor((xr-surfxmin)/di)+1;

                 ...check if ir is out of surface...

jr=floor((yr-surfymin)/dj)+1;

                   ...check if jr is out of surface...

NredepR(ir,jr)=NredepR(ir,jr)+1;

           NR=NR+1;

    end

            ... take averages ...

end

save: 'NredepSp', 'NredepR', 'nSppart', 'SpEout', 'SpThout', 'SpThout\_loc', 'aveSpThout', 'aveSpThout\_loc', 'SpPhiout', 'SpPhiout\_loc', 'aveSpPhiout' 'aveSpPhiout\_loc', 'sput\_redepos' 'nRpart', 'REout', 'RThout', 'RThout\_loc', 'aveRThout', 'aveRThout\_loc', 'RPhiout', 'RPhiout\_loc', 'refl\_redepos'

- - - - - - - - - - - - -

--------------- thompson\_distr---------------

thompson\_distr(Ein, Esb, Mtg, Mpr, npart, npoints)

u(1,npart) %array of npart random number, uniformly distributed (0,1)

func(1:npoints %array of discretized function

fsum(1:(npoints)) %array of cumulative values of the function

gamma=4\*Mtg\*Mpr/(Mtg+Mpr)^2;

Eth=Esb/(gamma\*(1-gamma));

Emax=Ein\*gamma\*(1-gamma)-Esb;

Exi=linspace(0,Emax, npoints);

%initial values:

func(1)=2\*Esb\*Exi(1)\*(Ein/Eth)\*(Ein/Eth)/(((Exi(1)+Esb)^3)\*((Ein/Eth-1)^2));

fsum(1)=0.0; %or func(1)?

for i=2:npoints

    func(i)=2\*Esb\*Exi(i)\*(Ein/Eth)\*(Ein/Eth)/(((Exi(i)+Esb)^3)\*((Ein/Eth-1)^2));

    fsum(i)=fsum(i-1)+func(i); %fsum(0)=0.0!

end

%normalize fsum, just in case:

for i=1:npoints; fsum(i)=fsum(i)/fsum(npoints); end

for j=1:npart %binary search

    u(j)=rand;

    kmin=1; kmax=npoints; kmid=floor(npoints/2.0); err=0;

    while kmin<kmax-1 %while there's a difference of 2 or more indexes

        if (u(j)>fsum(kmid) && u(j)<=fsum(kmax))

            kmin=kmid;

            kmid=floor((kmax+kmin)/2.0);

        elseif (u(j)<fsum(kmid) && u(j)>=fsum(kmin))

            kmax=kmid;

            kmid=floor((kmax+kmin)/2.0);

        else ['error in binary search’] end

    end

    output(j)=Exi(kmid);

end

- - - - - - - - - - - - -

--------------- cosn\_distr---------------

cosn\_distrib(a1, n1, a2, n2, npart, npoints)

%generate random numbers with an analytically given distribution:

u(1,npart) %array of npart random number, uniformly distributed (0,1)

func %array of discretized function

fsum %array of cumulative values of the function

cosxi=linspace(0,pi/2.0,npoints);

%initial values:

func(1)=a1\*(cos(cosxi(1)))^n1+a2\*(cos(cosxi(1)))^n2; fsum(1)=0.0;

for i=2:npoints

    func(i)=a1\*(cos(cosxi(i)))^n1+a2\*(cos(cosxi(i)))^n2;

    fsum(i)=fsum(i-1)+func(i);

end

%normalize fsum, just in case:

for i=1:npoints; fsum(i)=fsum(i)/fsum(npoints); end

for j=1:npart    %binary search

    u(j)=rand;

    kmin=1;    kmax=npoints;    kmid=floor(npoints/2.0);    err=0;

    while kmin<kmax-1 %while there's a difference of 2 or more indexes

        if (u(j)>fsum(kmid) && u(j)<=fsum(kmax))

            kmin=kmid;

            kmid=floor((kmax+kmin)/2.0);

        elseif (u(j)<fsum(kmid) && u(j)>=fsum(kmin))

            kmax=kmid;

            kmid=floor((kmax+kmin)/2.0);

        else ['error in binary search’] end

    end

    output(j)=cosxi(kmid);

end

- - - - - - - - - - - - -

--------------- emitted\_part\_surf\_intersec---------------

emitted\_part\_surf\_intersec(x0,y0,z0,phi,th,Ax,fx,Ay,fy,initz0)

%%given an analytical surface and particle trajectory, find the

%%intersection point

%%zp=z0+vz\*t -> t=(z-z0)/vp\*cos(th)

%%xp=x0+vx\*t = x0+ cos(phi)\*sin(th)\*((z-z0)/(cos(th)))=x0+(z-z0)\*(cos(phi)\*tan(th))

%%yp=y0+vy\*t = y0+ sin(phi)\*sin(th)\*((z-z0)/(cos(th)))=y0+(z-z0)\*(sin(phi)\*tan(th));

%%look for intersecting (redeposition) point (xs, ys, q), given by zp=zs

%%a) if the surface is given analytically:

%% as solution is along the particle's trajectory, x can be written as x=x(y): x=x0+(ys-y0)\*tan(phi) and pass only one variable to zs function

erreps=0.01; %error allowed when comparing the solution to initial position; ~cell size

check all possible directions:

    %%%%%%%%%%%%%%%%%% A) travelling towards +x, +y %%%%%%%%%%%%

... (th>0 && th<=pi) AND (sin(phi) > 0.0 && cos(phi) >0.0 ) ...

[ys,fval,exitflag,outinfo]=fzero(@(y) zpy(y,y0,z0,th,phi)-zs(x0+(y-y0)/tan(phi),y,Ax,fx,Ay,fy), y0+3\*pi/2.); %+2pi = as far as the solution can be from y0

xs=x0+(ys-y0)\*tan(phi);

q=zs(xs,ys,Ax,fx,Ay,fy);

... check if the only intersection point is the initial position or other possibilities of line-surface intersection could go wrong...

        %%%%%%%%%%%%%%%%%% B) travelling towards -x, +y %%%%%%%%%%%%

... (th>0 && th<=pi) AND (sin(phi) > 0.0 && cos(phi) <0.0 ) ...

[ys,fval,exitflag,outinfo]=fzero(@(y) zpy(y,y0,z0,th,phi)-zs(x0+(y-y0)/tan(phi),y,Ax,fx,Ay,fy), y0+3\*pi/2.); %+2pi = as far as the solution can be from y0

xs=x0+(ys-y0)\*tan(phi);

q=zs(xs,ys,Ax,fx,Ay,fy);

... check if the only intersection point is the initial position or other possibilities of line-surface intersection could go wrong...

        %%%%%%%%%%%%%%%%%% C) travelling towards -x, -y %%%%%%%%%%%%

... (th>0 && th<=pi) AND (sin(phi) < 0.0 && cos(phi) <0.0 ) ...

[ys,fval,exitflag,outinfo]=fzero(@(y) zpy(y,y0,z0,th,phi)-zs(x0+(y-y0)/tan(phi),y,Ax,fx,Ay,fy), y0-3\*pi/2.); %+2pi = as far as the solution can be from y0

xs=x0+(ys-y0)\*tan(phi);

q=zs(xs,ys,Ax,fx,Ay,fy);

... check if the only intersection point is the initial position or other possibilities of line-surface intersection could go wrong...

        %%%%%%%%%%%%%%%%%% D) travelling towards +x, -y %%%%%%%%%%%%

... (th>0 && th<=pi) AND (sin(phi) < 0.0 && cos(phi) >0.0 ) ...

[ys,fval,exitflag,outinfo]=fzero(@(y) zpy(y,y0,z0,th,phi)-zs(x0+(y-y0)/tan(phi),y,Ax,fx,Ay,fy), y0-3\*pi/2.); %+2pi = as far as the solution can be from y0

xs=x0+(ys-y0)\*tan(phi);

q=zs(xs,ys,Ax,fx,Ay,fy);

... check if the only intersection point is the initial position or other possibilities of line-surface intersection could go wrong...

else %if (th<0 OR th>pi)

    q=initz0;

    xs=x0+(q-z0)\*(cos(phi)\*tan(th));

    ys=y0+(q-z0)\*(sin(phi)\*tan(th));

    redep=7;

end

%%%%%%%%%%%%% THERE MIGHT BE MORE EXCEPTION CASES %%%%%%%%%%%%%

%%%%%%%%%%%%%% WITH PHI - THETA - AX- AY  %% TBD %%%%%%%%%%%%%%

%%SKIP CALCULATION OF THE ANGLE AT THE INTERSECTING POINT

output(1)=xs;

output(2)=ys;

output(3)=q;

output(4)=redep;

end

--------------- plot\_sputtering\_and\_reflection---------------

'plotting reflection and sputtering...'

%due to the way 3D surface plots auto-read axes from matrices, need to

%transpose each [x,y] matrix. Do within plot to not double variable names

%4-sputtering

'      ... 4: sputtering';

%4a-angular contribution to sputtering

'      ...   a: angular contribution to sputtering (wrt peak in angle)';

surf(xg,yg,transpose(Y\_angle));

caxis([0,1]);

title('4a: angular contribution to sputtering (wrt max)')

xlabel('x')

ylabel('y')

zlabel('sputt Y\_angle')

print('4a\_Yangle','-dpng')

%4b-average sputtering yield of each cell

'      ...   b: average sputtering yield of each cell';

surf(xg,yg,transpose(Y\_loc));

caxis([0,max(max(Y\_loc))]);

title('4b: average sputtering yield of each cell')

xlabel('x')

ylabel('y')

zlabel('ave sputt Y')

print('4b\_avesputtering','-dpng')

%4c-total erosion of each cell, or Ncounts hitting the surface

'      ...   c: total erosion of each cell (N particles)';

surf(xg,yg,transpose(cellNeros));

caxis([0,max(max(cellNeros))]);

title('4c: total erosion of each cell (N particles)')

xlabel('x')

ylabel('y')

zlabel('erosion (N particles)')

print('4c\_Nerosion','-dpng')

'      ... gross erosion (sput yield \* N impacts) = ', sum(sum(cellNeros));

'      ... gross erosion yield (wrt #impacts) = ', sum(sum(cellNeros))/NP;

%4d-total erosion of each cell

'      ...   c: total erosion (flux) of each cell';

surf(xg,yg,transpose(cellFeros));

caxis([0,max(max(cellFeros))]);

title('4d: total erosion (flux) of each cell')

xlabel('x')

ylabel('y')

zlabel('total erosion (flux)')

print('4d\_erosionflux','-dpng')

%5: reflection

'      ... 5: reflection';

%5a-angular contribution of reflection

'      ...   a: angular contribution to reflection (wrt peak in angle)';

surf(xg,yg,transpose(RN\_angle));

title('5a: angular contribution to reflection (wrt max)')

xlabel('x')

ylabel('y')

zlabel('refl yield, RN\_angle')

print('5a\_Rangle','-dpng')

%5b-average reflection yield of each cell

'      ...   b: average reflection yield of each cell';

surf(xg,yg,transpose(RN\_loc));

caxis([0,1.0]); %caxis([0,max(max(RN\_loc))]);

title('5b: average reflection yield of each cell')

xlabel('x')

ylabel('y')

zlabel('refl yield, RN')

print('5b\_avereflection','-dpng')

%5c-total reflection from each cell (N particles)

='      ...   c: total reflection from each cell (N particles)';

surf(xg,yg,transpose(cellNrefl));

caxis([0,max(max(cellNrefl))]);

title('5c: total reflection from each cell (N particles)')

xlabel('x')

ylabel('y')

zlabel('total reflection (N particles)')

print('5c\_Nreflection','-dpng')

%5d-total reflection from each cell (flux)

'      ...   d: total particle reflection (flux) from each cell';

contour(xg,yg,transpose(zg),1); %PLOT?

surf(xg,yg,transpose(cellFrefl));

caxis([0,max(max(cellFrefl))]);

title('5d: total particle reflection (flux) from each cell')

xlabel('x')

ylabel('y')

zlabel('total particle reflection (flux)')

print('5d\_reflectionflux','-dpng')

%6: emission by sputtering

'      ... 6: particles emitted by sputtering';

%6a-number of particles emitted by sputtering

'      ...   a: number of particles';

surf(xg,yg,transpose(nSppart));

caxis([0,max(max(nSppart))]);

title('6a: number of particles emitted by sputtering')

xlabel('x')

ylabel('y')

zlabel('number of particles ')

print('6a\_Nemittsputt','-dpng')

%6b-energy of particles emitted by sputtering

'      ...   b: average energy out (unused)';

surf(xg,yg,transpose(sum(SpEout,3)/size(SpEout,3)));

caxis([0,max(max(max(SpEout)))]);

title('6b: average energy of particles emitted by sputtering')

xlabel('x')

ylabel('y')

zlabel('energy ')

print('6b\_aveenergyemittsputt','-dpng')

%6c-theta angle of particles emitted by sputtering

'      ...   c: Average Theta out';

contour(xg,yg,transpose(zg),1); %PLOT?

surf(xg,yg,transpose(180/pi\*aveSpThout));

caxis([0,180]);

title('6c: average theta of particles emitted by sputtering')

xlabel('x')

ylabel('y')

zlabel('theta')

print('6c\_avethetaemittsputt','-dpng')

%6d-average theta angle of emitted particle, wrt surf normal

'      ...   d: ave theta wrt surf normal';

surf(xg,yg,transpose(180/pi\*aveSpThout\_loc));

caxis([0,90]); %180?

title('6d: ave theta angle wrt surf norm')

xlabel('x')

ylabel('y')

zlabel('theta')

print('6d\_avethetaemittsputt\_nsurf','-dpng')

%6e-phi angle of particles emitted by sputtering

'      ...   e: Average Phi out';

surf(xg,yg,transpose(180/pi\*aveSpPhiout));

caxis([-max(max(-180/pi\*aveSpPhiout)),max(max(180/pi\*aveSpPhiout))]);

title('6e: average phi of particles emitted by sputtering')

xlabel('x')

ylabel('y')

zlabel('phi')

print('6e\_avephiemittsputt','-dpng')

%6f-average phi angle of emitted particle, wrt surf normal

'      ...   f: phi wrt surf normal';

surf(xg,yg,transpose(180/pi\*aveSpPhiout\_loc));

caxis([-max(max(-180/pi\*aveSpPhiout\_loc)),max(max(180/pi\*aveSpPhiout\_loc))]);

title('6f: phi wrt surf normal')

xlabel('x')

ylabel('y')

zlabel('phi')

print('6f\_avephiemittsputt\_nsurf','-dpng')

%7: emission by reflection

'      ... 7: particles emitted by reflection';

%7a-number of particles emitted by sputtering

'      ...   a: number of particles';

surf(xg,yg,transpose(nRpart));

caxis([0,max(max(nRpart))]);

title('7a: number of particles emitted by reflection')

xlabel('x')

ylabel('y')

zlabel('number of particles ')

print('7a\_Nemittrefl','-dpng')

%7b-energy of particles emitted by reflection

'      ...   b: energy out (unused)';

surf(xg,yg,transpose(REout));

caxis([0,max(max(REout))]);

title('7b: energy of particles emitted by reflection')

xlabel('x')

ylabel('y')

zlabel('energy')

print('7b\_aveenergyemittrefl','-dpng')

%7c-theta of particles emitted by reflection

'      ...   c: Average Theta out';

surf(xg,yg,transpose(180/pi\*aveRThout));

caxis([0,180]);

title('7c: average theta of particles emitted by reflection')

xlabel('x')

ylabel('y')

zlabel('average theta')

print('7c\_avethetaemittrefl','-dpng'

%7d-average theta angle of emitted particle, wrt surf normal

'      ...   d: ave theta wrt surf normal';

surf(xg,yg,transpose(180/pi\*aveRThout\_loc));

caxis([0,90]); %180?

title('7d: theta angle wrt surf norm')

xlabel('x')

ylabel('y')

zlabel('theta')

print('7d\_avethetaemittrefl\_nsurf','-dpng')

%7e-phi of particles emitted by reflection

'      ...   e: Phi out';

surf(xg,yg,transpose(180/pi\*(RPhiout)));

caxis([-max(max(-180/pi\*RPhiout)),max(max(180/pi\*RPhiout))]);

title('7e: phi of particles emitted by reflection')

xlabel('x')

ylabel('y')

zlabel('phi')

print('7e\_avephiemittrefl','-dpng')

%7f-average phi angle of emitted particle, wrt surf normal

'      ...   f: phi wrt surf normal';

surf(xg,yg,transpose(180/pi\*RPhiout\_loc));

caxis([-max(max(-180/pi\*RPhiout\_loc)),max(max(180/pi\*RPhiout\_loc))]);

title('7f: phi wrt surf normal')

xlabel('x')

ylabel('y')

zlabel('phi')

print('7f\_avephiemittrefl\_nsurf','-dpng')

%8-redeposition

'      ... 8: redeposition';

%8a-number of particles redeposited after sputtering

'      ...   a: number of particles redeposited after sputtering';

surf(xg,yg,transpose(NredepSp));

caxis([0,max(max(NredepSp))]);

title('8a: number of redeposited particles (from sput)')

xlabel('x')

ylabel('y')

zlabel('# impacts')

print('8a\_redepossputt','-dpng')

' ... total redeposition (from sput) = ', sum(sum(NredepSp))

' ... fraction redeposited (from sput) = ',

sum(sum(NredepSp))/sum(sum(nSppart))

%8b-number of particles redeposited after reflection

'      ...   b: number of particles redeposited after reflection';

surf(xg,yg,transpose(NredepR));

caxis([0,max(max(NredepR))]);

title('8b: number of redeposited particles (from refl)')

xlabel('x')

ylabel('y')

zlabel('# impacts')

print('8b\_redeposrefl','-dpng')

' ... total redeposition (from refl) = ', sum(sum(NredepR));

' ... fraction redeposited (from refl) = ',

sum(sum(NredepR))/sum(sum(nRpart))

%8c-total number of particles redeposited

'      ...   c: total number of particles redeposited';

surf(xg,yg,transpose(NredepR+NredepSp));

caxis([0,max(max(NredepR+NredepSp))]);

title('8c: total number of redeposited particles')

xlabel('x')

ylabel('y')

zlabel('# impacts')

print('8c\_totalNredepos','-dpng')

%8d-net erosion (sput - redep by sput)

'      ...   d: net erosion (sput - redep by sput)';

surf(xg,yg,transpose(nSppart-NredepSp));

caxis([-max(max(-nSppart+NredepSp)),max(max(nSppart-NredepSp))]);

title('8d: net erosion (sput - redep by sput)')

xlabel('x')

ylabel('y')

zlabel('# net erosion')

print('8d\_neteros','-dpng')

' ... total net erosion (gross eros - redep by sput) = ',

sum(sum(nSppart-NredepSp))

' ... net erosion yield (wrt #impacts) = ',

sum(sum(nSppart-NredepSp))/NP

'      ... net erosion yield (gross eros yield \* (1- redep fract) = ',

sum(sum(cellNeros))/NP\*(1-sum(sum(NredepSp))/sum(sum(nSppart))))

**APPENDIX II: SCREEN OUTPUT EXAMPLE**

OUTPUT LOOKS LIKE:

plotting incoming particles output...

... 1: surface

... a: morphology

... b: normalized cell area (wrt flat surf)

... c: theta angle of cells normal

... d: phi angle of cells normal

... 2: angle

... a: at each impact point

... b: average impact angle of each cell

... c: profile along x (at y=0.7854+/-0.5)

... d: profile along y (at x=0.7854+/-0.5)

... e: cumulative distribution of angles

... 3: flux

... a: number of impacts per cell

... b: normalized local flux

... c: profile of number of impacts along x (integrated along y, normalized)

... d: profile of number of impacts along y (integrated along x, normalized)

... e: cumulative distribution of histogram

done!

erosion:

reflection:

Energy reflection:

---- calling particle emission -----

re-deposition: 0: YES, particle re-deposited in final position

1-6: NO; Final position = projection of trajectory in initial z0:

1: (x0,y0)=(xs,yx) 2: err in z-direction 3: err in x-direction 4: err in y-direction;

5: invalid exitflag 6: fzero(Exitflag)=6 7: invalid theta

...10% done

...20% done

...30% done

...40% done

...50% done

...60% done

...70% done

...80% done

...90% done

...100% done

---- particle emission done -----

plotting reflection and sputtering...

... 4: sputtering

... a: angular contribution to sputtering (wrt peak in angle)

... b: average sputtering yield of each cell

... c: total erosion of each cell (N particles)

... gross erosion (sput yield \* N impacts) = 535.7805

... gross erosion yiled (wrt #impacts) = 0.0053578

... c: total erosion (flux) of each cell

... 5: reflection

... a: angular contribution to reflection (wrt peak in angle)

... b: average reflection yield of each cell

... c: total reflection from each cell (N particles)

... d: total particle reflection (flux) from each cell

... 6: particles emitted by sputtering

... a: number of particles

... b: average energy out (unused)

... c: Average Theta out

... d: ave theta wrt surf normal

... e: Average Phi out

... f: phi wrt surf normal

... 7: particles emitted by reflection

... a: number of particles

... b: energy out (unused)

... c: Average Theta out

... d: ave theta wrt surf normal

... e: Phi out

... f: phi wrt surf normal

... 8: redeposition

... a: number of particles redeposited after sputtering

... total redeposition (from sput) = 133

... fraction redeposited (from sput) = 0.24767

... b: number of particles redeposited after reflection

... total redeposition (from refl) = 6376

... fraction redeposited (from refl) = 0.092071

... c: total number of particles redeposited

... d: net erosion (sput - redep by sput)

... total net erosion (gross eros - redep by sput) = 404

... net erosion yield (wrt #impacts) = 0.00404

... net erosion yield (gross eros yield \* (1- redep fract) = 0.0040308

done!