Problem 1

$$\frac{1}{2}\int_{-1}^{1}\frac{1-h^2}{(1+h^2-2h\mu_0)^{3/2}}\,d\mu_0 \qquad 1+h^2-2h\mu_0 = P$$

$$-2h\,d\mu = dP$$

$$d\mu = \frac{dP}{-2h}$$

$$= \frac{1}{4}\frac{1-h^2}{-h}\int \frac{1}{P^{3/2}}\,dP$$

$$= \frac{1}{4}\frac{(1-h^2)}{-h}\frac{-2}{\sqrt{P}}\Bigg| = \frac{1-h^2}{2h}\frac{1}{\sqrt{1+h^2-2h\mu_0}}\Bigg|_{-1}^{1}$$

$$= \frac{1-h^2}{2h}\left(\frac{1}{\sqrt{1+h^2-2h}} - \frac{1}{\sqrt{1+h^2+2h}}\right)$$

$$= \frac{1-h^2}{2h}\left(\frac{1}{\pm(h-1)} - \frac{1}{\pm(h+1)}\right)$$

$$= \frac{1-h^2}{2h}\left(\overset{\text{minus}}{\frac{1}{(1-h)}} - \overset{\text{plus}}{\frac{1}{h+1}}\right)$$

$$= \frac{1-h^2}{2h}\cdot\frac{h+1-1+h}{1-h^2} = \boxed{1}$$

First Moment

$$\frac{1}{2}\int_{-1}^{1}\frac{(1-h^2)\mu_0}{(1+h^2-2h\mu_0)^{3/2}}\,d\mu_0 = \frac{1}{2}\int_{(1+h)^2}^{(h-1)^2}\frac{(1-h^2)}{P^{3/2}}\frac{dP}{-2h}\mu_0 \qquad \frac{P-1-h^2}{-2h} = \mu_0$$

$$= \frac{(1-h^2)}{8h^2}\int_{(1+h^2)}^{(1-h^2)}\frac{P}{P^{3/2}} - \frac{1}{P^{3/2}} - \frac{h^2}{P^{3/2}} = \frac{(1-h^2)}{8h^2}\left(2\sqrt{P}\Big| + \frac{2}{\sqrt{P}}\Big| + \frac{2h^2}{\sqrt{P}}\Big|\right)$$

$$= \frac{(1-h^2)}{4h^2}\left(\overset{\text{minus}}{\pm(h-1)} - \overset{\text{Plus}}{\pm(1+h)} + \overset{\text{minus}}{\frac{1}{\pm(h-1)}} - \overset{\text{Plus}}{\frac{1}{\pm(1+h)}} + \overset{\text{minus}}{\frac{h^2}{\pm(h-1)}} - \overset{\text{Plus}}{\frac{h^2}{\pm(1+h)}}\right)$$

$$= \frac{(1-h^2)}{4h^2}\left(-2h + \frac{2h}{1-h^2} + \frac{2h^3}{1-h^2}\right) = h$$

$$\mu_o^2 = \frac{1}{4h^2}\left[P^2 - 2P(1+h^2) + h^4 + 2h^2 + 1\right]$$

Can I call it a second moment without a tensor?

Second Moment: $\frac{1}{2}\int_{-1}^{1} \frac{(1-h^2)\mu_o^2}{(1+h^2-2h\mu_o)^{3/2}}\, d\mu_o$

case① $\pm$

case② $\pm$ +

case③ $\mp$

case④ $=$

$$= \frac{(h^2-1)}{16h^3}\left[\frac{2}{3}P^{3/2}\Big|_{(h+1)^2}^{(h-1)^2} + 4(1+h^2)\sqrt{P}\Big|_{(h+1)^2}^{(h-1)^2} - \frac{2(h^2+1)^2}{\sqrt{P}}\Big|_{(h+1)^2}^{(h-1)^2}\right]$$

$$= \frac{(h^2-1)}{24h^3}\left[\underbrace{P^{3/2}\Big|}_{\text{term①}} + \underbrace{6(1+h^2)\sqrt{P}\Big|}_{\text{term②}} - \underbrace{\frac{3(h^2+1)^2}{\sqrt{P}}\Big|}_{\text{term③}}\right]$$

	Case①	Case②	Case③	Case④
Term ①	$2h^3+6h$	$-6h^2-2$	$-2h^3-6h$	$6h^2+2$
$\times h^2-1$	$2h^5+4h^3-6h$	$-6h^4+4h^2+2$	$-2h^5-4h^3+6h$	$6h^4-4h^2-2$
Term ②	$12h^3+12h$	$-12h^2-12$	$-12h^3-12h$	$12h^2+12$
$\times h^2-1$	$12h^5-12h$	$-12h^4+12$	$-12h^5+12h$	$12h^4-12$
Term ③	$\frac{6h^5+12h^3+6h}{(h^2-1)}$	$\frac{6h^4+12h^2+6}{(h^2-1)}$	$\frac{-6h^5-12h^3-6h}{(h^2-1)}$	$\frac{-6h^4-12h^2-6}{(h^2-1)}$

$$\frac{1}{24h^3}\left[12\cancel{h^5} - \cancel{12}h + 6h^5 + 12h^3 + \cancel{6h} - 2\cancel{h^5} - 4h^3 + \cancel{6h}\right]$$

$$\frac{1}{3}\left[2h^2+1\right]$$ ~ sorry couldn't get h^2

$$\int_{-1}^{1} K(\mu_o, v' \to v)\, d\mu = 1 \; ; \; \int_{-1}^{1} K(\mu_o, v' \to v)\mu_o\, d\mu_o = h$$

$$\int_{-1}^{1} K(\mu_o, v' \to v)\mu_o^2\, d\mu = \tfrac{1}{3}\left[2h^2+1\right]$$

$$K_\ell(E' \to E) = \int_{-1}^{1} d\mu_0 \, K(\mu_0, E' \to E) \frac{1}{2^\ell \ell!} \frac{d^\ell}{d\mu_0^\ell} \left[(\mu_0^2 - 1)^\ell \right]$$

$$K_\ell = \int \frac{K}{2^\ell \ell!} \frac{d^\ell}{d\mu_0^\ell} \left[(\mu_0^2 - 1)^\ell \right]$$

$$K_0 = \int K = \boxed{1}$$

$$K_1 = \int \frac{K}{2} 2\mu_0 = \int K \mu_0 = \boxed{h}$$

$$K_2 = \int \frac{K}{8} \frac{d^2}{d\mu_0^2} \left(\mu_0^4 - 2\mu_0^2 + 1 \right)$$

$$= \int \frac{K}{8} \frac{d}{d\mu_0} \left(4\mu_0^3 - 4\mu_0 \right)$$

$$= \int \frac{K}{2} \left(3\mu_0^2 - 1 \right)$$

$$= \frac{3}{2} \cdot \frac{1}{3} \left[2h^2 + 1 \right] - \frac{1}{2}$$

$$= h^2 + \frac{1}{2} - \frac{1}{2}$$

$$= \boxed{h^2}$$ — woo hoo!

Problem 2

60/670 DUDE   I'm not sure what you are asking for

If $Q=0$ little green book LGB pg 90

$V_{CM} = V_{CM}'$

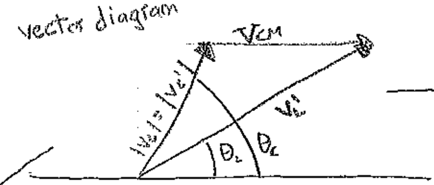
neutron scattering from stationary nuclei

$V_L = 0$, $m=1$; $M=A$

LGB Pg.90 $\quad V_{CM} = V_{CM}' = \dfrac{mV_L + MV_L}{m+M} = \dfrac{V_L}{1+A}$ $\quad \boxed{V_{CM} = \left(\dfrac{1}{1+A}\right) V_L}$

vector diagram

61/670



$\boxed{V_c' = V_c = -V_{CM} = -\left(\dfrac{1}{A+1}\right)V_L}$

Conservation Dude 61/670

$V_c' = V_c = \dfrac{A}{A+1} V_L \quad V_L = V_{CM}$

$\rightarrow V_L' \sin\theta_L = V_c' \sin\theta_c$

$V_L' \cos\theta_L \, d\theta_L = V_c' \cos\theta_c \, d\theta_c$

$\tan\theta_L \dfrac{d\theta_c}{d\theta_L} = \dfrac{\sin\theta_c}{\cos\theta_c}$

$V_L' \sin\theta_L = V_c' \sin\theta_c$

$V_L' \cos\theta_L = V_c' \cos\theta_c + V_{CM}$

Divide

$\tan\theta_L = \dfrac{V_c' \sin\theta_c}{V_{CM} + V_c' \cos\theta_c} = \dfrac{\sin\theta_c}{\frac{1}{A} + \cos\theta_c}$

$\sigma_L(\theta_L)\sin\theta_L \, d\theta_L = \sigma_{CM}(\theta_c)\sin\theta_c \, d\theta_c$

$\sigma_L(\theta_L) = \sigma_{CM}(\theta_c) \dfrac{\sin\theta_c}{\sin\theta_L} \dfrac{d\theta_c}{d\theta_L} = \sigma_{CM}(\theta_c) \dfrac{V_L'}{V_c'} \dfrac{\sin\theta_c}{\cos\theta_c} \cdot \dfrac{1}{\tan\theta_L}$

$= \sigma_{CM}(\theta_c) \dfrac{V_L'}{V_L} \cdot \dfrac{(A+1)}{A} \cdot \dfrac{1}{\cos\theta_c} \cdot \left(\frac{1}{A} + \cos\theta_c\right)$

$V_L'^2 = (V_{CM} + V_c \cos\theta_c)^2 + V_c^2 \sin\theta_c^2$

$V_L'^2 = V_{CM}^2 + V_c^2 + 2V_{CM} V_c \cos\theta_c$

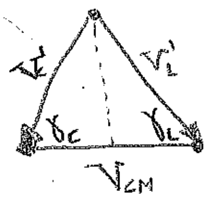
$\dfrac{V_L'^2}{V_L^2} = \left(\dfrac{1}{A+1}\right)^2 + \left(\dfrac{A}{A+1}\right)^2 + 2 \cdot \left(\dfrac{1}{A+1}\right)\left(\dfrac{A}{A+1}\right)\cos\theta_c \rightarrow \boxed{\dfrac{V_L'}{V_L} = \dfrac{\sqrt{A^2+1+2A\cos\theta_c}}{A+1}}$

$\sigma_L(\theta_L) = \sigma_{CM}(\theta_c) \sqrt{A^2+1+2A\cos\theta_c} \cdot \dfrac{1}{A}\left(\dfrac{1}{A}\cos\theta_c + 1\right)$

$= \sigma_{CM}(\theta_c) \sqrt{\frac{1}{A^2} + \frac{2\cos\theta_c}{A} + 1} \left(\frac{1}{A}\cos\theta_c + 1\right)$

$\approx \dfrac{\sigma_{CM}(\theta_c)\left(\frac{1}{A^2} + \frac{2}{A}\cos\theta_c + 1\right)^{3/2}}{1 + \frac{1}{A}\cos\theta_c}$ $\rightarrow$ if you say $\frac{1}{A^2}\cos\theta_c^2 \approx \frac{1}{A^2}$

The other Angle



$$\longrightarrow V_c' \sin\gamma_c = V_L' \sin\gamma_L$$

$$V_c' \cos\gamma_c \, d\gamma_c = V_L' \cos\gamma_L \, d\gamma_L$$

$$\frac{\sin\gamma_c}{\cos\gamma_c} = \frac{d\gamma_c}{d\gamma_L} \tan\gamma_L$$

$$V_L' \sin\gamma_L = V_c' \sin\gamma_c \qquad \xrightarrow{\text{divide}} \qquad \tan\gamma_L = \frac{V_c' \sin\gamma_c}{V_{CM} - V_c' \cos\gamma_c}$$

$$V_L' \cos\gamma_L = V_{CM} - V_c' \cos\gamma_c$$

$$= \frac{-\sin\gamma_c}{1+\cos\gamma_c}$$

Law of cosines

$$V_L'^2 = V_c'^2 + V_{CM}^2 - 2 V_c' V_{CM} \cos\gamma_c$$

$$\frac{V_L'^2}{V_L^2} = \frac{1}{(A+1)^2} + \frac{1}{(A+1)^2} - \frac{2}{(A+1)^2} \cos\gamma_c$$

$$\frac{V_L'}{V_L} = \frac{\sqrt{2(1-\cos\gamma_c)}}{(A+1)}$$

$$\sigma_L(\gamma_L) \sin\gamma_L \, d\gamma_L = \sigma_{CM}(\gamma_c) \sin\gamma_c \, d\gamma_c$$

$$\sigma_L(\gamma_L) = \sigma_{CM}(\gamma_c) \frac{\sin\gamma_c}{\sin\gamma_L} \frac{d\gamma_c}{d\gamma_L} = \sigma_{CM}(\gamma_c) \frac{V_L'}{V_c'} \frac{\sin\gamma_c}{\cos\gamma_c} \frac{1}{\tan\gamma_L}$$

$$= \sigma_{CM}(\gamma_c) \frac{V_L'}{V_L} (A+1) \frac{(1+\cos\gamma_c)}{\cos\gamma_c}$$

$$= \sigma_{CM}(\gamma_c) \sqrt{2(1-\cos\gamma_c)} \left(1/\cos\gamma_c + 1 \right)$$

$$\cos\theta_L = \frac{V_c' \cos\theta_c + V_{CM}}{V_L'} = \frac{V_c'}{V_L'}\cos\theta_c + \frac{V_{CM}}{V_L'}$$

$$= \left(\frac{A\cos\theta_c + 1}{A+1}\right)\frac{V_L}{V_L'}$$

$\beta = A \quad \mu = \cos\theta_c$

$$= \frac{A\cos\theta_c + 1}{A+1} \frac{A+1}{\sqrt{A^2+1+2A\cos\theta_c}}$$

$$\cos\theta_L = \frac{1+\beta\mu}{\sqrt{\beta^2+1+2\beta\mu}}$$

$$= \frac{A\cos\theta_c + 1}{\sqrt{A^2+1+2A\cos\theta_c}} =$$

$$\cos\gamma_L = \frac{V_{CM} - V_c'\cos\gamma_c}{V_L'} = \frac{V_L}{V_L'}\frac{(1-\cos\gamma_c)}{(A+1)} = \frac{(1-\cos\gamma_c)}{\sqrt{2(1-\cos\gamma_c)}}$$

$$\cos\gamma_L = \frac{1-\gamma\mu}{\sqrt{\beta^2+1-2\gamma\mu}}; \quad \gamma = 1, \; \mu = \cos\gamma_c$$

$$\frac{\cos\theta_L \cdot V_L' - V_{CM}}{V_c'} = \cos\theta_c = \frac{\cos\theta_L V_L' - \frac{1}{(1+A)}V_L}{A/_{A+1}V_L}$$

$$\cos\theta_c = \cos\theta_L \frac{V_L'}{V_L}\frac{(A+1)}{A} - 1/A$$

Conservation of momentum. In y direction LAB

$$\frac{V_L'}{V_L} = \frac{\sin\theta_c}{\sin\theta_L}\frac{A}{A+1} \qquad V_L'\sin\theta_L = V_L \sin\gamma_L A$$

$= \cos\theta_c$

Assuming: $K(\mu, E' \to E) = P(\theta_c) \cdot P(E' \to E)$

where $P(\theta_c) = \frac{\sigma_{CM}(\theta_c)}{\sigma_s} = \frac{\sigma_s}{4\pi} \cdot \frac{1}{\sigma_s} = \frac{1}{4\pi}$ s-wave

& $P(E' \to E) = \frac{1}{(1-\alpha)E'}$ $\alpha E' \leq E \leq E'$

I am honestly not sure this is correct. or this

Integrate over energy & Angle $\sim 2\pi \int_{\alpha E_i}^{E_i} dE_f \int_{-1}^{1} d\mu \, P(\theta_c) P(E_i \to E_f)$

$$= \frac{4\pi}{4\pi} \cdot \frac{1}{(1-\alpha)E_i} \cdot E_f \Big|_{\alpha E_i}^{E_i} = 1$$

$\sigma_L(\theta_L) \sin\theta_L \, d\theta_L = \sigma_{CM} \sin\theta_c \, d\theta_c$

$\sigma_L \mu_L d\mu_L = \sigma_c \mu_c d\mu_c$

$$\sigma_c \sqrt{\tfrac{1}{A^2} + \tfrac{2\mu_c}{A} + 1} \left(\tfrac{1}{A}\mu_c + 1\right) \cdot \frac{(A\cos\theta_c + 1)}{\sqrt{A^2 + 1 + 2\mu_c}} \, d\mu_L = \sigma_c \mu_c d\mu_c$$

$(\tfrac{1}{A}\mu_c + 1)(A\mu_c + 1) \, d\mu_L = \mu_c d\mu_c$

$\mu_c^2 + A\mu_c + \tfrac{1}{A}\mu_c + 1 \, . \, d\mu_L = \mu_c d\mu_c$

$\mu_c^2 + \frac{A^2+1}{A}\mu_c + 1$

sometimes a problem in Right_Direction Subroutine when starting in the middle of an assembly

$s_3 \quad d_3 \quad s_2 \quad d_2 \quad s_1 \quad d_1 \quad s_0$

3.) $$\hat{\Omega}\cdot\nabla\psi + \Sigma_t\psi = \frac{Q}{4\pi}$$

$$\left(\Omega_x\frac{\partial}{\partial x} + \Omega_y\frac{\partial}{\partial y}\right) + \Sigma_t\psi = \frac{Q}{4\pi}$$

$$\frac{d\psi}{ds} + \Sigma_t\psi = Q/4\pi$$

$$\frac{d}{ds}\left(e^{\Sigma_t s}\psi\right) = \frac{Q}{4\pi}e^{\Sigma_t s}$$

---

track 1 

$$\int_{-\infty}^{s_0} = \int_{-\infty}^{s_1} ds + \int_{s_1}^{s_0} ds$$

$$\int_{s_1}^{s_0} ds \rightarrow e^{\Sigma_t s}\psi\Big|_{s_1}^{s_0} = 0 \rightarrow e^{\Sigma_t s_0}\psi(s_0) - e^{\Sigma_t s_1}\psi(s_1) = 0$$

$$\boxed{\psi(s_0,\hat{\Omega}) = \psi(s_1,\hat{\Omega})e^{-\Sigma_t^{M} d_1}}$$

---

track 2 

$$\int_{s_2}^{s_1} ds \rightarrow e^{\Sigma_t s_1}\psi(s_1) - e^{\Sigma_t s_2}\psi(s_2) = \int_{s_2}^{s_1}\frac{Q}{4\pi}e^{\Sigma_t s}$$

$$e^{\Sigma_t s_1}\psi(s_1) = e^{\Sigma_t^{F} s_2}\psi(s_2) + \frac{Q}{\Sigma_t 4\pi}\left[e^{\Sigma_t s_1} - e^{\Sigma_t s_2}\right]$$

$$\boxed{\psi(s_1) = e^{-\Sigma_t^{F} d_2}\psi(s_2,\hat{\Omega}) + \frac{Q}{\Sigma_t 4\pi}\left[1 - e^{-\Sigma_t^{F} d_2}\right]}$$

---

track 3 

$$\boxed{\psi(s_2) = \psi(s_3,\hat{\Omega})e^{-\Sigma_t^{M} d_3}}$$

$$\psi(s_i,\hat{\Omega}) = \psi(s_{i+1},\hat{\Omega})e^{-\Sigma_t^{i+1} d_{i+1}} + \frac{Q_{i+1}}{\Sigma_t 4\pi}\left[1 - e^{-\Sigma_t^{i+1} d_{i+1}}\right]$$

Set $I = \frac{Q}{4\pi\Sigma_t^F}$

$$\psi(s_0,\hat{\Omega}) = e^{-\Sigma_t^F d_2 - \Sigma_t^M d_1}\,\psi(s_2,\hat{\Omega}) + I\left[e^{-\Sigma_t^M d_1} - e^{-\Sigma_t^F d_2 - \Sigma_t^M d_1}\right]$$

$$\psi(s_0,\hat{\Omega}) = e^{-\Sigma_t^F d_2 - \Sigma_t^M d_1 - \Sigma_t^M d_3}\,\psi(s_3,\hat{\Omega}) + I\left[e^{-\Sigma_t^M d_1} - e^{-\Sigma_t^F d_2 - \Sigma_t^M d_1}\right]$$

Make sure code alternates Between Moderater & fuel

and set:

$$P_i = -\sum_{j=1}^{i} \Sigma_t^j d_j$$

if Σ_t^j = odd $\Rightarrow \Sigma_t^j = \Sigma_t^M$

if Σ_t^j = even $\Rightarrow \Sigma_t^j = \Sigma_t^F$

This Makes

Always positive

$$\psi(s_0,\hat{\Omega}) = e^{P_3}\,\psi(s_3,\hat{\Omega}) + I\left[e^{P_1} - e^{P_2}\right]$$

lets Make Sure we Know our pattern: Fuel is next

$$\psi(s_0,\hat{\Omega}) = e^{P_4}\,\psi(s_4,\hat{\Omega}) + I\left[e^{P_3} - e^{P_4}\right] + I\left[e^{P_1} - e^{P_2}\right]$$

then Moderator:

$$\psi(s_0,\hat{\Omega}) = e^{P_5}\,\psi(s_5,\hat{\Omega}) + I\left[e^{P_3} - e^{P_4}\right] + I\left[e^{P_1} - e^{P_2}\right]$$

then fuel

$$\psi(s_0,\hat{\Omega}) = e^{P_6}\,\psi(s_6,\hat{\Omega}) + I\left[e^{P_5} - e^{P_6}\right] + I\left[e^{P_3} - e^{P_4}\right] + I\left[e^{P_1} - e^{P_2}\right]$$

$$\psi(s_0,\hat{\Omega}) = I\sum_{i=1}^{\infty}\left[(-1)^{i+1} e^{P_i}\right]$$

Stop when P_i is over 9,000!!!

---

if we Know all our d_j ~ this is trivial

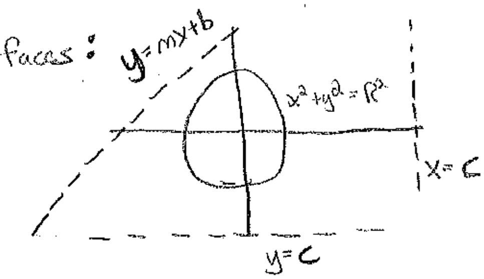
I could Be over simplifying, But I assumed Because the cylinder is infinitely tall that there will be no polar angle dependance ($\pi/2$ & $\pi/8$ are the same)

Need distances $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$; Current location: (x_1, y_1)

Direction: $x' = \cos\theta$; $y' = \sin\theta$

$\hat{\Omega} = \hat{e}_x \cos\theta + \hat{e}_y \sin\theta$

Different Surfaces:



Line Eq. for direction & location

$y - y_1 = m(x - x_1)$

$m = \frac{\sin\theta}{\cos\theta}$ $\quad y = m(x - x_1) + y_1$

$x = \frac{1}{m}(y - y_1) + x_1$

if $\cos\theta = 0$ $\quad x = C = x_1$

if $\sin\theta = 0$ $\quad y = C = y_1$

Points of intersection:

Common Line: $y_L - y_{1L} = m_L(x_L - x_{1L})$

$\cos\theta \neq 0$ $\quad m_L(x - x_{1L}) + y_{1L} = m(x - x_1) + y_1$

$\cos\theta_L \neq 0$ $\quad m_L x - m x = m_L x_{1L} - m x_1 + y_1 - y_{1L}$

$m_L \neq m$ $\quad \boxed{x_2 = \frac{m_L x_{1L} - m x_1 + y_1 - y_{1L}}{m_L - m}} \Rightarrow \boxed{y_2 = m(x_2 - x_1) + y_1}$

$\cos\theta = 0$ $\quad \boxed{x_2 = x_1}$

$\cos\theta_L \neq 0$ $\quad \boxed{y_2 = m_L(x_2 - x_{1L}) + y_{1L}}$

if $\sin\theta = 0 \Rightarrow m = 0$

& $\sin\theta_L = 0 \Rightarrow m_L = 0$

if $\cos\theta = 0$

& $\cos\theta_L = 0$ $\quad$ Parallel no intercept or on top of eachother

$d = 9000$

if $m_L = m$ $\quad$ Same as above

$\cos\theta \neq 0$ $\quad \boxed{x_2 = x_{1L} \Rightarrow y_2 = m(x_2 - x_1) + y_1}$

$\cos\theta_L = 0$

for a circle: $y = \pm\sqrt{R^2 - x^2}$

$x = \pm\sqrt{R^2 - y^2}$ $\quad$ if $\cos\theta = 0$

if $\cos\theta = 0$ $\quad x_2 = x_1 \Rightarrow y_2 = \pm\sqrt{R^2 - x_2^2}$ (if real)

if $\sin\theta = 0$ $\quad y_2 = y_1 \Rightarrow x_2 = \pm\sqrt{R^2 - y_2^2}$ (if real)

otherwise $\quad m(x_2 - x_1) = \pm\sqrt{R^2 - x_2^2}$ $\qquad x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$m^2(x_2^2 - 2x_2x_1 + x_1^2) + x_2^2 - R^2 = 0$

$x_2^2(1 + m^2) + x_2(2x_1m^2) + x_1^2m^2 - R^2 = 0$ $\qquad y = \pm\sqrt{R^2 - x^2}$

How do we know we are going in the right direction?

$$\hat{n} \cdot \langle x_2 - x_1 , y_2 - y_1 \rangle > 0$$

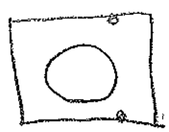
Multiple points on a circle?

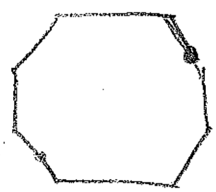
Take the shorter of the two that are heading in the right direction, and are real (not imaginary)

If Both points are real & in the right direction, you could skip the next irritive step

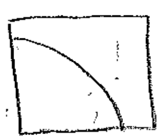
---

translation at Boundaries?

Can either do  translation with the same angle

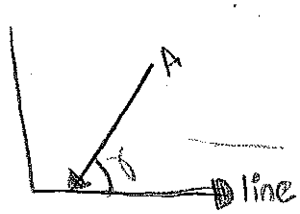


or reflection 90°

 and  keep the same position.

I think reflection would be easier because otherwise you need to hard code some geometric expectations.

With reflection you mirror every time you encounter a line



$A \cdot B = \|A\| \cdot \|B\| \cos\gamma$

Actually reflection is probably wrong. I'm doing translation.

$$y - y_v = m_v(x - x_v) \qquad x^2 + y^2 = R^2$$

$$y = m_v(x - x_v) + y_v$$

$$x^2 + [m_v(x - x_v) + y_v]^2 = R^2$$

$$x^2 + [m_v x - m_v x_v + y_v][m_v x - m_v x_v + y_v] = R^2$$

$$\cancel{x^2} + m_v^2 x^2 - m_v^2 x_v x + m_v x y_v - m_v^2 x_v x + m_v^2 x_v^2 - m_v x_v y_v$$

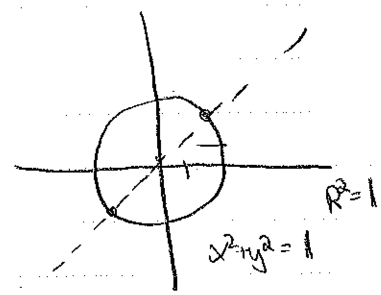
$$y_v m_v x - y_v m_v x_v + y_v^2 = R^2$$

$$x^2[1 + m_v^2] + x[-m_v^2 x_v + m_v y_v - m_v^2 x_v + y_v m_v] + m_v^2 x_v^2 - m_v x_v y_v - m_v x_v y_v + y_v^2 = R^2$$

$a = 1 + m_v^2$

$b = -2m_v^2 x_v + 2m_v y_v$

$c = m_v^2 x_v^2 + y_v^2 - 2m_v x_v y_v - R^2$



$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$y = x$

$y - y_v = m_v(x - x_v)$

$y_v = 0$

$m_v = 1 \quad x_v = 0$

expect points

$(1,1)$ & $(-1,-1)$

$$y = m_v(x_{1,2} - x_v) + y_v$$

$m_v = 1$

$R = 0.41$

$x_v = 0.63$

$y^2[1]$

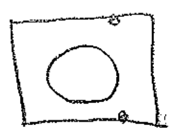
How do we know we are going in the right direction?

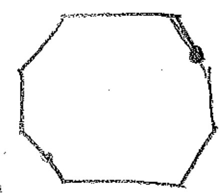
$$\hat{n} \cdot \langle x_2 - x_1, y_2 - y_1 \rangle > 0$$

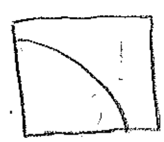
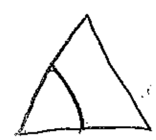
Multiple points on a circle?

take the shorter of the two that are heading in the right direction, and are real (not imaginary)
if Both points are real & in the right direction, you could skip the next iterative step

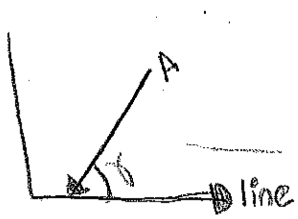
translation at Boundaries?

can either do  translation with the same angle



or reflection 90°  and  keep the same position.

I think reflection would be easier because otherwise you need to hard code some geometric expectations.
With reflection you mirror every time you encounter a line

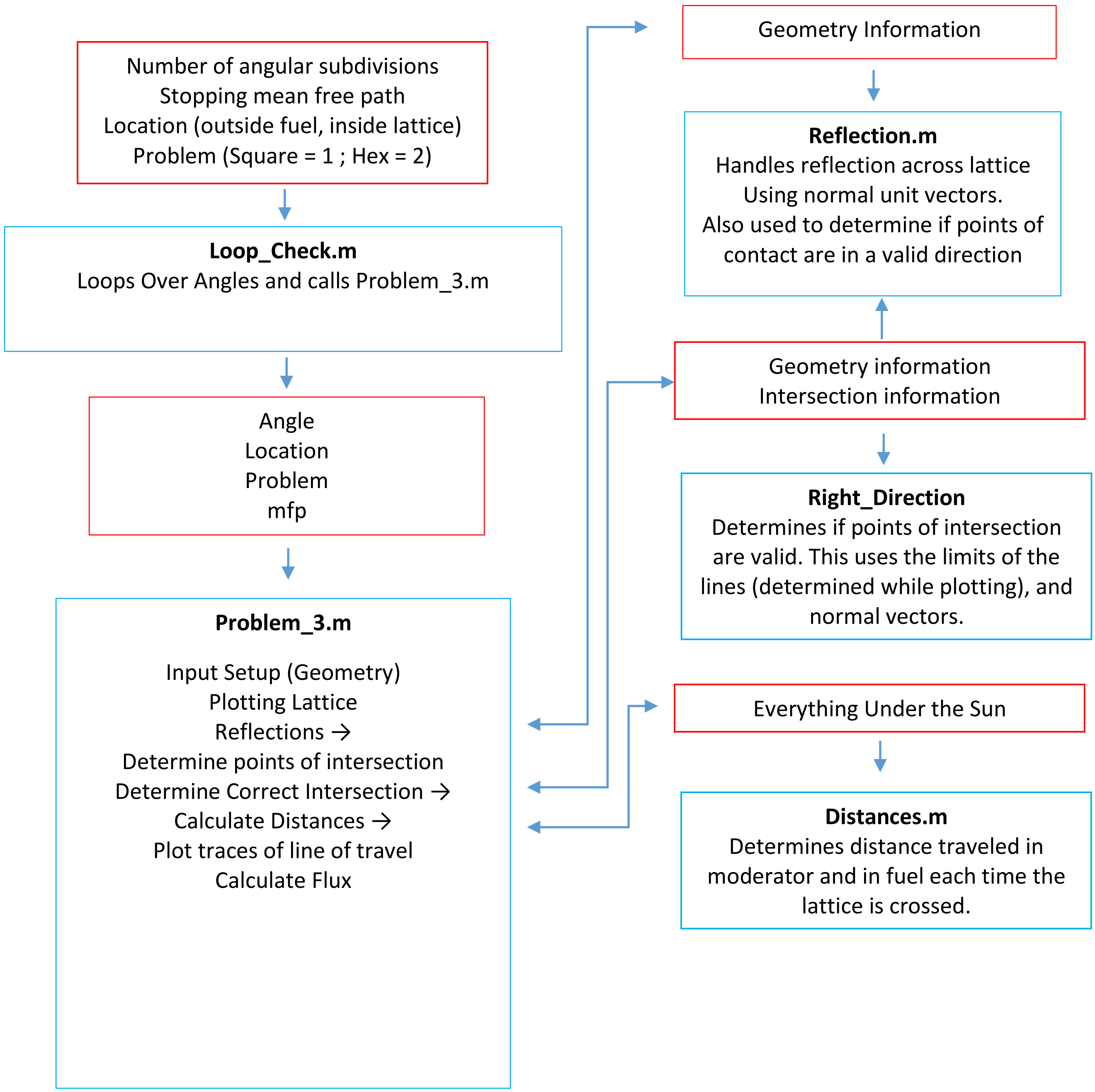


$A \cdot B = \|A\| \cdot \|B\| \cos\gamma$

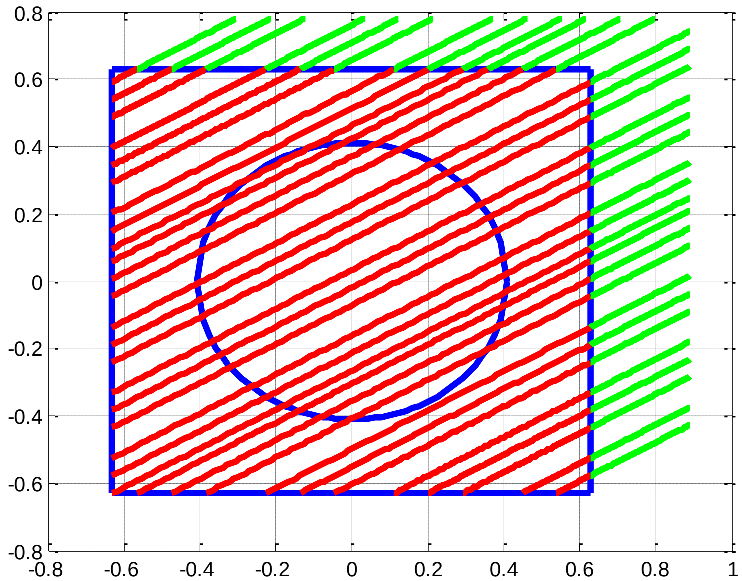
Actually reflection is probably wrong. I'm doing translation.

Problem 3 Continued

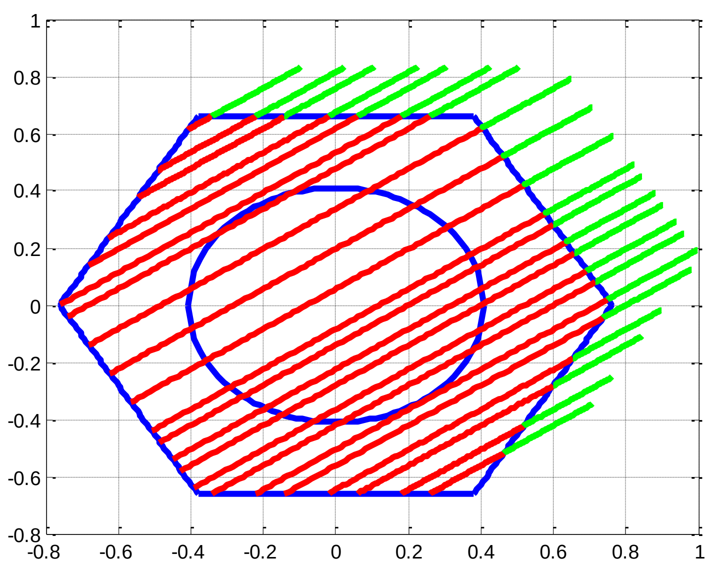
The code was made with several functions in matlab. The pseudo code is depicted below, highlighting program outline, inputs and outputs. Blue boxes depict programs, while red boxes coupled with arrows depict data flow.



Examples of visual tracing with the program.

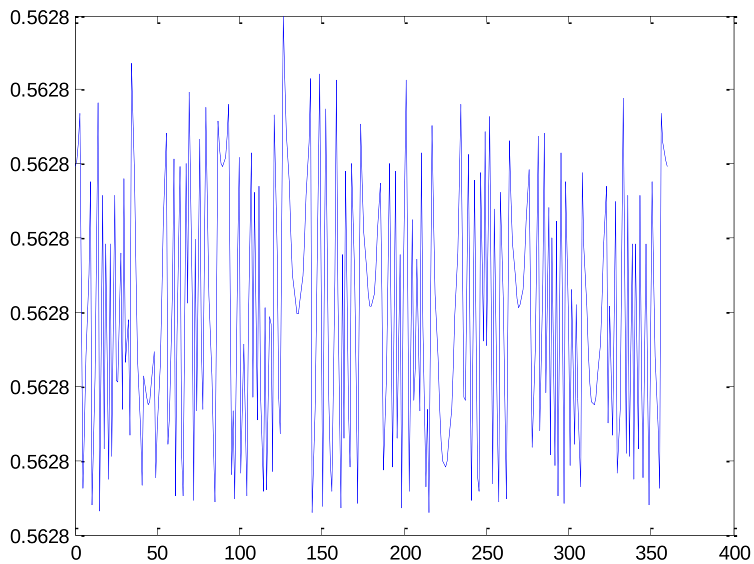


This Figure shows the square lattice with path tracing. The green shows direction of travel.

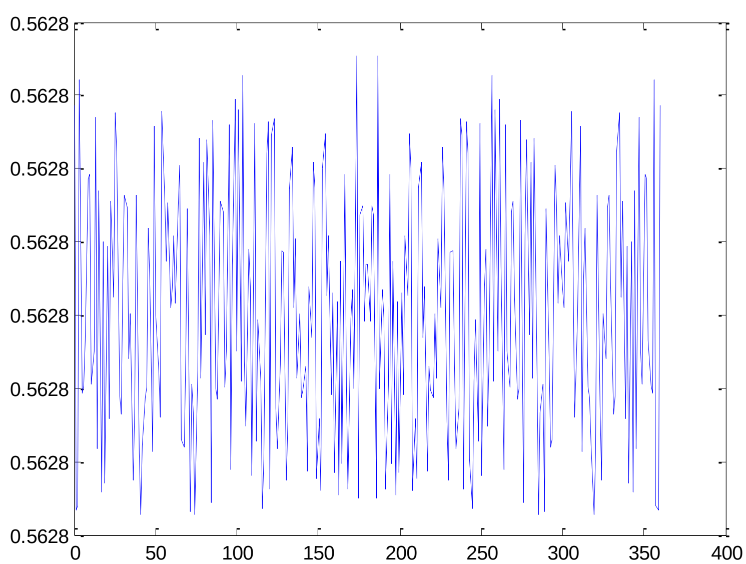


This figure shows the hex lattice.

2. The code was tested by assuming that the source and cross section values were constant throughout both pins. The plots for these cases are shown below.



Square Lattice

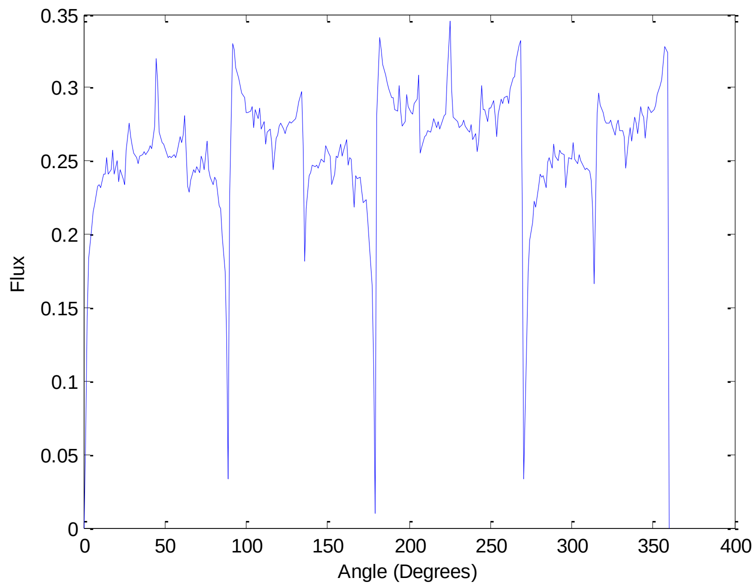


Hexagonal Lattice

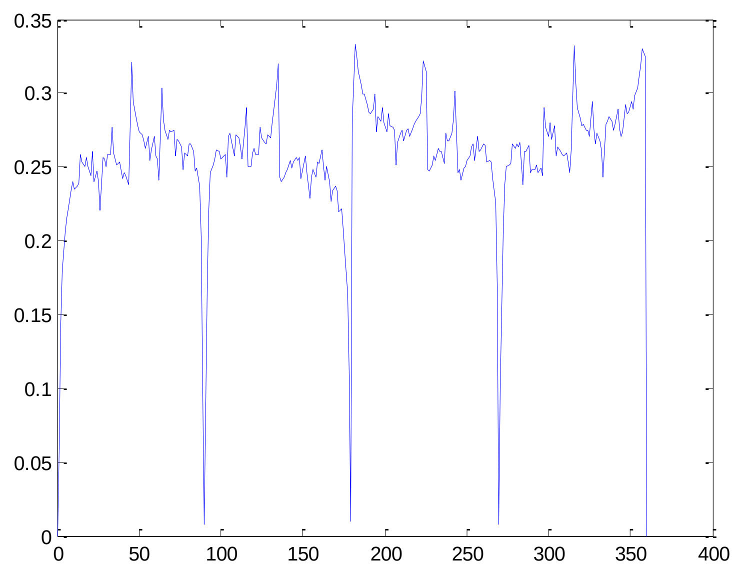
These plots show a flat distribution.

3. Plots at different points:

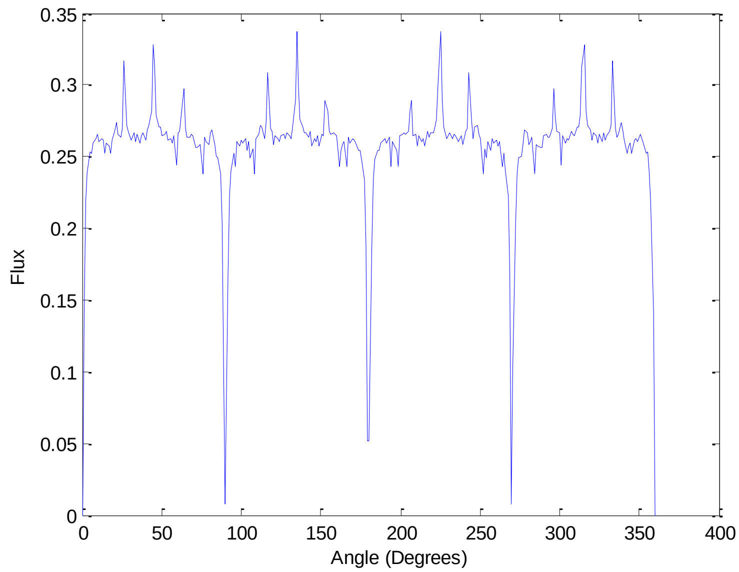
Square Lattice [0.41,0.41]



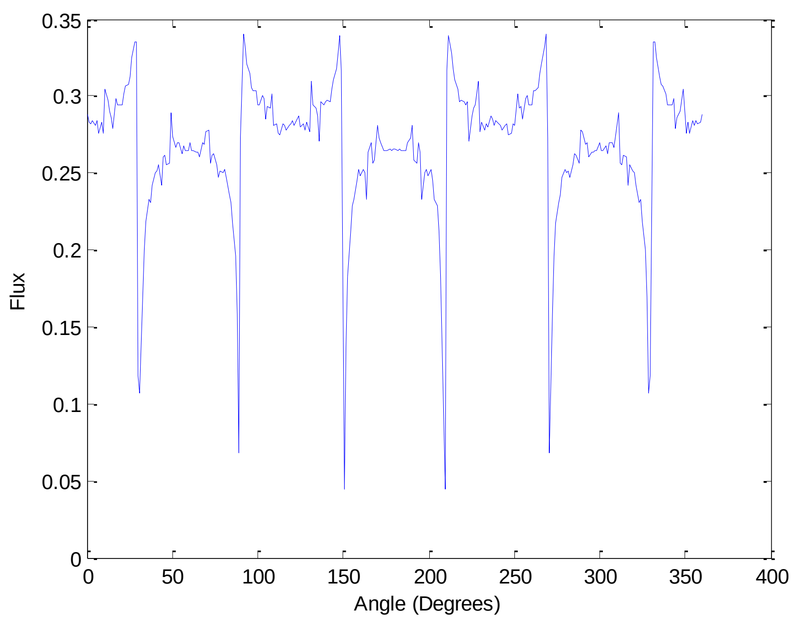
Square Lattice [0.63,0.41]



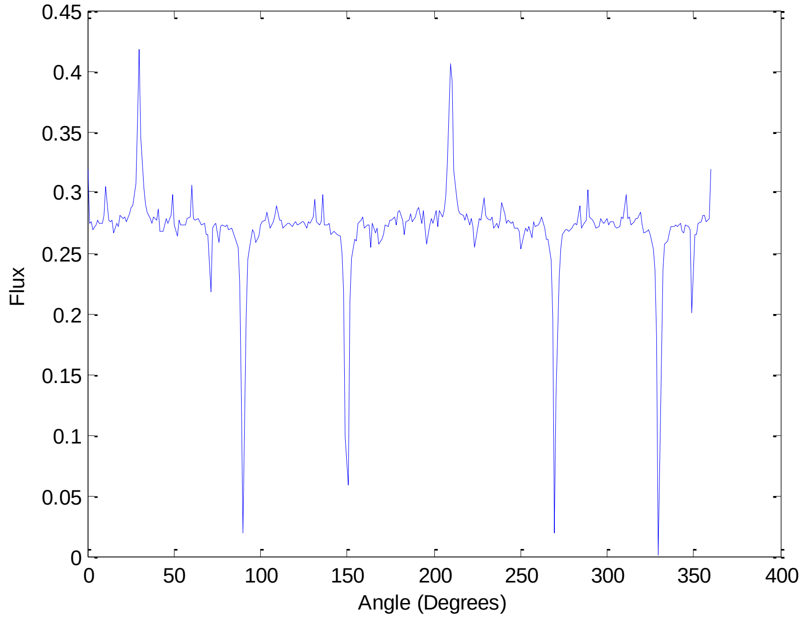
Square Lattice [0.63,0.63]



Hexagonal Lattice [0.381051,0.66]:



Hexagonal Lattice [0.572,0.33]:



These plots depict peaks and valleys, accurately representing my emotions in the completion of this homework.

Programs:

Loop_Check.m

```
function [Flux]=Loop_Check(n,mfp,r,CHOICE)
theta=linspace(0,360,n);
Flux=zeros(1,n);

%You can pick which problem to work on
for i=1:n
   %Flux(1,i)=Problem_3(theta(i),[0.63,0.63],1,mfp);
   %Flux(1,i)=Problem_3(theta(i),[0.381051,0.66],2,mfp);
   %Flux(1,i)=Problem_3(theta(i),[0.41,0.41],1,mfp);
   %Flux(1,i)=Problem_3(theta(i),[0.63,0.41],1,mfp);
   %Flux(1,i)=Problem_3(theta(i),[0.57157676649,0.33],2,mfp);
   Flux(1,i)=Problem_3(theta(i),r,CHOICE,mfp);
end


hold off
plot(theta,Flux);
xlabel 'Angle (Degrees)'
ylabel 'Flux'
```

Problem_3.m

```
function [Flux]=Problem_3(theta,r,CHOICE,mfp)
%This program attempts to solve problem 3 of homework 1
%of NUEN 629 Fall 2015, for details refer to the problem statement
format short
format compact
%Units cm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Lattice (Note: Hexigons should have flat faces with constant y values)
%Note Make sure x1 and y1 values are centered
%Form of input: [y1,x1,m,sin(b),cos(b)]

%Central Radius
R=0.41;
%Hexagon 1/2 pitch
RH=0.66;



%CHOICE=2;
if CHOICE==1
    %Square Lattice:
    Lines(1,1:5)=[0,0.63,nan,1,0]; %Positive X Line
    Lines(2,1:5)=[0,-0.63,nan,1,0]; %Negative X line
    Lines(3,1:5)=[0.63,0,0,0,1]; %Positive Y line
    Lines(4,1:5)=[-0.63,0,0,0,1];  %Negative Y line
else %Hex Lattice:
    Lines(1,1:5)=[0.66,0,0,0,1]; %Positive Y line
    Lines(2,1:5)=[-0.66,0,0,0,1];  %Negative Y line
```

```
    Lines(3,1:5)=[sind(30)*RH,cosd(30)*RH,tand(120),sind(120),cosd(120)]; %Right Side Negative
Slope
    Lines(4,1:5)=[-sind(30)*RH,cosd(30)*RH,tand(60),sind(60),cosd(60)]; %Right Side Positive
Slope
    Lines(5,1:5)=[-sind(30)*RH,-cosd(30)*RH,tand(120),sind(120),cosd(120)]; %Left Side Negative
Slope
    Lines(6,1:5)=[sind(30)*RH,-cosd(30)*RH,tand(60),sind(60),cosd(60)];  %Left Side Positive
Slope
end

Em=0.08;Ef=0.1414;I=1/(4*pi*Ef); %Values for Flux Determination
Q_CHECK=0;
%In Order to Check if my solution is working, need a flat profile
%Q_CHECK=1;Em=0.1414;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Find Length of Edges
if (any(Lines(:,3)))
   Lengths=(RH*2)/(3^0.5);
   Lengthx=Lengths*cosd(60);
   Pitch=2*RH;
   Min_dis=(Pitch/2)/sind(60)-R;
else
   Lengths=max(Lines(:,2))-min(Lines(:,2));
   Pitch=Lengths;
   Lengthx=1;
   Min_dis=(Pitch/(2^0.5))-R;
end

Rows=size(Lines,1); %Used for Looping
limits=zeros(Rows,4); %Limits for lines [ymin,ymax,xmin,xmax]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Plot Boundaries to visually observe %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xplot=zeros(1,100);
yplot=xplot;

for j=1:Rows %Loop over all Boundary Lines
    slope=1;
    if(Lines(j,3)<0) %Check for negative Slopes
       slope=-1;
    end
        if(Lines(j,3)==0) %m=0, sin(b)=0, cos(b)=1
            yplot(1,:)=Lines(j,1); %y is constant
            xplot(1,:)=linspace(Lines(j,2)-Lengths/2,Lines(j,2)+Lengths/2);
            limits(j,1)=Lines(j,1);
            limits(j,2)=Lines(j,1);
            limits(j,3)=Lines(j,2)-Lengths/2;
            limits(j,4)=Lines(j,2)+Lengths/2;
        elseif(isnan(Lines(j,3))) %m=INF,sin(b)=1,cos(b)=0
            xplot(1,:)=Lines(j,2); %x is constant
            yplot(1,:)=linspace(Lines(j,1)-Lengths/2,Lines(j,1)+Lengths/2);
            limits(j,1)=Lines(j,1)-Lengths/2;
            limits(j,2)=Lines(j,1)+Lengths/2;
            limits(j,3)=Lines(j,2);
            limits(j,4)=Lines(j,2);
        else %m is nonzero and non inf
            xplot(1,:)=linspace(Lines(j,2)-Lengthx/2,Lines(j,2)+Lengthx/2);
            yplot(1,:)=Lines(j,3).*(xplot-Lines(j,2))+Lines(j,1);
            limits(j,1)=slope.*Lines(j,3)*(xplot(1,1)-Lines(j,2))+Lines(j,1);
            limits(j,2)=slope.*Lines(j,3)*(xplot(end)-Lines(j,2))+Lines(j,1);
            limits(j,3)=Lines(j,2)-Lengthx/2;
            limits(j,4)=Lines(j,2)+Lengthx/2;
        end
    plot(xplot,yplot,'b','LineWidth',3);
    hold on
end
```

```
%Plot The Circle
xplot=linspace(-R,R,50);
yplot=(R.^2-xplot.^2).^0.5;
plot(xplot,-yplot,'b','LineWidth',3);
hold on
plot(xplot,yplot,'b','LineWidth',3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculations  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Direction and Magnitude This variable holds our location and direction
Vector(1,1:5)=[r(2),r(1),tand(theta),sind(theta),cosd(theta)];
Vector(Vector==inf)=nan;
Vector(Vector==-inf)=nan;

P=0;P_Store=zeros(1,2000);distance=0;nn=2;
while(P<mfp) %Loop until we have answers...
%For debugging purposes
%for kk=1:100

%If heading outside the system Turn Around: Do twice, just incase you need
%two translations
Vector=Reflection(Rows,Vector,Lines,Pitch,1);
Vector=Reflection(Rows,Vector,Lines,Pitch,1);

%%d will be variable for length till change medium

%Translation Matrix (I am sure there is a better way to do this)
XYT=zeros(Rows+1,3); %Values saved [Ytranslated,Xtranslated,Distance]
%The Plus 1 is for the circle consideration

%Lets go for one interaction to the next Boundary Line:
for j=1:Rows %Loop over all Boundary Lines
        %If we have a sloped line (This should include horizontal lines)
        if(Lines(j,5)~=0 && Vector(1,5)~=0 && Lines(j,3)~=Vector(1,3))
            %There is an intersection
            x2=(Lines(j,3)*Lines(j,2)-Vector(1,3)*Vector(1,2)+Vector(1,1)-
Lines(j,1))/(Lines(j,3)-Vector(1,3));
            y2=Vector(1,3)*(x2-Vector(1,2))+Vector(1,1);
            %Check if intersection is in the right direction
            XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
        end
        %If my vector is traveling up and down
        if(Lines(j,5)~=0 && Vector(1,5)==0)
            x2=Vector(1,2);
            y2=Lines(j,3)*(x2-Lines(j,2))+Lines(j,1);
            XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
        end
        %If my intersecting line is up and down
        if(Lines(j,5)==0 && Vector(1,5)~=0)
            x2=Lines(j,2);
            y2=Vector(1,3)*(x2-Vector(1,2))+Vector(1,1);
            XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
        end
end

if(any(XYT)) %If any elements of XYT are non zero
    A=XYT(:,3);
    n=find(A==min(A(A>0)));
    xt=XYT(n(1),2);
    yt=XYT(n(1),1);
    Length=XYT(n(1),3); %this is the track length
end

%%Checking If My tracing is done correctly
if(isnan(Vector(1,3))) %If we have an infinite slope
    xplot=ones(1,100).*Vector(1,2);
    yplot=linspace(Vector(1,1),Vector(1,1)+Vector(1,4)*Length,100);
else
```

```
    xplot=linspace(Vector(1,2),Vector(1,2)+Length*Vector(1,5),100);
    yplot=Vector(1,3).*(xplot-Vector(1,2))+Vector(1,1);
end
plot(xplot,yplot,'r','LineWidth',3);
hold on;

%Calculations for distances
[distance,P,P_Store,nn]=Distances(Vector,R,xt,yt,Em,Ef,distance,P,P_Store,nn,Length,mfp,Min_dis,Q
_CHECK);


Vector(1,1)=yt; %New Location for Vector
Vector(1,2)=xt;
%Plot Our Vector for its path
Length=0.3;
if(isnan(Vector(1,3))) %If we have an infinite slope
    xplot=ones(1,100)*Vector(1,2);
    yplot=linspace(Vector(1,1),Vector(1,1)+Vector(1,4)*Length);
else
    xplot=linspace(Vector(1,2),Vector(1,2)+Length*Vector(1,5));
    yplot=Vector(1,3).*(xplot-Vector(1,2))+Vector(1,1);
end
plot(xplot,yplot,'g','LineWidth',3);
hold on
grid on

end

sum=0;

for L=2:nn-1
    sum=sum+((-1)^L)*exp(-P_Store(L));
end
Flux=I*sum;
```

Reflection.m

```
function [Vector]=Reflection(Rows,Vector,Lines,Pitch,LT)
%This function will reflect across a cell given input values

%Translation Matrix (I am sure there is a better way to do this)
XYT=zeros(Rows,3); %Values saved [Ytranslated,Xtranslated,Angle]

tol=0.0001;
%Looping over all our lines
for j=1:Rows
%We need an inversion in logic: One for reflection, another for
%Right_Direction
if(LT==1)
    TF=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)))<tol;
elseif(LT==0)
    TF=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)))>tol;
end

%For Debugging Purposes:
%Value=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)));
%disp('     Cos_L(x) Sin_L(y)    Lx            Ly       Vx      Vy              Val');
%disp([Lines(j,5),Lines(j,4),Lines(j,2),Lines(j,1),Vector(1,2),Vector(1,1),Value]);

    %Check if on line (or check if not on line in the case of a
    %Right_Direction
    if(TF)
        %Find Normal to the line:
        XN=-1*Lines(j,4);
        YN=Lines(j,5);
        %Find the inward normal
        d1=((Lines(j,2)+XN)^2+(Lines(j,1)+YN)^2)^0.5;
        d2=((Lines(j,2)-XN)^2+(Lines(j,1)-YN)^2)^0.5;
```

```matlab
        %For Debugging Purposes
        %disp('      j         LXN         VXN        d1     d2       LX          LY');
        %disp([j,XN,YN,d1,d2,Lines(j,2),Lines(j,1)]);

        if(d2<d1)
            XN=-XN;
            YN=-YN;
        end
        %Check if we are going the right way (dot product should be less
        %than 90. It it is greater, then we store a matrix which holds
        %reflection coordinates. The reason we have multiple reflection
        %coordinates is because we might be on top of two lines

        dot=acosd(XN*Vector(1,5)+YN*Vector(1,4));

        %For Debugging Purposes
        %disp('      j         LXN       VXN          LYN     VYN        dot');
        %disp([j,XN,Vector(1,5),YN,Vector(1,4),dot]);



    if(LT==1)
        TF2=dot>90;
    elseif(LT==0)
        TF2=dot>90;
    end

        if(TF2)
            XYT(j,2)=Vector(1,2)+XN*Pitch;
            XYT(j,1)=Vector(1,1)+YN*Pitch;
            XYT(j,3)=acosd(XN*Vector(1,5)+YN*Vector(1,4));
        end

    end
end

if(any(XYT)) %If any elements of XYT are non zero
    n=find(XYT(:,3)==max(XYT(:,3))); %Find the line of the max angle
    Vector(1,2)=XYT(n(1),2);
    Vector(1,1)=XYT(n(1),1);
end

%For Debugging purposes
%disp('XYT...Did we find anything?');
%disp([XYT]);
```

Right_Direction.m

```matlab
function [XYT]=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines)

tol=0.000000001;CHECK=1;

%Find the distance you need to go
distance=((Vector(1,2)-x2)^2+(Vector(1,1)-y2)^2)^0.5;

if (distance>tol)
    %If you are within the bounds of habitiation, This is all you need
    %after the first step, but if you do not start on the edge, you
    %need to check normals to make sure you are going in the right
    %direction.
    if(y2+tol<limits(j,1)||y2>limits(j,2)+tol||x2+tol<limits(j,3)||x2>limits(j,4)+tol)
        CHECK=0;
    end
    if(CHECK==1)%Check Normals and directions
        CHECK2(1:2)=Vector(1,1:2);
        DOS=Reflection(1,Vector,Lines(j,:),90,0); %will update
    else
        DOS=1;CHECK2=1;
    end
```

```
    if(CHECK==1 && (CHECK2(1)~=DOS(1,1)||CHECK2(2)~=DOS(1,2))) %Update worthy Values
         XYT(j,2)=x2;
         XYT(j,1)=y2;
         XYT(j,3)=distance;
    end
end

%For Debugging purposes
%disp('       x2         x1     xmin       xmax       y2          y1       ymin     ymax')
%disp([x2,Vector(1,2),limits(j,3),limits(j,4),y2,Vector(1,1),limits(j,1),limits(j,2)]);
%disp('    Line     Slope       Dist_cal   CHECK(1=keep)  CHECK2/=DOS(1,1)');
%disp([j,Vector(1,3),distance,CHECK,CHECK2,DOS(1,1)]);
```

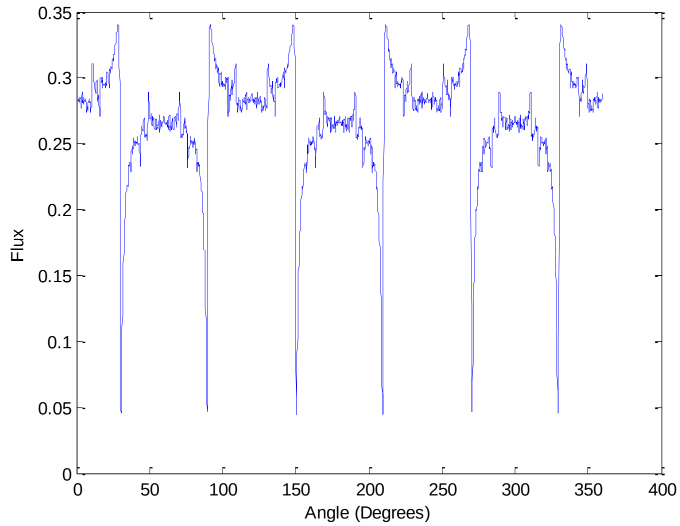
Distances.m

```
function
[distance,P,P_Store,nn]=Distances(Vector,R,xt,yt,Em,Ef,distance,P,P_Store,nn,Length,mfp,Min_dis,Q
_CHECK)
passed=0;

%Please note...this code will fail if you start in the fuel. Min_dis will
%need to be modified.



%Determine Distance/FLUX Stuff
a=1+Vector(1,3)^2;
b=2*(Vector(1,3)*Vector(1,1)-Vector(1,2)*(Vector(1,3)^2));
c=(Vector(1,3)^2)*(Vector(1,2)^2)+Vector(1,1)^2-2*Vector(1,3)*Vector(1,2)*Vector(1,1)-R^2;
%Determine if we passed through fuel

debug=0;
%For Debugging purposes
%disp('       mv        x1        y1      R            a        b          c');
%disp([Vector(1,3),Vector(1,2),Vector(1,1),R,a,b,c]);
%debug=1;
%disp(' V_sin(x4)    V_cos(y5)');
%disp([Vector(1,4),Vector(1,5)]);

if (abs(Vector(1,5))<0.000001 && isreal((R^2-xt^2)^0.5)) %If cos(b)=0 %Vertical
    xcmin=xt;xcmax=xt;
    ycmin=-1*(R^2-xt^2)^0.5;
    ycmax=(R^2-xt^2)^0.5;
    passed=1;
elseif (abs(Vector(1,4))<0.000001 && isreal((R^2-yt^2)^0.5)) %If sin(b)=0 %Horizontal
    ycmin=yt;ycmax=yt;
    xcmin=-1*(R^2-yt^2)^0.5;
    xcmax=(R^2-yt^2)^0.5;
    passed=1;
elseif (isreal((b^2-4*a*c)^0.5) && abs(Vector(1,5))>0.000001 && abs(Vector(1,4))>0.000001)
    xcmin=(-b-(b^2-4*a*c)^0.5)/(2*a);
    xcmax=(-b+(b^2-4*a*c)^0.5)/(2*a);
    ycmin=Vector(1,3)*(xcmin-Vector(1,2))+Vector(1,1);
    ycmax=Vector(1,3)*(xcmax-Vector(1,2))+Vector(1,1);
    passed=1;
end

%For Debugging purposes
%if(passed==1)
%disp('       x2        y2       xcmax    ycmax     xcmin    ycmin');
%disp([xt,yt,xcmax,ycmax,xcmin,ycmin]);
%end

if(passed==1)%If we passed through fuel
    %If the vector is left traveling the dot with -x should be less than 90
    dot=acosd(-1*Vector(1,5));
    distance_fuel=((xcmax-xcmin)^2+(ycmax-ycmin)^2)^0.5;
    if(distance_fuel<0.00001||Length<Min_dis) %Either on edge of circle
```

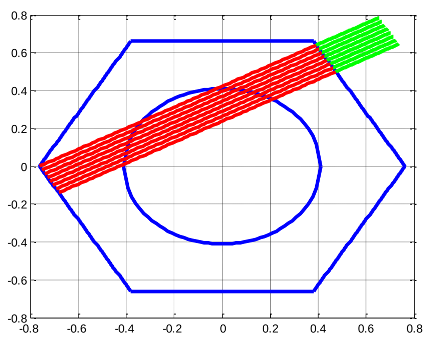
```
        passed=0; %Or starting in the moderator somewhere
    %Below is redundant, and sometimes causes errors, but I want to keep it
    %to remind myself about where I came from.
    %elseif(Vector(1,4)==0)%traveling at 180 or 0 degrees
    %    disp('I was here')
    %    distance_before=((xcmax-Vector(1,2))^2+(ycmax-Vector(1,1))^2)^0.5;
    %    distance_after=((xt-xcmin)^2+(yt-ycmin)^2)^0.5;
    elseif(dot<90) %Traveling leftward
        distance_before=((xcmax-Vector(1,2))^2+(ycmax-Vector(1,1))^2)^0.5; %distance traveled
before the fuel
        distance_after=((xt-xcmin)^2+(yt-ycmin)^2)^0.5;
    elseif(dot>90) %Traveling rightward
        distance_before=((xcmin-Vector(1,2))^2+(ycmin-Vector(1,1))^2)^0.5;
        distance_after=((xt-xcmax)^2+(yt-ycmax)^2)^0.5;
    end

    if(passed==1) %Some redundancy is good.
        if(abs(Length-distance_fuel-distance_before-distance_after)>0.000001)
          disp('Your lengths are not adding up...do not worry about it');
          disp('    Fuel D    Before     After    Length');
          disp([distance_fuel,distance_before,distance_after,Length]);
        end
        distance=distance+distance_before;
        if(Q_CHECK==0)
            P_Store(nn)=distance*Em+P_Store(nn-1); %part for moderator
            nn=nn+1; %for the fuel
            P_Store(nn)=distance_fuel*Ef+P_Store(nn-1);
            P=P_Store(nn);
            nn=nn+1; %new era of moderator
            distance=distance_after; %Resetting distance
        else
            P_Store(nn)=distance*Em*0+P_Store(nn-1); %part for moderator
            nn=nn+1; %for the fuel
            P_Store(nn)=distance_fuel*Ef+P_Store(nn-1)+distance*Ef;
            P=P_Store(nn);
            nn=nn+1; %new era of moderator
            distance=distance_after; %Resetting distance
        end
    else %Some Redundancy is bad
        distance=distance+Length;
    end
else %If we did not pass through fuel
distance=distance+Length;
end

if(Q_CHECK==0) %Q check is to check if code is working
    if(P_Store(nn-1)+distance*Em>mfp)   %Make sure we don't go up to our mfps
        P_Store(nn)=distance*Em+P_Store(nn-1); %part for moderator
        nn=nn+1; %for the fuel
        P_Store(nn)=P_Store(nn-1)+0*Ef;
        P=P_Store(nn);
        nn=nn+1; %new era of moderator
        distance=0; %Resetting distance
    end
else
    if(P_Store(nn-1)+distance*Ef>mfp)   %Make sure we don't go up to our mfps
        P_Store(nn)=distance*Em*0+P_Store(nn-1); %part for moderator
        nn=nn+1; %for the fuel
        P_Store(nn)=P_Store(nn-1)+distance*Ef;
        P=P_Store(nn);
        nn=nn+1; %new era of moderator
        distance=0; %Resetting distance
    end
end
%For Debugging Purposes:
%disp('    Distance  Length       P_Store      P     mfp, theta');
%disp([distance,Length,P_Store(nn-1),P,distance*Em,acosd(Vector(1,4))]);
```

```
A=exist('distance_fuel','var');
B=exist('distance_before','var');
if(A && debug==1 && B)
    disp('    Fuel D    Before     After    Length');
    disp([distance_fuel,distance_before,distance_after,Length]);
    disp('  Distance      P1        P2        P3      P4');
    findingstuff=find(P_Store(2:end)==0,2);
    disp([distance,P_Store(2),P_Store(3:findingstuff(2))]);
elseif(debug==1)
    disp('  Distance      P1        P2        P3      P4');
    findingstuff=find(P_Store(2:end)==0,2);
    disp([distance,P_Store(2),P_Store(3:findingstuff(2))]);
end
```

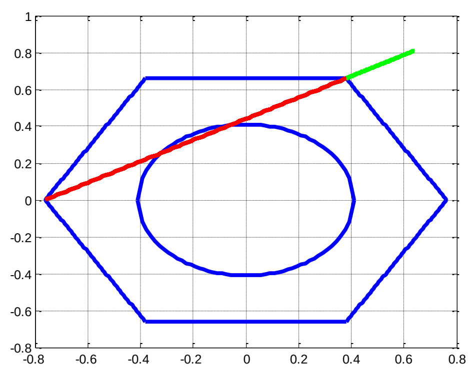
Hex Lattice: [0.381051,0.66]



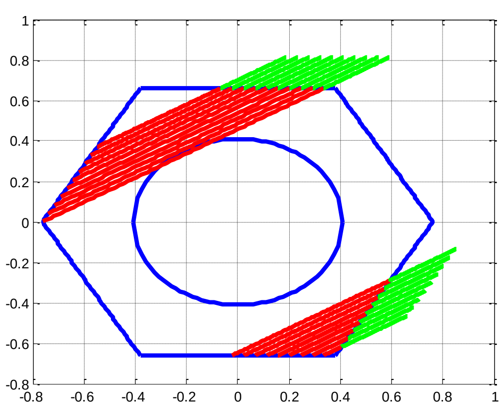
Dead spot every 60 degrees



Right before the “dead spot” (29 degrees)



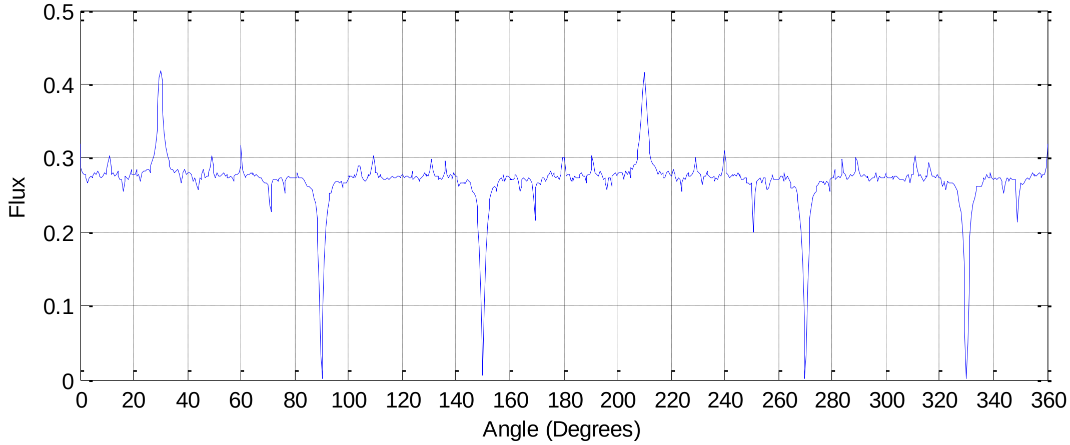
At the “dead Spot” (30 degrees)



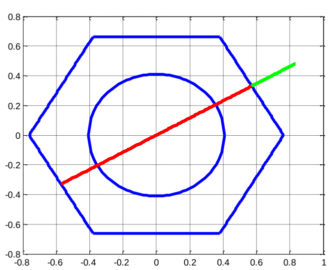
Right after the “dead Spot” (31 degrees)

Every Picture is shown at 10 mean free paths. It should be noted that I took the dead spot at 30 degrees, but these pictures show the dead spot is nearer the 31 degree mark. At 30 degrees, the trace goes through the fuel every pass through the lattice, at 31 degrees, there is a single pass through the fuel, then many subsequent passes through the fuel lattice without any interactions in the fuel.

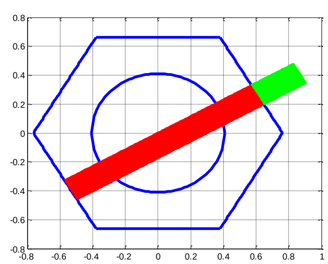
Hex Lattice: [0.57157676649,0.33]



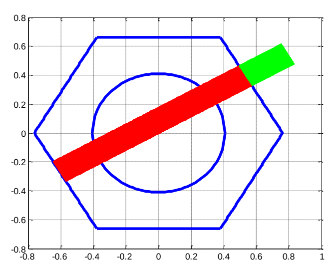
Every 60 degrees there is either a peak or a dead spot



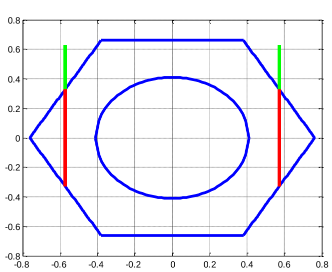
At peak spot (30 degrees)



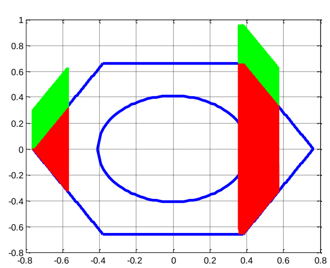
Right before peak (29.9)



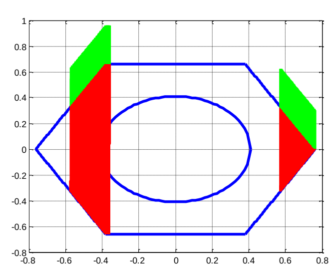
Right after peak (30.1)



At dead spot (90 degrees)



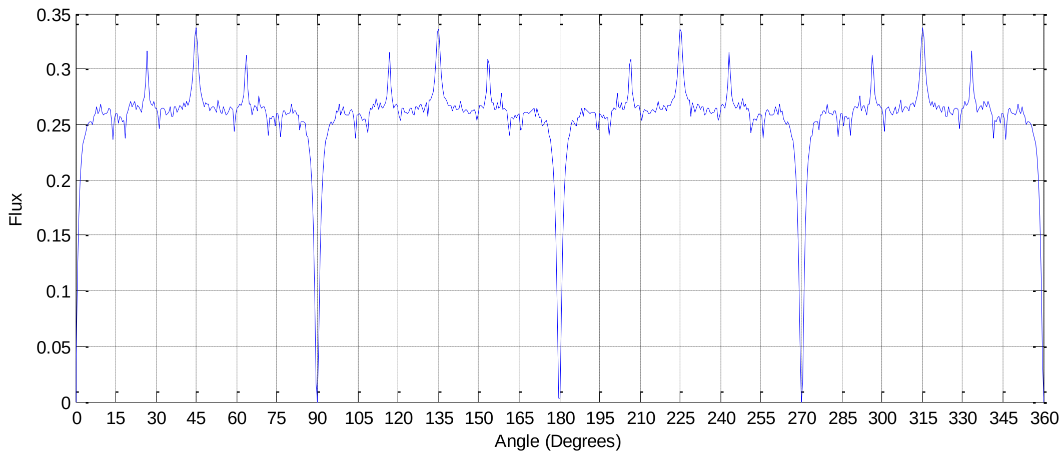
Right after dead spot (90.1)



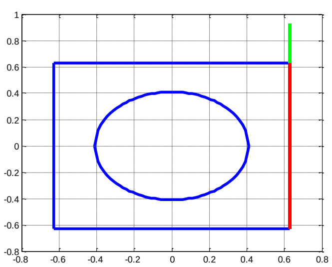
Right after dead spot (89.9)

These figures better show why we have peaks and dead spots in the angular distribution. All pictures are shown at 10 mfps.

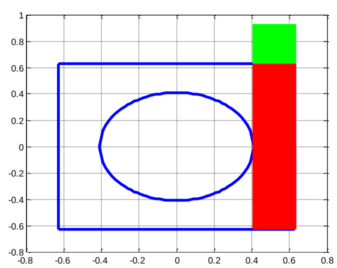
Square Lattice: [0.63, 0.63]



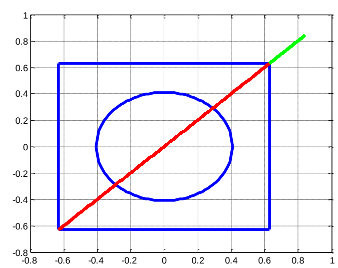
Dead spots and peaks every 90 degrees.



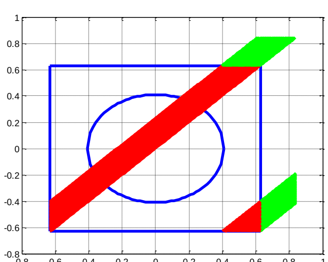
At dead spot (90 degrees)



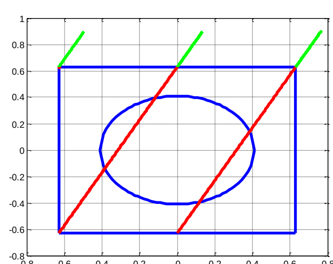
Right after dead spot (90.1)



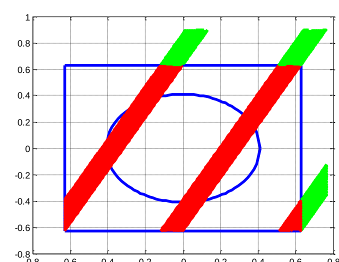
At peak spot (45 degrees)



After peak spot (45.1)



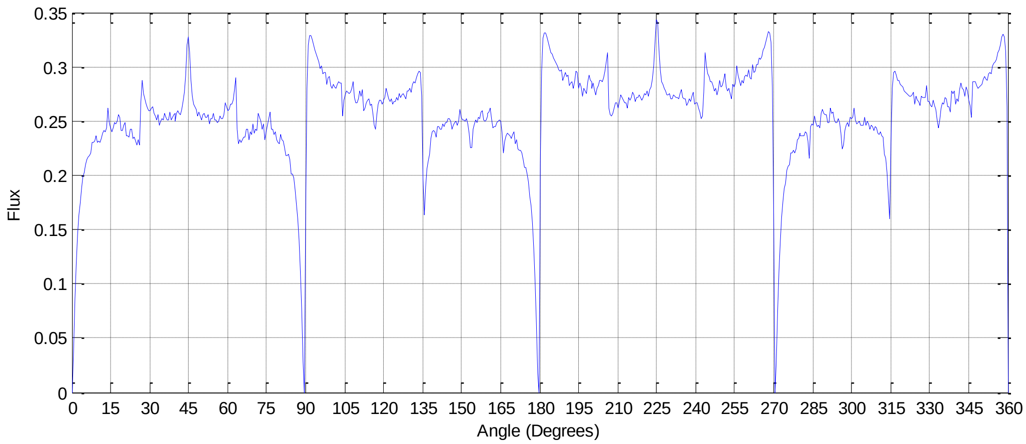
At Peak Spot (63.43 degrees)



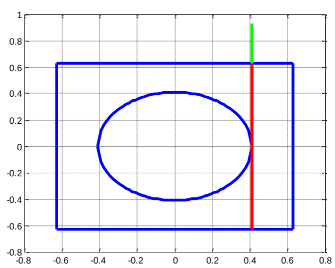
At peak spot (63.5 degrees)

These pictures were done all at 10 mfps and show the origin of the peaks and valleys.

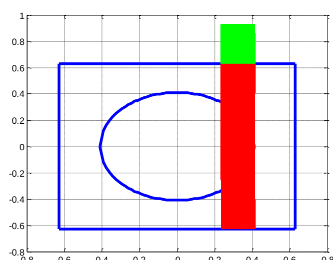
Square Lattice: [0.41,0.41]



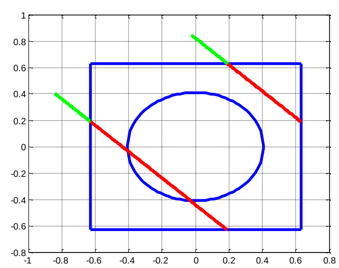
Valleys every 90 degrees, and varying peaks.



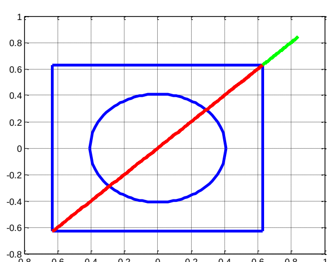
At dead spot (90 degrees)



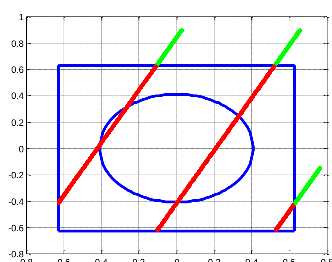
Right after dead spot (90.1)



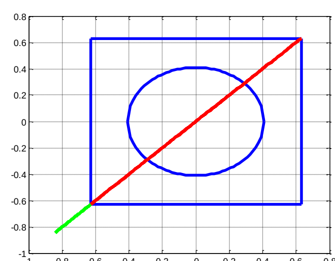
At dead spot (135 degrees)



Peak (45 degrees)



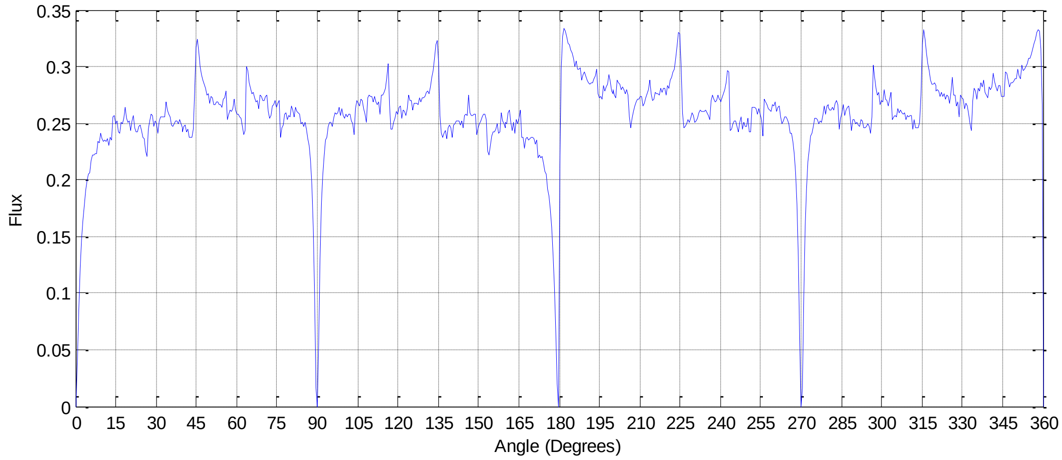
At peak (63.43 degrees)



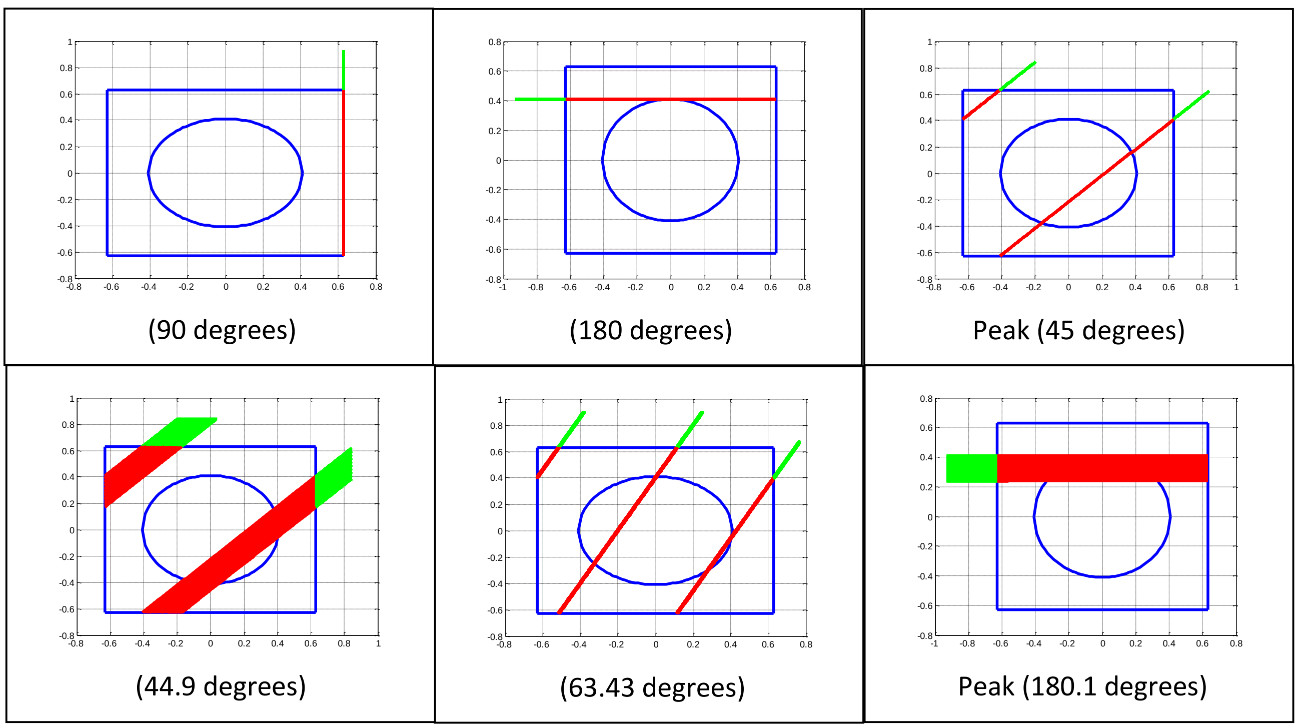
Peak (225 degrees)

The peaks vary in height with angles 180 to 270 being the highest because they initially start going towards the fuel. 10 mfp

Square Lattice: [0.63,0.41]



Valleys every 90 degrees, and varying peaks.



Geometry graphs shown at 10 mfps.