Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Doble titulació GETI - ADE

Títol: Estudi i implementació de la refrigeració magnètica en ANSYS.

Document: Annexos

Alumne: Joel Alayo Vargas Machuca

Tutor: Joan J. Suñol Martínez (Lino Montoro Moreno)
Departament: Física (Enginyeria mecànica i de la construcció industrial)
Àrea: Física aplicada (Màquines i motors tèrmics)

Convocatòria: Febrer 2021

Índex

A. Materials magnetocalòrics a prop de T_{amb}

Materials de transició de primer ordre:

- Aliatges $La(Fe, Co, Si)_{13}$
- Aliatges $La(Fe, Co)_{13-x}Si_x$
- Aliatges $MnFeP$
- Aliatges $(La, Ca, Sr)MnO_3$
- Aliatges $La(FeCoSi)_{13}B_x$
- Aliatges $LaFeSiH$
- Aliatges $LaFe_{13-x-y}Co_xSi_y$
- Aliatges $La(Fe, Mn, Si)_{13}H_z$
- Aliatges $La(Fe, Mn, Si)_{13}H_y$
- Aliatges $MnFeAs_{1-x}P_x$

Materials de transició de segon ordre:

- Elements:

Gd	293K
Tb	235K

- Aliatges $Gd_{1-x}Tb_x$

$Gd_{0.74}Tb_{0.26}$	278K

- Aliatges $Gd_{1-x}Er_x$

$Gd_{0.85}Er_{0.15}$	265K
$Gd_{0.94}Er_{0.06}$	280K

- Aliatges $Gd_{1-x}Dy_x$

$Gd_{0.89}Dy_{0.11}$	283.5K
$Gd_{0.87}Dy_{0.13}$	280.5K
$Gd_{0.84}Dy_{0.16}$	277.5K

- Aliatges $Gd_{1-x}Y_x$

$Gd_{0.92}Y_{0.08}$	278.5K
$Gd_{0.985}Y_{0.015}$	290K
$Gd_{0.95}Y_{0.5}$	283.5K

- Aliatges $Gd_{1-x}Ho_x$

$Gd_{0.9}Ho_{0.1}$	277.5K

Dades Gadolini (SI)
$M_m = 0.15725\ kg/mol$
$\rho = 7900\ kg/m^3$
$k = 10.5\ W/m \cdot K$
$J = 7/2$
$g = 2$
$T_C = 293\ K$
$T_D = 184\ K$

B. Càlculs numèrics dels models

B.1. Model 1: Ezan et al. (2017) modificat

1. Hem donat un valor de $B_J(X)$ qualsevol en un rang entre -1 i 1, que son els valors possibles per la funció de Brillouin.

$$X = a \cdot \frac{B}{T} + b \cdot \frac{B_J(X)}{T}$$

On els valors a i b son constants per un determinat material:

$$a = \frac{g_J \cdot \mu_B \cdot J}{k_B}$$

$$b = \frac{3 \cdot T_C \cdot J}{J+1}$$

2. Amb el valor de X, es torna a buscar el valor de $B_J(X)$. Aquest procés és coneix com a iteració del punt fixe.

$$B_J(X) = \frac{c}{\tanh\,(c \cdot X)} - \frac{d}{\tanh\,(d \cdot X)}$$

On els valors de c i d son constants per un determinat material:

$$c = \frac{2J+1}{2J}$$

$$d = \frac{1}{2J}$$

Es continua iterant els dos valors fins tenir un error més petit que un 0.1%. Es necessita prou precisió en els càlculs, ja que pas del temps triat en el programa també pot ser molt petit.

3. Partim de la equació de l'entropia magnètica.

$$S_m(T,B) = \left(\frac{R_u}{M_m}\right) \cdot \left[\ln(\sinh(c \cdot X)) - \ln(\sinh(d \cdot X)) - X \cdot B_J(X)\right]$$

   a. Fem la derivada parcial de l'entropia magnètica respecte la temperatura, mantenint constant el camp magnètic.

$$\left(\frac{\partial S_m}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right) \cdot \left[\frac{c \cdot \cosh(c \cdot X)}{\sinh(c \cdot X)} \cdot \left(\frac{\partial X}{\partial T}\right) - \frac{d \cdot \cosh(d \cdot X)}{\sinh(d \cdot X)} \cdot \left(\frac{\partial X}{\partial T}\right) - \left(\frac{\partial X}{\partial T}\right) \cdot B_J(X) - X \cdot \frac{\partial B_J}{\partial T}\right]$$

$$\left(\frac{\partial S_m}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right)\cdot\left[\left(\frac{\partial X}{\partial T}\right)\cdot\underbrace{\left(\frac{c}{\tanh(c\cdot X)}-\frac{d}{\tanh(d\cdot X)}\right)}_{B_J(X)}-\left(\frac{\partial X}{\partial T}\right)\cdot B_J(X)-X\cdot\frac{\partial B_J}{\partial T}\right]$$

$$\left(\frac{\partial S_m}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right)\cdot\left[-X\cdot\frac{\partial B_J}{\partial T}\right]$$

b. Busquem la derivada de l'equació de Brillouin.

$$\frac{\partial B_J(X)}{\partial T} = \frac{-c}{\sinh^2(c\cdot X)}\cdot c\cdot\frac{\partial X}{\partial T}-\frac{-d}{\sinh^2(d\cdot X)}\cdot d\cdot\frac{\partial X}{\partial T}$$

$$\frac{\partial B_J(X)}{\partial T} = \underbrace{\left(\frac{d^2}{\sinh^2(d\cdot X)}-\frac{c^2}{\sinh^2(c\cdot X)}\right)}_{e}\cdot\frac{\partial X}{\partial T}$$

$$\frac{\partial B_J(X)}{\partial T} = e\cdot\frac{\partial X}{\partial T}$$

c. Busquem la derivada de l'equació X.

$$\frac{\partial X}{\partial T} = -a\cdot\frac{B}{T^2}+\left(-b\cdot\frac{B_J(X)}{T^2}+\frac{b}{T}\cdot\frac{\partial B_J(X)}{\partial T}\right)$$

d. Resolem la derivada parcial.

$$\frac{\partial B_J(X)}{\partial T} = e\cdot\left(a\cdot\frac{B}{T^2}-b\cdot\frac{B_J(X)}{T^2}+\frac{b}{T}\cdot\frac{\partial B_J(X)}{\partial T}\right)$$

$$\frac{\partial B_J(X)}{\partial T}\left(1-e\cdot\frac{b}{T}\right) = -\frac{e}{T^2}\cdot\left(a\cdot B+b\cdot B_J(X)\right)$$

$$\frac{\partial B_J(X)}{\partial T} = -\frac{e\cdot(a\cdot B+b\cdot B_J(X))}{T(T-e\cdot b)}$$

$$\boxed{\left(\frac{\partial S_m}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right)\cdot X\cdot e\cdot\frac{a\cdot B+b\cdot B_J(X)}{T(T-e\cdot b)}}$$

4. A partir de la equació de l'entropia de xarxa derivem.

$$S_l(T) = \left(\frac{R_u}{M_m}\right)\cdot\left[-3\cdot\ln\left(1-e^{-T_D/T}\right)+12\cdot\left(\frac{T}{T_D}\right)^3\cdot C\right]$$

On, per abreviar, hem anomenat a la integració C i la x d'aquesta equació és la variable d'integració.

$$C = \int_{x=0}^{T_D/T}\frac{x^3}{e^x-1}dx$$

Derivada:

$$\left(\frac{\partial S_l}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right)\cdot\left[-3\cdot\frac{e^{-T_D/T}}{e^{-T_D/T}-1}\cdot(-1)\cdot\left(-\frac{T_D}{T^2}\right)+36\cdot\frac{T^2}{T_D^3}\cdot C-12\cdot\frac{T_D}{T^2(e^{T_D/T}-1)}\right]$$

$$\boxed{\left(\frac{\partial S_l}{\partial T}\right)_B = \left(\frac{R_u}{M_m}\right)\cdot\left[-3\cdot\frac{T_D}{T^2}\cdot\frac{e^{-T_D/T}}{e^{-T_D/T}-1}+36\cdot\frac{T^2}{T_D^3}\cdot C-12\cdot\frac{T_D}{T^2(e^{T_D/T}-1)}\right]}$$

5. Un cop calculat totes les entalpies, podem calcular la calor específica:

$$\left(\frac{\partial S}{\partial T}\right)_B = \left(\frac{\partial S_m}{\partial T}\right)_B + \left(\frac{\partial S_l}{\partial T}\right)_B$$

$$\boxed{c_{p,B}(B,T) = T\cdot\left(\frac{\partial S}{\partial T}\right)_B}$$

6. El càlcul de la variació de temperatura adiabàtica:

$$\Delta T_{ad} = -\int_{B_0}^{B_1}\frac{T}{c_{p,B}(B,T)}\cdot\left(\frac{\partial S_m(B,T)}{\partial B}\right)_T dB$$

$$\Delta T_{ad} = -\int_{B_0}^{B_1}\frac{\left({}^{\partial S_m}/_{\partial B}\right)_T}{\left({}^{\partial S_l}/_{\partial T}\right)_B+\left({}^{\partial S_m}/_{\partial T}\right)_B}dB$$

   a. Cal calcular la derivada parcial de entropia magnètica respecte el camp magnètic a temperatura constant.

$$\left(\frac{\partial S_m}{\partial B}\right)_T = \left(\frac{R_u}{M_m}\right)\cdot\left[\frac{c\cdot\cosh(c\cdot X)}{\sinh(c\cdot X)}\cdot\left(\frac{\partial X}{\partial B}\right)-\frac{d\cdot\cosh(d\cdot X)}{\sinh(d\cdot X)}\cdot\left(\frac{\partial X}{\partial B}\right)-\left(\frac{\partial X}{\partial B}\right)\cdot B_J(X)-X\cdot\frac{\partial B_J}{\partial B}\right]$$

$$\left(\frac{\partial S_m}{\partial B}\right)_T = \left(\frac{R_u}{M_m}\right)\cdot\left[\left(\frac{\partial X}{\partial B}\right)\cdot\underbrace{\left(\frac{c}{\tanh(c\cdot X)}-\frac{d}{\tanh(d\cdot X)}\right)}_{B_J(X)}-\left(\frac{\partial X}{\partial B}\right)\cdot B_J(X)-X\cdot\frac{\partial B_J}{\partial B}\right]$$

$$\left(\frac{\partial S_m}{\partial B}\right)_T = \left(\frac{R_u}{M_m}\right)\cdot\left[-X\cdot\frac{\partial B_J}{\partial B}\right]$$

$$\frac{\partial B_J(X)}{\partial B} = \frac{-c}{\sinh^2(c\cdot X)}\cdot c\cdot\frac{\partial X}{\partial B}-\frac{-d}{\sinh^2(d\cdot X)}\cdot d\cdot\frac{\partial X}{\partial B}$$

$$\frac{\partial B_J(X)}{\partial B} = \underbrace{\left(\frac{d^2}{\sinh^2(d\cdot X)}-\frac{c^2}{\sinh^2(c\cdot X)}\right)}_{e}\cdot\frac{\partial X}{\partial B}$$

$$\frac{\partial B_J(X)}{\partial B} = e \cdot \frac{\partial X}{\partial B}$$

$$\frac{\partial X}{\partial B} = \frac{a}{T} + \frac{b}{T} \cdot \frac{\partial B_J(X)}{\partial B}$$

$$\frac{\partial B_J(X)}{\partial B} = e \cdot \left(\frac{a}{T} + \frac{b}{T} \cdot \frac{\partial B_J(X)}{\partial B}\right)$$

$$\frac{\partial B_J(X)}{\partial B} \cdot \left(1 - \frac{e \cdot b}{T}\right) = \frac{e \cdot a}{T} \rightarrow \frac{\partial B_J(X)}{\partial B} = \frac{e \cdot a}{T - e \cdot b}$$

$$\boxed{\left(\frac{\partial S_m}{\partial B}\right)_T = -\left(\frac{R_u}{M_m}\right) \cdot X \cdot \left(\frac{e \cdot a}{T - e \cdot b}\right)}$$

b. Finalment, l'equació quedarà:

$$\Delta T_{ad} = \int_{B_0}^{B_1} \frac{\left(\frac{R_u}{M_m}\right) \cdot X \cdot \left(\frac{e \cdot a}{T - e \cdot b}\right)}{\left({}^{\partial S_l}/_{\partial T}\right)_B + \left(\frac{R_u}{M_m}\right) \cdot X \cdot e \cdot \frac{a \cdot B + b \cdot B_J(X)}{T(T - e \cdot b)}} dB$$

$$\boxed{\Delta T_{ad} = \int_{B_0}^{B_1} \frac{X \cdot e \cdot a \cdot T \cdot \left(\frac{R_u}{M_m}\right)}{\left({}^{\partial S_l}/_{\partial T}\right)_B \cdot T \cdot (T - e \cdot b) + X \cdot \left(\frac{R_u}{M_m}\right) \cdot e \cdot (a \cdot B + b \cdot B_J(X))} dB}$$

7. Per últim, s'ha introduït aquesta diferència de temperatura en el ANSYS Fluent en l'equació d'energia del sòlid, com una font de calor volumètrica [W/m^3]

$$\boxed{mce = \frac{\rho_s \cdot c_{p,B}(T,B) \cdot \Delta T_{ad}(T,B)}{\Delta t}}$$

B.2. Model 3: Petersen et al. (2008)

Resolució de la calor específica magnètica:

$$\sigma^2 = f^2[a^2 \coth^2(aX) + b^2 \coth^2(bX) - 2ab \cdot \coth(aX) \cdot \coth(bX)]$$

On f fa referència a les constants del material: $N_s g J \mu_B$.

Simplificant l'equació, ens queda:

$$\sigma = f \cdot B_J(X)$$

Per tant, per fer la següent derivada:

$$\frac{\partial(\sigma^2)}{\partial T}$$

Hem dividit l'equació en tres troços:

$$\sigma^2 = f^2 \left[\underbrace{c^2 \coth^2(cX)}_{1} + \underbrace{d^2 \coth^2(dX)}_{2} - \underbrace{2cd \cdot \coth(cX) \cdot \coth(dX)}_{3} \right]$$

- 1r tros:

$$\frac{\partial}{\partial T}(c^2[\coth(cX) \cdot \coth(cX)])$$

$$2c^2 \left(-\frac{c}{\sinh^2(cX)}\right) \cdot \coth(cX) \cdot \frac{\partial X}{\partial T}$$

- 2n tros:

$$\frac{\partial}{\partial T}(d^2[\coth(dX) \cdot \coth(dX)])$$

$$2d^2 \left(-\frac{d}{\sinh^2(dX)}\right) \cdot \coth(dX) \cdot \frac{\partial X}{\partial T}$$

- 3r tros:

$$\frac{\partial}{\partial T}(2cd \cdot \coth(cX) \cdot \coth(dX))$$

$$2cd \left[\left(-\frac{c}{\sinh^2(cX)}\right) \cdot \coth(dX) \cdot \frac{\partial X}{\partial T} + \left(-\frac{d}{\sinh^2(dX)}\right) \cdot \coth(cX) \cdot \frac{\partial X}{\partial T} \right]$$

$$-2cd \left[\frac{c}{\sinh^2(cX)} \coth(dX) + \frac{d}{\sinh^2(dX)} \coth(cX) \right] \cdot \frac{\partial X}{\partial T}$$

L'equació quedarà:

$$\sigma^2 = f^2 \left[-\frac{2a^3}{\sinh^2(aX)} \coth(aX) - \frac{2b^3}{\sinh^2(bX)} \coth(bX) + \frac{2a^2 b}{\sinh^2(aX)} \coth(bX) + \frac{2ab^2}{\sinh^2(bX)} \coth(aX) \right] \cdot \frac{\partial X}{\partial T}$$

La derivada parcial de X és la mateixa que la trobada en el model anterior:

$$\frac{\partial X}{\partial T} = -\frac{a}{T^2}B - \frac{b}{T^2}B_J(X) + \frac{b}{T}\cdot\frac{\partial B_J(X)}{\partial T}$$

De la mateixa manera amb l'equació de Brillouin:

$$\frac{\partial B_J(X)}{\partial T} = e\cdot\frac{\partial X}{\partial T}$$

Resolent les dos equacions, obtenim que finalment:

$$\frac{\partial X}{\partial T} = \frac{(-aB - bB_J(X))}{T(T-be)}$$

Per calcular la calor específica magnètica també calcular la derivada de la magnetització específica respecte la temperatura:

$$\frac{\partial \sigma}{\partial T} = f\cdot\frac{\partial B_J(X)}{\partial T}$$

$$\frac{\partial \sigma}{\partial T} = f\cdot e\cdot\frac{(-aB - bB_J(X))}{T(T-be)}$$

Aquesta mateixa funció serà la utilitzada en el càlcul de la temperatura adiabàtica:

$$\Delta T_{ad} = -\mu_0\int\frac{T}{c_p}\cdot\left(\frac{\partial\sigma}{\partial T}\right)_B dB$$

C. Scripts en Matlab dels models

C.1. Model 1: Ezan et al. (2017) modificat

```
T = 293;
Td = 184;
Tc = 293;
den = 7900;
RuMm = 8.31434/0.15725;
a = (2*(9.2740154e-24)*3.5)/(1.380662e-23);
b = (3*Tc*3.5)/(3.5+1);
c = (2*3.5+1)/(2*3.5);
d = 1/(2*3.5);
mu_0 = pi*4e-7;
B_ant = 0;
H_appl = 2;
timestep = 0.001;

B_J = 0.5;
X = 0.1;
for n_iter=1:1500
    B_J_ant = B_J;
    X_ant = X;
    X = (a*H_appl)/T + (b*B_J)/T;
    B_J = c/(tanh(c*X)) - d/(tanh(d*X));
    if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.0001) && (((abs(X - X_ant))/abs(X)) < 0.0001))
            break;
    end
end
e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);
dSm = (RuMm)*(X*e)*(a*H_appl+b*B_J)/((T-e*b)*T);
Sm = RuMm*(log(sinh(c*X))-log(sinh(d*X))-X*B_J);

M = (3.83e24)*2*(9.274009e-24)*3.5*B_J*den;
H_dem = 0.16*M*(pi*(4e-7));
B = H_appl; %- H_dem;

int_result = 0.0;
for x_int=0.00001:0.0001:(Td/T)
    int_result = ((x_int^3)/(exp(x_int)-1))*0.0001 + int_result;
end

dS_l = (RuMm)*((-3*Td*exp(-Td/T))/(T^2*(exp(-Td/T)-1)) + 36*(T^2)*int_result/(Td^3) - 12*Td/((T^2)*(exp(Td/T)-1)));

cap = T*(dSm + dS_l);

delta_T = 0.0;
B_inter = 0.0;
for b_iter=1:10000000
    for m_iter=1:10000
        B_J_ant = B_J;
        X_ant = X;
        X = (a*B_inter)/T + (b*B_J)/T;
        B_J = c/(tanh(c*X)) - d/(tanh(d*X));
        if ((((abs(B_J - B_J_ant))/abs(B_J)) < 0.000001) && (((abs(X - X_ant))/abs(X)) < 0.000001))
            break;
        end
    end
    e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);
    delta_T = ((X*e*a*RuMm*T)/(dS_l*T*(T-e*b)+X*e*RuMm*(a*B_inter+b*B_J)))*(timestep*0.001) + delta_T;
    B_inter = timestep*0.001 + B_inter;
    if (B_inter >= B)
        break;
    end
end
mce = den*cap*delta_T/timestep;
delta_T;
```

C.2. Model 2: Ezan et al. (2017) modificat aplicat en un període de temps

```
T = 293;
Td = 184;
Tc = 293;
den = 7900;
RuMm = 8.31434/0.15725;
a = (2*(9.2740154e-24)*3.5)/(1.380662e-23);
b = (3*Tc*3.5)/(3.5+1);
c = (2*3.5+1)/(2*3.5);
d = 1/(2*3.5);
mu_0 = pi*4e-7;
timestep = 0.0001;

B_fin = 1.1;
B = 0;
delta_T_ac = 0;
for iter_step=1:100000000
    B_J = 0.5;
    X = 0.1;
    for n_iter=1:1500
        B_J_ant = B_J;
        X_ant = X;
        X = (a*B)/T + (b*B_J)/T;
        B_J = c/(tanh(c*X)) - d/(tanh(d*X));
        if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.0001) && (((abs(X - X_ant))/abs(X)) < 0.0001))
            break;
        end
    end
    e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);

    int_result = 0.0;
    for x_int=0.00001:0.0001:(Td/T)
        int_result = ((x_int^3)/(exp(x_int)-1))*0.0001 + int_result;
    end
    dS_l = (RuMm)*((-3*Td*exp(-Td/T))/(T^2*(exp(-Td/T)-1)) + 36*(T^2)*int_result/(Td^3) - 12*Td/((T^2)*(exp(Td/T)-1)));

    B_int = B;
    delta_T = 0;
    for b_iter=1:100000
        for m_iter=1:100000
            B_J_ant = B_J;
            X_ant = X;
            X = (a*B_int)/T + (b*B_J)/T;
            B_J = c/(tanh(c*X)) - d/(tanh(d*X));
            if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.0001) && (((abs(X - X_ant))/abs(X)) < 0.0001))
                break;
            end
        end

        e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);
        delta_T = delta_T + ((X*e*a*RuMm*T)/(dS_l*T*(T-e*b)+X*e*RuMm*(a*B_int+b*B_J)))*(timestep*0.0001);
        B_int = B_int + timestep*0.0001;
        if B_int >= (B + timestep)
            break;
        end
    end
    delta_T_ac = delta_T_ac + delta_T;
    T = T + delta_T;
    B = B + timestep;
    if B >= B_fin
        break;
    end
end
delta_T_ac;
```

C.3. Model 3: Petersen et al. (2008)

```
T = 220;
H_appl = 2;

Td = 169;
Tc = 293;
den = 7900;
N_s = 3.83e24;
N = 3.83e24;
g = 2;
J = 7/2;
gam_e = 6.93e-2;

mu_0 = pi*(4e-7);
k_b = 1.380662e-23;
mu_b = 9.2740154e-24;
a = (2*J+1)/(2*J);
b = 1/(2*J);
c = (g*mu_b*J)/k_b;
d = (3*Tc*J)/(J+1);
e = N_s*g*J*mu_b;
N_int = (3*k_b*Td)/(N_s*(g^2)*(mu_b^2)*J*(J+1));
Nd = 0.16;

int_result = 0.0;
for x_int=0.000001:0.00001:(Td/T)
    int_result = (((x_int^4)*(exp(x_int)))/((exp(x_int)-1)^2))*0.00001 + int_result;
end
c_l = 9*N*k_b*((T/Td)^3)*int_result;

c_e = gam_e*T;

B_J = 0.5;
x= 0.1;
for n_iter=1:10000
    B_J_ant = B_J;
    x_ant = x;
    x = (c*H_appl)/T + (d*B_J)/T;
    B_J = a/(tanh(a*x)) - b/(tanh(b*x));
    if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.00001) && (((abs(x - x_ant))/abs(x)) <
    0.00001))
        break;
    end
end
```

```
43
44  M = e*B_J*den*mu_0;
45  H_int = H_appl; %-Nd*M;
46
47  T_result = 0.0;
48  for t_int=0.000001:0.00001:H_int
49      for m_iter=1:10000
50          B_J_ant = B_J;
51          x_ant = x;
52          x = (c*t_int)/T + (d*B_J)/T;
53          B_J = a/(tanh(a*x)) - b/(tanh(b*x));
54          if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.00001) && (((abs(x - x_ant))/abs(x))
            < 0.00001))
55                  break;
56          end
57      end
58      f = (b^2)/(sinh(b*x)^2)-(a^2)/(sinh(a*x)^2);
59
60      m_T1 = f*((-c*t_int-d*B_J)/(T*(T-d*f)));
61      m_T2 =
        (-2*a^3*coth(a*x)/(sinh(a*x))^2-2*b^3*coth(b*x)/(sinh(b*x))^2+2*a^2*b*coth(b*x)/(s
        inh(a*x))^2+2*b^2*a*coth(a*x)/(sinh(b*x))^2)*((-c*t_int-d*B_J)/(T*(T-d*f)));
62      c_m = -t_int*e*m_T1-N_int*(e^2)*m_T2/2;
63
64      c_p = c_m+c_l+c_e;
65
66      T_result = T_result + ((T*e*m_T1)/(c_p))*0.00001;
67  end
68
69  Delta_T = -T_result;
```

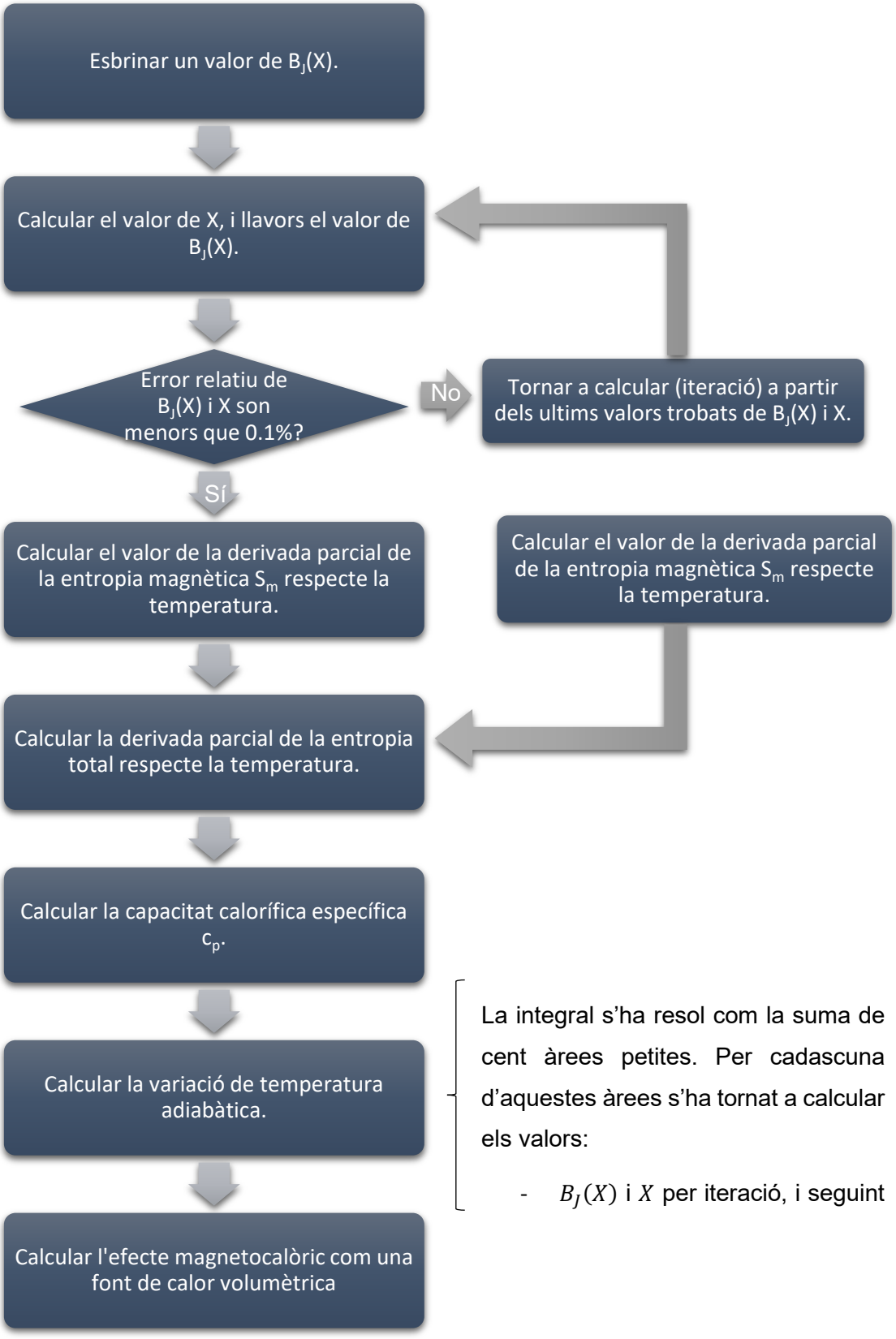
C.4. Model 4: Bouchard et al. (2009) i Ezan et al (2017)

```
1   T = 293;
2   H_appl = 1.1;
3
4   Td = 184;
5   Tc = 293;
6   den = 7900;
7   N_s = 3.83e24;
8   N = 3.83e24;
9   g = 2;
10  J = 7/2;
11  gam_e = 6.93e-2;
12  RuMm = 8.31434/0.15725;
13  k_b = 1.380662e-23;
14  mu_b = 9.2740154e-24;
15  a = (2*(9.2740154e-24)*3.5)/(1.380662e-23);
16  b = (3*Tc*3.5)/(3.5+1);
17  c = (2*3.5+1)/(2*3.5);
18  d = 1/(2*3.5);
19  f = N_s*g*J*mu_b;
20  mu_0 = pi*4e-7;
21  B_ant = 0;
22  timestep = 0.001;
23
24  B_J = 0.5;
25  X = 0.1;
26  for m_iter=1:100000
27      B_J_ant = B_J;
28      X_ant = X;
29      X = (b*B_J)/T;
30      B_J = c/(tanh(c*X)) - d/(tanh(d*X));
31      if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.00001) && (((abs(X - X_ant))/abs(X)) <
        0.00001))
32              break;
33      end
34  end
35  e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);
36  dSm_0 = (RuMm)*(X*e)*(b*B_J)/((T-e*b)*T);
37
38  if (T <= 291)
39      cap_0 = (7.22705396e-6)*T^4-(6.76611611e-3)*T^3+2.37368*T^2-368.997*T+(2.16496e4);
40  else
41      cap_0 =
        (-5.62735110e-6)*T^5+(9.17772113e-3)*T^4-5.98335371*T^3+(1.94914902e3)*T^2-(3.1727
        6931e5)*T+(2.06453722e7);
42  end
```

```
43
44  delta_T = 0;
45  for b_iter=0.0001:0.0001:H_appl
46      for n_iter=1:1500
47          B_J_ant = B_J;
48          X_ant = X;
49          X = (a*b_iter)/T + (b*B_J)/T;
50          B_J = c/(tanh(c*X)) - d/(tanh(d*X));
51          if (((abs(B_J - B_J_ant)/abs(B_J)) < 0.0001) && (((abs(X - X_ant))/abs(X)) <
            0.0001))
52              break;
53          end
54      end
55      e = (d^2)/((sinh(d*X))^2) - (c^2)/((sinh(c*X))^2);
56      dSm_H = (RuMm)*(X*e)*(a*b_iter+b*B_J)/((T-e*b)*T);
57      cap = cap_0 - T*(dSm_0-dSm_H);
58      M = (f*e/T)*(-a*b_iter-b*B_J)/(T-e*b);
59      delta_T = delta_T + (T*M/cap)*0.0001;
60  end
61  delta_T = -delta_T;
```

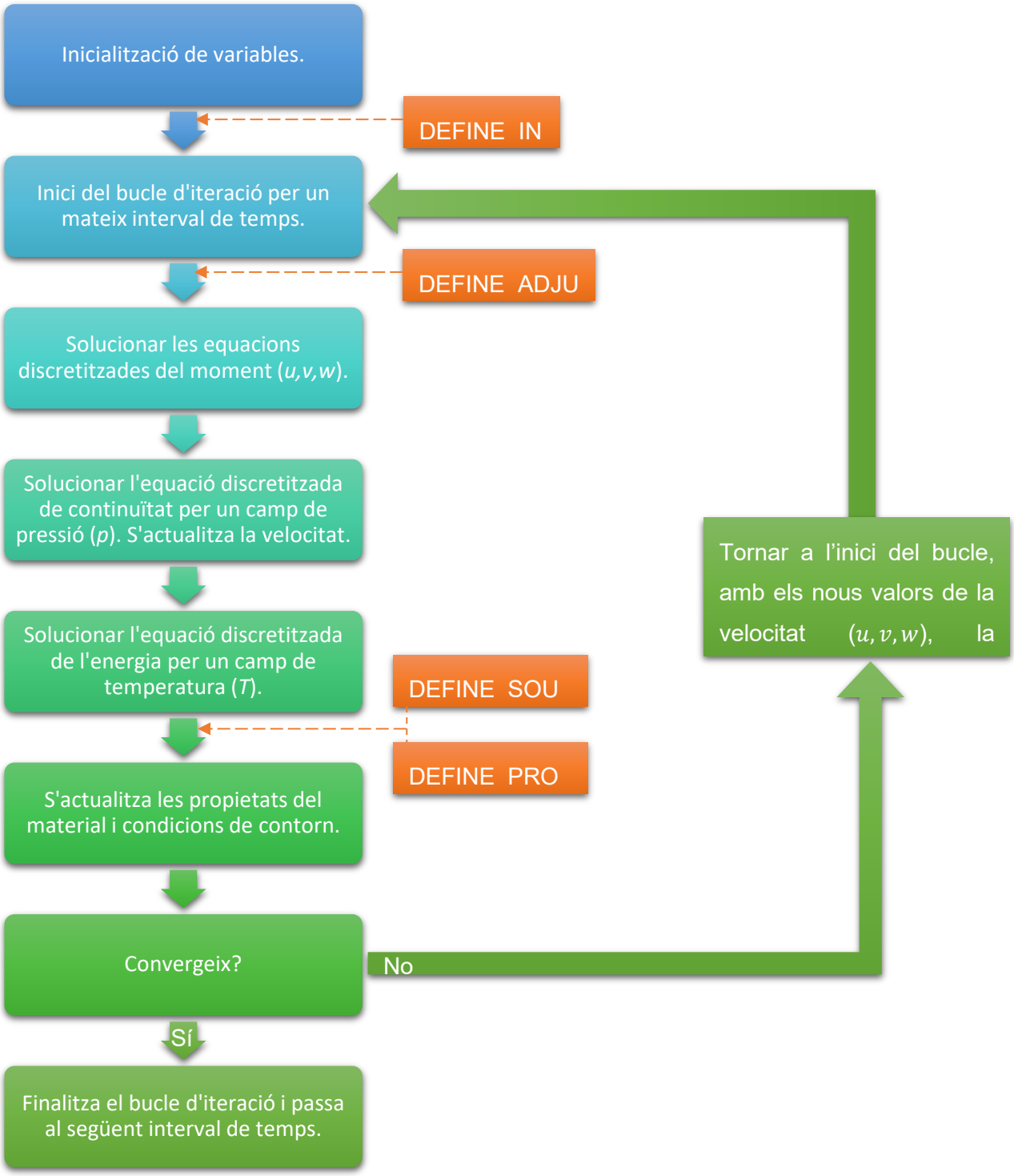
D. ANSYS FLUENT

D.1. Esquema d'iteració de l'UDF amb el ANSYS FLUENT.



La integral s'ha resol com la suma de cent àrees petites. Per cadascuna d'aquestes àrees s'ha tornat a calcular els valors:

- $B_J(X)$ i X per iteració, i seguint


Inicialització de variables.
DEFINE IN
Inici del bucle d'iteració per un mateix interval de temps.
DEFINE ADJU
Solucionar les equacions discretitzades del moment (u,v,w).
Solucionar l'equació discretitzada de continuïtat per un camp de pressió (p). S'actualitza la velocitat.
Solucionar l'equació discretitzada de l'energia per un camp de temperatura (T).
DEFINE SOU
DEFINE PRO
S'actualitza les propietats del material i condicions de contorn.
Convergeix?
No
Sí
Finalitza el bucle d'iteració i passa al següent interval de temps.
Tornar a l'inici del bucle, amb els nous valors de la velocitat (u, v, w), la

D.2. UDF del model 2

```
#include "udf.h"
#include "mem.h"

#define J 3.5                /* Total angular momentum */
#define g 2                  /* Spectroscopic splitting factor */
#define mu_B 9.2740154*pow(10,-24)   /* Bohr magneton */
#define k_B 1.380662*pow(10,-23)     /* Boltzmann constant */
#define n 6.023*pow(10,23)       /* Avogadro number */
#define Tc 293               /* Curie Temperature */
#define Td 184               /* Debye Temperature */
#define PI 3.141592654           /* Pi number */
#define den 7900.0

static real B;
static real dSm;
static real B_ant;
static real B_J;
static real X;
static real RuMm;
static real a;
static real b;
static real c;
static real d;
static real e;
static real T;
static real delta_T;
static real mce;
static real mce_ac;
static real cap;
static real dS_l;
static int last_ts = -1;
static real T_cold;

/*   Define initial variables   */

static real mu_0;   /* Permeability */
static real n_cicles;   /* N° cicles */
static real tm;     /* Magnetization time */
static real tf;     /* FluidFlow time */
static real tc;     /* Cycle time */
static real t_act;  /* Actual cycle time */
static real B_vel;  /* Magnetic field applied change respect the time */

DEFINE_INIT(init_code, domain)
{
    RuMm = 8.31434/0.15725;
    a = (g*mu_B*J)/(k_B);
    b = (3*Tc*J)/(J+1);
    c = (2*J+1)/(2*J);
    d = 1/(2*J);

```

```
52      mu_0 = (4.0*PI)*pow(10,-7);
53      n_cicles = 1.0;
54      t_act = 0.0;
55      tm = 3.0;
56      tf = 3.0;
57      tc = 2.0*tm + 2.0*tf;
58      B = 0.0;
59      B_ant = 0.0;
60      T = 298.0;
61      T_cold = 298.0;
62      delta_T = 0.0;
63      mce = 0.0;
64      mce_ac = 0.0;
65      cap = 155.7363;
66      B_vel = 0.33333;    /* Velocitat de canvi del camp mangètic amb el temps [T/s] */
67  }
68
69  DEFINE_ADJUST(my_adjust,domain)
70  {
71      real tt = CURRENT_TIME;
72      if (tt >= n_cicles*tc)
73      {

74          n_cicles++;
75      }
76      t_act = tt - (n_cicles - 1.0)*tc;
77
78
79      if ((t_act >= 0.0) && (t_act <= tm))
80      {
81          B = B_vel*t_act;
82          if (t_act > 0.0)
83          {
84              B_ant = B - CURRENT_TIMESTEP;
85          }
86      }
87      else if ((t_act > tm) && (t_act <= (tm+tf)))
88      {
89          B = B_vel*tm;
90          B_ant = B;
91      }
92      else if ((t_act > (tm+tf)) && (t_act <= (2*tm+tf)))
93      {
94          B = B_vel*(2*tm + tf - t_act);
95          B_ant = B + CURRENT_TIMESTEP;
96      }
97      else if ((t_act > (2*tm+tf)) && (t_act < tc))
98      {
99          B = 0.0;
100         B_ant = 0.0;
101     }
102
103     int curr_ts;
104     curr_ts = N_TIME;
105     cell_t cell;
106     int ID = 4;
107     Thread *thread = Lookup_Thread(domain, ID);
```

```
107     Thread *thread = Lookup_Thread(domain, ID);
108     if (last_ts != curr_ts)
109     {
110         last_ts = curr_ts;
111         begin_c_loop(cell,thread)
112         {
113             T = C_T(cell,thread);
114             B_J = 0.5;
115             X = 0.1;
116             int n_iter;
117             for (n_iter=1;n_iter!=1500;n_iter++)
118             {
119                 real B_J_ant = B_J;
120                 real X_ant = X;
121                 X = (a*B)/T + (b*B_J)/T;
122                 B_J = c/(tanh(c*X)) - d/(tanh(d*X));
123                 if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) && (((fabs(X -
                    X_ant))/fabs(X)) < 0.00001))
124                 {
125                     break;
126                 }
127             }
128             e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
129             dSm = (RuMm)*(X*e)*(a*B+b*B_J)/((T-e*b)*T);
130 
131             real int_result = 0.0;
132             real z = 0.00001;
133             real div = Td/T;
134             int x_int;
135             for (x_int=1;x_int!=10000;x_int++)
136             {
137                 int_result += ((pow(z,3))/(exp(z)-1))*0.0001;
138                 z += 0.0001;
139                 if (z >= div)
140                 {
141                     break;
142                 }
143             }
144             dS_l = (RuMm)*((-3*Td*exp(-Td/T))/(pow(T,2)*(exp(-Td/T)-1)) +
                36*(pow(T,2))*int_result/(pow(Td,3)) - 12*Td/((pow(T,2))*(exp(Td/T)-1)));
```

```
145
146             cap = T*(dSm + dS_l);
147             C_UDMI(cell,thread,0) = cap;
148
149             if (((t_act > 0.0) && (t_act <= tm)) || ((t_act > (tm+tf)) && (t_act <=
                (2*tm+tf))))
150             {
151
152                 real B_inter = B_ant;
153                 real B_max;
154                 delta_T = 0.0;
155                 int b_iter;
156                 for (b_iter=1;b_iter!=20000;b_iter++)
157                 {
158                     real m_iter;
159                     for (m_iter=1;m_iter=1000;m_iter++)
160                     {
161                         real B_J_ant = B_J;
162                         real X_ant = X;
163                         X = (a*B_inter)/T + (b*B_J)/T;
164                         B_J = c/(tanh(c*X)) - d/(tanh(d*X));
165                         if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) &&
                          (((fabs(X - X_ant))/fabs(X)) < 0.00001))
166                         {
167                             break;
168                         }
169                     }
170                     e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
171                     delta_T +=
                        ((X*e*a*RuMm*T)/(dS_l*T*(T-e*b)+X*e*RuMm*(a*B_inter+b*B_J)))*((B-B
                        _ant)*0.01);
172                     if ((B > B_ant) && (t_act > 0.0) && (t_act <= tm))
173                     {
174                         B_inter += CURRENT_TIMESTEP*0.01;
175                         if (B_inter >= B)
176                         {
177                             break;
178                         }
179                     }
180                     else if ((B < B_ant) && (t_act > (tm+tf)) && (t_act <= (2*tm+tf)))
181                     {
182                         B_inter -= CURRENT_TIMESTEP*0.01;
183                         if (B_inter <= B)
184                         {
185                             break;
186                         }
187                     }
188                 }
189
190                  C_UDMI(cell,thread,1) += delta_T;
191                  C_UDMI(cell,thread,2) = den*cap*delta_T/CURRENT_TIMESTEP;
192              }
193              else
194              {
195                  C_UDMI(cell,thread,2) = 0.0;
196              }
197
198          }
199          end_c_loop(cell,thread)
200      }
201
202  }
203
204  DEFINE_SOURCE(mce_source,cell,thread,dS,eqn)
205  {
206      real x[ND_ND];
207      C_CENTROID(x,cell,thread);
208      dS[eqn] = 0.0;
209      mce_ac = C_UDMI(cell,thread,2);
210      return mce_ac;
211  }
212
213  DEFINE_SPECIFIC_HEAT(my_spec_cap,T,Tref,h,yi)
```

```
214  {
215      Domain *domain;
216      domain = Get_Domain(1);
217      cell_t cell;
218      int ID = 4;
219      Thread *thread = Lookup_Thread(domain, ID);
220      begin_c_loop(cell,thread)
221      {
222          real cap_prop = C_UDMI(cell,thread,0);
223          *h = cap_prop*(T-Tref);
224          return cap_prop;
225      }
226      end_c_loop(cell,thread)
227  }
228
229  DEFINE_PROFILE(change_fluid_dir,thread2,i)
230  {
231      face_t f;
232      real x[ND_ND];
233      real v = 0.02/3;
234      begin_f_loop(f,thread2)
235      {
236          F_CENTROID(x,f,thread2);
237          if ((t_act >= 0.0) && (t_act <= tm))
238              F_PROFILE(f,thread2,i) = 0;
239          else if ((t_act > tm) && (t_act <= (tm+tf)))
240              F_PROFILE(f,thread2,i) = v;
241          else if ((t_act > (tm+tf)) && (t_act <= (2*tm+tf)))
242              F_PROFILE(f,thread2,i) = 0.0;
243          else if ((t_act > (2*tm+tf)) && (t_act <= tc))
244              F_PROFILE(f,thread2,i) = -v;
245      }
246      end_f_loop(f,thread2)
247  }
248
249


250  DEFINE_PROFILE(change_fluid_T_cold,thread3,i)
251  {
252      face_t f;
253      real x[ND_ND];
254      begin_f_loop(f,thread3)
255      {
256          F_CENTROID(x,f,thread3);
257          if ((t_act >= 0.0) && (t_act < CURRENT_TIMESTEP))
258          {
259              T_cold=F_T(f,thread3);
260          }
261          F_PROFILE(f,thread3,i)=T_cold;
262
263      }
264      end_f_loop(f,thread3)
265
266  }
267
268  DEFINE_DELTAT(mydeltat,d)
269  {
270      real time_step;
271      if ((t_act >= 8.98 && t_act <= 9.02) || (t_act >= 11.97 && t_act <= 12.03))
272      {
273          time_step = 0.033;
274      }
275      else
276      {
277          time_step = 0.011;
278      }
279      return time_step;
280  }
281
```

D.3. UDF del model 4 (prototip 1)

```
#include "udf.h"
#include "mem.h"

#define J 3.5                /* Total angular momentum */
#define g 2                  /* Spectroscopic splitting factor */
#define mu_B 9.2740154*pow(10,-24)  /* Bohr magneton */
#define k_B 1.380662*pow(10,-23)    /* Boltzmann constant */
#define n 6.023*pow(10,23)      /* Avogadro number */
#define Tc 293               /* Curie Temperature */
#define Td 184               /* Debye Temperature */
#define PI 3.141592654           /* Pi number */
#define den 7900.0           /* Densitat Gd */
#define N_s 3.83*pow(10,24)      /* Number of magnetic spins per unit of mass */
#define N 3.83*pow(10,24)        /* Number of atoms per unit of mass */
#define gam_e 6.93*pow(10,-2)        /* Sommerfeld constant */


static real B_max;
static real B_min;
static real B;
static real B_ant;
static real B_J;
static real X;
static real RuMm;
static real a;
static real b;
static real c;
static real d;
static real e;
static real f;
static real T;
static real delta_T;
static real mce;
static real mce_ac;
static real cap;
static int last_ts = -1;
static real T_cold;

/*   Define initial variables   */

static real mu_0;   /* Permeability */
static real n_cicles;   /* N° cicles */
static real tm;     /* Magnetization time */
static real tf;     /* FluidFlow time */
static real tc;     /* Cycle time */
static real t_act;  /* Actual cycle time */
static real B_vel;  /* Magnetic field applied change respect the time */

```

```
49  DEFINE_INIT(init_code, domain)
50  {
51      /* Definició d'algunes variables */
52      mu_0 = (4.0*PI)*pow(10,-7);
53      RuMm = 8.31434/0.15725;
54      a = (g*mu_B*J)/(k_B);
55      b = (3*Tc*J)/(J+1);
56      c = (2*J+1)/(2*J);
57      d = 1/(2*J);
58      f = N_s*g*J*mu_B;
59
60      n_cicles = 1.0;
61      t_act = 0.0;
62      tm = 3;
63      tf = 3;
64      tc = 2.0*tm + 2.0*tf;
65
66      /* Initzialització de algunes variables */
67      B_max = 0.97;
68      B_min = 0.16;
69
70      T = 298.0;
71      T_cold = 298.0;
72      delta_T = 0.0;
73      mce = 0.0;

74      mce_ac = 0.0;
75  }
76
77  DEFINE_ADJUST(my_adjust,domain)
78  {
79      real tt = CURRENT_TIME;
80      if (tt >= n_cicles*tc)
81      {
82          n_cicles++;
83      }
84      t_act = tt - (n_cicles - 1.0)*tc;
85
86
87      if ((t_act >= 0.0) && (t_act < (tm+tf)))
88      {
89          B = B_max;
90          B_ant = B_min;
91      }
92      else if ((t_act >= (tm+tf)) && (t_act < tc))
93      {
94          B = B_min;
95          B_ant = B_max;
96      }
97
98      int curr_ts;
99      curr_ts = N_TIME;
100     cell_t cell;
101     int ID = 4;
102     Thread *thread = Lookup_Thread(domain, ID);
103     if (last_ts != curr_ts)
104     {
105         real dSm_0;
106         real dSm_H;
107         real cap_0;
108         real M;
109         last_ts = curr_ts;
110         begin_c_loop(cell,thread)
111         {
112             T = C_T(cell,thread);
113             if (T <= 291)
114             {
115                     cap_0 =
                        (7.22705396*pow(10,-6))*pow(T,4)-(6.76611611*pow(10,-3))*pow(T,3)+
                        2.37368*pow(T,2)-368.997*T+(2.16496*pow(10,4));
116             }
117             else
```

```
118             {
119                     cap_0 =
                        (-5.62735110*pow(10,-6))*pow(T,5)+(9.17772113*pow(10,-3))*pow(T,4)
                        -5.98335371*pow(T,3)+(1.94914902*pow(10,3))*pow(T,2)-(3.17276931*p
                        ow(10,5))*T+(2.06453722*pow(10,7));
120             }
121
122             if ((t_act >= 0.0) && (t_act < (tm+tf+CURRENT_TIMESTEP)))
123             {
124                 B_J = 0.5;
125                 X = 0.1;
126                 int c0_iter;
127                 for (c0_iter=1;c0_iter!=1500;c0_iter++)
128                 {
129                     real B_J_ant = B_J;
130                     real X_ant = X;
131                     X = (b*B_J)/T;
132                     B_J = c/(tanh(c*X)) - d/(tanh(d*X));
133                     if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) && (((fabs(X -
                        X_ant))/fabs(X)) < 0.00001))
134                     {
135                         break;
136                     }
137                 }
138                 e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
139                 dSm_0 = (RuMm)*(X*e)*(b*B_J)/((T-e*b)*T);
140
141                 int cH_iter;
142                 for (cH_iter=1;cH_iter=10000;cH_iter++)
143                 {
144                     real B_J_ant = B_J;
145                     real X_ant = X;
146                     X = (a*B)/T + (b*B_J)/T;
147                     B_J = c/(tanh(c*X)) - d/(tanh(d*X));
148                     if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) && (((fabs(X -
                        X_ant))/fabs(X)) < 0.00001))
149                     {
150                         break;
151                     }
152                 }
153                 e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
154                 dSm_H = (RuMm)*(X*e)*(a*B+b*B_J)/((T-e*b)*T);
155                 cap = cap_0 - T*(dSm_0-dSm_H);
156
157                 C_UDMI(cell,thread,0) = cap;
158
159                 if (((t_act > CURRENT_TIMESTEP) && (t_act <= (2*CURRENT_TIMESTEP)))
                    || ((t_act >= (tm+tf)) && (t_act < (tm+tf+CURRENT_TIMESTEP))))
160                 {
161                     real B_inter = B_ant;
162                     delta_T = 0.0;
163                     int b_iter;
164                     for (b_iter=1;b_iter!=20000;b_iter++)
165                     {
166                         real m_iter;
167                         for (m_iter=1;m_iter=10000;m_iter++)
168                         {
169                             real B_J_ant = B_J;
170                             real X_ant = X;
171                             X = (a*B_inter)/T + (b*B_J)/T;
172                             B_J = c/(tanh(c*X)) - d/(tanh(d*X));
173                             if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) &&
                                (((fabs(X - X_ant))/fabs(X)) < 0.00001))
174                             {
175                                 break;
176                             }
177                         }
178                         e = (pow(d,2))/(pow(sinh(d*X),2)) -
                            (pow(c,2))/(pow(sinh(c*X),2));
179                         dSm_H = (RuMm)*(X*e)*(a*B_inter+b*B_J)/((T-e*b)*T);
180                         cap = cap_0 - T*(dSm_0-dSm_H);
181                         M = (f*e/T)*(-a*B_inter-b*B_J)/(T-e*b);
182                         delta_T = delta_T - (T*M/cap)*CURRENT_TIMESTEP*0.01;
```

```
183
184                         if ((B > B_ant) && (t_act > CURRENT_TIMESTEP) && (t_act <=
                            (2*CURRENT_TIMESTEP)))
185                         {
186                             B_inter += CURRENT_TIMESTEP*0.01;
187                             if (B_inter >= B)
188                             {
189                                 break;
190                             }
191                         }
192                         else if ((B < B_ant) && (t_act >= (tm+tf)) && (t_act <
                            (tm+tf+CURRENT_TIMESTEP)))
193                         {
194                             B_inter -= CURRENT_TIMESTEP*0.01;
195                             if (B_inter <= B)
196                             {
197                                 delta_T = - delta_T;
198                                 break;
199                             }
200                         }
201                     }
202                     C_UDMI(cell,thread,1) = delta_T;
203                     C_UDMI(cell,thread,2) = den*cap*delta_T/CURRENT_TIMESTEP;
204                 }
205                 else
206                 {
207                     C UDMI(cell,thread,2) = 0.0;

208                  }
209
210             }
211             else
212             {
213                 C_UDMI(cell,thread,0) = cap_0;
214                 C_UDMI(cell,thread,2) = 0.0;
215             }
216         }
217         end_c_loop(cell,thread)
218     }
219
220 }
221
222 DEFINE_SOURCE(mce_source,cell,thread,dS,eqn)
223 {
224     real x[ND_ND];
225     C_CENTROID(x,cell,thread);
226     dS[eqn] = 0.0;
227     mce_ac = C_UDMI(cell,thread,2);
228     return mce_ac;
229 }
230
231 DEFINE_SPECIFIC_HEAT(my_spec_cap,T,Tref,h,yi)
232 {
233     Domain *domain = Get_Domain(1);
234     int ID = 4;
235     Thread *thread = Lookup_Thread(domain,ID);
236     cell_t cell;
237     real cap_prop;
238     begin_c_loop(cell,thread)
239     {
240         cap_prop = C_UDMI(cell,thread,0);
241         *h = cap_prop*(T-Tref);
242         return cap_prop;
243     }
244     end_c_loop(cell,thread)
245 }
246
```

```
247  DEFINE_PROFILE(change_fluid_dir,thread2,i)
248  {
249      face_t f;
250      real x[ND_ND];
251      real v = 0.02/3;
252      begin_f_loop(f,thread2)
253      {
254          F_CENTROID(x,f,thread2);
255          if ((t_act >= 0.0) && (t_act <= tm))
256              F_PROFILE(f,thread2,i) = 0;
257          else if ((t_act > tm) && (t_act <= (tm+tf)))
258              F_PROFILE(f,thread2,i) = v;
259          else if ((t_act > (tm+tf)) && (t_act <= (2*tm+tf)))
260              F_PROFILE(f,thread2,i) = 0.0;
261          else if ((t_act > (2*tm+tf)) && (t_act <= tc))
262              F_PROFILE(f,thread2,i) = -v;
263      }
264      end_f_loop(f,thread2)
265  }
266
267
268  DEFINE_PROFILE(change_fluid_T_cold,thread3,i)
269  {
270      face_t f;
271      real x[ND_ND];
272      begin_f_loop(f,thread3)
273      {
274          F_CENTROID(x,f,thread3);
275          if ((t_act >= 0.0) && (t_act <= CURRENT_TIMESTEP))
276          {
277              T_cold=F_T(f,thread3);
278          }
279          F_PROFILE(f,thread3,i)=T_cold;
280
281      }
282      end_f_loop(f,thread3)
283
284  }
285
286  DEFINE_DELTAT(mydeltat,d)
287  {
288      real time_step;
289      if ((t_act >= 8.98 && t_act <= 9.02) || (t_act >= 11.99 && t_act <= 12.03))
290      {
291          time_step = 0.031;
292      }
293      else
294      {
295          time_step = 0.01;
296      }
297      return time_step;
298  }
299
300
```

D.4. UDF amb moviment del mallat (prototip 2)

```
#include "udf.h"
#include "mem.h"
#include "dynamesh_tools.h"

#define J 3.5                 /* Total angular momentum */
#define g 2                   /* Spectroscopic splitting factor */
#define mu_B 9.2740154*pow(10,-24)   /* Bohr magneton */
#define k_B 1.380662*pow(10,-23)     /* Boltzmann constant */
#define n 6.023*pow(10,23)        /* Avogadro number */
#define Tc 293                /* Curie Temperature */
#define Td 184                /* Debye Temperature */
#define PI 3.141592654            /* Pi number */
#define den 7900.0            /* Densitat Gd */
#define N_s 3.83*pow(10,24)       /* Number of magnetic spins per unit of mass */
#define N 3.83*pow(10,24)         /* Number of atoms per unit of mass */
#define gam_e 6.93*pow(10,-2)         /* Sommerfeld constant */


static real B_max;
static real B_min;
static real B;
static real B_ant;
static real B_J;
static real X;
static real RuMm;
static real a;
static real b;
static real c;
static real d;
static real e;
static real f;
static real T;
static real delta_T;
static real mce;
static real mce_ac;
static real cap;
static int last_ts = -1;
static real T_cold;

/*   Define initial variables   */

static real mu_0;   /* Permeability */
static real n_cicles;   /* N° cicles */
static real tm;     /* Magnetization time */
static real tf;     /* FluidFlow time */
static real tc;     /* Cycle time */
static real t_act;  /* Actual cycle time */
static real B_vel;  /* Magnetic field applied change respect the time */

```

```
50  DEFINE_INIT(init_code, domain)
51  {
52      /* Definició d'algunes variables */
53      mu_0 = (4.0*PI)*pow(10,-7);
54      RuMm = 8.31434/0.15725;
55      a = (g*mu_B*J)/(k_B);
56      b = (3*Tc*J)/(J+1);
57      c = (2*J+1)/(2*J);
58      d = 1/(2*J);
59      f = N_s*g*J*mu_B;
60
61      n_cicles = 1.0;
62      t_act = 0.0;
63      tm = 3;
64      tf = 3;
65      tc = 2.0*tm + 2.0*tf;
66
67      /* Initzialització de algunes variables */
68      B_max = 0.97;
69      B_min = 0.16;
70
71      T = 298.0;
72      T_cold = 298.0;
73      delta_T = 0.0;


74      mce = 0.0;
75      mce_ac = 0.0;
76  }
77
78  DEFINE_ADJUST(my_adjust,domain)
79  {
80      real tt = CURRENT_TIME;
81      if (tt >= n_cicles*tc)
82      {
83          n_cicles++;
84      }
85      t_act = tt - (n_cicles - 1.0)*tc;
86
87
88      if ((t_act >= 0.0) && (t_act < (tm+tf)))
89      {
90          B = B_max;
91          B_ant = B_min;
92      }
93      else if ((t_act >= (tm+tf)) && (t_act < tc))
94      {
95          B = B_min;
96          B_ant = B_max;
97      }
98
```

```
 99      int curr_ts;
100      curr_ts = N_TIME;
101      cell_t cell;
102      int ID = 5;
103      Thread *thread = Lookup_Thread(domain, ID);
104      if (last_ts != curr_ts)
105      {
106          real dSm_0;
107          real dSm_H;
108          real cap_0;
109          real M;
110          last_ts = curr_ts;
111          begin_c_loop(cell,thread)
112          {
113              T = C_T(cell,thread);
114              if (T <= 291)
115              {
116                      cap_0 =
                         (7.22705396*pow(10,-6))*pow(T,4)-(6.76611611*pow(10,-3))*pow(T,3)+
                         2.37368*pow(T,2)-368.997*T+(2.16496*pow(10,4));
117              }
118              else
119              {
120                      cap_0 =
                         (-5.62735110*pow(10,-6))*pow(T,5)+(9.17772113*pow(10,-3))*pow(T,4)
                         -5.98335371*pow(T,3)+(1.94914902*pow(10,3))*pow(T,2)-(3.17276931*p
                         ow(10,5))*T+(2.06453722*pow(10,7));
121              }
122
123              if ((t_act >= 0.0) && (t_act < (tm+tf+CURRENT_TIMESTEP)))
124              {
125                  B_J = 0.5;
126                  X = 0.1;
127                  int c0_iter;
128                  for (c0_iter=1;c0_iter!=1500;c0_iter++)
129                  {
130                      real B_J_ant = B_J;
131                      real X_ant = X;
132                      X = (b*B_J)/T;
133                      B_J = c/(tanh(c*X)) - d/(tanh(d*X));
134                      if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) && (((fabs(X -
                         X_ant))/fabs(X)) < 0.00001))
135                      {
136                          break;
137                      }
138                  }
139                  e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
140                  dSm_0 = (RuMm)*(X*e)*(b*B_J)/((T-e*b)*T);
```

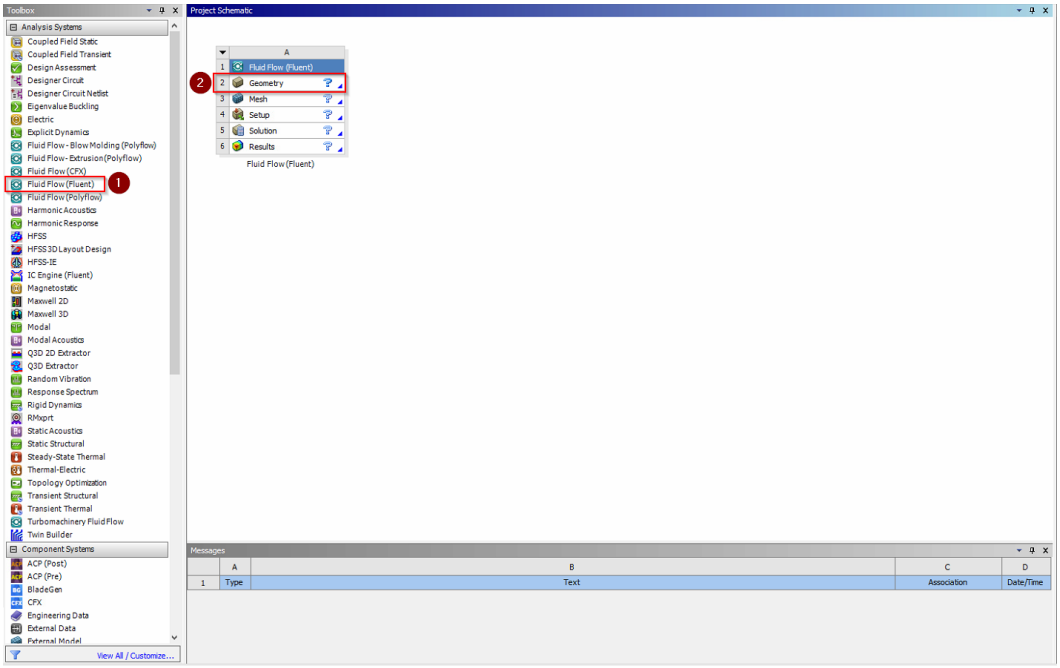
```
141
142                 int cH_iter;
143                 for (cH_iter=1;cH_iter=10000;cH_iter++)
144                 {
145                     real B_J_ant = B_J;
146                     real X_ant = X;
147                     X = (a*B)/T + (b*B_J)/T;
148                     B_J = c/(tanh(c*X)) - d/(tanh(d*X));
149                     if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) && (((fabs(X -
                        X_ant))/fabs(X)) < 0.00001))
150                     {
151                         break;
152                     }
153                 }
154                 e = (pow(d,2))/(pow(sinh(d*X),2)) - (pow(c,2))/(pow(sinh(c*X),2));
155                 dSm_H = (RuMm)*(X*e)*(a*B+b*B_J)/((T-e*b)*T);
156                 cap = cap_0 - T*(dSm_0-dSm_H);
157
158                 C_UDMI(cell,thread,0) = cap;
159
160                 if (((t_act > CURRENT_TIMESTEP) && (t_act <= (2*CURRENT_TIMESTEP)))
                    || ((t_act >= (tm+tf)) && (t_act < (tm+tf+CURRENT_TIMESTEP))))
161                 {
162                     real B_inter = B_ant;
163                     delta_T = 0.0;
164                     int b_iter;
165                     for (b_iter=1;b_iter!=20000;b_iter++)
166                     {
167                         real m_iter;
168                         for (m_iter=1;m_iter=10000;m_iter++)
169                         {
170                             real B_J_ant = B_J;
171                             real X_ant = X;
172                             X = (a*B_inter)/T + (b*B_J)/T;
173                             B_J = c/(tanh(c*X)) - d/(tanh(d*X));
174                             if ((((fabs(B_J - B_J_ant))/fabs(B_J)) < 0.00001) &&
                                (((fabs(X - X_ant))/fabs(X)) < 0.00001))
175                             {
176                                 break;
177                             }
178                         }
179                         e = (pow(d,2))/(pow(sinh(d*X),2)) -
                            (pow(c,2))/(pow(sinh(c*X),2));
180                         dSm_H = (RuMm)*(X*e)*(a*B_inter+b*B_J)/((T-e*b)*T);
181                         cap = cap_0 - T*(dSm_0-dSm_H);
182                         M = (f*e/T)*(-a*B_inter-b*B_J)/(T-e*b);
183                         delta_T = delta_T - (T*M/cap)*CURRENT_TIMESTEP*0.01;
184
185                         if ((B > B_ant) && (t_act > CURRENT_TIMESTEP) && (t_act <=
                            (2*CURRENT_TIMESTEP)))

186                         {
187                             B_inter += CURRENT_TIMESTEP*0.01;
188                             if (B_inter >= B)
189                             {
190                                 break;
191                             }
192                         }
193                         else if ((B < B_ant) && (t_act >= (tm+tf)) && (t_act <
                            (tm+tf+CURRENT_TIMESTEP)))
194                         {
195                             B_inter -= CURRENT_TIMESTEP*0.01;
196                             if (B_inter <= B)
197                             {
198                                 delta_T = - delta_T;
199                                 break;
200                             }
201                         }
202                     }
203                     C_UDMI(cell,thread,1) = delta_T;
204                     C_UDMI(cell,thread,2) = den*cap*delta_T/CURRENT_TIMESTEP;
205                 }
206                 else
207                 {
```

```
208                     C_UDMI(cell,thread,2) = 0.0;
209                 }
210
211             }
212             else
213             {
214                 C_UDMI(cell,thread,0) = cap_0;
215                 C_UDMI(cell,thread,2) = 0.0;
216             }
217         }
218         end_c_loop(cell,thread)
219     }
220
221 }
222
223 DEFINE_SOURCE(mce_source,cell,thread,dS,eqn)
224 {
225     real x[ND_ND];
226     C_CENTROID(x,cell,thread);
227     dS[eqn] = 0.0;
228     mce_ac = C_UDMI(cell,thread,2);
229     return mce_ac;
230 }
231
232 DEFINE_SPECIFIC_HEAT(my_spec_cap,T,Tref,h,yi)
233 {
234     Domain *domain = Get_Domain(1);
235     int ID = 5;
236     Thread *thread = Lookup_Thread(domain,ID);
237     cell_t cell;
238     real cap_prop;
239     begin_c_loop(cell,thread)
240     {
241         cap_prop = C_UDMI(cell,thread,0);
242         *h = cap_prop*(T-Tref);
243         return cap_prop;
244     }
245     end_c_loop(cell,thread)
246 }
247
248 DEFINE_ZONE_MOTION(piston_inlet,omega,axis,origin,velocity,time,dtime)
249 {
250     real velz = 0.02/3;
251     if ((t_act >= 0.0) && (t_act <= tm))
252     {
253         N3V_D(velocity,=,0.0,0.0,0.0);
254     }
255     else if ((t_act > tm) && (t_act <= (tm+tf)))
256     {
257         N3V_D(velocity,=,0.0,0.0,-velz);
258     }
259     else if ((t_act > (tm+tf)) && (t_act <= (2*tm+tf)))
260     {
261         N3V_D(velocity,=,0.0,0.0,0.0);
262     }
263     else if ((t_act > (2*tm+tf)) && (t_act <= tc))
264     {
265         N3V_D(velocity,=,0.0,0.0,velz);
266     }
267 }
```

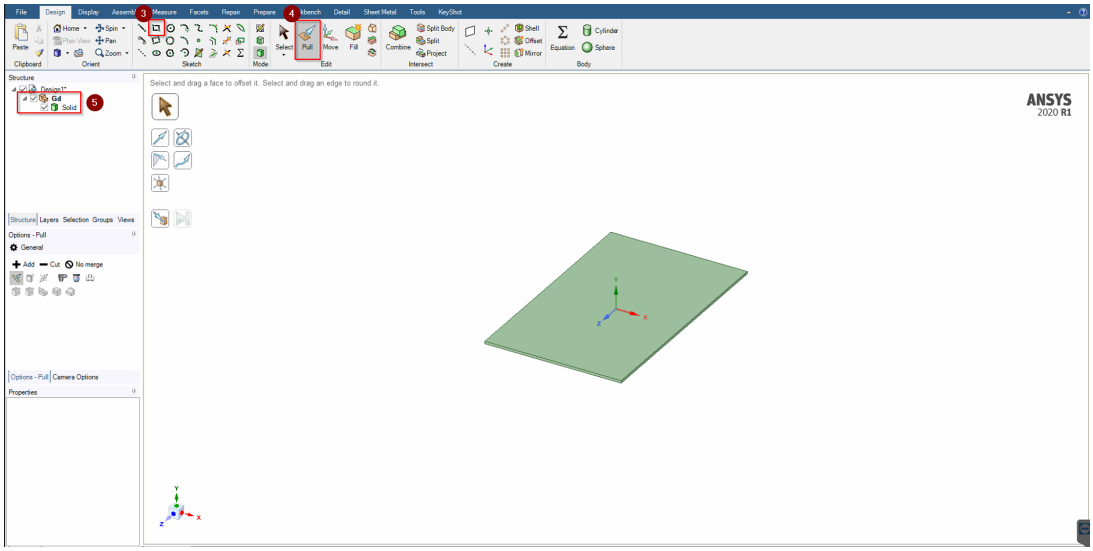
D.5. Pas a pas per la simulació en ANSYS FLUENT

Aquesta secció es mostra com s'ha realitzat la simulació d'un sistema de refrigeració en ANSYS Fluent 2020 R1. Com anirem veiem al llarg del procés l'ANSYS té certes limitacions que tindrem tenir en compte. Donat que s'ha realitzat dos prototips diferents que es diferencien sobretot en la configuració de l'ANSYS i l'UDF, partirem de la realització del prototip 2 que és més complex.

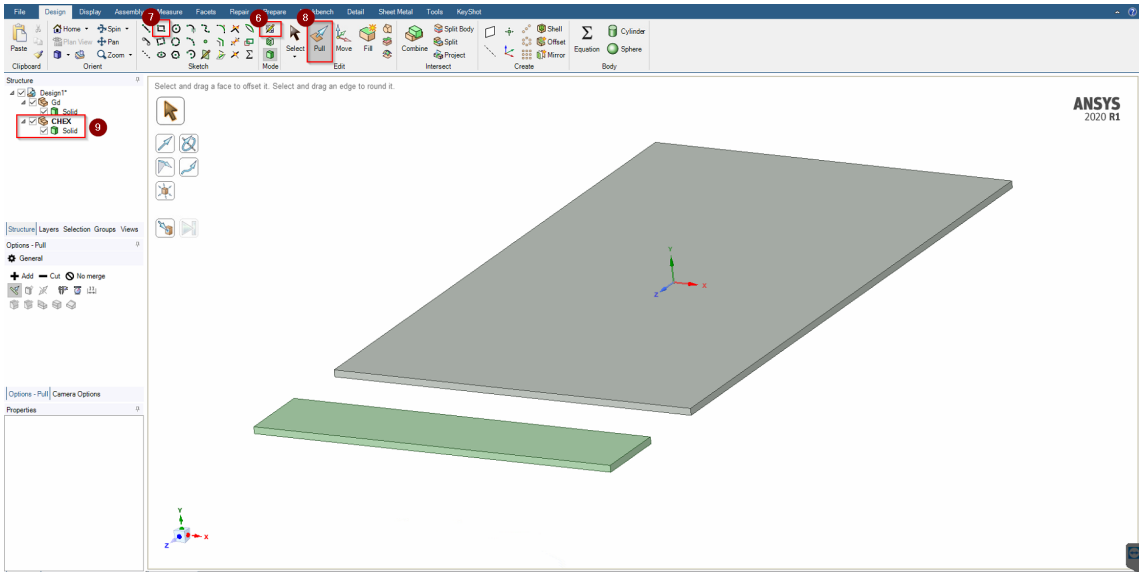
1. *ANSYS Workbench* trobem una bloc de *FluidFlow (Fluent)*.

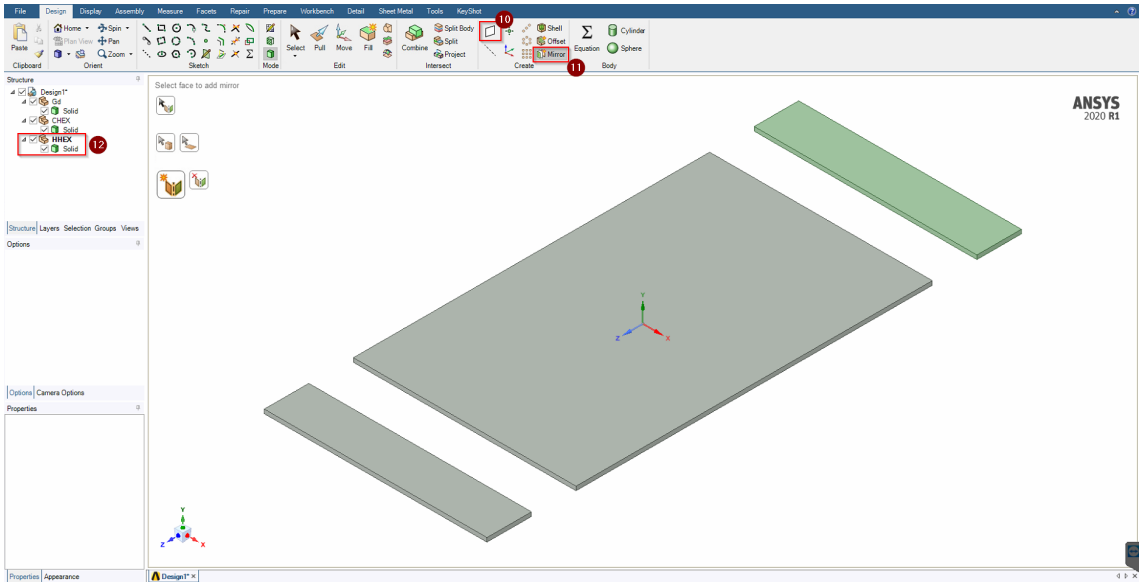


2. La geometria s'ha realitzat amb *ANSYS Spaceclaim*. Primer es realitza un rectangle en el pla xz, i s'extrudeix en la direcció normal positiva al pla. Es guarda com un nou component.

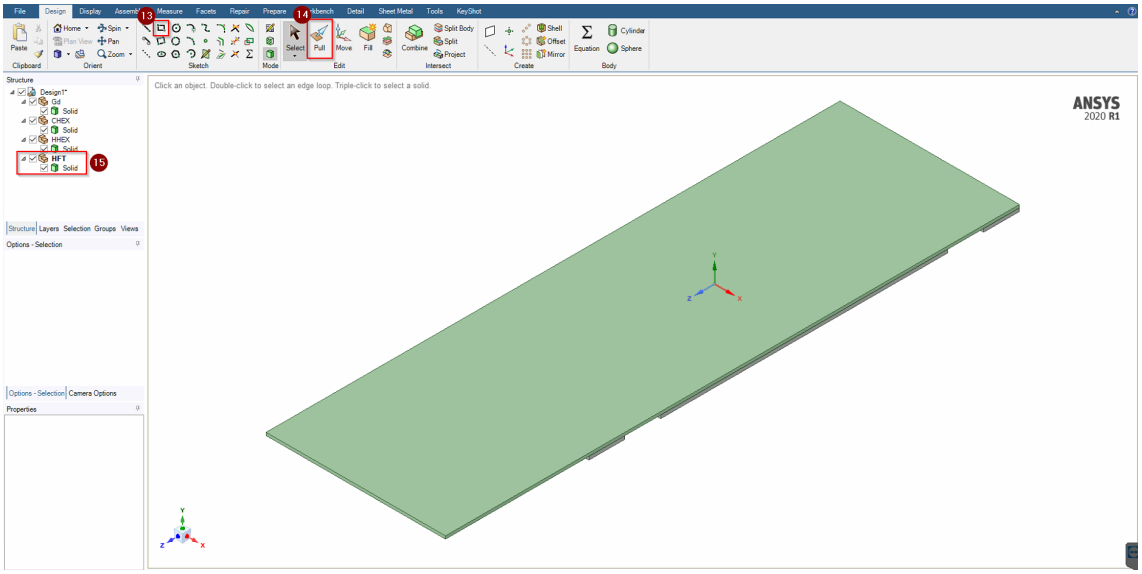


3. Es fa el mateix pels components CHEX i HHEX, amb la funció mirall s'escurça esforços. En el cas del prototip 1 només caldria el regenerador.

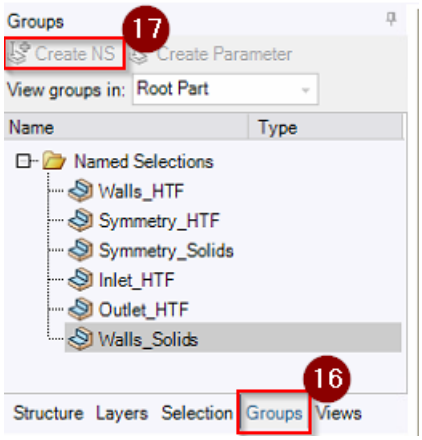




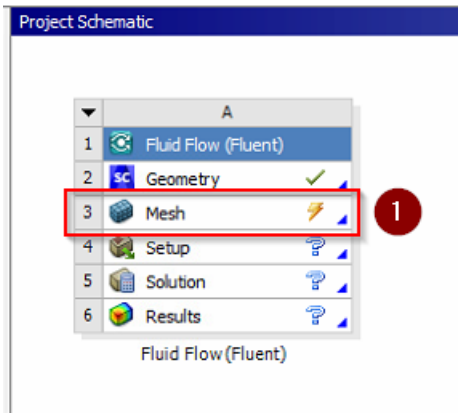
4. Seguidament creem el HTF que cobreixi tota la superfície de les plaques, a més, s'ha de allargar segons la distància que volem desplaçar durant la fase del bufat. Donat que es buscar desplaçar el HTF el 50% de la longitud del regenerador, aquesta distància sobrant ha de ser 20mm, el temps s'ajustarà mitjançant l'UDF.



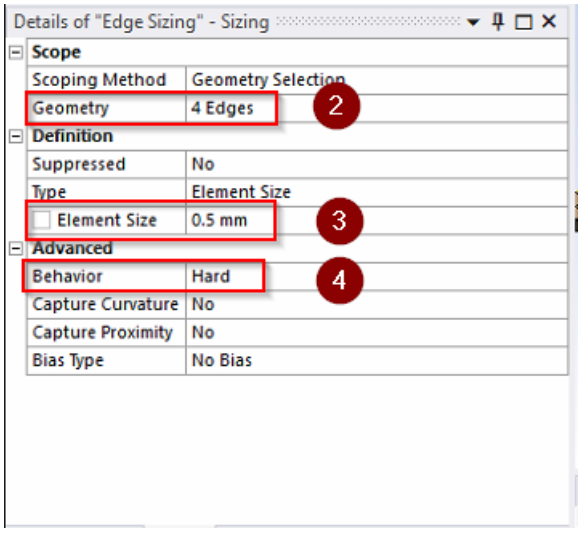
5. Normalment se li assigna uns noms prèviament per poder-los identificar més fàcilment durant la configuració de les condicions de contorn.

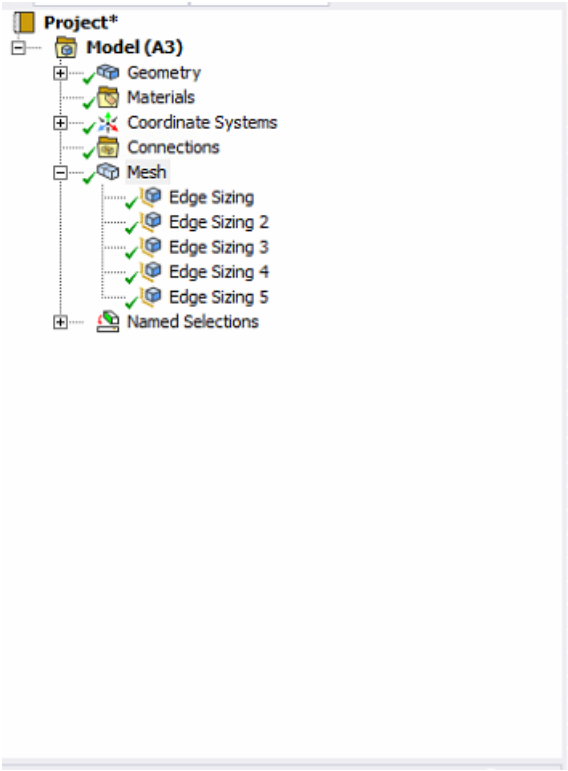


6. Un cop tenim la geometria, podem anar a fer el mallat a partir del bloc del *Workbench*. I per defecte es fa servir amb *ANSYS Meshing*.

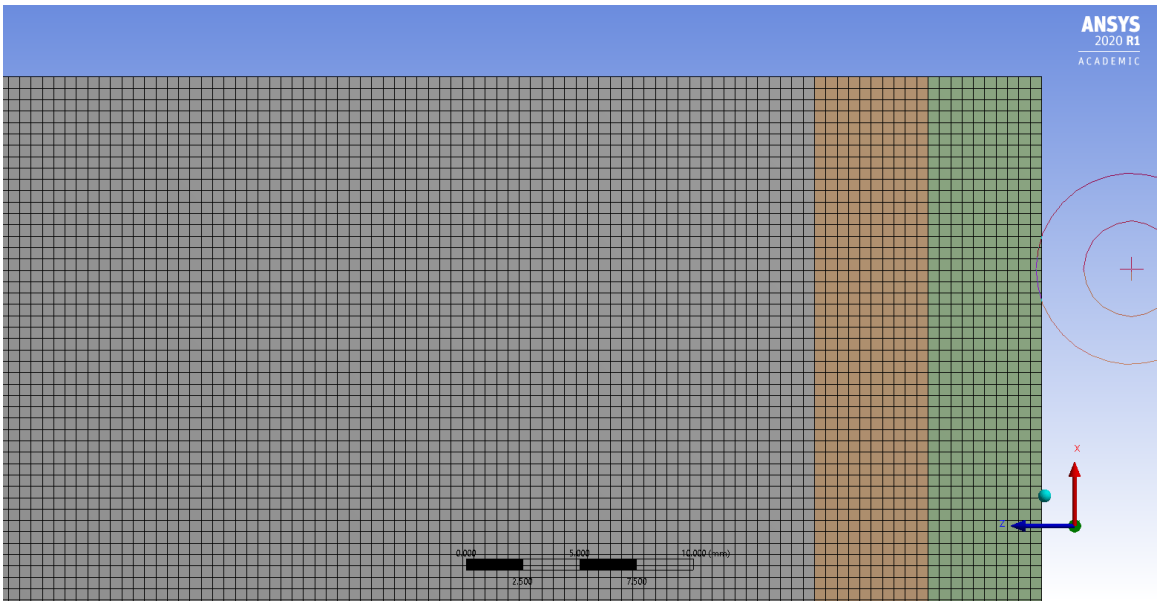


7. En aquest cas, donat que la geometria no es complexa i per no tenir problemes en moure el mallat necessitem d'una estructura quadri-lateral (2D) o hexaedre (3D). Hem triat un mallat de 0.5x0.5mm per tots els marges en el pla xz, però lo ideal seria comprovar els resultats amb un mallat més petit.

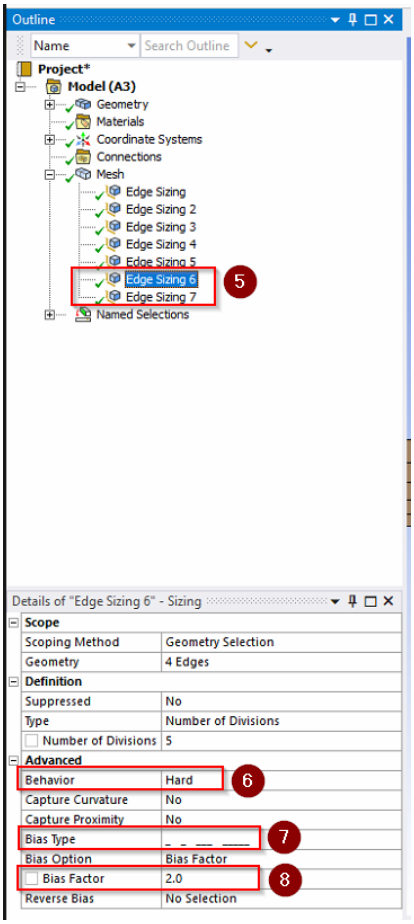




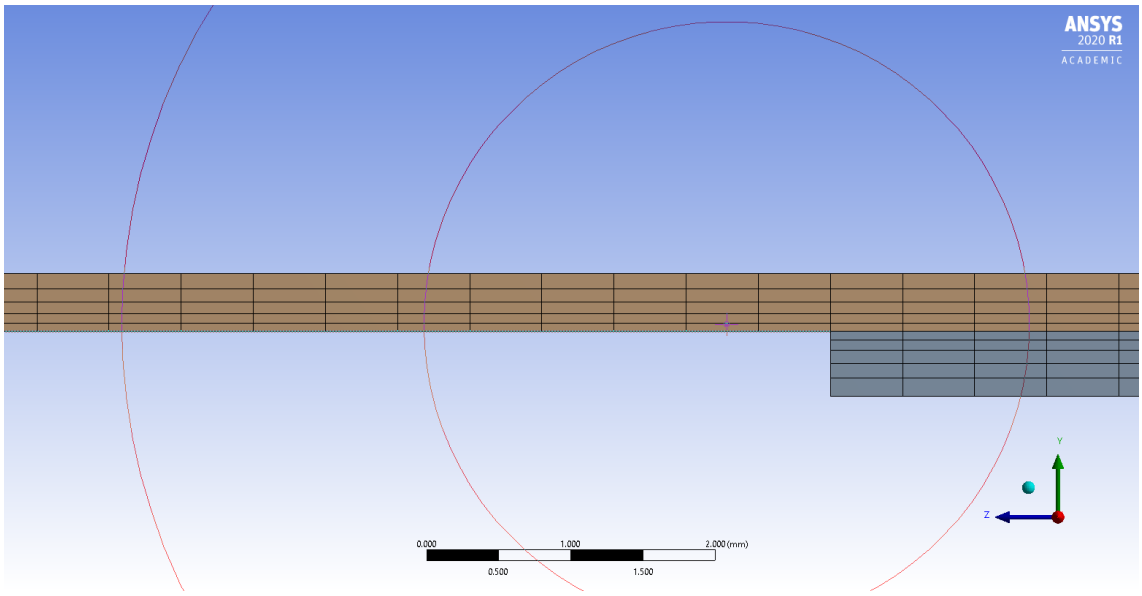
Observem en la següent figura com quedaria el mallat vist per sobre de la geometria (eix y).



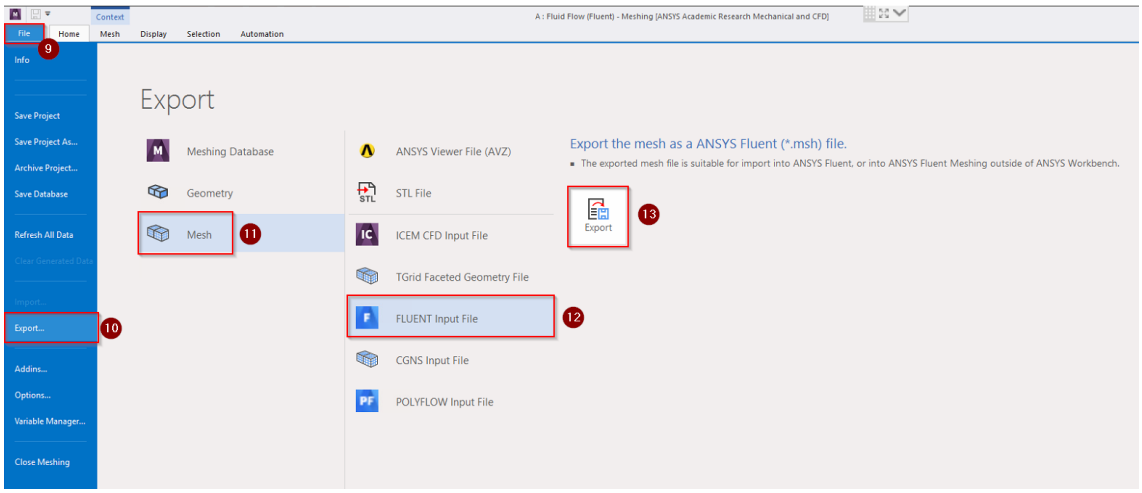
8. Tal com s'ha indicat a la memòria, cal que les cel·les adjacent a la interfície líquid-sòlid compleixi les condicions de capa límit. Amb un Sizing en els marges del pla xy i un factor Bias de 2.



Cal estar al cas de si el mallat dona distorsions, es possible aplicar “Reverse Bias” en els marges que es necessitin. En la següent figura podem observar el resultat:

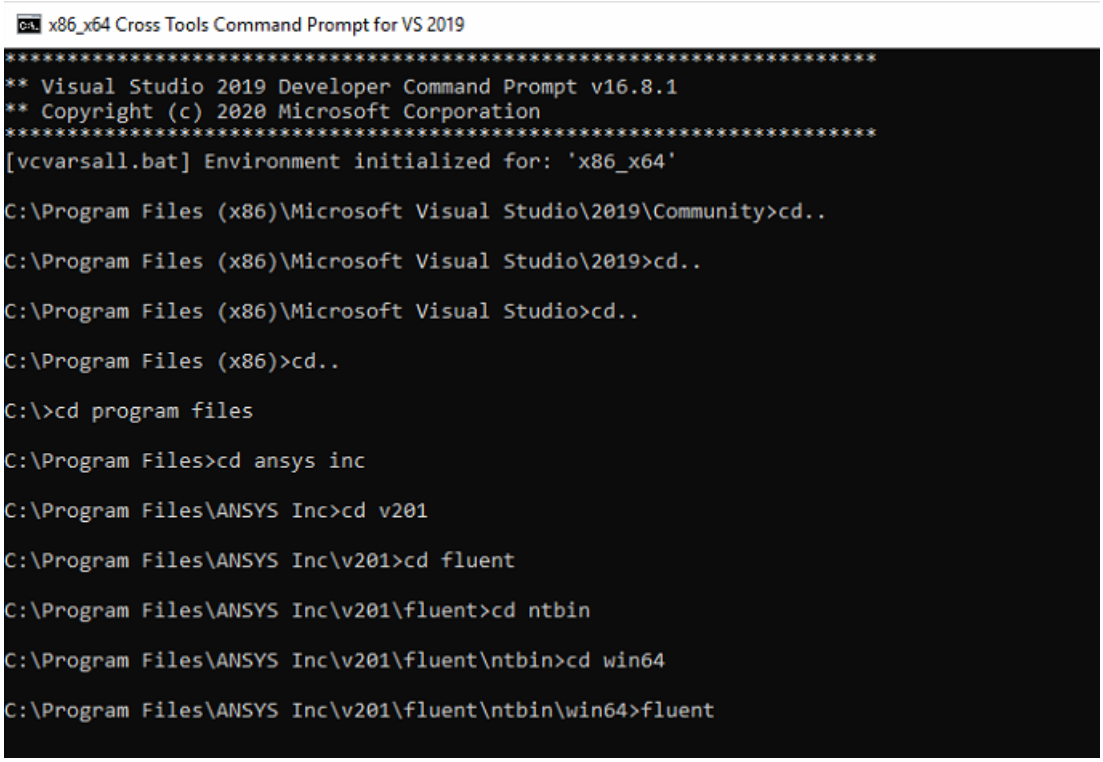


9. Cal fer l’exportació del mallat, ja que deixarem de treballar en el bloc de *Workbench*.

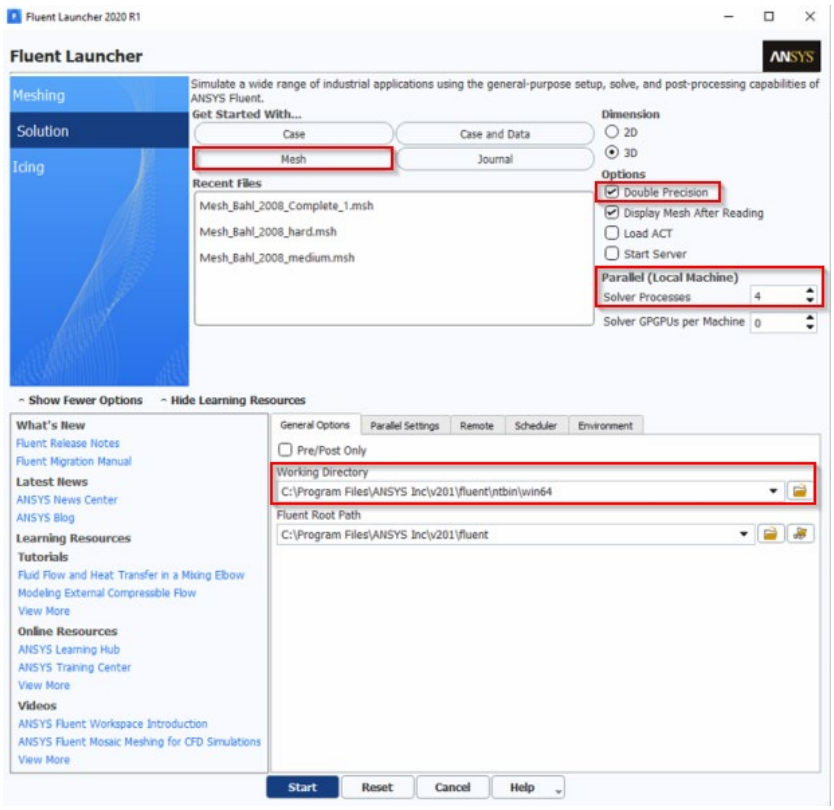


Cal veure la comptabilitat de l’ANSYS FLUENT amb el Visual Studio, fer les instal·lacions necessàries i obrir “*Cross Tools Command Prompt for VS*”.

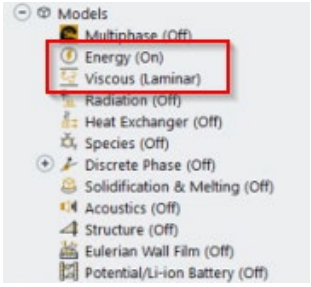
Un cop obert el prompt s’hi pot introduir aquesta sèrie de comandes, si la instal·lació s’ha realitzat en la ubicació per defecte. L’objectiu d’això és obrir el ANSYS Fluent x64 i podem carregar UDFs.



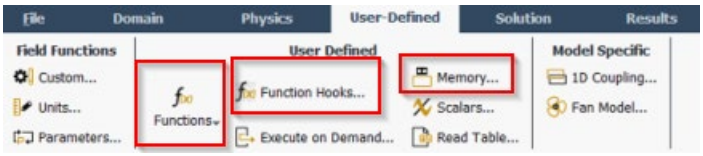
1. Carregem el mallat, escollim "*Double Precision*" per una millor resolució de dades, escollim quants processadors volem dedicar a Fluent i canviem "Working Directory" per la carpeta de treball on hem de tenir l'UDF.

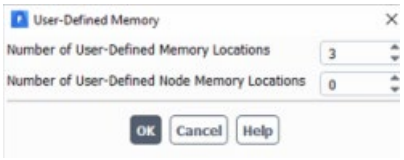


2. Comprovem el mallat i la qualitat.
3. Activem els models:

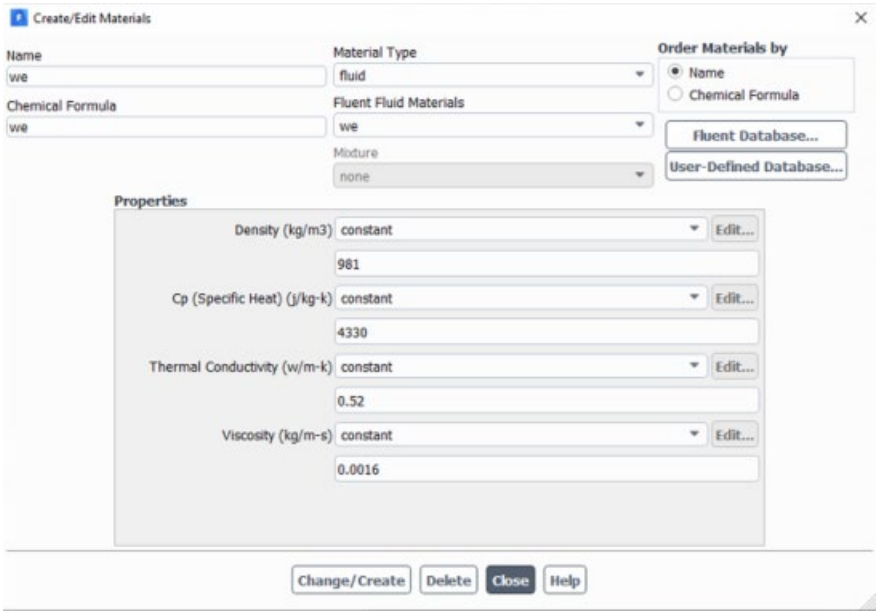


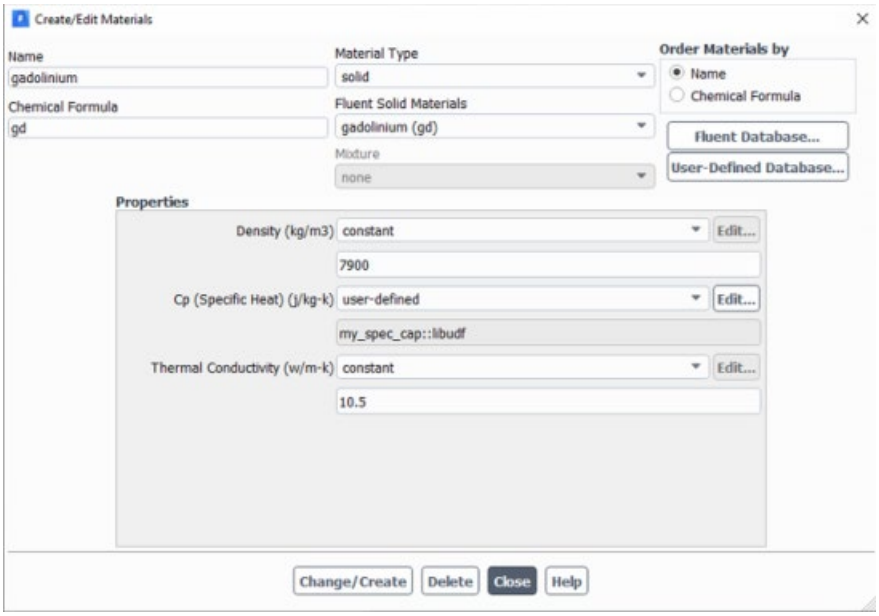
4. Carregar l'UDM i l'UDF.



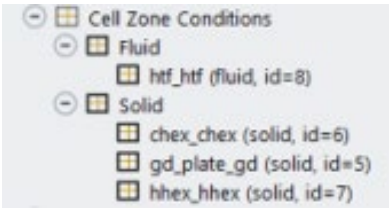


5. Crear els nous materials: WE i Gd



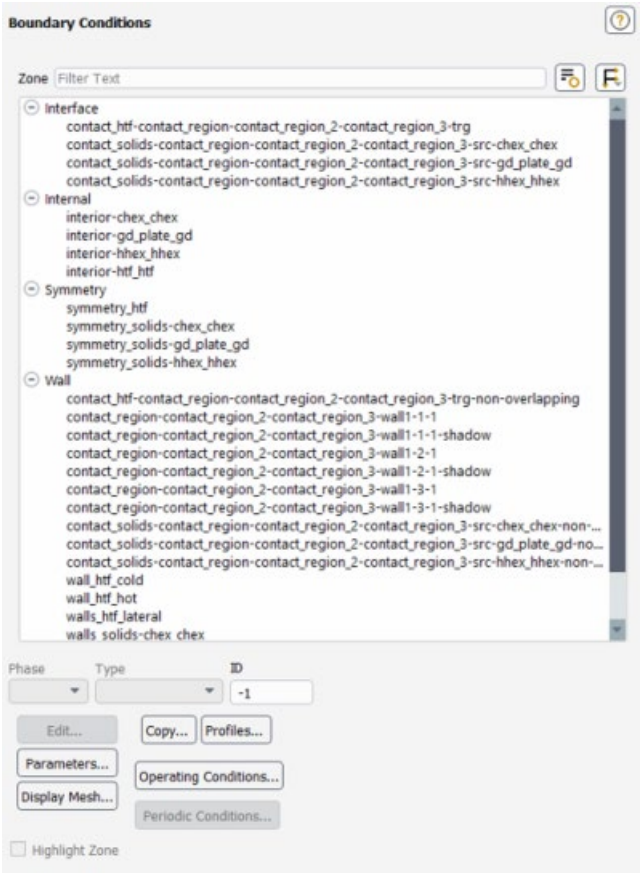


6. Confirmem que totes les assignacions siguin correctes i cerquem l'ID del regenerador



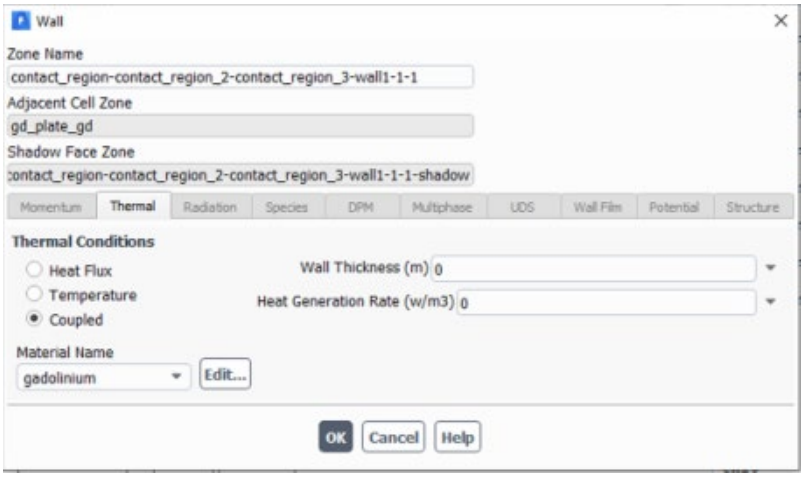
En aquest cas, el regenerador es correspon a ID = 6, que serà el que hem d'introduir en el UDF: "my_adjust" i "my_spec_cap".

7. Automàticament l'ANSYS Meshing detecten les interfícies (superfícies en contacte), i és important verificar-lo en el ANSYS Fluent.



En el prototip 2 només hi ha zones de contacte, superfícies de simetria i parets. Cal destacar que la zona de contacte de una regió té una segona part acoblada, que anomena *shadow*; nosaltres no la farem servir, però indirectament es definirà les seves condicions de contorn.

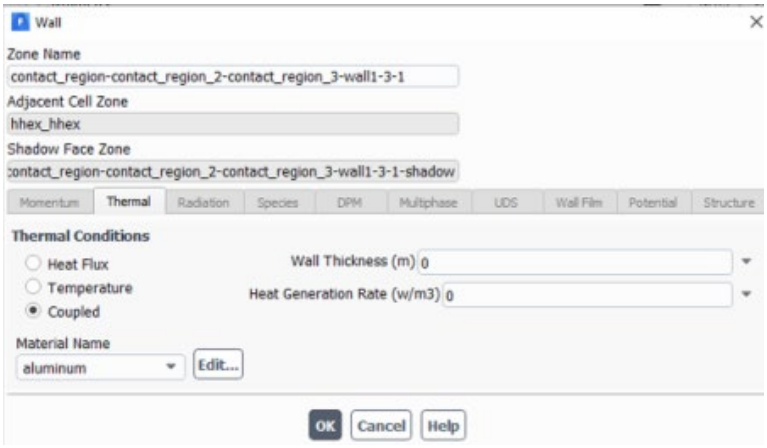
- Per la regió de contacte regenerador-HTF



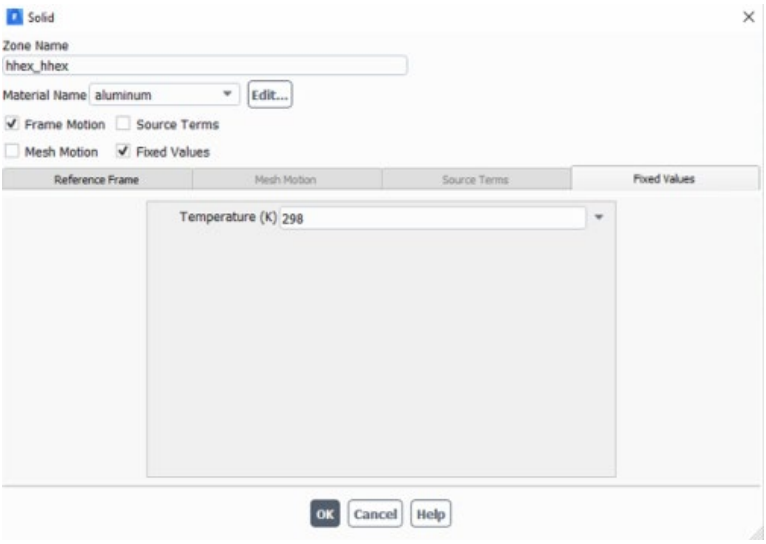
- Per la regió de contacte CHEX-HTF (si només es vol mesurar la temperatura adiabàtica).

En altres paraules, no transfereix calor al fluid.

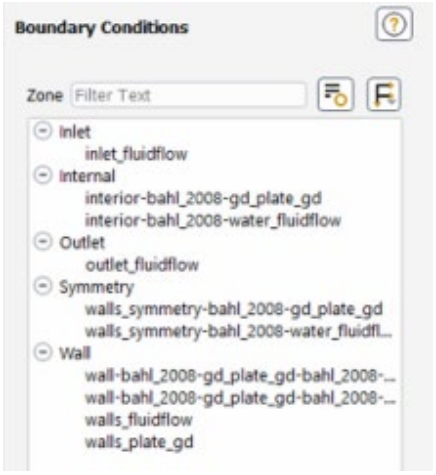
- Per la regió de contacte HHEX-HTF



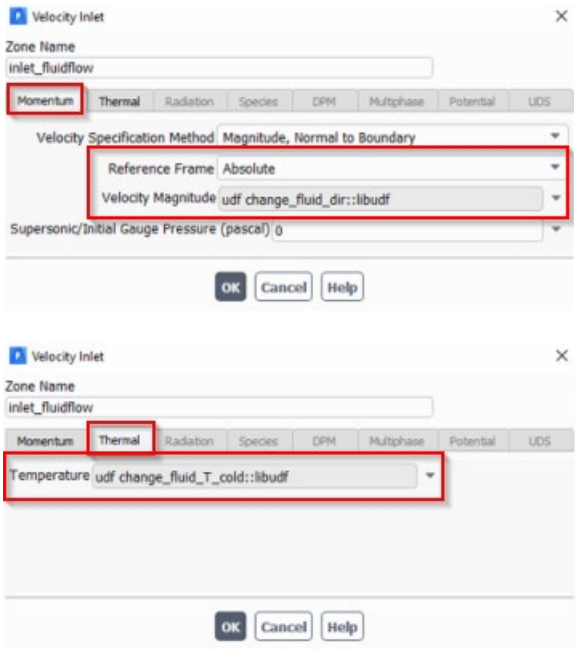
I a més a la cel·la fixem el valor de la temperatura ambient.



En aquest el prototip 1 es diferència que tindrem una entrada i sortida de fluid, per tant s'haurà de definir les condicions de contorn d'aquestes àrees:

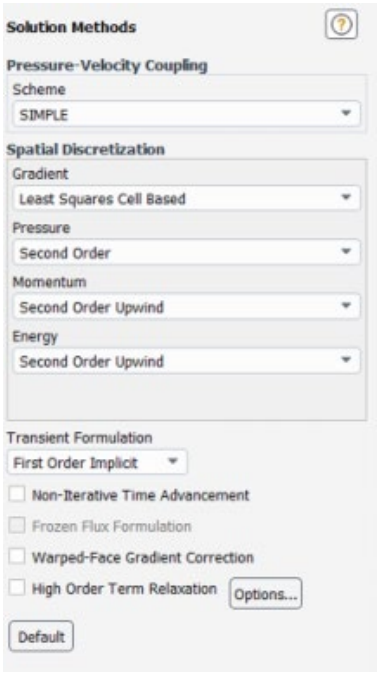


Respecte a l'entrada de fluid el definirem amb un UDF pel perfil de velocitats i la temperatura d'entrada:

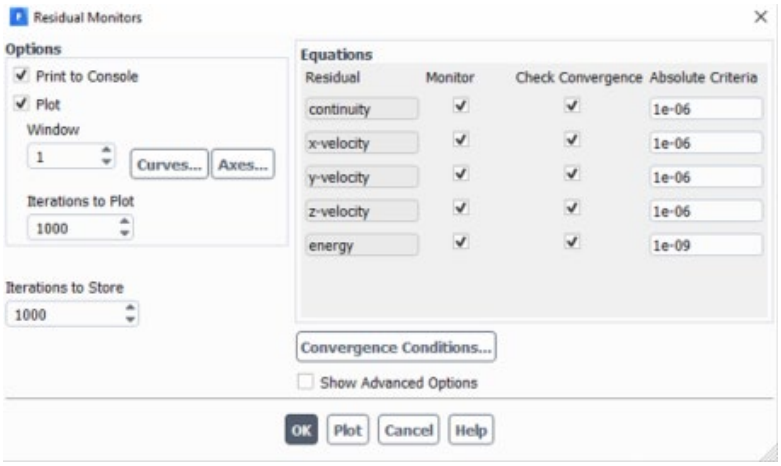


D'altre banda la sortida de fluid la considerarem com una pressió de sortida de 0 Pa i un temperatura fixe de 298K.

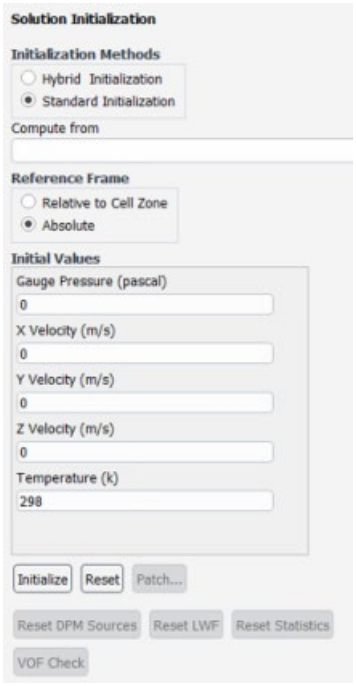
8. Definirem els mètodes de la solució



9. Redefinirem els criteris per la convergència, d'aquesta manera tindrem resultats més precisos.



10. Escollirem una inicialització estàndard a temperatura ambient, i totes les velocitats a 0 m/s.



Com a suggeriment, per anar comprovant els resultats durant la simulació hem creat un pla yz que talla l'origen: *Results* > *Surfaces* > *New* > *Plane* > $x = 0$. Llavors a *Results* > *Graphics* > *Contours* > *New* > *Temperature* > *Static Temperature* del pla creat.