**Algorithm 1:** 2D ingot evaporation model - Ingot definition

**%% Request user input parameters**;
Request $t_p$ = time required for e-beam to draw a pattern & time step used in time discretization;
Request;
**for** $i \in Oxidelist$ **do**
  $X_i$ = mass fraction for oxide i (in wt%)
**end**
Request $N$ = number of points used to discretize the target height;
Request $M$ = number of points used to discretize the target width-wise;
Request $t_{dep}$ = deposition time;
**%% Initialize target matrices and physical parameters for constituent oxides**;
Given $n$ = number of oxides in oxide list;
Given $N_A$ = the Avogadro Number;
Given $k_B$ = the Boltzmann constant;
Given $D_t$ = Target diameter;
Given $H_t$ = Target height;
Given $T(y)$ = Temperature distribution on ingot surface;
`/* Define dimensions of finite elements */`
$\delta_X = H_t/N$ ;
$\delta_Y = D_t/M$ ;
**for** $i \in Oxidelist$ **do**
  `/* Define physical properties of the oxides */`
  $\rho_i$ = oxide density of oxide i;
  $Mmol_i$ = molar mass of oxide i;
  $V_{P_i}(T)$ = Vapour pressure as a function of temperature ;
  `/* Calculate volume fraction of the oxide */`
  $V_i = \frac{X_i}{\rho_i} \frac{1}{\sum_{i=1}^{n} \frac{X_i}{\rho_i}}$;
  `/* Calculate number of molecules per finite element */`
  $N_{M_i} = N_A \rho_i \delta_X {\delta_Y}^2 X_i / Mmol_i$;
  `/* Initialize target matrix for each oxide */`
  $TM_i = N_{M_i} * J[N, M]$, where J[N,M] is an all-ones matrix of dimensions N, M;
**end**

**Algorithm 2:** 2D ingot evaporation model - Evaporation loop

```
%% Calculate the evaporation rate distribution on the surface for each oxide;
for i ∈ Oxidelist do
    for y ∈ M do
        δ_{m_i}[y] = sqrt(Mmol_i / (2π k_B T(y))) V_{P_i}(T(y)); where δ_{m_i} is a vector of dimension M representing the
         molecules evaporated for oxide i from the ingot surface;
    end
end
%% Calculate the evaporation in the specified time;
while t ≤ t_dep) do
    for i ∈ Oxidelist do
        while y ≤ M do
            while x ≤ N do
                if TM_i[x, y] > 0 then
                    TM_i[x, y] = TM_i[x, y] − δ_{m_i}[y];
                    if TM_i[x, y] >= 0 then
                        y = y + 1;
                    else if TM_i[x, y] < 0 then
                        TM_i[x + 1, y] = TM_i[x + 1, y] − TM_i[x, y];
                        y = y + 1;
                    end
                else if TM_i[x, y] ≤ 0 then
                    x = x + 1;
                end
                Break;
            end
            Break
        end
    end
    x = 0;
    y = 0;
    t = t + t_p;
end
```