1 Генерация

Для интегрирования методом МК нужно соответствующие дифференциалы заменить на веса и функции от генераторов.

Положим $G()$ — генератор случайнгого числа от 0..1 не включительно.

Полная скорость захвата это интеграл

$$C_+ = \int d^3\vec{r} \cdot d^3\vec{v} f_k\left(r,v\right) \cdot n_p f_B\left(\vec{v}_1\right) d^3\vec{v}_1 \cdot \Gamma\left(\vec{v},\vec{v}_1,r\right)$$

где

$$\Gamma\left(\vec{v},\vec{v}_1,r\right) = \int\limits_{v'<v_{esc}} d^3\vec{v}\,'\delta\left(E_f - E_{in}\right) \cdot \frac{m_k^3\left|\mathcal{M}\right|^2}{64\pi^2 m_i^2 m_k^2} = \int |\vec{v} - \vec{v}_1| d\sigma$$

— сечение, умноженное на разность скоростей.

$$\Gamma\left(\vec{v},\vec{v}_1,r\right) = \nu' d\vec{n}\,' \frac{G_F^2}{\pi} \frac{m_i m_k^2}{(m_i + m_k)} \Phi dF$$

- Генерация $d^3\vec{r}$

  $$d^3\vec{r} = V \cdot 3\xi^2 d\xi$$

  где ξ — безразмерный радиус

  Положим $\xi = G()^n = g_1^n$

  $$d^3\vec{r} = V \cdot 3n g_1^{3n-1} dg_1 = V3n\xi^{\frac{3n-1}{p}} dg_1$$

```
double r_nd = pow(G(),pow_r);
factor *= (3*pow_r* pow(r_nd,(3*pow_r-1.0)/pow_r));
```

- Генерация $d^3\vec{v} f_k\left(\vec{r},\vec{v}\right)$

  $$d^3\vec{v} = 2\pi v dv^2 d\vec{n} = 2\pi v du^2 d\vec{n} = 4\pi v u du d\vec{n}$$

  $$f_k\left(\vec{r},\vec{v}\right) = n_\chi f(u)$$

  $$f(u) = \frac{1}{2(2\pi)^{3/2} u v_\odot \overline{v}} \left(e^{-\dfrac{(u - v_\odot)^2}{2\overline{v}^2}} - e^{-\dfrac{(u + v_\odot)^2}{2\overline{v}^2}} \right)$$

  $$v^2 = u^2 + v_{esc}^2$$

  За генерацию скорости отвечает функция Velocity(). Причем напрвление скоростей считается изотропным. Под $d\vec{n}$ подразумевается генерация направления.

  Проверка генератора с правильным распределением при $v_\odot = 0.73e-3, \overline{v} = 0.53e-3, v_{esc} = 2e-3$

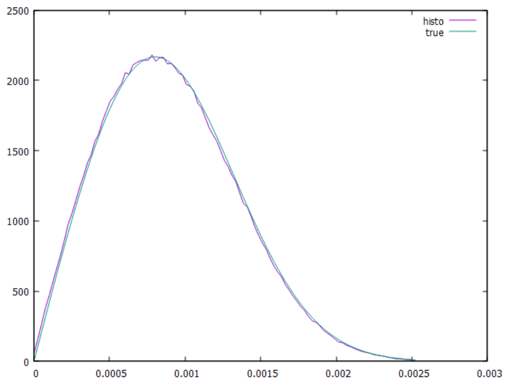

Рис. 1: compare of u distribution

- $f_B\left(\vec{v}_1\right)$:

$$\vec{v}_1 = Gauss3\left(\sqrt{\frac{T(\xi)}{m_i}}\right)$$

- Концентрация ядер n_i: $\widehat{\rho}(r) \cdot \widetilde{\rho}_i(r)$ учитывается в коде:

```
auto /*std::vector<double>*/ RhoND = BM["Rho"];
...
decltype(Therm) /*grid function*/ Element_N(R,RhoND*BM[element]);
...
double n_nd = nR(r_nd);
factor *=  n_nd;
```

- Массовые константы: из формулы безразмерного захвата осталось только $\frac{m_k}{m_k+m_i}$

```
double const_fact_rd = mk/(mk+mi)/Nmk;
```

- Выходная скорость соответствует интегралу $\nu' d\vec{n}$, где $d\vec{n} = \frac{1}{4\pi} dcos(\theta') d\phi$. С разницей лишь в том, что $cos(\theta')$ генерируется до $cos(\theta_{max})$

```
double Nu1_squared =Nu.quad()-deltaE*2*mi/(mk*(mi+mk));
if(Nu1_squared<=0.0)
return MC::MCResult<vec3>(vec3(0,0,0),0);

double Nu1 = sqrt(Nu1_squared);

double cosTh1max = (Vesc*Vesc-Nu1_squared-VcmN*VcmN)/(2*VcmN*Nu1)
    ;
...
double cosTh1 = (1+cosTh1max)*G()-1;
...
vec3 vNu1 = Nu1*(n_v*cosTh1+n_1*sinTh1*cos(phi1)+n_2*sinTh1*sin(
    phi1));
...
return MC::MCResult<vec3>(vNu1,0.5*(1.0+cosTh1max)*Nu1);
```

Из закона сохранения импульса и энергии, выходная скорость частицы в СЦМ изменится

$$\nu'^2 = \nu^2 - \Delta E \cdot \frac{2m_i}{m_k(m_i + m_k)}$$

- Форм фактор ядра учитывается в структуре dF.

```
constexpr double fermi_GeV = 5;
inline double BesselX(double x) noexcept{
        if(x<0.01){
                return 1.0/3-x*x*(1-x*x/28)/10;
        }else{
                return (sin(x)-x*cos(x))/(x*x*x);
        }
}
...
s =  fermi_GeV*0.9;
double b = (1.23*pow(M,1.0/3)-0.6)*fermi_GeV;
double a = 0.52*fermi_GeV;
R = sqrt(b*b+7*M_PI*M_PI*a*a/3-5*s*s);
...
double bf = 3*BesselX(q*R)*exp(-q*q*s*s/2);
return bf*bf;
```