

Anexos

Anexo A: Tablas de datos de sensibilización

Casos	Temperatura de borde					
	Ambos		Reducción		Oxidación	
	mol H2	%	mol H2	%	mol H2	%
Caso 1	1,80	143,49	0,98	31,64	1,15	54,66
Caso 2	1,50	102,10	0,92	24,03	1,08	45,91
Caso 3	1,23	66,02	0,87	17,46	0,99	33,95
Caso 4	0,98	32,59	0,82	10,15	0,88	18,60
Caso 5	0,74	0,00	0,74	0,00	0,74	0,00

Figura 7.1: Sensibilización temperatura de borde.

Casos	Velocidad del gas					
	Ambos		Reducción		Oxidación	
	mol H2	%	mol H2	%	mol H2	%
Caso 1	0,85	15,26	0,79	6,60	0,80	7,42
Caso 2	0,82	10,24	0,77	4,50	0,78	5,15
Caso 3	0,79	6,23	0,76	2,78	0,76	3,22
Caso 4	0,76	2,88	0,75	1,31	0,75	1,52
Caso 5	0,74	0,00	0,74	0,00	0,74	0,00

Figura 7.2: Sensibilización velocidad del gas.

	Temperatura inicial	
	Oxidación	
Casos	mol H2	%
Caso 1	0,94	26,95
Caso 2	0,88	19,17
Caso 3	0,83	11,54
Caso 4	0,78	5,05
Caso 5	0,74	0,00

Figura 7.3: Sensibilización temperatura inicial (oxidación).

Casos	Tiempo de reacción					
	Ambos		Reducción		Oxidación	
	mol H2	%	mol H2	%	mol H2	%
Caso 1	1,63	119,49	0,87	18,03	1,14	53,49
Caso 2	1,51	103,34	0,86	16,64	1,10	49,07
Caso 3	1,35	81,78	0,85	14,47	1,05	42,13
Caso 4	1,12	50,57	0,82	10,38	0,96	29,58
Caso 5	0,74	0,00	0,74	0,00	0,74	0,00

Figura 7.4: Sensibilización tiempo de reacción.

Aporte de calor volumétrico [W/m ³]	Aporte de calor volumétrico					
	Ambos		Reducción		Oxidación	
	mol H2	%	mol H2	%	mol H2	%
1*10 ⁹	1,43	92,43	0,84	13,00	1,04	40,07
5*10 ⁸	1,02	38,01	0,78	5,82	0,92	24,24
1*10 ⁸	0,79	6,70	0,75	1,09	0,78	5,35
5*10 ⁷	0,77	3,30	0,75	0,54	0,76	2,69
1*10 ⁷	0,75	0,65	0,74	0,11	0,75	0,54
5*10 ⁶	0,74	0,32	0,74	0,05	0,74	0,27
1*10 ⁶	0,74	0,06	0,74	0,01	0,74	0,05
5*10 ⁵	0,74	0,03	0,74	0,01	0,74	0,03
1*10 ⁵	0,74	0,01	0,74	0,00	0,74	0,01
1*10 ²	0,74	0,00	0,74	0,00	0,74	0,00

Figura 7.5: Sensibilización aporte de calor volumétrico.

Anexo B: Código en Python

Debido a la extensión de los códigos, estos se incluyen como archivos de pdf adjuntos a este informe, es posible que esto implique diferencias en la paginación señaladas en el índice.

Anexo B1

May 31, 2021

1 Modelo 1-D: determinación dt

1.1 Constantes

```
[1]: R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases.
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción. Bulfin
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción. Bulfin
lp=0.1 # [m] Longitud del medio poroso. ***DATO CÁMARA DE REACCIÓN ***
vol=7.3*10**(-4) # [m3] Volumen de la cámara de reacción. ***DATO CÁMARA DE
    ↳ REACCIÓN
e=0.65 # [-] Porosidad del medio poroso. ***DATO PELLETS BULFIN***
dp=0.01 # [m] diámetro de partícula ***DATO PELLETS BULFIN***
cCe=(1-e)*34.55 # [kmol/m3] concentración de cerio. Cálculo
molCeO2=(1-e)*30.6225 # =10.7179 moles totales de CeO2.
sigma=5.67*10**(-8) # [W/m2K4] Constante de Stefan-Boltzmann.

#Cálculo moles totales CeO2
#7.22 g/cm3 * 730 cm3 / 172.115(g/mol)= 30.6225 moles de CeO2 totales

yox=1 #Fracción molar de oxidante
Aox=1.0 # Factor pre-exponencial de oxidación. Arifin
Eox=29.0 # Energía de activación de oxidación. Arifin
gamma=0.89 # [-] Orden de reacción para la oxidación. Arifin
```

1.2 Función de Reducción

```
[2]: #Definición de función reducción
import numpy as np
def reduccion(dt,t_reaccion): #Los inputs se eligen desde el apartado
    ↳ 'variables' ubicado mas abajo. Al seleccionar se debe marcar con #

    #Funciones entalpía de reacción
    def entalpia_red(delta): #Valores EN kJ/mol

        if delta==0:
            entalpia=430
        elif 395-31.4*np.log10(delta) > 430:
            entalpia=430
```

```

else:
    entalpia= 395-31.4*np.log10(delta)
return entalpia

#.....Variables.....

u_in=1. # [m/s] Velocidad inicial del gas
Ts_border=2200 #[K] Temperatura del solido en x=0 en la reducción.
Qce=1000000 #[W/m3] Aporte de calor volumétrico externo.
nx=800 #[nodos] Cantidad de nodos temporales
#dt=0.02 #[segundos] Intervalo de tiempo
#t_reaccion=15 #[minutos] Tiempo de reacción.
#.....

#Constantes
Tg_border=300. #Temperatura del gas en x=0 en la reduccion.
d_border=0.35 #Concentracion de vacancias en el sólido en x=0 en la
→reducción.
d_max=0.35 #Concentración de vacancias máxima.
dx=lp/nx #[-] Espaciamiento
molxnodo=molCeO2/nx #cantidad de moles de CeO2 por nodo.

#####
#RESOLUCION DE ECUACIONES

#condiciones iniciales
d=np.zeros(nx)
d[0]=d_border
d_final=np.zeros(nx)+0.35 #condición final de reducción (consideración de
→reducción completa)

Ts=np.zeros(nx)+300
Ts[0]=Ts_border
Tg=np.zeros(nx)+300
Tg[0]=Tg_border

kred=np.zeros(nx) #mod eq vacancias
kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se usan
→para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)

```

```

alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt #contador para el tiempo de reacción

    #primero calculo todas las propiedades en función de la temperatura

    #sólido
    rho_s=(-0.0002*Ts+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref. Nelson 2014
    cp_s= (0.0599*np.log(Ts)+0.0390)*1000 #[J/kgK]calor específico del sólido * ref. Nelson 2014
    lambda_s= (6.7760*10**(-2)+2.7930*10**(-4)*Ts)**(-1)#[W/mK] Coeficiente de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Ts**3.)/(9.*(1.-e)) #[W/mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

    #gas (argón)
    rho_ar=1.603*300/Tg #[kg/m3]. Gases ideales
    cp_ar=(20.7860+Tg*2.8259*10**(-7)-Tg**2*1.4642*10**(-7)+Tg**3*1.0921*10**(-8)-Tg**(-1)*3.6614*10**(-8))*1000/39.948 #[J/kgK]. ref. webbook. nist.gob. Chase 1998
    lambda_ar=(0.0442*Tg + 4.7827)/1000 #[W/Km] Conductividad térmica. Ajuste Younglove 1986.
    mu_ar= (0.0566*Tg+6.0645)*10**(-6) ##[Pa*s] viscosidad dinámica del argón. Ajuste Younglove 1986.
    nu_ar=mu_ar/rho_ar # viscosidad cinematica del argon

```

```

u_ar=1.603*u_in/rho_ar # m/s velocidad del argon (depende de la
→ densidad)

#parámetros para a_ar. ref. Toledo
ags_ar=6./(1.-e) #superficie específica interna del medio poroso
Pr_ar=(mu_ar*cp_ar)/lambda_ar #Número de Prandtl
Re_ar=(e*u_ar*dp)/mu_ar #Número de Reynolds
hgs_ar=(lambda_ar/dp)*(2.0+1.1*Pr_ar**(1/3)*Re_ar**0.6) #Coeficiente
→ convectivo interfásico
a_ar = ags_ar*hgs_ar #Coeficiente intersticial de intercambio de calor

#Cálculo de vacancias
d_n=d.copy() #d_n sería d anterior (d^n), d sería el valor actual
→ d^(n+1)

kred=(1-d_n/d_final)**(1/3)*Ared*np.exp(-Ered/(R*Ts)) #La temperatura
→ todavía no se calcula para n+1, por tanto Ts corresponde a la temperatura en
→ n.
kox=y_ox**gamma*(1-d_n/d_final)*Aox*np.exp(-Eox/(R*Ts)) Para el caso
→ de la reducción será cero en la reducción

d=(d_n/dt+d_max*kred)/(kred+kox+1/dt) #como son estáticas no requiere
→ recorrer x

#Cálculo de temperaturas
for i in range(nx):
    if i==0:
        #solido
        beta_s[i+1]=Ts_border

        #gas
        beta_g[i+1]=Tg_border

    elif i<(nx-1):
        #solido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i] + A_s[i]+B_s[i]
        F_s[i]=Ts[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i]*Tg[i]+Qce/
→ nx-(d[i]-d_n[i])/dt*(cCe/nx)*(10**6)*entalpia_red(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

```

```

        #gas
        A_g[i]=(lambda_ar[i-1]+lambda_ar[i])/
        ↪ (2*dx**2)+u_ar[i]*cp_ar[i]*rho_ar[i]/dx
        B_g[i]=(lambda_ar[i]+lambda_ar[i+1])/(2*dx**2)
        C_g[i]=(cp_ar[i]*rho_ar[i])/dt + a_ar[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tg[i]*(cp_ar[i]*rho_ar[i])/dt + a_ar[i]*Ts[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Ts[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tg[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Ts[j-1]=alfa_s[j]*Ts[j]+beta_s[j]
            #gas
            Tg[j-1]=alfa_g[j]*Tg[j]+beta_g[j]

        if j==0:
            break
    tiempo=nt/60 #[minutos]
    return Ts,Tg,d

```

```

[3]: from numba import njit #Numba es una biblioteca que optimiza códigos y ahorra
    ↪ mucho tiempo de iteración.
    #Optimización de función reducción
    reduccion_njit=njit(reduccion)

```

1.3 Gráficos y otros procedimientos

```

[4]: #Listas vacías de resultados a obtener
    dt=[2.0,0.2,0.02,0.002] #Intervalos de tiempo
    t_reaccion=[2.5, 5, 7.5, 10]
    Ts_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
    Tg_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
    d_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

```

```
[5]: for i in range(len(dt)):
      for j in range(len(t_reaccion)):
          resultados=reduccion_njit(dt[i],t_reaccion[j])
          print(i,j)
          Ts_dt[i][j]=resultados[0]
          Tg_dt[i][j]=resultados[1]
          d_dt[i][j]=resultados[2]
      print('Iteración número:', i)
```

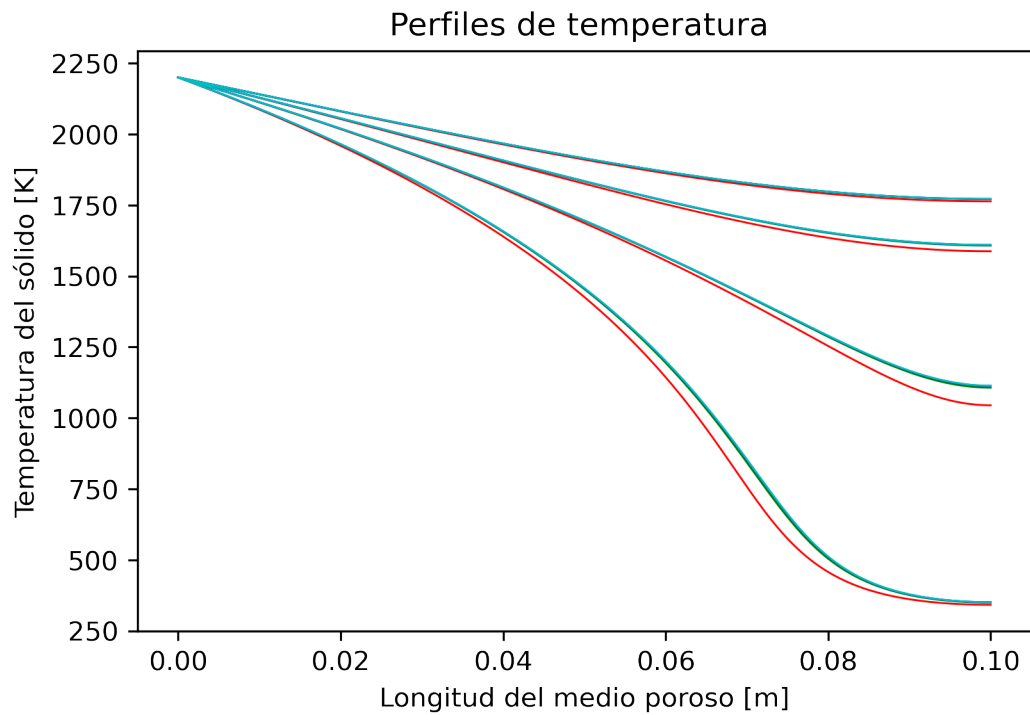
```
0 0
0 1
0 2
0 3
Iteración número: 0
1 0
1 1
1 2
1 3
Iteración número: 1
2 0
2 1
2 2
2 3
Iteración número: 2
3 0
3 1
3 2
3 3
Iteración número: 3
```

```
[10]: #Importes y gráfica de Ts
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200
lp=0.1 #Tengo que definirlo fuera de la función
nodos=800
colores=['r','g','b','c','m','k']
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,Ts_dt[i][j],colores[i],linewidth=0.7)

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del sólido [K]')
#plt.legend(('tiempo=2.5 [min]','tiempo=5 [min]','tiempo=7.5 [min]','tiempo=10 [min]'), #Se agregó manualmente la leyenda correcta.
```

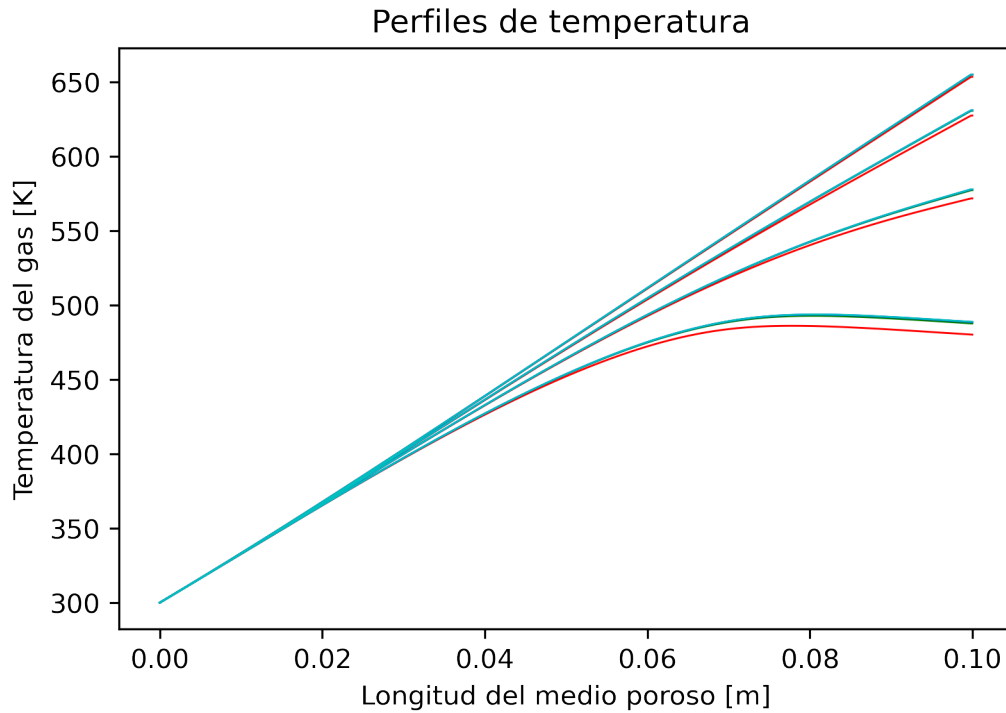
```
#prop = {'size': 7}, loc='lower left')
plt.show()
```



```
[13]: #Gráfica de Tg
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,Tg_dt[i][j],colores[i],linewidth=0.7)

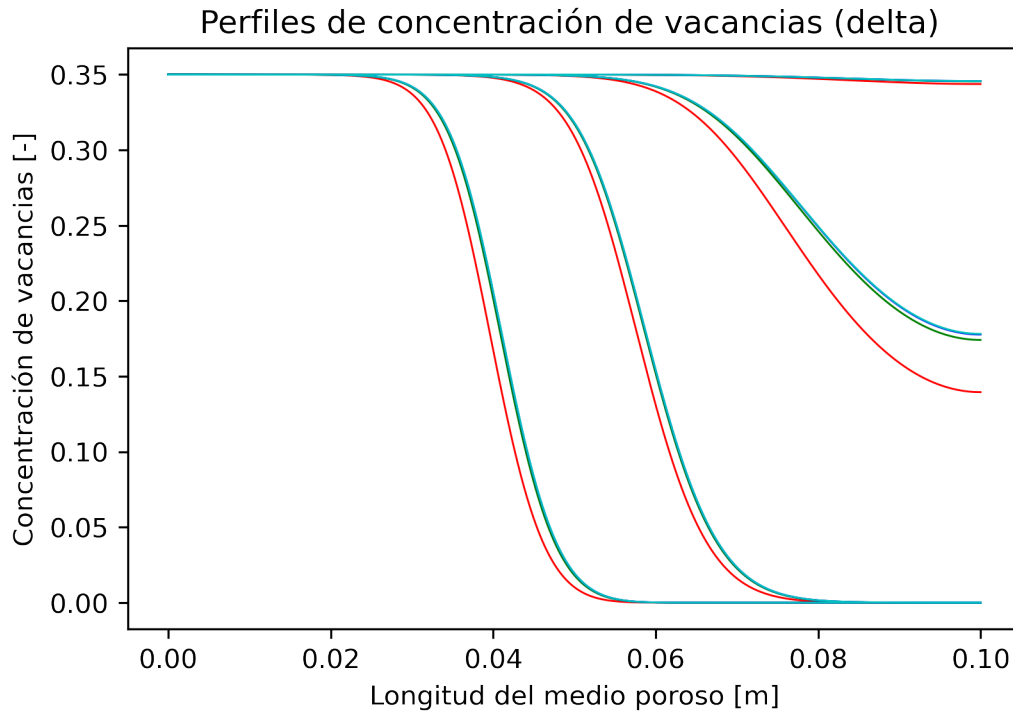
#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
#plt.legend(('dt=2 [s]','dt=0.2 [s]','dt=0.02 [s]','dt=0.002 [s]'),
#prop = {'size': 10}, loc='upper left')
plt.show()
```



```
[17]: #Gráfica de d
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,d_dt[i][j],colores[i],linewidth=0.7)

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
#plt.legend(('dt=2 [s]','dt=0.2 [s]','dt=0.02 [s]','dt=0.002 [s]'),
#           #prop = {'size': 10}, loc='lower left')
plt.show()
```



1.4 Función de Oxidación

```
[18]: #Definición de función oxidación
def oxidacion(dt,t_reaccion):
    def entalpia_ox(delta):
        if delta==0:
            entalpia=-430+285.85
        elif 395-31.4*np.log10(delta) > 430:
            entalpia=-430+285.85
        else:
            entalpia= -(395-31.4*np.log10(delta))+285.85
        return entalpia

    #.....Variables.....

    u_in=1 #[m/s] Velocidad inicial del gas.
    Ts_inicial=700 #[K] Temperatura inicial del sólido para la
    Ts_bordeo=1200 #[K] Temperatura del sólido en x=0 en la oxidación.
    Qce=1000000 #[W/m3] Aporte de calor volumétrico externo.
    nx=800 #[nodos] Cantidad de nodos espaciales.
    #dt=0.02 #[s] Intervalo de tiempo.
```

```

#t_reaccion=15 #[min] Tiempo de reacción.
#.....

#Constantes
dx=lp/nx #[-] Espaciamiento

#Condiciones de borde de temperatura y delta.

Ts_bordeo=float(Ts_bordeo) #Temperatura del solido en x=0 en la oxidación.
Tg_bordeo=400. #Temperatura del gas en x=0 en la oxidación.
d_bordeo=0 #Concentración de vacancias en el sólido en x=0 en la oxidación.

#####
#RESOLUCION DE ECUACIONES

#condiciones iniciales
d=reduccion_njit(dt,10)[2] #Las condiciones finales de la reducción se
→ asignan aquí.
d_inicial=d.copy()
dox=d.copy()
dox[0]=0

Tsox=np.zeros(nx)+float(Ts_inicial)
Tsox[0]=Ts_bordeo
Tgox=np.zeros(nx)+300
Tgox[0]=Tg_bordeo

kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se
→ usan para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)

```

```

F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

#condición final de reducción
dox_final=np.zeros(nx) #Asumimos que todo el material se oxidará

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt

    #sólido
    rho_s=(-0.0002*Tsox+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref.
    ↪ Nelson 2014
    cp_s= (0.0599*np.log(Tsox)+0.039)*1000 #[J/kgK]calor específico del
    ↪ sólido * ref. Nelson 2014
    lambda_s= (6.776*10**(-2)+2.793*10**(-4)*Tsox)**(-1)#[W/mK] Coeficiente
    ↪ de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Tsox**3.)/(9.*(1.-e)) #[W/
    ↪ mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

    #gas
    #nota: La temperatura del vapor varía muy poco, por ende se
    ↪ considerarán propiedades constantes.

    rho_h2o=np.zeros(nx)+1/1.673 #kg/m3 #1/v volumen específico
    cp_h2o=np.zeros(nx)+2.01*10**3 #j/kgK calor específico
    lambda_h2o=np.zeros(nx)+24.8*10**(-3) #W/mK conductividad térmica del
    ↪ vapor
    mu_h2o= np.zeros(nx)+12.0*10**(-6) #Pa*s o kg/ms Viscosidad dinámica del
    ↪ vapor
    nu_h2o=mu_h2o/rho_h2o #m**2/s viscosidad cinemática
    u_h2o=np.zeros(nx)+u_in

    #parámetros para a_ar. ref. Toledo
    ags_h2o=6./(1.-e) #superficie específica interna del medio poroso
    Pr_h2o=(mu_h2o*cp_h2o)/lambda_h2o #Número de Prandtl
    Re_h2o=(e*u_h2o*dp)/mu_h2o #Número de Reynolds
    hgs_h2o=(lambda_h2o/dp)*(2.0+1.1*Pr_h2o**(1/3)*Re_h2o**0.6)
    ↪ #Coeficiente convectivo interfásico
    a_h2o = ags_h2o*hgs_h2o #Coeficiente intersticial de intercambio de
    ↪ calor

```

```

#Cálculo de vacancias
dox_n=dox.copy() #d_n sería d anterior (d~n), d sería el valor actual
↪ d^(n+1)
kox=yox**gamma*(1-(d_inicial-dox_n)/(d_inicial))*Aox*np.exp(-Eox/
↪ (R*Tsox))
dox=(dox_n/dt)/(kox+1/dt) #como son estáticas no requiere recorrer x

#Cálculo de temperaturas
for i in range (nx):
    if i==0:
        #solido
        beta_s[i+1]=Ts_bordeo

        #gas
        beta_g[i+1]=Tg_bordeo

    elif i<(nx-1):
        #solido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_h2o[i] + A_s[i]+B_s[i]
        F_s[i]=Tsox[i]*(1-e)*(cp_s[i]*rho_s[i])/dt +
↪ a_h2o[i]*Tgox[i]+Qce/nx+(dox[i]-dox_n[i])/dt*(cCe/
↪ nx)*(10**6)*entalpia_ox(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

        #gas
        A_g[i]=(lambda_h2o[i-1]+lambda_h2o[i])/
↪ (2*dx**2)+u_h2o[i]*cp_h2o[i]*rho_h2o[i]/dx
        B_g[i]=(lambda_h2o[i]+lambda_h2o[i+1])/(2*dx**2)
        C_g[i]=(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tgox[i]*(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]*Tsox[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Tsox[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tgox[i]=beta_g[i]/(1-alfa_g[i])

```

```

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Tsox[j-1]=alfa_s[j]*Tsox[j]+beta_s[j]
            #gas
            Tgox[j-1]=alfa_g[j]*Tgox[j]+beta_g[j]

        if j==0:

            break
    tiempo=nt/60
    return Tsox,Tgox,dox

```

```

[19]: #Optimización de función oxidación
oxidacion_njit=njit(oxidacion)

```

1.5 Gráficos y otros procedimientos

```

[20]: #Listas vacías de resultados a obtener

Tsox_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
Tgox_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
dox_dt=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]

```

```

[21]: #Calculo de oxidación
for i in range(len(dt)):
    for j in range(len(t_reaccion)):
        resultadosox=oxidacion_njit(dt[i],t_reaccion[j])
        print(i,j)
        Tsox_dt[i][j]=resultadosox[0]
        Tgox_dt[i][j]=resultadosox[1]
        dox_dt[i][j]=resultadosox[2]
    print('Iteración número:', i)

```

```

0 0
0 1
0 2
0 3
Iteración número: 0
1 0
1 1
1 2
1 3
Iteración número: 1
2 0
2 1

```

```

2 2
2 3
Iteración número: 2
3 0
3 1
3 2
3 3
Iteración número: 3

```

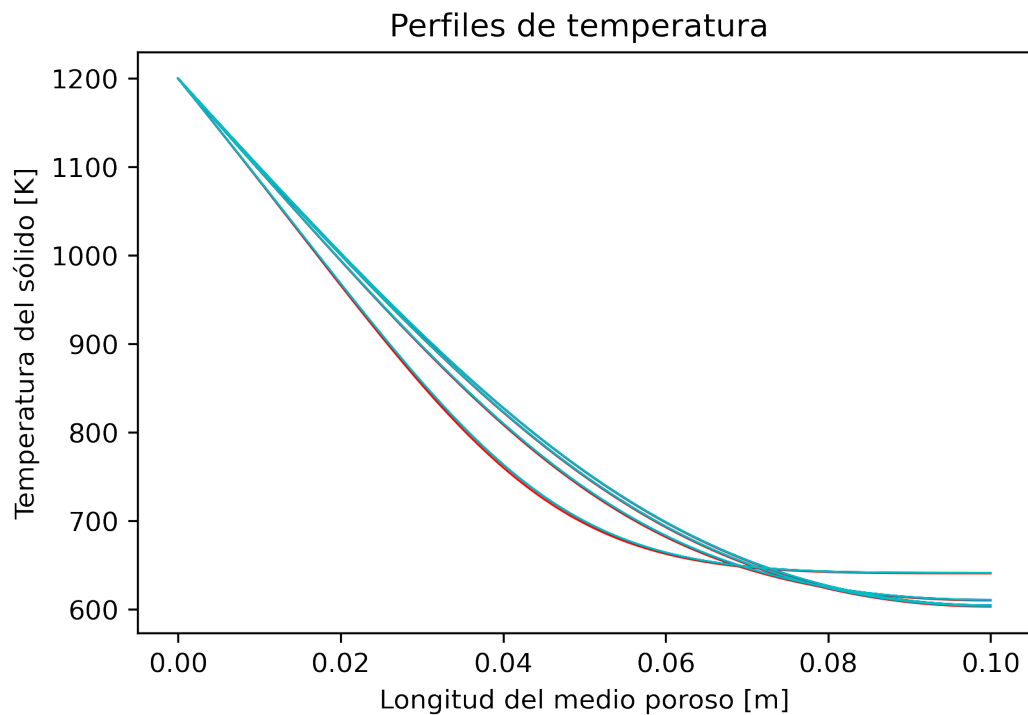
```

[22]: #Gráfica de Ts
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,Tsox_dt[i][j],colores[i],linewidth=0.7)

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del sólido [K]')
#plt.legend(('dt=2 [s]','dt=0.2 [s]','dt=0.02 [s]','dt=0.002 [s]'), #Se agregó
→manualmente la leyenda correcta.
        #prop = {'size': 10}, loc='lower left')
plt.show()

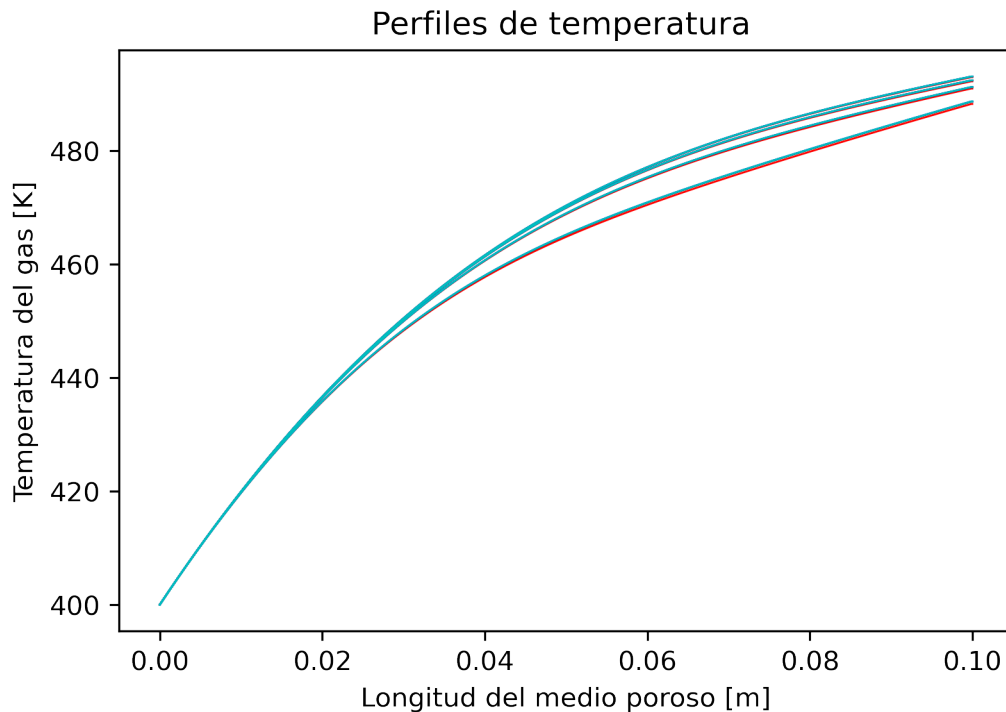
```



```
[23]: #Gráfica de Tg
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,Tgox_dt[i][j],colores[i],linewidth=0.7)

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
#plt.legend(('dt=2 [s]', 'dt=0.2 [s]', 'dt=0.02 [s]', 'dt=0.002 [s]'),
            #prop = {'size': 10}, loc='upper left')
plt.show()
```

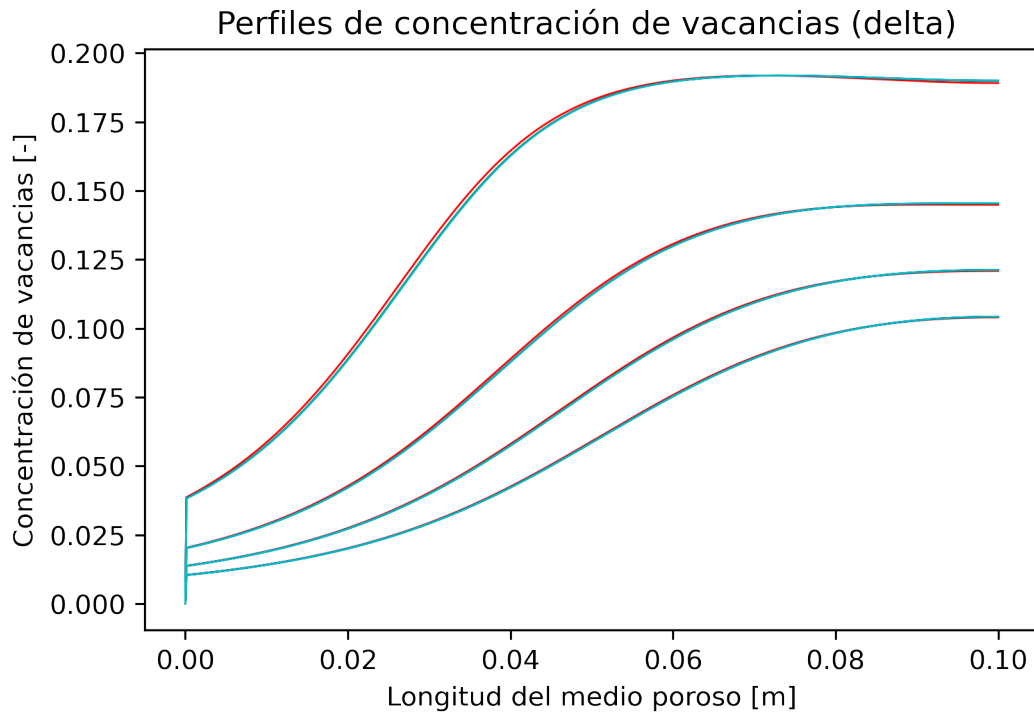


```
[24]: #Gráfica de d
for i in range(len(t_reaccion)):
    for j in range(len(dt)):

        puntos=np.linspace(0,lp,nodos)
        plt.plot(puntos,dox_dt[i][j],colores[i],linewidth=0.7)

#Datos extra
```

```
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
#plt.legend(('dt=2 [s]', 'dt=0.2 [s]', 'dt=0.02 [s]', 'dt=0.002 [s]'),
#           #prop = {'size': 10}, loc='upper right')
plt.show()
```



Anexo B2

May 31, 2021

1 Modelo 1-D: determinación nx

1.1 Constantes

```
[2]: R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases.
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción. Bulfin.
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción. Bulfin.
lp=0.1 #[m] Longitud del medio poroso. ***DATO CÁMARA DE REACCIÓN ***
vol=7.3*10**(-4) #[m3] Volumen de la cámara de reacción. ***DATO CÁMARA DE
    ↪ REACCIÓN***
e=0.65 #[-] Porosidad del medio poroso.***DATO PELLETS BULFIN***
dp=0.01 #[m] Diámetro de partícula. ***DATO PELLETS BULFIN***
cCe=(1-e)*34.55 #[kmol/m3] Concentración de cerio. Cálculo.
molCeO2=(1-e)*30.6225 #=10.7179 Moles totales de CeO2.
sigma=5.67*10**(-8) #[W/m2K4] Constante de Stefan-Boltzmann.
#Cálculo moles totales CeO2
#7.22 g/cm3 * 730 cm3 / 172.115(g/mol)= 30.6225 moles de CeO2 totales.
yox=1 #Fracción molar de oxidante
Aox=1.0 # Factor pre-exponencial de oxidación. Arifin
Eox=29.0 # Energía de activación de oxidación. Arifin
gamma=0.89 #[-] Orden de reacción para la oxidación. Arifin
```

1.2 Función de Reducción

```
[ ]: #Definición de función reducción
import numpy as np
def reduccion(nx): #Los inputs se eligen desde el apartado 'variables' ubicado
    ↪ mas abajo. Al seleccionar se debe marcar #
    #Funciones entalpía de reacción
    def entalpia_red(delta): #Valores en kJ/mol
        if delta==0:
            entalpia=430
        elif 395-31.4*np.log10(delta) > 430:
            entalpia=430
        else:
            entalpia= 395-31.4*np.log10(delta)
        return entalpia
    #.....Variables.....
```

```

u_in=1. # [m/s] Velocidad inicial del gas.
Ts_border=2200 #[K] Temperatura del solido en x=0 en la reducci3n.
Qce=1000000 #[W/m3] Aporte de calor volum3trico externo.
#nx=800 #[nodos] Cantidad de nodos temporales.
dt=0.02 #[segundos] Intervalo de tiempo.
t_reaccion=5 #[minutos] Tiempo de reacci3n.
#.....
→

#Constantes
Tg_border=300. #Temperatura del gas en x=0 en la reducci3n.
d_border=0.35 #Concentraci3n de vacancias en el s3lido en x=0 en la
→reducci3n.
d_max=0.35 #Concentraci3n de vacancias m3xima.
dx=lp/nx #[-] Espaciamiento.
molxnodo=molCeO2/nx #Cantidad de moles de CeO2 por nodo.
#####
#RESOLUCI3N DE ECUACIONES
d=np.zeros(nx)
d[0]=d_border
d_final=np.zeros(nx)+0.35 #condici3n final de reducci3n (consideraci3n de
→reducci3n completa).
Ts=np.zeros(nx)+300
Ts[0]=Ts_border
Tg=np.zeros(nx)+300
Tg[0]=Tg_border
kred=np.zeros(nx)
kox=np.zeros(nx)
#se crean los arreglos de A, B, C, F, alfa y beta vac3as, los cuales se usan
→para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)
A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

tiempo=0 #[min] tiempo de reacci3n
nt=0
while tiempo < t_reaccion:
    nt+=dt #Contador para el tiempo de reacci3n

```

```

#Primero calculo todas las propiedades en función de la temperatura
→
#sólido
rho_s=(-0.0002*Ts+6.8561)*1000
cp_s= (0.0599*np.log(Ts)+0.0390)*1000
lambda_s= (6.7760*10**(-2)+2.7930*10**(-4)*Ts)**(-1)
lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Ts**3.)/(9.*(1.-e))
#gas (argón)
rho_ar=1.603*300/Tg
cp_ar=(20.7860+Tg*2.8259*10**(-7)-Tg**2*1.4642*10**(-7)+Tg**3*1.
→0921*10**(-8)-Tg**(-1)*3.6614*10**(-8))*1000/39.948
lambda_ar=(0.0442*Tg + 4.7827)/1000
mu_ar= (0.0566*Tg+6.0645)*10**(-6)
nu_ar=mu_ar/rho_ar
u_ar=1.603*u_in/rho_ar
#parámetros para a_ar. ref. Toledo
ags_ar=6./(1.-e) #Superficie específica interna del medio poroso.
Pr_ar=(mu_ar*cp_ar)/lambda_ar #Número de Prandtl.
Re_ar=(e*u_ar*dp)/mu_ar #Número de Reynolds.
hgs_ar=(lambda_ar/dp)*(2.0+1.1*Pr_ar**(1/3)*Re_ar**0.6) #Coeficiente
→convectivo interfásico.
a_ar = ags_ar*hgs_ar #Coeficiente intersticial de intercambio de calor.
→
#Cálculo de vacancias
d_n=d.copy()
kred=(1-d_n/d_final)**(1/3)*Ared*np.exp(-Ered/(R*Ts)) #La temperatura
→todavía no se calcula para n+1, por tanto Ts corresponde a la temperatura en
→n.

d=(d_n/dt+d_max*kred)/(kred+kox+1/dt)

#Cálculo de temperaturas
for i in range (nx):
    if i==0:
        #sólido
        beta_s[i+1]=Ts_border

        #gas
        beta_g[i+1]=Tg_border

    elif i<(nx-1):
        #sólido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i] + A_s[i]+B_s[i]

```

```

        F_s[i]=Ts[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i]*Tg[i]+Qce/
        ↪ nx-(d[i]-d_n[i])/dt*(cCe/nx)*(10**6)*entalpia_red(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

        #gas
        A_g[i]=(lambda_ar[i-1]+lambda_ar[i])/
        ↪ (2*dx**2)+u_ar[i]*cp_ar[i]*rho_ar[i]/dx
        B_g[i]=(lambda_ar[i]+lambda_ar[i+1])/(2*dx**2)
        C_g[i]=(cp_ar[i]*rho_ar[i])/dt + a_ar[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tg[i]*(cp_ar[i]*rho_ar[i])/dt + a_ar[i]*Ts[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #sólido
        Ts[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tg[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #sólido
            Ts[j-1]=alfa_s[j]*Ts[j]+beta_s[j]
            #gas
            Tg[j-1]=alfa_g[j]*Tg[j]+beta_g[j]

        if j==0:
            break
    tiempo=nt/60 #[minutos]
    return Ts,Tg,d

```

```

[4]: from numba import njit #Numba es una biblioteca que optimiza códigos y ahorra
    ↪ mucho tiempo de iteración.
    #Optimización de función reducción
    reduccion_njit=njit(reduccion)

```

1.3 Gráficos y otros procedimientos

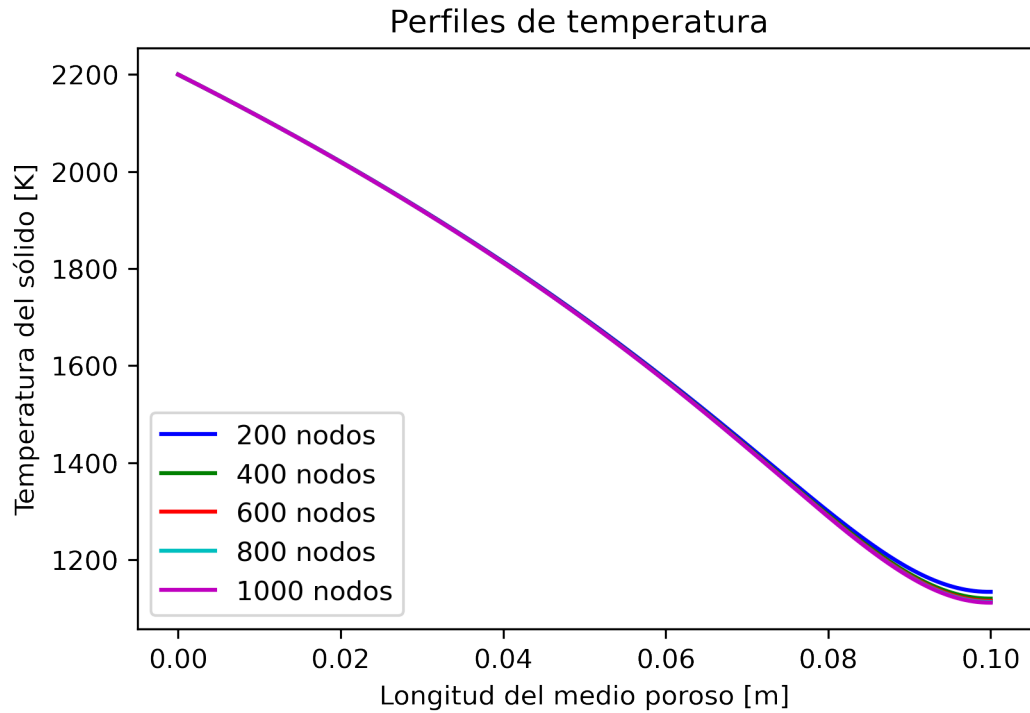
```
[5]: #Listas vacías de resultados a obtener
Ts_nx=[]
Tg_nx=[]
d_nx=[]
```

```
[6]: #Cálculo de reducción
nodos=[200, 400, 600, 800, 1000]
lp=0.1 #Tengo que definirlo fuera de la función
for i in range(len(nodos)):
    print(i)
    resultados=reduccion_njit(nodos[i])
    Ts_nx.append(resultados[0])
    Tg_nx.append(resultados[1])
    d_nx.append(resultados[2])
```

0
1
2
3
4

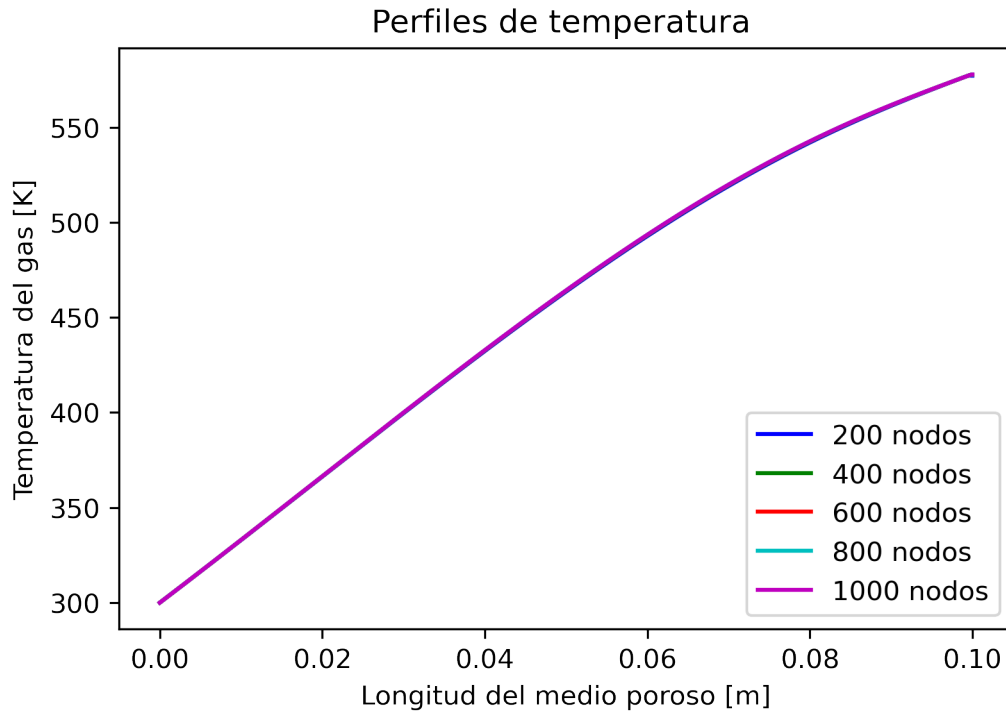
```
[7]: #Importes y gráfica de Ts
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
puntos=[]
colores=['b','g','r','c','m','k']
for i in range(len(nodos)):
    puntos.append(np.linspace(0,lp,nodos[i]))
    plt.plot(puntos[i],Ts_nx[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del sólido [K]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='lower left')
plt.show()
```



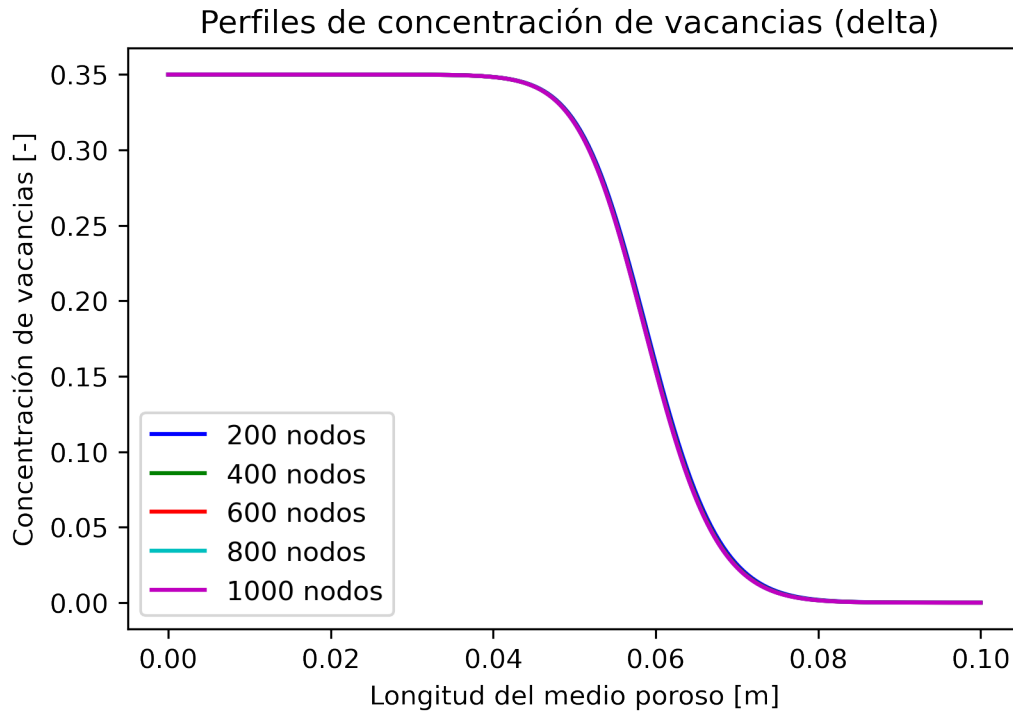
```
[8]: #Gráfica de Tg
colores=['b','g','r','c','m','k']
for i in range(len(nodos)):
    plt.plot(puntos[i],Tg_nx[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='lower right')
plt.show()
```



```
[9]: #Gráfica de d
colores=['b','g','r','c','m','k']
for i in range(len(nodos)):
    plt.plot(puntos[i],d_nx[i],colores[i])

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='lower left')
plt.show()
```



1.4 Función de Oxidación

```
[10]: #Definición de función oxidación
def oxidacion(nx):
    def entalpia_ox(delta):
        if delta==0:
            entalpia=-430+285.85
        elif 395-31.4*np.log10(delta) > 430:
            entalpia=-430+285.85
        else:
            entalpia= -(395-31.4*np.log10(delta))+285.85
        return entalpia

    #.....Variables.....

    u_in=1 #[m/s] Velocidad inicial del gas.
    Ts_inicial=700 #[K] Temperatura inicial del sólido para la
    Ts_bordeo=1200 #[K] Temperatura del sólido en x=0 en la oxidación.
    Qce=1000000 #[W/m3] Aporte de calor volumétrico externo.
    #nx=800 #[nodos] Cantidad de nodos espaciales.
    dt=0.02 #[s] Intervalo de tiempo.
```

```

t_reaccion=5 #[min] Tiempo de reacción.
#.....

#Constantes
dx=lp/nx #[-] Espaciamiento

#Condiciones de borde de temperatura y delta.

Ts_bordeo=float(Ts_bordeo) #Temperatura del sólido en x=0 en la oxidación.
Tg_bordeo=400. #Temperatura del gas en x=0 en la oxidación.
d_bordeo=0 #Concentración de vacancias en el sólido en x=0 en la oxidación.

#####
#RESOLUCION DE ECUACIONES

#Condiciones iniciales
d=reduccion_njit(nx)[2] #Las condiciones finales de la reducción se asignan
→aquí.
d_inicial=d.copy()
dox=d.copy()
dox[0]=0

Tsox=np.zeros(nx)+float(Ts_inicial)
Tsox[0]=Ts_bordeo
Tgox=np.zeros(nx)+300
Tgox[0]=Tg_bordeo

kox=np.zeros(nx)

A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)

```

```

beta_g=np.zeros(nx)

dox_final=np.zeros(nx)

tiempo=0
nt=0

while tiempo < t_reaccion:

    nt+=dt

    #sólido
    rho_s=(-0.0002*Tsox+6.8561)*1000
    cp_s= (0.0599*np.log(Tsox)+0.039)*1000
    lambda_s= (6.776*10**(-2)+2.793*10**(-4)*Tsox)**(-1)
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Tsox**3.)/(9.*(1.-e))

    #gas
    #nota: La temperatura del vapor varía muy poco, por ende se
    →considerarán propiedades constantes.

    rho_h2o=np.zeros(nx)+1/1.673 #kg/m3 #1/v Columna específico.
    cp_h2o=np.zeros(nx)+2.01*10**3 #j/kgK Calor específico.
    lambda_h2o=np.zeros(nx)+24.8*10**(-3) #W/mK Conductividad térmica del
    →vapor.
    mu_h2o= np.zeros(nx)+12.0*10**(-6) #Pa*s o kg/ms Viscosidad dinámica del
    →vapor.
    nu_h2o=mu_h2o/rho_h2o #m**2/s Viscosidad cinemática.
    u_h2o=np.zeros(nx)+u_in

    #parámetros para a_ar. ref. Toledo
    ags_h2o=6./(1.-e)
    Pr_h2o=(mu_h2o*cp_h2o)/lambda_h2o
    Re_h2o=(e*u_h2o*dp)/mu_h2o
    hgs_h2o=(lambda_h2o/dp)*(2.0+1.1*Pr_h2o**(1/3)*Re_h2o**0.6)
    a_h2o = ags_h2o*hgs_h2o

    #Cálculo de vacancias
    dox_n=dox.copy()
    kox=yox**gamma*(1-(d_inicial-dox_n)/(d_inicial))*Aox*np.exp(-Eox/
    →(R*Tsox))
    dox=(dox_n/dt)/(kox+1/dt)

    #Cálculo de temperaturas
    for i in range (nx):
        if i==0:

```

```

        #sólido
        beta_s[i+1]=Ts_bordeo

        #gas
        beta_g[i+1]=Tg_bordeo

    elif i<(nx-1):
        #sólido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_h2o[i] + A_s[i]+B_s[i]
        F_s[i]=Tsox[i]*(1-e)*(cp_s[i]*rho_s[i])/dt +
        ↪a_h2o[i]*Tgox[i]+Qce/nx+(dox[i]-dox_n[i])/dt*(cCe/
        ↪nx)*(10**6)*entalpia_ox(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

        #gas
        A_g[i]=(lambda_h2o[i-1]+lambda_h2o[i])/
        ↪(2*dx**2)+u_h2o[i]*cp_h2o[i]*rho_h2o[i]/dx
        B_g[i]=(lambda_h2o[i]+lambda_h2o[i+1])/(2*dx**2)
        C_g[i]=(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tgox[i]*(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]*Tsox[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #sólido
        Tsox[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tgox[i]=beta_g[i]/(1-alfa_g[i])

for j in range(nx-1,-1,-1):
    if j!=0:
        #sólido
        Tsox[j-1]=alfa_s[j]*Tsox[j]+beta_s[j]
        #gas
        Tgox[j-1]=alfa_g[j]*Tgox[j]+beta_g[j]

    if j==0:

```

```

        break
    tiempo=nt/60
    return Tsox,Tgox,dox

```

```

[11]: #Optimización de función oxidación
oxidacion_njit=njit(oxidacion)

```

1.5 Gráficos y otros procedimientos

```

[12]: #Listas vacías de resultados a obtener
Tsox_nx=[]
Tgox_nx=[]
dox_nx=[]

```

```

[13]: #Calculo de oxidación
nodos=[200, 400, 600, 800, 1000]
lp=0.1 #Tengo que definirlo fuera de la función
for i in range(len(nodos)):
    print (i)
    #cálculo oxidación
    resultadosox=oxidacion_njit(nodos[i])
    Tsox_nx.append(resultadosox[0])
    Tgox_nx.append(resultadosox[1])
    dox_nx.append(resultadosox[2])

```

```

0
1
2
3
4

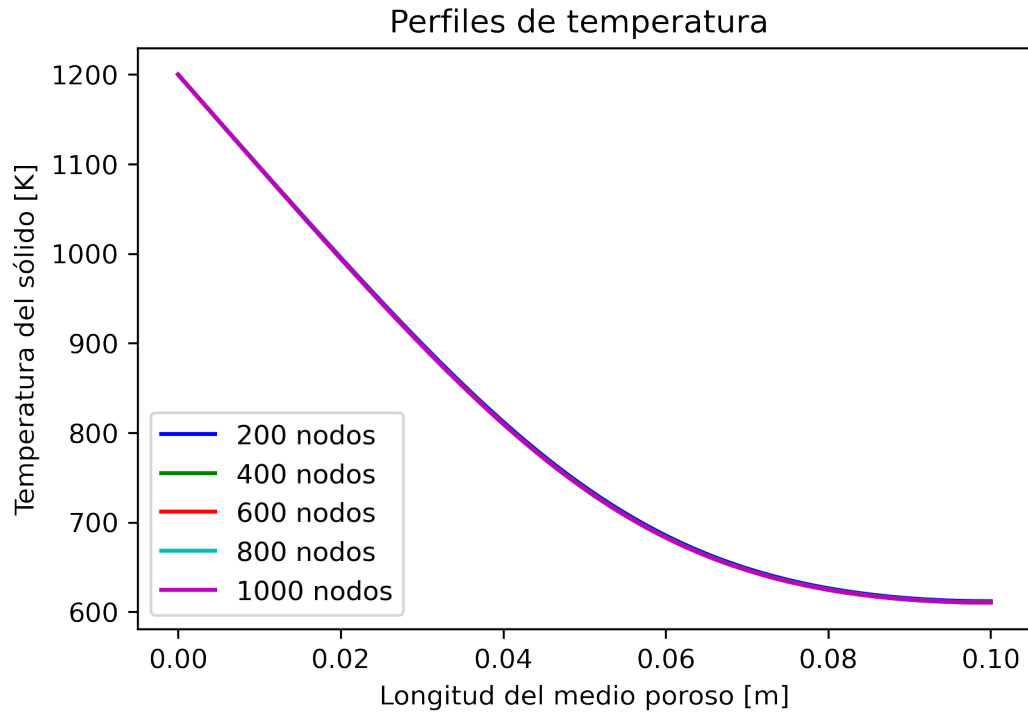
```

```

[14]: #Gráfica de Ts
for i in range(len(nodos)):
    puntos.append(np.linspace(0,lp,nodos[i]))
    plt.plot(puntos[i],Tsox_nx[i],colores[i])

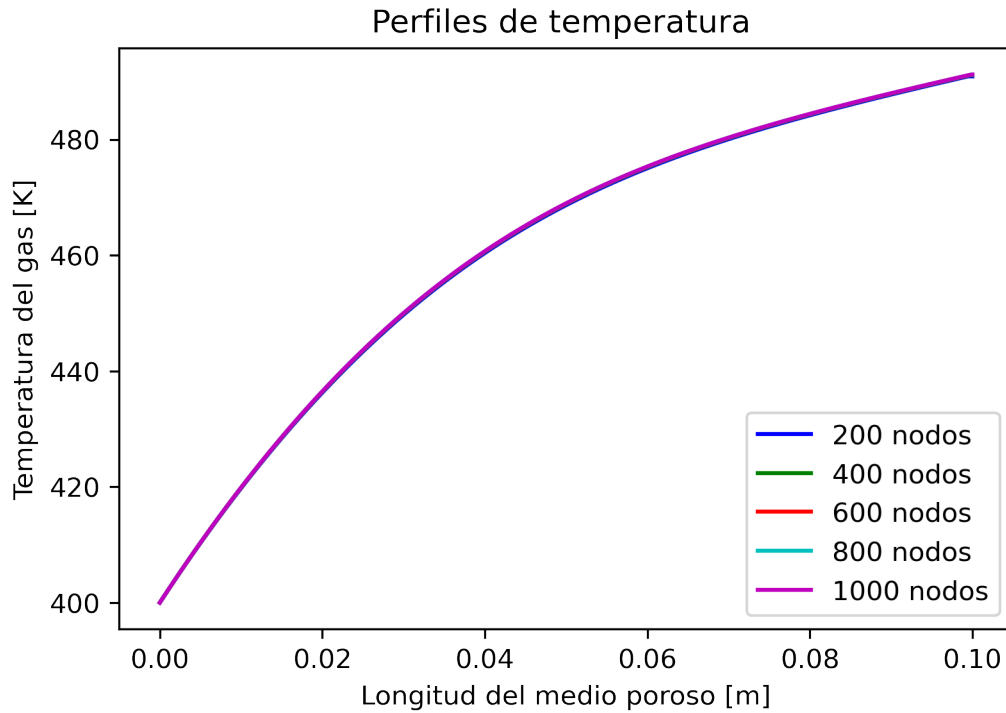
#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del sólido [K]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='lower left')
plt.show()

```



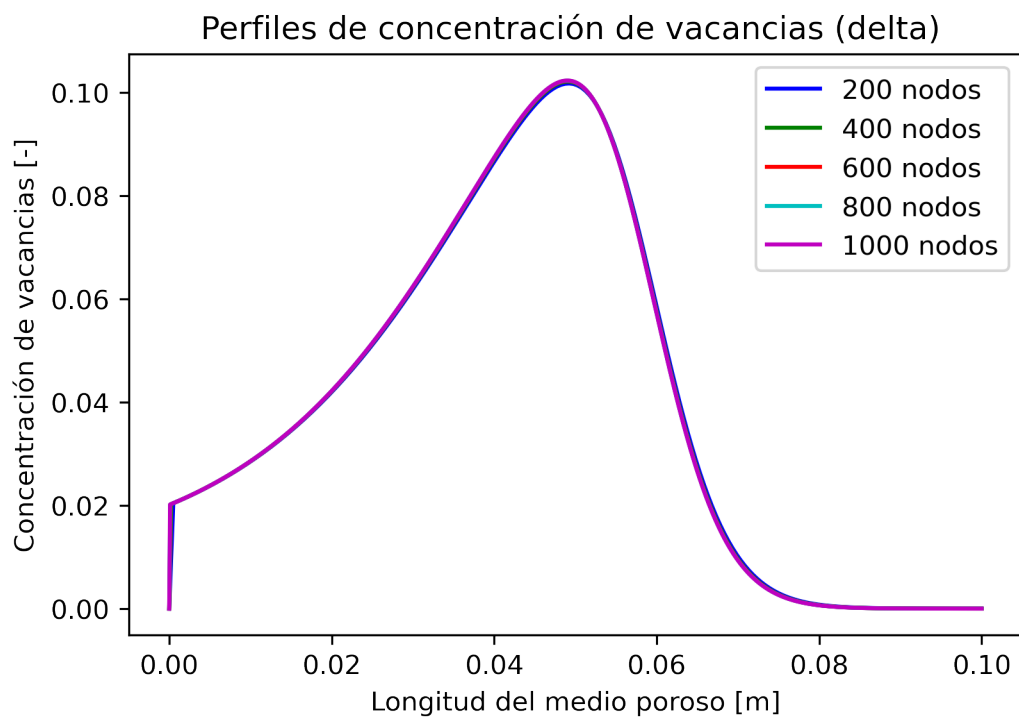
```
[15]: #Gráfica de Tg
colores=['b','g','r','c','m','k']
for i in range(len(nodos)):
    plt.plot(puntos[i],Tgox_nx[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='lower right')
plt.show()
```



```
[17]: #Gráfica de d
colores=['b','g','r','c','m','k']
for i in range(len(nodos)):
    plt.plot(puntos[i],dox_nx[i],colores[i])

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
plt.legend(('200 nodos','400 nodos','600 nodos','800 nodos','1000 nodos'),
           prop = {'size': 10}, loc='upper right')
plt.show()
```



Anexo B3

May 31, 2021

1 Modelo 1-D: Casos extremos

1.1 Constantes

```
[1]: R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases.
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción. Bulfin
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción. Bulfin
lp=0.1 # [m] Longitud del medio poroso. ***DATO CÁMARA DE REACCIÓN ***
vol=7.3*10**(-4) # [m3] Volumen de la cámara de reacción. ***DATO CÁMARA DE
    ↪ REACCIÓN
e=0.65 # [-] Porosidad del medio poroso. ***DATO PELLETS BULFIN***
dp=0.01 # [m] diámetro de partícula ***DATO PELLETS BULFIN***
cCe=(1-e)*34.55 # [kmol/m3] concentración de cerio. Cálculo
molCeO2=(1-e)*30.6225 # =10.7179 moles totales de CeO2 considerando pellets de
    ↪ CeO2.
sigma=5.67*10**(-8) # [W/m2K4] Constante de Stefan-Boltzmann.

#Cálculo moles totales CeO2
# 7.22 g/cm3 * 730 cm3 / 172.115(g/mol)= 30.6225 moles de CeO2 totales (si todo
    ↪ el volumen fuera CeO2)

yox=1 # Fracción molar de oxidante
Aox=1.0 # Factor pre-exponencial de oxidación. Arifin
Eox=29.0 # Energía de activación de oxidación. Arifin
gamma=0.89 # [-] Orden de reacción para la oxidación. Arifin
```

1.2 Reducción

```
[2]: import numpy as np
def reduccion(Qce,u_in,Ts_border,t_reaccion): # Los inputs se eligen desde el
    ↪ apartado 'variables' ubicado mas abajo. Al seleccionar se debe marcar con #

    # Funciones entalpía de reacción
    def entalpia_red(delta): # Valores EN kJ/mol

        if delta==0:
            entalpia=430
        elif 395-31.4*np.log10(delta) > 430:
```

```

        entalpia=430
    else:
        entalpia= 395-31.4*np.log10(delta)
    return entalpia

#.....Variables.....
#Qce=1000000 #[W/m3] Aporte de calor volumétrico externo.
#u_in=1. # [m/s] Velocidad inicial del gas
#Ts_border=2100 #[K] Temperatura del solido en x=0 en la reducción.

nx=800 #[nodos] Cantidad de nodos temporales
dt=0.02 #[segundos] Intervalo de tiempo
#t_reaccion=15 #[minutos] Tiempo de reacción.
#.....

#Constantes
Tg_border=300.#Temperatura del gas en x=0 en la reducción.
d_border=0.35 #Concentracion de vacancias en el sólido en x=0 en la
↪reducción.
d_max=0.35 #Concentración de vacancias máxima.
dx=lp/nx #[-] Espaciamiento
molxnodo=molCeO2/nx #cantidad de moles de CeO2 por nodo.

#####
#RESOLUCION DE ECUACIONES

#condiciones iniciales
d=np.zeros(nx)
d[0]=d_border
d_final=np.zeros(nx)+0.35 #condición final de reducción (consideración de
↪reducción completa)

Ts=np.zeros(nx)+300
Ts[0]=Ts_border
Tg=np.zeros(nx)+300
Tg[0]=Tg_border

kred=np.zeros(nx) #mod eq vacancias
kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se usan
↪para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)

```

```

F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt #contador para el tiempo de reacción

    #primero calculo todas las propiedades en función de la temperatura

    #sólido
    rho_s=(-0.0002*Ts+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref. Nelson 2014
    cp_s= (0.0599*np.log(Ts)+0.0390)*1000 #[J/kgK]calor específico del sólido * ref. Nelson 2014
    lambda_s= (6.7760*10**(-2)+2.7930*10**(-4)*Ts)**(-1)#[W/mK] Coeficiente de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Ts**3.)/(9.*(1.-e)) #[W/mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

    #gas (argón)
    rho_ar=1.603*300/Tg #[kg/m3]. Gases ideales
    cp_ar=(20.7860+Tg*2.8259*10**(-7)-Tg**2*1.4642*10**(-7)+Tg**3*1.0921*10**(-8)-Tg**(-1)*3.6614*10**(-8))*1000/39.948 #[J/kgK]. ref. webbook. nist.gob. Chase 1998
    lambda_ar=(0.0442*Tg + 4.7827)/1000 #[W/Km] Conductividad térmica. Ajuste Younglove 1986.
    mu_ar= (0.0566*Tg+6.0645)*10**(-6) ##[Pa*s] viscosidad dinámica del argón. Ajuste Younglove 1986.

```

```

nu_ar=mu_ar/rho_ar # viscosidad cinemática del argon
u_ar=1.603*u_in/rho_ar # m/s velocidad del argon (depende de la
↳ densidad)

#parámetros para a_ar. ref. Toledo
ags_ar=6./(1.-e) #superficie específica interna del medio poroso
Pr_ar=(mu_ar*cp_ar)/lambda_ar #Número de Prandtl
Re_ar=(e*u_ar*dp)/mu_ar #Número de Reynolds
hgs_ar=(lambda_ar/dp)*(2.0+1.1*Pr_ar**(1/3)*Re_ar**0.6) #Coeficiente
↳ convectivo interfásico
a_ar = ags_ar*hgs_ar #Coeficiente intersticial de intercambio de calor

#Cálculo de vacancias
d_n=d.copy() #d_n sería d anterior (d^n), d sería el valor actual
↳ d^(n+1)

kred=(1-d_n/d_final)**(1/3)*Ared*np.exp(-Ered/(R*Ts)) #La temperatura
↳ todavía no se calcula para n+1, por tanto Ts corresponde a la temperatura en
↳ n.
kox=y_ox**gamma*(1-d_n/d_final)*Aox*np.exp(-Eox/(R*Ts)) Para el caso
↳ de la reducción será cero en la reducción

d=(d_n/dt+d_max*kred)/(kred+kox+1/dt) #como son estáticas no requiere
↳ recorrer x

#Cálculo de temperaturas
for i in range (nx):
    if i==0:
        #sólido
        beta_s[i+1]=Ts_border

        #gas
        beta_g[i+1]=Tg_border

    elif i<(nx-1):
        #sólido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i] + A_s[i]+B_s[i]
        F_s[i]=Ts[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i]*Tg[i]+Qce/
↳ nx-(d[i]-d_n[i])/dt*(cCe/nx)*(10**6)*entalpia_red(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

```

```

        #gas
        A_g[i]=(lambda_ar[i-1]+lambda_ar[i])/
        ↪(2*dx**2)+u_ar[i]*cp_ar[i]*rho_ar[i]/dx
        B_g[i]=(lambda_ar[i]+lambda_ar[i+1])/(2*dx**2)
        C_g[i]=(cp_ar[i]*rho_ar[i])/dt + a_ar[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tg[i]*(cp_ar[i]*rho_ar[i])/dt + a_ar[i]*Ts[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Ts[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tg[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Ts[j-1]=alfa_s[j]*Ts[j]+beta_s[j]
            #gas
            Tg[j-1]=alfa_g[j]*Tg[j]+beta_g[j]

        if j==0:

            break
    tiempo=nt/60 #[minutos]

    #Cálculo de producción de oxígeno

    prod_o2_nodo=molxnodo*d/2 #producción de oxígeno por nodo
    prod_o2=np.sum(prod_o2_nodo) #producción total de oxígeno
    ##ecuación de producción de o2
    return Ts,Tg,d,prod_o2

```

```

[3]: from numba import njit #Numba es una biblioteca que optimiza códigos y ahorra
    ↪mucho tiempo de iteración.
    reduccion_njit=njit(reduccion)

```

1.3 Oxidación

```
[4]: def oxidacion(Qce,u_in,Ts_bordeo,Ts_inicial,t_reaccion,Qred,ured,Tred,tred):  
    ↪#reduccion_tuple es una tupla con los parámetros de entrada de reducción  
    def entalpia_ox(delta):  
        if delta==0:  
            entalpia=-430+285.85  
        elif 395-31.4*np.log10(delta) > 430:  
            entalpia=-430+285.85  
        else:  
            entalpia= -(395-31.4*np.log10(delta))+285.85  
        return entalpia  
  
    #.....Variables.....  
    #=1000000 #[W/m3] Aporte de calor volumétrico externo.  
    #u_in=1 #[m/s] Velocidad inicial del gas.  
    #Ts_bordeo=1200 #[K] Temperatura del sólido en x=0 en la oxidación.  
    #Ts_inicial=700 #[K] Temperatura inicial del sólido para la  
  
    nx=800 #[nodos] Cantidad de nodos espaciales.  
    dt=0.02 #[s] Intervalo de tiempo.  
    #t_reaccion=15 #[min] Tiempo de reacción.  
    #.....  
  
    #Constantes  
    dx=lp/nx #[-] Espaciamiento  
    molxnodo=molCeO2/nx #cantidad de moles de CeO2 por nodo.  
  
    #Condiciones de borde de temperatura y delta.  
  
    Ts_bordeo=float(Ts_bordeo) #Temperatura del solido en x=0 en la oxidación.  
    Tg_bordeo=400. #Temperatura del gas en x=0 en la oxidación.  
    d_bordeo=0 #Concentración de vacancias en el sólido en x=0 en la oxidación.  
  
    #####  
    #RESOLUCION DE ECUACIONES  
  
    #condiciones iniciales  
    d=reduccion_njit(Qred,ured,Tred,tred)[2] #Las condiciones finales de la  
    ↪reducción se asignan aquí.  
    d_inicial=d.copy()  
    dox=d.copy()  
    dox[0]=0
```

```

Tsox=np.zeros(nx)+float(Ts_inicial)
Tsox[0]=Ts_bordeo
Tgox=np.zeros(nx)+300
Tgox[0]=Tg_bordeo

kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se
→usan para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

#condición final de reducción
dox_final=np.zeros(nx) #Asumimos que todo el material se oxidará

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt

    #sólido
    rho_s=(-0.0002*Tsox+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref.
→Nelson 2014
    cp_s= (0.0599*np.log(Tsox)+0.039)*1000 #[J/kgK]calor específico del
→sólido * ref. Nelson 2014
    lambda_s= (6.776*10**(-2)+2.793*10**(-4)*Tsox)**(-1)#[W/mK] Coeficiente
→de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Tsox**3.)/(9.*(1.-e)) #[W/
→mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

```

```

#gas
#nota: La temperatura del vapor varía muy poco, por ende se
→considerarán propiedades constantes.

rho_h2o=np.zeros(nx)+1/1.673 #kg/m3 #1/v volumen específico
cp_h2o=np.zeros(nx)+2.01*10**3 #j/kgK calor específico
lambda_h2o=np.zeros(nx)+24.8*10**(-3) #W/mK conductividad térmica del
→vapor
mu_h2o= np.zeros(nx)+12.0*10**(-6)#Pa*s o kg/ms Viscosidad dinámica del
→vapor
nu_h2o=mu_h2o/rho_h2o #m**2/s viscosidad cinemática
u_h2o=np.zeros(nx)+u_in

#parámetros para a_ar. ref. Toledo
ags_h2o=6./(1.-e) #superficie específica interna del medio poroso
Pr_h2o=(mu_h2o*cp_h2o)/lambda_h2o #Número de Prandtl
Re_h2o=(e*u_h2o*dp)/mu_h2o #Número de Reynolds
hgs_h2o=(lambda_h2o/dp)*(2.0+1.1*Pr_h2o**(1/3)*Re_h2o**0.6)
→#Coeficiente convectivo interfásico
a_h2o = ags_h2o*hgs_h2o #Coeficiente intersticial de intercambio de
→calor

#Cálculo de vacancias
dox_n=dox.copy() #d_n sería d anterior (d~n), d sería el valor actual
→d^(n+1)
kox=yox**gamma*(1-(d_inicial-dox_n)/(d_inicial))*Aox*np.exp(-Eox/
→(R*Tsox))
dox=(dox_n/dt)/(kox+1/dt) #como son estáticas no requiere recorrer x

#Cálculo de temperaturas
for i in range (nx):
    if i==0:
        #solido
        beta_s[i+1]=Ts_bordeo

        #gas
        beta_g[i+1]=Tg_bordeo

    elif i<(nx-1):
        #solido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_h2o[i] + A_s[i]+B_s[i]

```

```

        F_s[i]=Tsox[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + u
        a_h2o[i]*Tgox[i]+Qce/nx+(dox[i]-dox_n[i])/dt*(cCe/
        nx)*(10**6)*entalpia_ox(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

        #gas
        A_g[i]=(lambda_h2o[i-1]+lambda_h2o[i])/
        (2*dx**2)+u_h2o[i]*cp_h2o[i]*rho_h2o[i]/dx
        B_g[i]=(lambda_h2o[i]+lambda_h2o[i+1])/(2*dx**2)
        C_g[i]=(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tgox[i]*(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]*Tsox[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Tsox[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tgox[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Tsox[j-1]=alfa_s[j]*Tsox[j]+beta_s[j]
            #gas
            Tgox[j-1]=alfa_g[j]*Tgox[j]+beta_g[j]

        if j==0:
            break
    tiempo=nt/60

    #Cálculo de producción de oxígeno
    prod_h2_nodo=molxnodo*(d-dox) #producción de hidrógeno por nodo
    prod_h2=np.sum(prod_h2_nodo) #producción total de hidrógeno
    return Tsox,Tgox,dox,prod_h2

```

```
[5]: oxidacion_njit=njit(oxidacion)
```

1.4 Perfiles de temperatura y vacancias para casos extremos

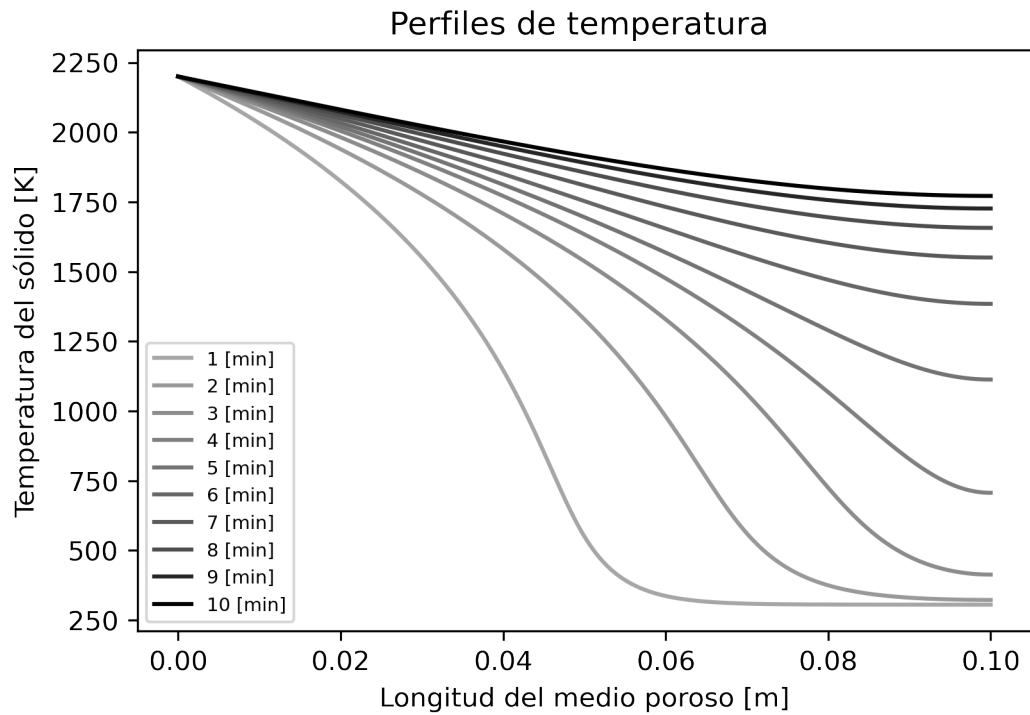
1.4.1 Caso más favorable:

Reduccion

```
[6]: t_reac=[1,2,3,4,5,6,7,8,9,10]
Ts_red=[]
Tg_red=[]
d_red=[]
for i in range(len(t_reac)):
    resultados=reduccion_njit(10**6,1,2200,t_reac[i])
    Ts_red.append(resultados[0])
    Tg_red.append(resultados[1])
    d_red.append(resultados[2])

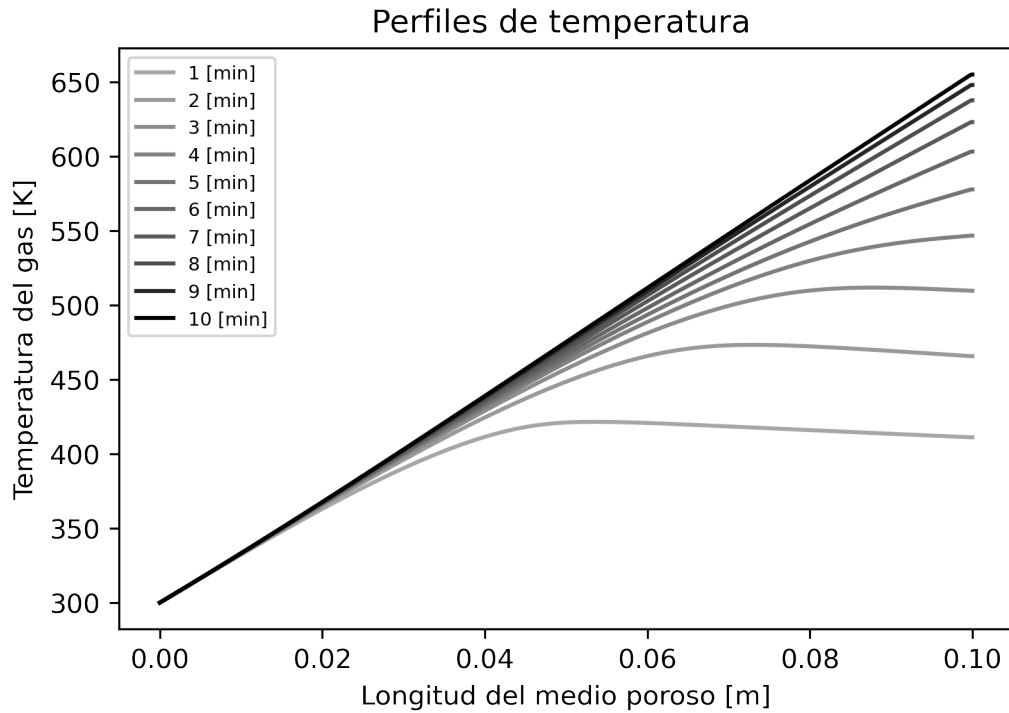
[7]: import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200
#colores=['b','g','r','c','m','k','y']
colores=['0.65','0.60','0.55','0.50','0.45','0.40','0.35','0.30','0.15','0.0']
nodos=np.linspace(0,lp,800)
for i in range(len(t_reac)):
    plt.plot(nodos,Ts_red[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del sólido [K]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='lower left')
plt.show()
```



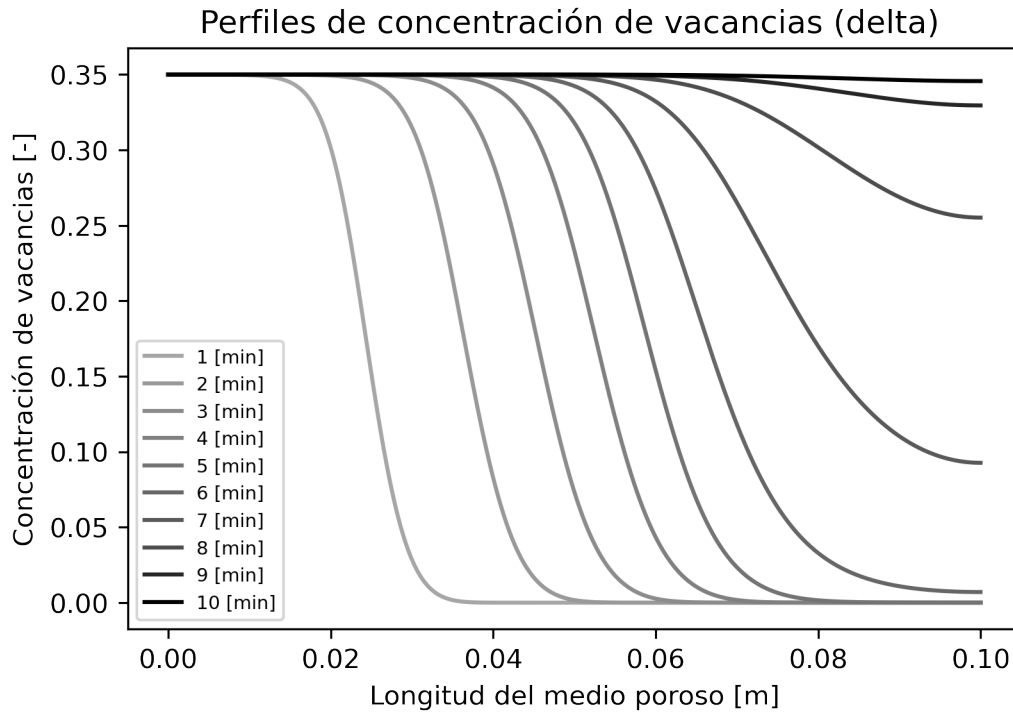
```
[8]: for i in range(len(t_reac)):
      plt.plot(nodos,Tg_red[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper left')
plt.show()
```



```
[9]: for i in range(len(t_reac)):
      plt.plot(nodos,d_red[i],colores[i])

      #Datos extra
      plt.title('Perfiles de concentración de vacancias (delta) ')
      plt.xlabel('Longitud del medio poroso [m]')
      plt.ylabel('Concentración de vacancias [-]')
      plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
                prop = {'size': 7}, loc='lower left')
      plt.show()
```

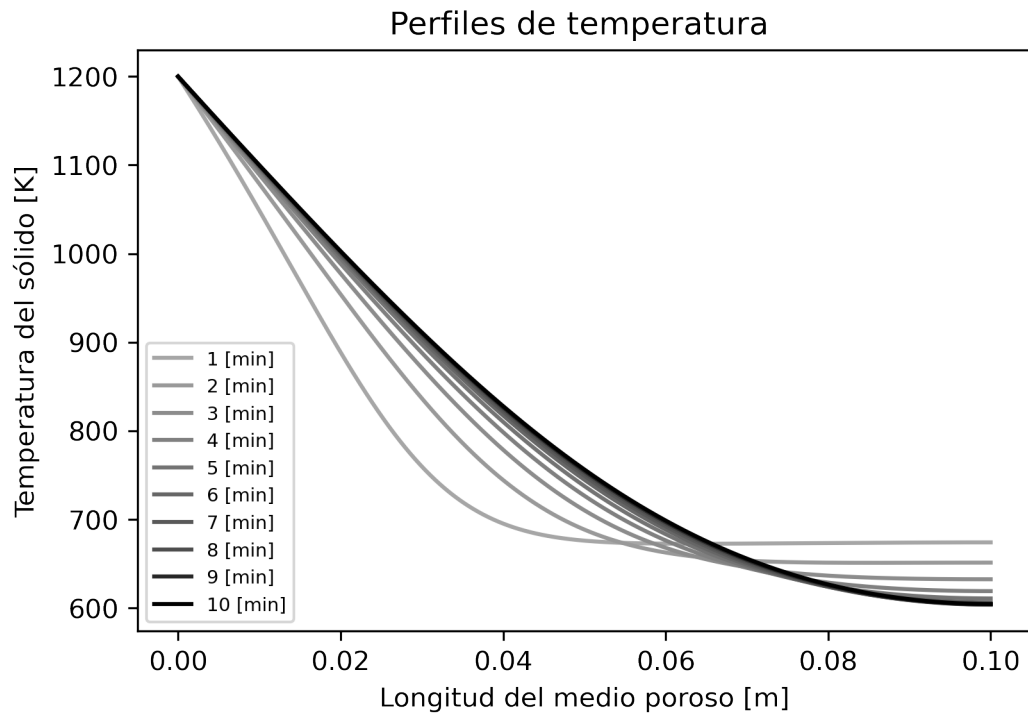


Oxidación

```
[10]: Ts_ox=[]
      Tg_ox=[]
      d_ox=[]
      for i in range(len(t_reac)):
          resultados=oxidacion_njit(10**6,1,1200,700,t_reac[i],10**6,1,2200,10)
          Ts_ox.append(resultados[0])
          Tg_ox.append(resultados[1])
          d_ox.append(resultados[2])
```

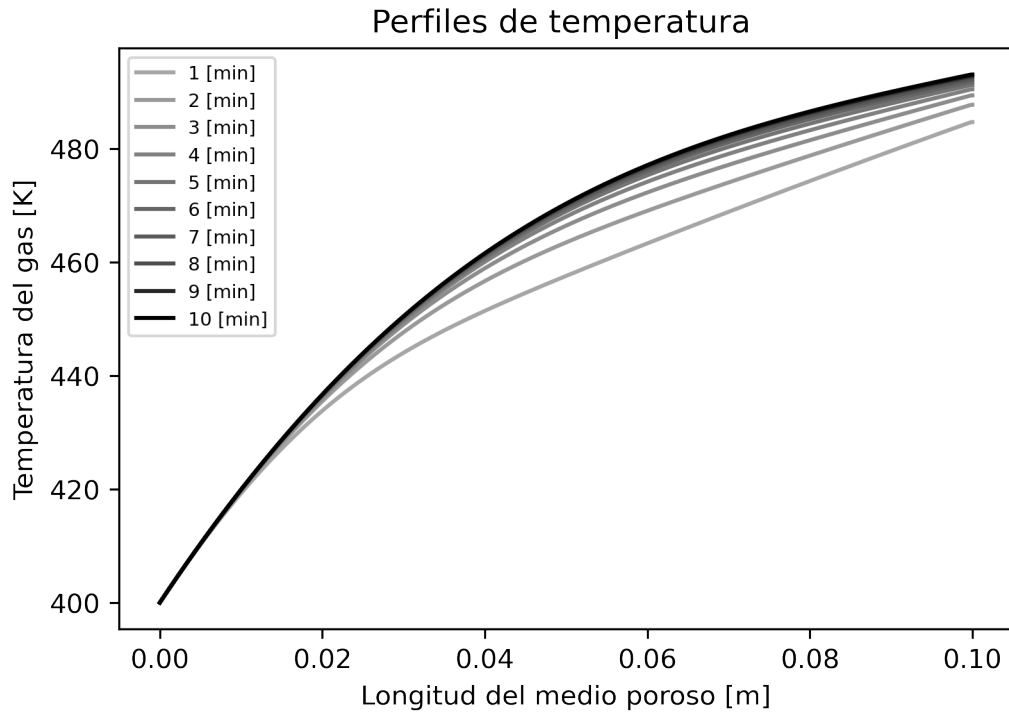
```
[11]: for i in range(len(t_reac)):
      plt.plot(nodos,Ts_ox[i],colores[i])

      #Datos extra
      plt.title('Perfiles de temperatura ')
      plt.xlabel('Longitud del medio poroso [m]')
      plt.ylabel('Temperatura del sólido [K]')
      plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
                  prop = {'size': 7}, loc='lower left')
      plt.show()
```



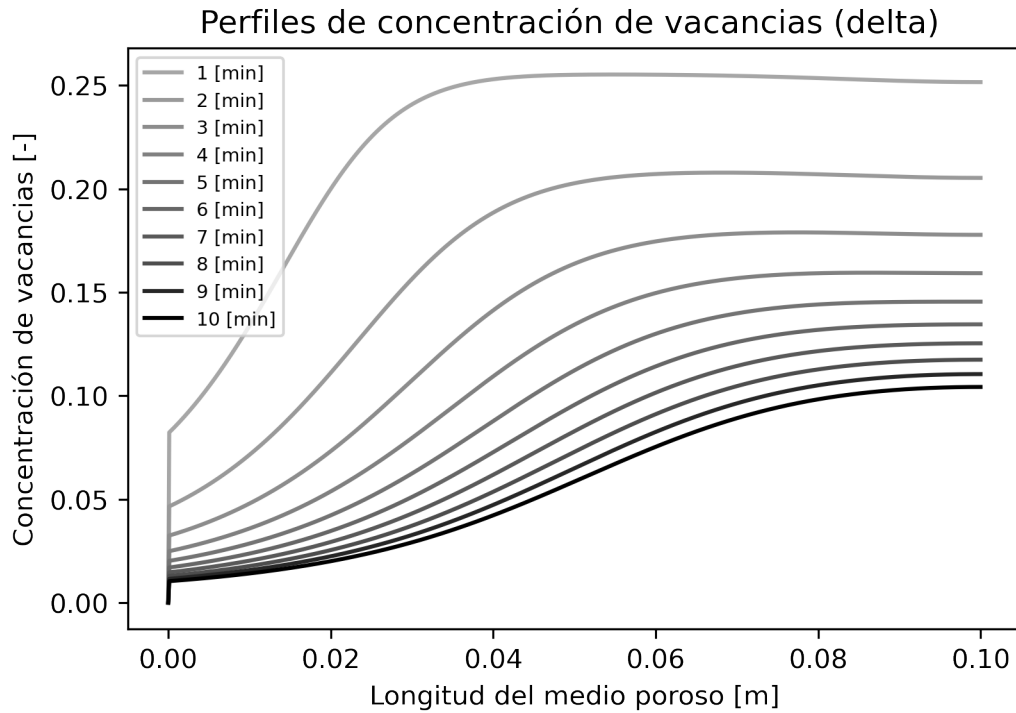
```
[12]: for i in range(len(t_reac)):
        plt.plot(nodos,Tg_ox[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper left')
plt.show()
```



```
[13]: for i in range(len(t_reac)):
        plt.plot(nodos,d_ox[i],colores[i])

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper left')
plt.show()
```



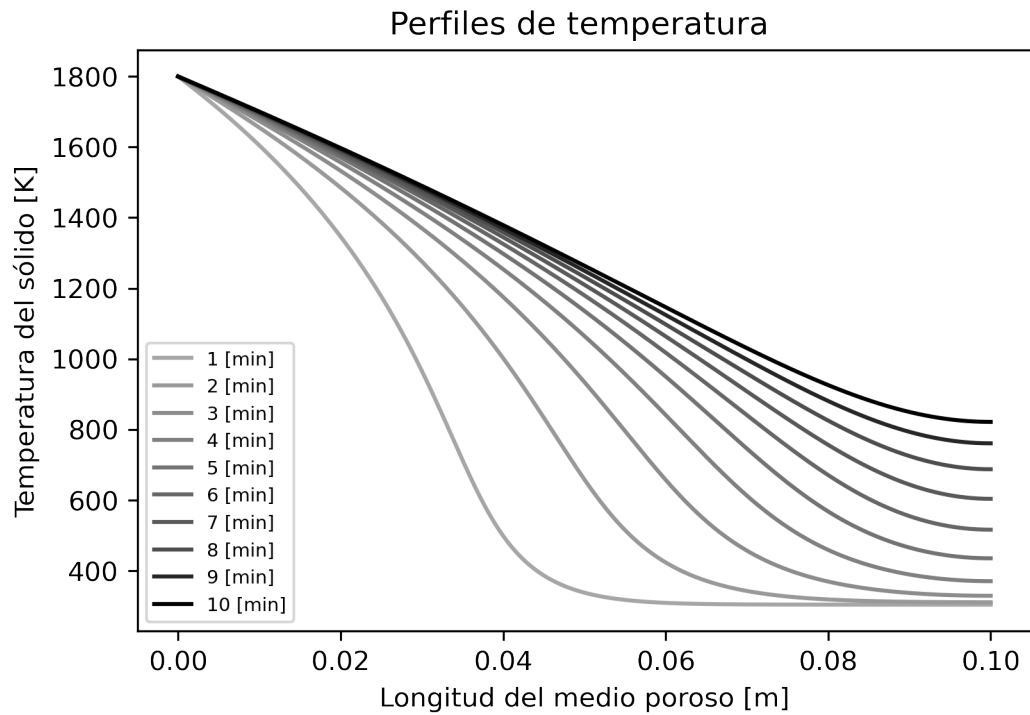
1.4.2 Caso mas desfavorable

Reducción

```
[32]: Ts_red=[]
      Tg_red=[]
      d_red=[]
      for i in range(len(t_reac)):
          resultados=reduccion_njit(10**2,3,1800,t_reac[i])
          Ts_red.append(resultados[0])
          Tg_red.append(resultados[1])
          d_red.append(resultados[2])

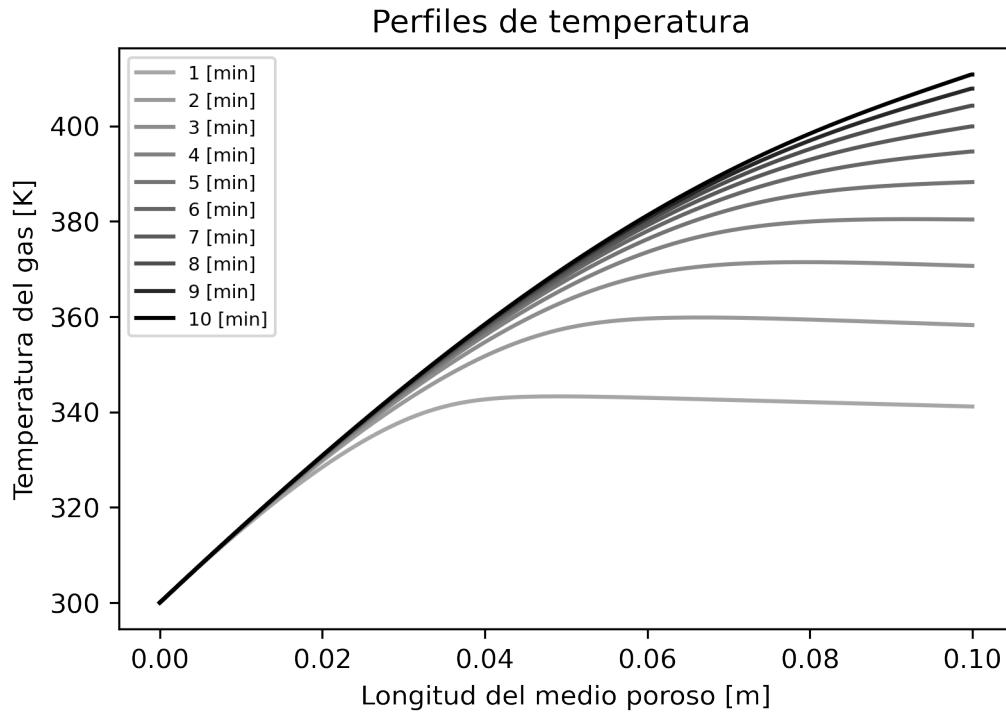
[34]: for i in range(len(t_reac)):
      plt.plot(nodos,Ts_red[i],colores[i])

      #Datos extra
      plt.title('Perfiles de temperatura ')
      plt.xlabel('Longitud del medio poroso [m]')
      plt.ylabel('Temperatura del sólido [K]')
      plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
                  prop = {'size': 7}, loc='lower left')
      plt.show()
```



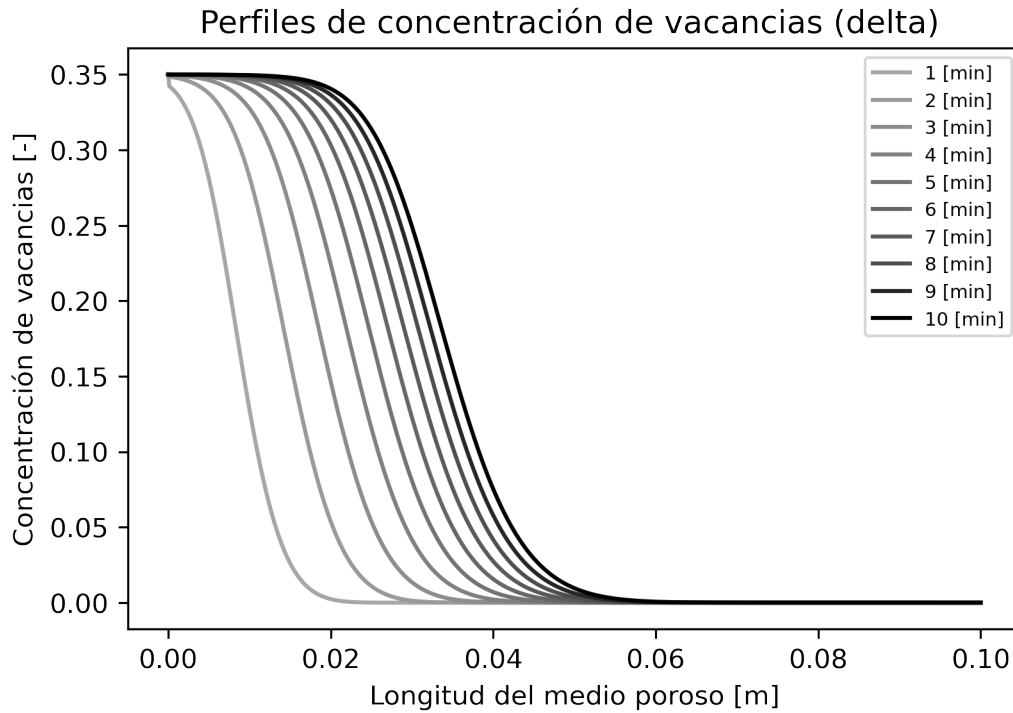
```
[35]: for i in range(len(t_reac)):
        plt.plot(nodos,Tg_red[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper left')
plt.show()
```



```
[36]: for i in range(len(t_reac)):
        plt.plot(nodos,d_red[i],colores[i])

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper right')
plt.show()
```

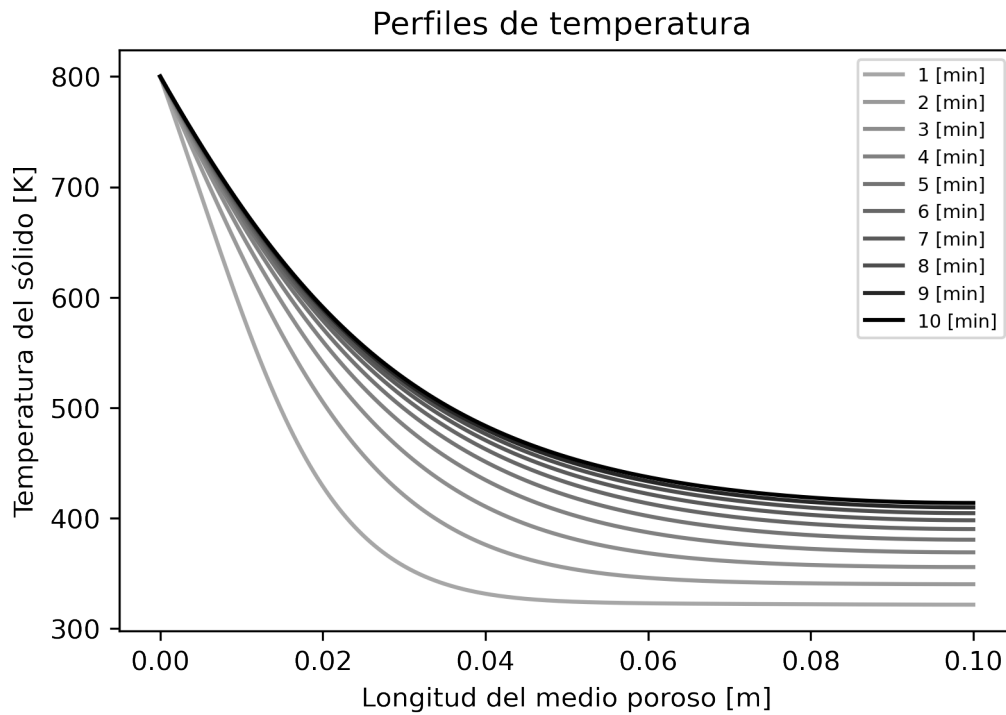


Oxidación

```
[37]: Ts_ox=[]
      Tg_ox=[]
      d_ox=[]
      for i in range(len(t_reac)):
          resultados=oxidacion_njit(10**2,3,800,300,t_reac[i],10**2,3,1800,10)
          Ts_ox.append(resultados[0])
          Tg_ox.append(resultados[1])
          d_ox.append(resultados[2])

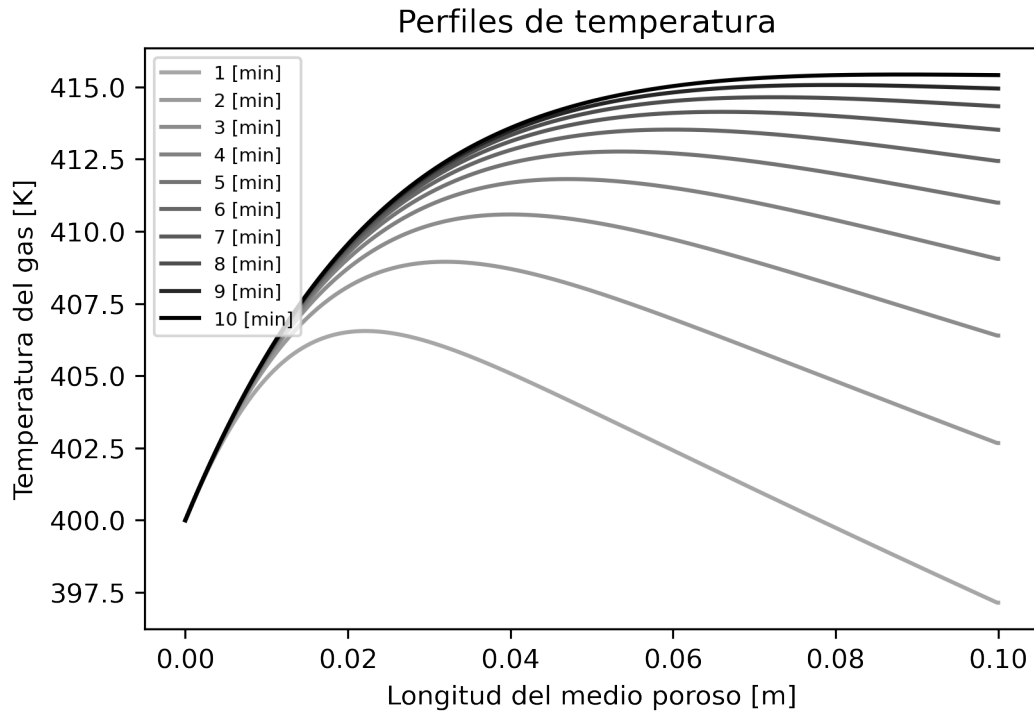
[39]: for i in range(len(t_reac)):
      plt.plot(nodos,Ts_ox[i],colores[i])

      #Datos extra
      plt.title('Perfiles de temperatura ')
      plt.xlabel('Longitud del medio poroso [m]')
      plt.ylabel('Temperatura del sólido [K]')
      plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
                  prop = {'size': 7}, loc='upper right')
      plt.show()
```



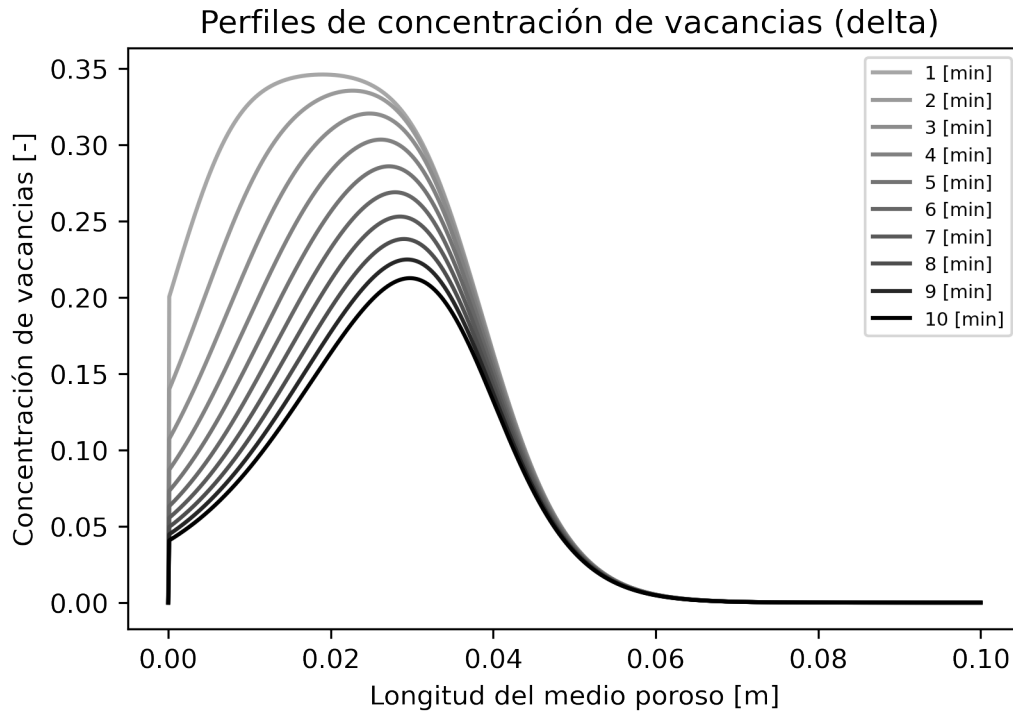
```
[40]: for i in range(len(t_reac)):
        plt.plot(nodos,Tg_ox[i],colores[i])

#Datos extra
plt.title('Perfiles de temperatura ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Temperatura del gas [K]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper left')
plt.show()
```



```
[41]: for i in range(len(t_reac)):
        plt.plot(nodos,d_ox[i],colores[i])

#Datos extra
plt.title('Perfiles de concentración de vacancias (delta) ')
plt.xlabel('Longitud del medio poroso [m]')
plt.ylabel('Concentración de vacancias [-]')
plt.legend(('1 [min]','2 [min]','3 [min]','4 [min]','5 [min]','6 [min]','7 [min]','8 [min]','9 [min]','10 [min]'),
           prop = {'size': 7}, loc='upper right')
plt.show()
```



1.5 Producción de O₂/H₂ para casos extremos y combinación

1.5.1 Caso +R +O

Reducción: $Q_{ce}=10^{**6}$ W/m³ $u_{gas}=1$ m/s $T_{border}=2200$ K tiempo=10 min

Oxidación: $Q_{ce}=10^{**6}$ W/m³ $u_{gas}=1$ m/s $T_{sborde}=1200$ K $T_{sinicial}=700$ K tiempo=10 min

```
[6]: red1=reduccion_njit(10**6,1,2200,10)
```

```
[7]: ox1=oxidacion_njit(10**6,1,1200,700,10,10**6,1,2200,10)
```

```
[8]: print('Produccion total de oxígeno:',red1[3], 'moles')
      print('Produccion total de hidrógeno:',ox1[3], 'moles')
```

Produccion total de oxígeno: 1.8708797308986194 moles

Produccion total de hidrógeno: 3.112462998949285 moles

1.5.2 Caso +R -O

Reducción: $Q_{ce}=10^{**6}$ W/m³ $u_{gas}=1$ m/s $T_{border}=2200$ K tiempo=10 min

Oxidación: $Q_{ce}=10^{**2}$ W/m³ $u_{gas}=3$ m/s $T_{sborde}=800$ K $T_{sinicial}=300$ K tiempo=10 min

```
[9]: red2=reduccion_njit(10**6,1,2200,10)
```

```
[10]: ox2=oxidacion_njit(10**2,3,800,300,10,10**6,1,2200,10)
```

```
[11]: print('Produccion total de oxígeno:',red2[3], 'moles')
      print('Produccion total de hidrógeno:',ox2[3], 'moles')
```

Produccion total de oxígeno: 1.8708797308986194 moles

Produccion total de hidrógeno: 1.00545185986426 moles

1.5.3 Caso -R +O

Reducción: $Q_{ce}=10^{**2}$ W/m³ $u_{gas}=3$ m/s $T_{border}=1800$ K tiempo=10 min

Oxidación: $Q_{ce}=10^{**6}$ W/m³ $u_{gas}=1$ m/s $T_{sborde}=1200$ K $T_{sinicial}=700$ K tiempo=10 min

```
[12]: red3=reduccion_njit(10**2,3,1800,10)
```

```
[13]: ox3=oxidacion_njit(10**6,1,1200,700,10,10**2,3,1800,10)
```

```
[14]: print('Produccion total de oxígeno:',red3[3], 'moles')
      print('Produccion total de hidrógeno:',ox3[3], 'moles')
```

Produccion total de oxígeno: 0.6435200959771489 moles

Produccion total de hidrógeno: 1.2114944745887772 moles

1.5.4 Caso -R -O

Reducción: $Q_{ce}=10^{**2}$ W/m³ $u_{gas}=3$ m/s $T_{border}=1800$ K tiempo=10 min

Oxidación: $Q_{ce}=10^{**2}$ W/m³ $u_{gas}=3$ m/s $T_{sborde}=800$ K $T_{sinicial}=300$ K tiempo=10 min

```
[15]: red4=reduccion_njit(10**2,3,1800,10)
```

```
[16]: ox4=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[17]: print('Produccion total de oxígeno:',red4[3], 'moles')
      print('Produccion total de hidrógeno:',ox4[3], 'moles')
```

Produccion total de oxígeno: 0.6435200959771489 moles

Produccion total de hidrógeno: 0.7411406343418204 moles

Anexo B4

May 31, 2021

1 Modelo 1-D: Sensibilización

1.1 Constantes

```
[2]: R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases.
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción. Bulfin
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción. Bulfin
lp=0.1 # [m] Longitud del medio poroso. ***DATO CÁMARA DE REACCIÓN ***
vol=7.3*10**(-4) # [m3] Volumen de la cámara de reacción. ***DATO CÁMARA DE
    ↪ REACCIÓN
e=0.65 # [-] Porosidad del medio poroso. ***DATO PELLETS BULFIN***
dp=0.01 # [m] diámetro de partícula ***DATO PELLETS BULFIN***
cCe=(1-e)*34.55 # [kmol/m3] Concentración de cerio. Cálculo
molCeO2=(1-e)*30.6225 # =10.7179 Moles totales de CeO2.
sigma=5.67*10**(-8) # [W/m2K4] Constante de Stefan-Boltzmann.

#Cálculo moles totales CeO2
#7.22 g/cm3 * 730 cm3 / 172.115(g/mol)= 30.6225 moles de CeO2 totales

yox=1 #Fracción molar de oxidante
Aox=1.0 # Factor pre-exponencial de oxidación. Arifin
Eox=29.0 # Energía de activación de oxidación. Arifin
gamma=0.89 # [-] Orden de reacción para la oxidación. Arifin
```

1.2 Función de la Reducción

```
[3]: import numpy as np
def reduccion(Qce,u_in,Ts_border,tred): #Los inputs se eligen desde el
    ↪ apartado 'variables' ubicado mas abajo. Al seleccionar se debe marcar con #

    #Funciones entalpía de reacción
    def entalpia_red(delta): #Valores EN kJ/mol

        if delta==0:
            entalpia=430
        elif 395-31.4*np.log10(delta) > 430:
            entalpia=430
        else:
```

```

        entalpia= 395-31.4*np.log10(delta)
    return entalpia

#.....Variables.....
#Qce=1000000 #[W/m3] Aporte de calor volumétrico externo.
#u_in=1. # [m/s] Velocidad inicial del gas
#Ts_border=2100 #[K] Temperatura del solido en x=0 en la reducción.

nx=800 #[nodos] Cantidad de nodos temporales
dt=0.02 #[segundos] Intervalo de tiempo
t_reaccion=tred #[minutos] Tiempo de reacción.
#.....

#Constantes
Tg_border=300. #Temperatura del gas en x=0 en la reducción.
d_border=0.35 #Concentración de vacancias en el sólido en x=0 en la
↳reducción.
d_max=0.35 #Concentración de vacancias máxima.
dx=lp/nx #[-] Espaciamiento
molxnodo=molCeO2/nx #cantidad de moles de CeO2 por nodo.

#####
#RESOLUCION DE ECUACIONES

#condiciones iniciales
d=np.zeros(nx)
d[0]=d_border
d_final=np.zeros(nx)+0.35 #condición final de reducción (consideración de
↳reducción completa)

Ts=np.zeros(nx)+300
Ts[0]=Ts_border
Tg=np.zeros(nx)+300
Tg[0]=Tg_border

kred=np.zeros(nx) #mod eq vacancias
kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se usan
↳para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)

```

```

beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt #contador para el tiempo de reacción

    #primero calculo todas las propiedades en función de la temperatura

    #sólido
    rho_s=(-0.0002*Ts+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref. Nelson 2014
    cp_s= (0.0599*np.log(Ts)+0.0390)*1000 #[J/kgK]calor específico del sólido * ref. Nelson 2014
    lambda_s= (6.7760*10**(-2)+2.7930*10**(-4)*Ts)**(-1)#[W/mK] Coeficiente de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Ts**3.)/(9.*(1.-e)) #[W/mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

    #gas (argón)
    rho_ar=1.603*300/Tg #[kg/m3]. Gases ideales
    cp_ar=(20.7860+Tg*2.8259*10**(-7)-Tg**2*1.4642*10**(-7)+Tg**3*1.0921*10**(-8)-Tg**(-1)*3.6614*10**(-8))*1000/39.948 #[J/kgK]. ref. webbook. nist.gov. Chase 1998
    lambda_ar=(0.0442*Tg + 4.7827)/1000 #[W/Km] Conductividad térmica. Ajuste Younglove 1986.
    mu_ar= (0.0566*Tg+6.0645)*10**(-6) ##[Pa*s] viscosidad dinámica del argón. Ajuste Younglove 1986.
    nu_ar=mu_ar/rho_ar # viscosidad cinematica del argon

```

```

u_ar=1.603*u_in/rho_ar # m/s velocidad del argon (depende de la
→ densidad)

#parámetros para a_ar. ref. Toledo
ags_ar=6./(1.-e) #superficie específica interna del medio poroso
Pr_ar=(mu_ar*cp_ar)/lambda_ar #Número de Prandtl
Re_ar=(e*u_ar*dp)/mu_ar #Número de Reynolds
hgs_ar=(lambda_ar/dp)*(2.0+1.1*Pr_ar**(1/3)*Re_ar**0.6) #Coeficiente
→ convectivo interfásico
a_ar = ags_ar*hgs_ar #Coeficiente intersticial de intercambio de calor

#Cálculo de vacancias
d_n=d.copy() #d_n sería d anterior (d^n), d sería el valor actual
→ d^(n+1)

kred=(1-d_n/d_final)**(1/3)*Ared*np.exp(-Ered/(R*Ts)) #La temperatura
→ todavía no se calcula para n+1, por tanto Ts corresponde a la temperatura en
→ n.
kox=y_ox**gamma*(1-d_n/d_final)*Aox*np.exp(-Eox/(R*Ts)) Para el caso
→ de la reducción será cero en la reducción

d=(d_n/dt+d_max*kred)/(kred+kox+1/dt) #como son estáticas no requiere
→ recorrer x

#Cálculo de temperaturas
for i in range(nx):
    if i==0:
        #solido
        beta_s[i+1]=Ts_border

        #gas
        beta_g[i+1]=Tg_border

    elif i<(nx-1):
        #solido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i] + A_s[i]+B_s[i]
        F_s[i]=Ts[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + a_ar[i]*Tg[i]+Qce/
→ nx-(d[i]-d_n[i])/dt*(cCe/nx)*(10**6)*entalpia_red(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

```

```

        #gas
        A_g[i]=(lambda_ar[i-1]+lambda_ar[i])/
        ↪ (2*dx**2)+u_ar[i]*cp_ar[i]*rho_ar[i]/dx
        B_g[i]=(lambda_ar[i]+lambda_ar[i+1])/(2*dx**2)
        C_g[i]=(cp_ar[i]*rho_ar[i])/dt + a_ar[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tg[i]*(cp_ar[i]*rho_ar[i])/dt + a_ar[i]*Ts[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Ts[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tg[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Ts[j-1]=alfa_s[j]*Ts[j]+beta_s[j]
            #gas
            Tg[j-1]=alfa_g[j]*Tg[j]+beta_g[j]

        if j==0:

            break
    tiempo=nt/60 #[minutos]

    #Cálculo de producción de oxígeno

    prod_o2_nodo=molxnodo*d/2 #producción de oxígeno por nodo
    prod_o2=np.sum(prod_o2_nodo) #producción total de oxígeno
    ##ecuación de producción de o2
    return Ts,Tg,d,prod_o2

```

```

[4]: from numba import njit #Numba es una biblioteca que optimiza códigos y ahorra
    ↪ mucho tiempo de iteración.
    reduccion_njit=njit(reduccion)

```

1.3 Función de la Oxidación

```
[6]: def oxidacion(Qce,u_in,Ts_bordeo,Ts_inicial,tox,Qred,ured,Tred,tred):  
    ↪#reduccion_tuple es una tupla con los parámetros de entrada de reducción  
    def entalpia_ox(delta):  
        if delta==0:  
            entalpia=-430+285.85  
        elif 395-31.4*np.log10(delta) > 430:  
            entalpia=-430+285.85  
        else:  
            entalpia= -(395-31.4*np.log10(delta))+285.85  
        return entalpia  
  
    #.....Variables.....  
    #=1000000 #[W/m3] Aporte de calor volumétrico externo.  
    #u_in=1 #[m/s] Velocidad inicial del gas.  
    #Ts_bordeo=1200 #[K] Temperatura del sólido en x=0 en la oxidación.  
    #Ts_inicial=700 #[K] Temperatura inicial del sólido para la  
  
    nx=800 #[nodos] Cantidad de nodos espaciales.  
    dt=0.02 #[s] Intervalo de tiempo.  
    t_reaccion=tox #[min] Tiempo de reacción.  
    #.....  
  
    #Constantes  
    dx=lp/nx #[-] Espaciamiento  
    molxnodo=molCeO2/nx #cantidad de moles de CeO2 por nodo.  
  
    #Condiciones de borde de temperatura y delta.  
  
    Ts_bordeo=float(Ts_bordeo) #Temperatura del solido en x=0 en la oxidación.  
    Tg_bordeo=400. #Temperatura del gas en x=0 en la oxidación.  
    d_bordeo=0 #Concentración de vacancias en el sólido en x=0 en la oxidación.  
  
    #####  
    #RESOLUCION DE ECUACIONES  
  
    #condiciones iniciales  
    d=reduccion_njit(Qred,ured,Tred,tred)[2] #Las condiciones finales de la  
    ↪reducción se asignan aquí.  
    d_inicial=d.copy()  
    dox=d.copy()  
    dox[0]=0
```

```

Tsox=np.zeros(nx)+float(Ts_inicial)
Tsox[0]=Ts_bordeo
Tgox=np.zeros(nx)+300
Tgox[0]=Tg_bordeo

kox=np.zeros(nx) #mod eq vacancias

#se crean los arreglos de A, B, C, F, alfa y beta vacías, los cuales se
→usan para aplicar TDMA.
A_s=np.zeros(nx)
B_s=np.zeros(nx)
C_s=np.zeros(nx)
F_s=np.zeros(nx)
alfa_s=np.zeros(nx)
beta_s=np.zeros(nx)

A_g=np.zeros(nx)
B_g=np.zeros(nx)
C_g=np.zeros(nx)
F_g=np.zeros(nx)
alfa_g=np.zeros(nx)
beta_g=np.zeros(nx)

#condición final de reducción
dox_final=np.zeros(nx) #Asumimos que todo el material se oxidará

tiempo=0 #[min] tiempo de reacción
nt=0

while tiempo < t_reaccion:

    nt+=dt

    #sólido
    rho_s=(-0.0002*Tsox+6.8561)*1000 #7220 #[kg/m3]densidad del sólido *ref.
→Nelson 2014
    cp_s= (0.0599*np.log(Tsox)+0.039)*1000 #[J/kgK]calor específico del
→sólido * ref. Nelson 2014
    lambda_s= (6.776*10**(-2)+2.793*10**(-4)*Tsox)**(-1)#[W/mK] Coeficiente
→de transferencia de calor del sólido. ref. Nelson 2014
    lambda_ef= (1.-e)*lambda_s + (32.*sigma*dp*e*Tsox**3.)/(9.*(1.-e)) #[W/
→mK] Coeficiente de transferencia de calor efectivo. ref. Toledo

```

```

#gas
#nota: La temperatura del vapor varía muy poco, por ende se
→considerarán propiedades constantes.

rho_h2o=np.zeros(nx)+1/1.673 #kg/m3 #1/v volumen específico
cp_h2o=np.zeros(nx)+2.01*10**3 #j/kgK calor específico
lambda_h2o=np.zeros(nx)+24.8*10**(-3) #W/mK conductividad térmica del
→vapor
mu_h2o= np.zeros(nx)+12.0*10**(-6)#Pa*s o kg/ms Viscosidad dinámica del
→vapor
nu_h2o=mu_h2o/rho_h2o #m**2/s viscosidad cinemática
u_h2o=np.zeros(nx)+u_in

#parámetros para a_ar. ref. Toledo
ags_h2o=6./(1.-e) #superficie específica interna del medio poroso
Pr_h2o=(mu_h2o*cp_h2o)/lambda_h2o #Número de Prandtl
Re_h2o=(e*u_h2o*dp)/mu_h2o #Número de Reynolds
hgs_h2o=(lambda_h2o/dp)*(2.0+1.1*Pr_h2o**(1/3)*Re_h2o**0.6)
→#Coeficiente convectivo interfásico
a_h2o = ags_h2o*hgs_h2o #Coeficiente intersticial de intercambio de
→calor

#Cálculo de vacancias
dox_n=dox.copy() #d_n sería d anterior (d~n), d sería el valor actual
→d^(n+1)
kox=yox**gamma*(1-(d_inicial-dox_n)/(d_inicial))*Aox*np.exp(-Eox/
→(R*Tsox))
dox=(dox_n/dt)/(kox+1/dt) #como son estáticas no requiere recorrer x

#Cálculo de temperaturas
for i in range (nx):
    if i==0:
        #solido
        beta_s[i+1]=Ts_bordeo

        #gas
        beta_g[i+1]=Tg_bordeo

    elif i<(nx-1):
        #solido
        A_s[i]=(lambda_ef[i-1]+lambda_ef[i])/(2*dx**2)
        B_s[i]=(lambda_ef[i]+lambda_ef[i+1])/(2*dx**2)
        C_s[i]=(1-e)*(cp_s[i]*rho_s[i])/dt + a_h2o[i] + A_s[i]+B_s[i]

```

```

        F_s[i]=Tsox[i]*(1-e)*(cp_s[i]*rho_s[i])/dt + u
↪ a_h2o[i]*Tgox[i]+Qce/nx+(dox[i]-dox_n[i])/dt*(cCe/
↪ nx)*(10**6)*entalpia_ox(d[i])

        alfa_s[i+1]=B_s[i]/(C_s[i]-A_s[i]*alfa_s[i])
        beta_s[i+1]=(F_s[i]+beta_s[i]*A_s[i])/(C_s[i]-A_s[i]*alfa_s[i])

        #gas
        A_g[i]=(lambda_h2o[i-1]+lambda_h2o[i])/
↪ (2*dx**2)+u_h2o[i]*cp_h2o[i]*rho_h2o[i]/dx
        B_g[i]=(lambda_h2o[i]+lambda_h2o[i+1])/(2*dx**2)
        C_g[i]=(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]/e + A_g[i]+B_g[i]
        F_g[i]=Tgox[i]*(cp_h2o[i]*rho_h2o[i])/dt + a_h2o[i]*Tsox[i]/e

        alfa_g[i+1]=B_g[i]/(C_g[i]-A_g[i]*alfa_g[i])
        beta_g[i+1]=(F_g[i]+beta_g[i]*A_g[i])/(C_g[i]-A_g[i]*alfa_g[i])

    else:
        #solido
        Tsox[i]=beta_s[i]/(1-alfa_s[i])

        #gas
        Tgox[i]=beta_g[i]/(1-alfa_g[i])

    for j in range(nx-1,-1,-1):
        if j!=0:
            #solido
            Tsox[j-1]=alfa_s[j]*Tsox[j]+beta_s[j]
            #gas
            Tgox[j-1]=alfa_g[j]*Tgox[j]+beta_g[j]

        if j==0:

            break
    tiempo=nt/60

    #Cálculo de producción de oxígeno
    prod_h2_nodo=molxnodo*(d-dox) #producción de hidrógeno por nodo
    prod_h2=np.sum(prod_h2_nodo) #producción total de hidrógeno
    return Tsox,Tgox,dox,prod_h2

```

```
[7]: oxidacion_njit=njit(oxidacion)
```

1.4 Sensibilización: Base caso desfavorable

Reducción: $Q_{ce}=10^{**6}$ W/m³ $u_{gas}=1$ m/s $T_{border}=2200$ K tiempo=10 min

Oxidación: $Q_{ce}=10^{**2}$ W/m³ $u_{gas}=3$ m/s $T_{borde}=800$ K $T_{inicial}=300$ K tiempo=10 min

Solo se calcularán las cantidades de H₂ producido.

```
[8]: #Producción considerando el caso desfavorable:
base=oxidacion_njit(1*10**2,3,800,300,10,1*10**2,3,1800,10)[3]
```

```
[9]: def porcent(prod):
      r=(prod-base)/base*100
      return r
```

1.4.1 Aporte de calor volumétrico (Q_{ce})

Ambas etapas

```
[10]: #def oxidacion(Qce,u_in,Ts_bordeo,Ts_inicial,tox,Qred,ured,Tred,tred)
qce_prod1=oxidacion_njit(1*10**9,3,800,300,10,1*10**9,3,1800,10)
qce_prod2=oxidacion_njit(5*10**8,3,800,300,10,5*10**8,3,1800,10)
qce_prod3=oxidacion_njit(1*10**8,3,800,300,10,1*10**8,3,1800,10)
qce_prod4=oxidacion_njit(5*10**7,3,800,300,10,5*10**7,3,1800,10)
qce_prod5=oxidacion_njit(1*10**7,3,800,300,10,1*10**7,3,1800,10)
qce_prod6=oxidacion_njit(5*10**6,3,800,300,10,5*10**6,3,1800,10)
qce_prod7=oxidacion_njit(1*10**6,3,800,300,10,1*10**6,3,1800,10)
qce_prod8=oxidacion_njit(5*10**5,3,800,300,10,5*10**5,3,1800,10)
qce_prod9=oxidacion_njit(1*10**5,3,800,300,10,1*10**5,3,1800,10)
qce_prod10=oxidacion_njit(1*10**2,3,800,300,10,1*10**2,3,1800,10)
```

```
[11]: print('Caso 1: ' , qce_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod1[3]),'%')
print('Caso 2: ' , qce_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod2[3]),'%')
print('Caso 3: ' , qce_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod3[3]),'%')
print('Caso 4: ' , qce_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod4[3]),'%')
print('Caso 5: ' , qce_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod5[3]),'%')
print('Caso 6: ' , qce_prod6[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod6[3]),'%')
print('Caso 7: ' , qce_prod7[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod7[3]),'%')
print('Caso 8: ' , qce_prod8[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod8[3]),'%')
print('Caso 9: ' , qce_prod9[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(qce_prod9[3]),'%')
```

```
print('Caso 10: ' , qce_prod10[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qce_prod10[3]),'%')
```

Caso 1: 1.426156188811162 moles de hidrógeno. Se tiene un incremento de 92.42720244015207 %
 Caso 2: 1.0228799005706388 moles de hidrógeno. Se tiene un incremento de 38.014278690713084 %
 Caso 3: 0.7908051047640906 moles de hidrógeno. Se tiene un incremento de 6.701085883163775 %
 Caso 4: 0.7655660576275878 moles de hidrógeno. Se tiene un incremento de 3.2956529643606265 %
 Caso 5: 0.7459605320352195 moles de hidrógeno. Se tiene un incremento de 0.650335101067485 %
 Caso 6: 0.7435464851844742 moles de hidrógeno. Se tiene un incremento de 0.32461461849144385 %
 Caso 7: 0.741621114175023 moles de hidrógeno. Se tiene un incremento de 0.0648297787138924 %
 Caso 8: 0.7413808094937868 moles de hidrógeno. Se tiene un incremento de 0.032406150848776995 %
 Caso 9: 0.7411886244235343 moles de hidrógeno. Se tiene un incremento de 0.006475165372152867 %
 Caso 10: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Reducción

```
[17]: #def oxidacion(Qce,u_in,Ts_bordeo,Ts_inicial,Qred,ured,Tred):
qcer_prod1=oxidacion_njit(1*10**2,3,800,300,10,1*10**9,3,1800,10)
qcer_prod2=oxidacion_njit(1*10**2,3,800,300,10,5*10**8,3,1800,10)
qcer_prod3=oxidacion_njit(1*10**2,3,800,300,10,1*10**8,3,1800,10)
qcer_prod4=oxidacion_njit(1*10**2,3,800,300,10,5*10**7,3,1800,10)
qcer_prod5=oxidacion_njit(1*10**2,3,800,300,10,1*10**7,3,1800,10)
qcer_prod6=oxidacion_njit(1*10**2,3,800,300,10,5*10**6,3,1800,10)
qcer_prod7=oxidacion_njit(1*10**2,3,800,300,10,1*10**6,3,1800,10)
qcer_prod8=oxidacion_njit(1*10**2,3,800,300,10,5*10**5,3,1800,10)
qcer_prod9=oxidacion_njit(1*10**2,3,800,300,10,1*10**5,3,1800,10)
qcer_prod10=oxidacion_njit(1*10**2,3,800,300,10,1*10**2,3,1800,10)
```

```
[18]: print('Caso 1: ' , qcer_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod1[3]),'%')
print('Caso 2: ' , qcer_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod2[3]),'%')
print('Caso 3: ' , qcer_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod3[3]),'%')
print('Caso 4: ' , qcer_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod4[3]),'%')
print('Caso 5: ' , qcer_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod5[3]),'%')
```

```

print('Caso 6: ' , qcer_prod6[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod6[3]),'%')
print('Caso 7: ' , qcer_prod7[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod7[3]),'%')
print('Caso 8: ' , qcer_prod8[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod8[3]),'%')
print('Caso 9: ' , qcer_prod9[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod9[3]),'%')
print('Caso 10: ' , qcer_prod10[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qcer_prod10[3]),'%')

```

Caso 1: 0.8375125722325913 moles de hidrógeno. Se tiene un incremento de 13.003191759463476 %

Caso 2: 0.784281239192641 moles de hidrógeno. Se tiene un incremento de 5.820839237769251 %

Caso 3: 0.7492522051790446 moles de hidrógeno. Se tiene un incremento de 1.0944712057823844 %

Caso 4: 0.7451696473604646 moles de hidrógeno. Se tiene un incremento de 0.5436232790315485 %

Caso 5: 0.7419422882618432 moles de hidrógeno. Se tiene un incremento de 0.10816488570143737 %

Caso 6: 0.7415412026258085 moles de hidrógeno. Se tiene un incremento de 0.05404754042986393 %

Caso 7: 0.7412207009783977 moles de hidrógeno. Se tiene un incremento de 0.010803163781234224 %

Caso 8: 0.7411806611224269 moles de hidrógeno. Se tiene un incremento de 0.005400699779736873 %

Caso 9: 0.741148632887401 moles de hidrógeno. Se tiene un incremento de 0.0010792210290341815 %

Caso 10: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Oxidación

```

[19]: qceo_prod1=oxidacion_njit(1*10**9,3,800,300,10,1*10**2,3,1800,10)
qceo_prod2=oxidacion_njit(5*10**8,3,800,300,10,1*10**2,3,1800,10)
qceo_prod3=oxidacion_njit(1*10**8,3,800,300,10,1*10**2,3,1800,10)
qceo_prod4=oxidacion_njit(5*10**7,3,800,300,10,1*10**2,3,1800,10)
qceo_prod5=oxidacion_njit(1*10**7,3,800,300,10,1*10**2,3,1800,10)
qceo_prod6=oxidacion_njit(5*10**6,3,800,300,10,1*10**2,3,1800,10)
qceo_prod7=oxidacion_njit(1*10**6,3,800,300,10,1*10**2,3,1800,10)
qceo_prod8=oxidacion_njit(5*10**5,3,800,300,10,1*10**2,3,1800,10)
qceo_prod9=oxidacion_njit(1*10**5,3,800,300,10,1*10**2,3,1800,10)
qceo_prod10=oxidacion_njit(1*10**2,3,800,300,10,1*10**2,3,1800,10)

```

```

[20]: print('Caso 1: ' , qceo_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
↪incremento de ',porcent(qceo_prod1[3]),'%')

```

```

print('Caso 2: ' , qceo_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod2[3]),'%')
print('Caso 3: ' , qceo_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod3[3]),'%')
print('Caso 4: ' , qceo_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod4[3]),'%')
print('Caso 5: ' , qceo_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod5[3]),'%')
print('Caso 6: ' , qceo_prod6[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod6[3]),'%')
print('Caso 7: ' , qceo_prod7[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod7[3]),'%')
print('Caso 8: ' , qceo_prod8[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod8[3]),'%')
print('Caso 9: ' , qceo_prod9[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod9[3]),'%')
print('Caso 10: ' , qceo_prod10[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(qceo_prod10[3]),'%')

```

Caso 1: 1.0380963985023381 moles de hidrógeno. Se tiene un incremento de 40.06739752223047 %
 Caso 2: 0.9208281220830642 moles de hidrógeno. Se tiene un incremento de 24.244722177568576 %
 Caso 3: 0.7807721888017558 moles de hidrógeno. Se tiene un incremento de 5.347373038739226 %
 Caso 4: 0.7610725042313876 moles de hidrógeno. Se tiene un incremento de 2.689350572077045 %
 Caso 5: 0.7451408142456025 moles de hidrógeno. Se tiene un incremento de 0.5397329087662976 %
 Caso 6: 0.7431414173590593 moles de hidrógeno. Se tiene un incremento de 0.26995996772132846 %
 Caso 7: 0.7415408681092219 moles de hidrógeno. Se tiene un incremento de 0.054002405057298745 %
 Caso 8: 0.7413407378812252 moles de hidrógeno. Se tiene un incremento de 0.026999402020712326 %
 Caso 9: 0.7411806240880675 moles de hidrógeno. Se tiene un incremento de 0.005395702838854005 %
 Caso 10: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

1.4.2 Temperatura de borde

Ambas etapas

```

[13]: tb_prod1=oxidacion_njit(10**2,3,1200,300,10,10**2,3,2200,10)
      tb_prod2=oxidacion_njit(10**2,3,1100,300,10,10**2,3,2100,10)
      tb_prod3=oxidacion_njit(10**2,3,1000,300,10,10**2,3,2000,10)
      tb_prod4=oxidacion_njit(10**2,3,900,300,10,10**2,3,1900,10)

```

```
tb_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[27]: print('Caso 1: ' , tb_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪de ',porcent(tb_prod1[3]), '%')
print('Caso 2: ' , tb_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪de ',porcent(tb_prod2[3]), '%')
print('Caso 3: ' , tb_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪de ',porcent(tb_prod3[3]), '%')
print('Caso 4: ' , tb_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪de ',porcent(tb_prod4[3]), '%')
print('Caso 5: ' , tb_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪de ',porcent(tb_prod5[3]), '%')
```

Caso 1: 1.8045755718999006 moles de hidrógeno. Se tiene un incremento de 143.48625460301164 %

Caso 2: 1.4978329766535017 moles de hidrógeno. Se tiene un incremento de 102.09834776953927 %

Caso 3: 1.2304350568324232 moles de hidrógeno. Se tiene un incremento de 66.0191062017706 %

Caso 4: 0.9826796917791244 moles de hidrógeno. Se tiene un incremento de 32.5901787387229 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Reducción

```
[29]: tbr_prod1=oxidacion_njit(10**2,3,800,300,10,10**2,3,2200,10)
tbr_prod2=oxidacion_njit(10**2,3,800,300,10,10**2,3,2100,10)
tbr_prod3=oxidacion_njit(10**2,3,800,300,10,10**2,3,2000,10)
tbr_prod4=oxidacion_njit(10**2,3,800,300,10,10**2,3,1900,10)
tbr_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[50]: print('Caso 1: ' , tbr_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(tbr_prod1[3]), '%', '3ER')
print('Caso 2: ' , tbr_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(tbr_prod2[3]), '%')
print('Caso 3: ' , tbr_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(tbr_prod3[3]), '%')
print('Caso 4: ' , tbr_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(tbr_prod4[3]), '%')
print('Caso 5: ' , tbr_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪incremento de ',porcent(tbr_prod5[3]), '%')
```

Caso 1: 0.9756210605253199 moles de hidrógeno. Se tiene un incremento de 31.637777679229924 % 3ER

Caso 2: 0.9192017870985753 moles de hidrógeno. Se tiene un incremento de 24.025285418992624 %

Caso 3: 0.8705109803084967 moles de hidrógeno. Se tiene un incremento de 17.455573203264628 %

Caso 4: 0.8163297963698938 moles de hidrógeno. Se tiene un incremento de

10.145059998612282 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Oxidación

```
[31]: tbo_prod1=oxidacion_njit(10**2,3,1200,300,10,10**2,3,1800,10)
tbo_prod2=oxidacion_njit(10**2,3,1100,300,10,10**2,3,1800,10)
tbo_prod3=oxidacion_njit(10**2,3,1000,300,10,10**2,3,1800,10)
tbo_prod4=oxidacion_njit(10**2,3,900,300,10,10**2,3,1800,10)
tbo_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[48]: print('Caso 1: ' , tbo_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tbo_prod1[3]),'%', '1ER')
print('Caso 2: ' , tbo_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tbo_prod2[3]),'%')
print('Caso 3: ' , tbo_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tbo_prod3[3]),'%')
print('Caso 4: ' , tbo_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tbo_prod4[3]),'%')
print('Caso 5: ' , tbo_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tbo_prod5[3]),'%')
```

Caso 1: 1.1462352581671034 moles de hidrógeno. Se tiene un incremento de 54.65826660348107 % 1ER

Caso 2: 1.0814263878262074 moles de hidrógeno. Se tiene un incremento de 45.91378986885291 %

Caso 3: 0.9927253835988035 moles de hidrógeno. Se tiene un incremento de 33.945615393279056 %

Caso 4: 0.8790209247507035 moles de hidrógeno. Se tiene un incremento de 18.603795827674386 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

1.4.3 Velocidad del gas

Ambas etapas

```
[34]: u_prod1=oxidacion_njit(10**2,1,800,300,10,10**2,1,1800,10)
u_prod2=oxidacion_njit(10**2,1.5,800,300,10,10**2,1.5,1800,10)
u_prod3=oxidacion_njit(10**2,2,800,300,10,10**2,2,1800,10)
u_prod4=oxidacion_njit(10**2,2.5,800,300,10,10**2,2.5,1800,10)
u_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[35]: print('Caso 1: ' , u_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪ de ',porcent(u_prod1[3]),'%')
print('Caso 2: ' , u_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪ de ',porcent(u_prod2[3]),'%')
print('Caso 3: ' , u_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪ de ',porcent(u_prod3[3]),'%')
print('Caso 4: ' , u_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
      ↪ de ',porcent(u_prod4[3]),'%')
```

```
print('Caso 5: ' , u_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(u_prod5[3]),'%')
```

Caso 1: 0.8542648268215864 moles de hidrógeno. Se tiene un incremento de 15.26352587322748 %
 Caso 2: 0.8170302151328686 moles de hidrógeno. Se tiene un incremento de 10.239565512211168 %
 Caso 3: 0.7872972897510617 moles de hidrógeno. Se tiene un incremento de 6.227786370157847 %
 Caso 4: 0.76246642820671 moles de hidrógeno. Se tiene un incremento de 2.8774287735320545 %
 Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Reducción

```
[12]: ur_prod1=oxidacion_njit(10**2,3,800,300,10,10**2,1,1800,10)
ur_prod2=oxidacion_njit(10**2,3,800,300,10,10**2,1.5,1800,10)
ur_prod3=oxidacion_njit(10**2,3,800,300,10,10**2,2,1800,10)
ur_prod4=oxidacion_njit(10**2,3,800,300,10,10**2,2.5,1800,10)
ur_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[11]: print('Caso 1: ' , ur_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ur_prod1[3]),'%')
print('Caso 2: ' , ur_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ur_prod2[3]),'%')
print('Caso 3: ' , ur_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ur_prod3[3]),'%')
print('Caso 4: ' , ur_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ur_prod4[3]),'%')
print('Caso 5: ' , ur_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ur_prod5[3]),'%')
```

Caso 1: 0.8757731681172327 moles de hidrógeno. Se tiene un incremento de 18.16558525292227 %
 Caso 2: 0.7744802889539656 moles de hidrógeno. Se tiene un incremento de 4.498424869357335 %
 Caso 3: 0.7617621292223669 moles de hidrógeno. Se tiene un incremento de 2.7823997126887554 %
 Caso 4: 0.7508259997533435 moles de hidrógeno. Se tiene un incremento de 1.30681883609368 %
 Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Oxidación

```
[38]: uo_prod1=oxidacion_njit(10**2,1,800,300,10,10**2,3,1800,10)
uo_prod2=oxidacion_njit(10**2,1.5,800,300,10,10**2,3,1800,10)
uo_prod3=oxidacion_njit(10**2,2,800,300,10,10**2,3,1800,10)
uo_prod4=oxidacion_njit(10**2,2.5,800,300,10,10**2,3,1800,10)
uo_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[39]: print('Caso 1: ' , uo_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(uo_prod1[3]),'%')
print('Caso 2: ' , uo_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(uo_prod2[3]),'%')
print('Caso 3: ' , uo_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(uo_prod3[3]),'%')
print('Caso 4: ' , uo_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(uo_prod4[3]),'%')
print('Caso 5: ' , uo_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(uo_prod5[3]),'%')
```

Caso 1: 0.7961198458307156 moles de hidrógeno. Se tiene un incremento de 7.418188794589596 %
 Caso 2: 0.7793134529350599 moles de hidrógeno. Se tiene un incremento de 5.150549952930239 %
 Caso 3: 0.7649952199690732 moles de hidrógeno. Se tiene un incremento de 3.2186314610097058 %
 Caso 4: 0.7524118376296595 moles de hidrógeno. Se tiene un incremento de 1.5207914349276743 %
 Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

1.4.4 Temperatura inicial de oxidación

```
[40]: ti_prod1=oxidacion_njit(10**2,3,800,700,10,10**2,3,1800,10)
ti_prod2=oxidacion_njit(10**2,3,800,600,10,10**2,3,1800,10)
ti_prod3=oxidacion_njit(10**2,3,800,500,10,10**2,3,1800,10)
ti_prod4=oxidacion_njit(10**2,3,800,400,10,10**2,3,1800,10)
ti_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[51]: print('Caso 1: ' , ti_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ti_prod1[3]),%', '4T0')
print('Caso 2: ' , ti_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ti_prod2[3]),'%')
print('Caso 3: ' , ti_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ti_prod3[3]),'%')
print('Caso 4: ' , ti_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ti_prod4[3]),'%')
print('Caso 5: ' , ti_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(ti_prod5[3]),'%')
```

Caso 1: 0.9408676937771343 moles de hidrógeno. Se tiene un incremento de 26.948604648115683 % 4T0
 Caso 2: 0.8831899464035025 moles de hidrógeno. Se tiene un incremento de 19.166310073908008 %
 Caso 3: 0.8266421171987948 moles de hidrógeno. Se tiene un incremento de 11.536472147813761 %
 Caso 4: 0.7785495358368107 moles de hidrógeno. Se tiene un incremento de 5.047476789369643 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

1.4.5 Tiempo de reacción

Reducción

```
[42]: trr_prod1=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,50)
trr_prod2=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,40)
trr_prod3=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,30)
trr_prod4=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,20)
trr_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[52]: print('Caso 1: ' , trr_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(trr_prod1[3]),'%','5T0')
print('Caso 2: ' , trr_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(trr_prod2[3]),'%')
print('Caso 3: ' , trr_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(trr_prod3[3]),'%')
print('Caso 4: ' , trr_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(trr_prod4[3]),'%')
print('Caso 5: ' , trr_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(trr_prod5[3]),'%')
```

Caso 1: 0.8747465620336865 moles de hidrógeno. Se tiene un incremento de 18.02706821094981 % 5T0

Caso 2: 0.8644690690665221 moles de hidrógeno. Se tiene un incremento de 16.64035528617657 %

Caso 3: 0.8484152574448596 moles de hidrógeno. Se tiene un incremento de 14.474260097519249 %

Caso 4: 0.8180560074750318 moles de hidrógeno. Se tiene un incremento de 10.377972758370898 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Oxidación

```
[44]: tro_prod1=oxidacion_njit(10**2,3,800,300,50,10**2,3,1800,10)
tro_prod2=oxidacion_njit(10**2,3,800,300,40,10**2,3,1800,10)
tro_prod3=oxidacion_njit(10**2,3,800,300,30,10**2,3,1800,10)
tro_prod4=oxidacion_njit(10**2,3,800,300,20,10**2,3,1800,10)
tro_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[49]: print('Caso 1: ' , tro_prod1[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tro_prod1[3]),'%','2D0')
print('Caso 2: ' , tro_prod2[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tro_prod2[3]),'%')
print('Caso 3: ' , tro_prod3[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tro_prod3[3]),'%')
print('Caso 4: ' , tro_prod4[3], 'moles de hidrógeno.' , 'Se tiene un_
      ↪ incremento de ',porcent(tro_prod4[3]),'%')
```

```
print('Caso 5: ' , tro_prod5[3], 'moles de hidrógeno.' , 'Se tiene un_
↳incremento de ',porcent(tro_prod5[3]),'%')
```

Caso 1: 1.1375680270467687 moles de hidrógeno. Se tiene un incremento de 53.48882173448779 % 2D0

Caso 2: 1.104817093954971 moles de hidrógeno. Se tiene un incremento de 49.069831387145335 %

Caso 3: 1.0533718400920051 moles de hidrógeno. Se tiene un incremento de 42.1284694540417 %

Caso 4: 0.9603465856542029 moles de hidrógeno. Se tiene un incremento de 29.576836184005916 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Ambas etapas

```
[46]: tr_prod1=oxidacion_njit(10**2,3,800,300,50,10**2,3,1800,50)
tr_prod2=oxidacion_njit(10**2,3,800,300,40,10**2,3,1800,40)
tr_prod3=oxidacion_njit(10**2,3,800,300,30,10**2,3,1800,30)
tr_prod4=oxidacion_njit(10**2,3,800,300,20,10**2,3,1800,20)
tr_prod5=oxidacion_njit(10**2,3,800,300,10,10**2,3,1800,10)
```

```
[47]: print('Caso 1: ' , tr_prod1[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(tr_prod1[3]),'%')
print('Caso 2: ' , tr_prod2[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(tr_prod2[3]),'%')
print('Caso 3: ' , tr_prod3[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(tr_prod3[3]),'%')
print('Caso 4: ' , tr_prod4[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(tr_prod4[3]),'%')
print('Caso 5: ' , tr_prod5[3], 'moles de hidrógeno.' , 'Se tiene un incremento_
↳de ',porcent(tr_prod5[3]),'%')
```

Caso 1: 1.626764666529521 moles de hidrógeno. Se tiene un incremento de 119.4947343528385 %

Caso 2: 1.5070087126404712 moles de hidrógeno. Se tiene un incremento de 103.33640375538036 %

Caso 3: 1.3472684585762689 moles de hidrógeno. Se tiene un incremento de 81.78310514207983 %

Caso 4: 1.1159125244195431 moles de hidrógeno. Se tiene un incremento de 50.56690629445028 %

Caso 5: 0.7411406343418204 moles de hidrógeno. Se tiene un incremento de 0.0 %

Anexo B5

May 31, 2021

1 Modelo de Comparación: Cinética Bulfin+Arifin

1.1 Etapa de Reducción

```
[1]: import numpy as np
dT=17.
Tred=1500.+273.15 #temperatura a la que se fijan los reactantes
Ts=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15 #Este arreglo
    ↳ contiene las distintas Tox que ocupa Arifin
time=240.0 #Segundos que se mantiene a 240°C una vez que alcanza la temperatura
    ↳ de reducción
timer=np.zeros(len(Ts))
dt=0.02
d=np.zeros(len(Ts))
d_max=0.35
R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción
tiempo_total=np.zeros(len(Ts))
y_ox=0 #Para la reducción este valor es cero. En el trabajo de Arifin se señala
    ↳ que la presion parcial de O2 es inferior a 10ppm.
Aox=1.0
Eox=29.0
gamma=0.89
kred=np.zeros(len(Ts))
kox=np.zeros(len(Ts))
print_counter=0 #

#d_final=np.array([0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35]) #para la
    ↳ iteración de f(a) se consideró como
#condición inicial de iteración la reducción completa.

#Luego de 10 iteraciones se obtuvo una distribución de vacancias de
#D [0.34933276 0.34933308 0.3493334 0.34933371 0.34933402 0.3493343 0.34933472]
```

```

d_final=np.array([0.34933276, 0.34933308, 0.3493334, 0.34933371, 0.34933402, 0.
↪3493343, 0.34933472])
#este d_final es obtenido mediante proceso iterativo.

#----Al ejecutar este código se obtendrá el resultado de la iteración 11 de
↪f(a) considerando la reducción completa como condicion inicial

for i in range(len(Ts)):
    tt=0
    while timer[i] < time:
        d_n=d.copy()
        tt+=dt

        if Ts[i] < Tred:
            Ts[i]+=dT*dt

        if Ts[i] >= Tred:

            timer[i]+=dt

        kred[i]=(1-d_n[i]/d_final[i])** (1/3)*Ared*np.exp(-Ered/(R*Ts[i]))
        kox[i]=0 #y_ox=0 implica que kox=0

        d[i]=(d_n[i]/dt+d_max*kred[i])/(kred[i]+kox[i]+1/dt)

        tiempo_total[i]=tt

```

```

[2]: print('d_anterior: ',d_final)
      print('d_actual: ',d)
      print('diferencia: ',d_final-d)
      #iteracion 1

```

```

d_anterior: [0.34933276 0.34933308 0.3493334 0.34933371 0.34933402 0.3493343
0.34933472]
d_actual: [0.34933016 0.34933048 0.3493308 0.34933111 0.34933143 0.34933171
0.34933213]
diferencia: [2.59906523e-06 2.59814024e-06 2.59743308e-06 2.59543545e-06
2.59487111e-06 2.59213760e-06 2.59178144e-06]

```

```

[3]: umol_o2=(d/2)/172.115*10**6
      print('MODELO NUMÉRICO - micro-moles de O2 por g de CeO2:',umol_o2)

```

MODELO NUMÉRICO - micro-moles de O2 por g de CeO2: [1014.81614309 1014.81707538
1014.81800705 1014.81891341 1014.81981561
1014.82063696 1014.82185811]

1.1.1 Cálculos según estequiometría de la reducción

```
[4]: print('Considerando una reacción estequiométrica:')
print('-----')

umol_o2=(d/2)/172.115*10**6
print('MODELO NUMÉRICO - micro-moles de O2 por g de CeO2:',umol_o2)
umol_o2_arifin=0.056*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+28.
    ↪99
print('RESULTADOS DE ARIFIN - micro-moles de O2 por g de CeO2:' ,umol_o2_arifin_
    ↪)
#print('Producción total de O2 en micro moles con 0,2g de CeO2', (d/2)/172.
    ↪115*10**6*0.2)
d_arifin=(0.056*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+28.
    ↪99)*172.115*2*10**-6
```

Considerando una reacción estequiométrica:

MODELO NUMÉRICO - micro-moles de O2 por g de CeO2: [1014.81614309 1014.81707538
1014.81800705 1014.81891341 1014.81981561
1014.82063696 1014.82185811]

RESULTADOS DE ARIFIN - micro-moles de O2 por g de CeO2: [70.99 73.79 76.59 79.39
82.19 84.99 90.59]

1.2 Etapa de Oxidación

```
[5]: d_final=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]) #Asumimos inicialmente
    ↪condición final como oxidación completa de la ceria.
    #d_final se obtiene mediante iteración

#Luego de 1100 iteraciones se obtiene un d_final de [0.19972006 0.18600724 0.
    ↪17338739 0.16185165 0.1513547 0.14182989 0.12538496]
d_final=np.array([0.19972006, 0.18600724, 0.17338739, 0.16185165, 0.1513547, 0.
    ↪14182989, 0.12538496])
#El dox obtenido será la iteración 1101 de f(a).

dox=d.copy() #'d' corresponde a la distribución de vacancias final de la etapa
    ↪de reducción
#Luego de correr el programa, dox será la distribución de vacancias final de la
    ↪etapa de oxidación.
```

```

#La oxidación es isotérmica en el experimento de arifin:
Tox=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15

#d_inicial es fijo, son los valores que se obtuvieron al final de la reducción
d_inicial=d.copy()

for i in range(len(Tox)):
    tt=0
    while tt < 600:
        d_n=dox.copy()
        kox[i]=0.3**gamma*(1-(d_inicial[i]-d_n[i])/
→(d_inicial[i]-d_final[i]))*Aox*np.exp(-Eox/(R*Tox[i]))

        if dox[i]>=0.35:
            break
        dox[i]=(d_n[i]/dt)/(1/dt+kox[i])

        tt+=dt
print('dox:',dox)
print('d_final',d_final)
print('diferencia entre dox y d_final;',dox-d_final)

```

```

dox: [0.19972986 0.1860172  0.17339739 0.1618616  0.15136452 0.14183954
      0.12539417]
d_final [0.19972006 0.18600724 0.17338739 0.16185165 0.1513547  0.14182989
      0.12538496]
diferencia entre dox y d_final; [9.79597246e-06 9.95965358e-06 1.00013409e-05
      9.94809173e-06
      9.82381484e-06 9.64852690e-06 9.20585035e-06]

```

1.3 Resultados segun estequiometría

```

[6]: print('Considerando una reacción estequiométrica:')
      print('-----')

      print('Moles de hidrógeno producidos:')
      umol_h2=(d-dox)/172.115*10**6
      print('MODELO NUMÉRICO - micro-moles de H2 por g de CeO2:',umol_h2)

```

```

umol_h2_arifin=0.059*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+56.
↪45
print('RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2:', umol_h2_arifin)

print('Diferencia de resultados:', umol_h2-umol_h2_arifin)

```

Considerando una reacción estequiométrica:

```

-----
Moles de hidrógeno producidos:
MODELO NUMÉRICO - micro-moles de H2 por g de CeO2: [ 869.18807171  948.86141363
1022.18523212 1089.21079786 1150.20132652
1205.54378953 1301.09497933]
RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2: [100.7  103.65 106.6
109.55 112.5  115.45 121.35]
Diferencia de resultados: [ 768.48807171  845.21141363  915.58523212
979.66079786 1037.70132652
1090.09378953 1179.74497933]

```

```

[7]: #Razon de moles de H2 sobre moles de O2 producidos
print('Razon de moles de H2 sobre moles de O2 producidos:')
print('-----')
#Arifin:
razon_arifin=(umol_h2_arifin)/(umol_o2_arifin)
print('moles de H2/moles de O2 segun arifin:', razon_arifin)

razon_modelo=umol_h2/umol_o2
print('moles de H2/moles de O2 segun el modelo numérico:', razon_modelo)

```

Razon de moles de H2 sobre moles de O2 producidos:

```

-----
moles de H2/moles de O2 segun arifin: [1.41850965 1.40466188 1.39182661
1.37989671 1.36877966 1.35839511
1.33955183]
moles de H2/moles de O2 segun el modelo numérico: [0.85649807 0.93500734
1.00725965 1.07330558 1.13340448 1.18793779
1.28209199]

```

1.4 Función para ajuste lineal

```

[8]: import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200
def b1(x,y):
    t1=x-np.average(x)
    t2=y-np.average(y)
    sxy=sum(t1*t2)
    sxx=sum(t1*t1)
    return sxy/sxx
def b0(x,y):

```

```

    return np.average(y)-b1(x,y)*np.average(x)
def reg_lin(x,y):

    return b0(x,y)+b1(x,y)*x

```

1.5 Modelo vs Experimento de Arifin: oxígeno producido

```

[9]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)

puntosy1=b0(T_ox,umol_o2)+b1(T_ox,umol_o2)*puntosx
puntosy2=b0(T_ox,umol_o2_arifin)+b1(T_ox,umol_o2_arifin)*puntosx

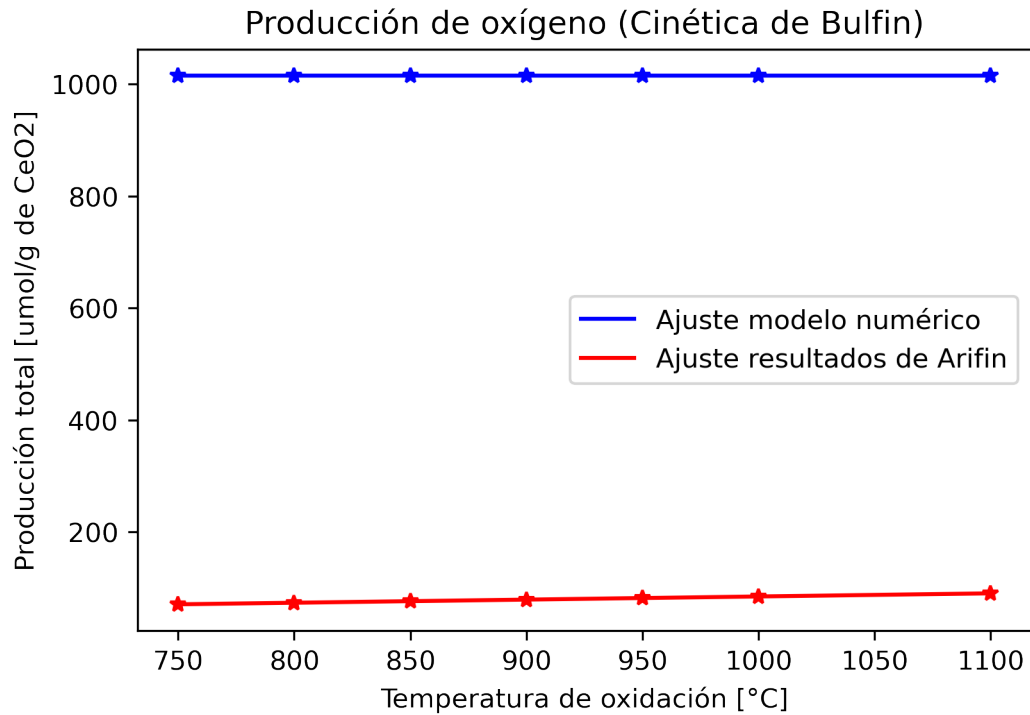
plt.plot(puntosx,puntosy1,'b')#rectas
plt.plot(puntosx,puntosy2,'r') #rectas

plt.plot(T_ox,umol_o2,'b*') #puntos
plt.plot(T_ox,umol_o2_arifin,'r*')#puntos

#Datos extra
plt.title('Producción de oxígeno (Cinética de Bulfin)')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('Ajuste modelo numérico','Ajuste resultados de Arifin'),
          prop = {'size': 10}, loc='center right')

plt.show()

```



1.6 Modelo vs Experimento de Arifin: hidrógeno producido

```
[10]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)

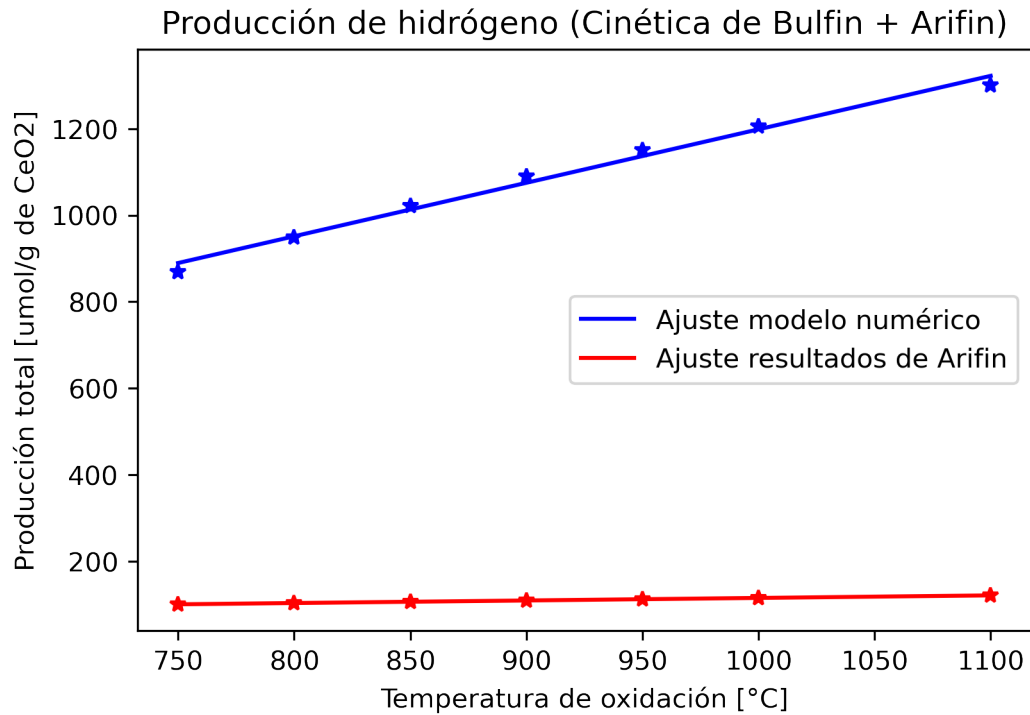
puntosy1=b0(T_ox,umol_h2)+b1(T_ox,umol_h2)*puntosx
puntosy2=b0(T_ox,umol_h2_arifin)+b1(T_ox,umol_h2_arifin)*puntosx

plt.plot(puntosx,puntosy1,'b')#rectas
plt.plot(puntosx,puntosy2,'r') #rectas

plt.plot(T_ox,umol_h2,'b*') #puntos
plt.plot(T_ox,umol_h2_arifin,'r*')#puntos

#Datos extra
plt.title('Producción de hidrógeno (Cinética de Bulfin + Arifin)')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('Ajuste modelo numérico','Ajuste resultados de Arifin'),
          prop = {'size': 10}, loc='center right')

plt.show()
```



1.7 Producción de O₂ y H₂

```
[22]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)
#Oxígeno
puntosy102=b0(T_ox,umol_o2)+b1(T_ox,umol_o2)*puntosx #Ajuste lineal resultados
puntosy202=b0(T_ox,umol_o2_arifin)+b1(T_ox,umol_o2_arifin)*puntosx #Ajuste
↪ lineal arifin
#Hidrógeno
puntosy1h2=b0(T_ox,umol_h2)+b1(T_ox,umol_h2)*puntosx
puntosy2h2=b0(T_ox,umol_h2_arifin)+b1(T_ox,umol_h2_arifin)*puntosx

plt.plot(puntosx,puntosy102,'r') #Modelo O2
plt.plot(puntosx,puntosy202,'r--') #Arifin O2

plt.plot(puntosx,puntosy1h2,'b') #Modelo H2
plt.plot(puntosx,puntosy2h2,'b--') #Arifin H2

plt.plot(T_ox,umol_o2,'r.')
plt.plot(T_ox,umol_o2_arifin,'r.')
```

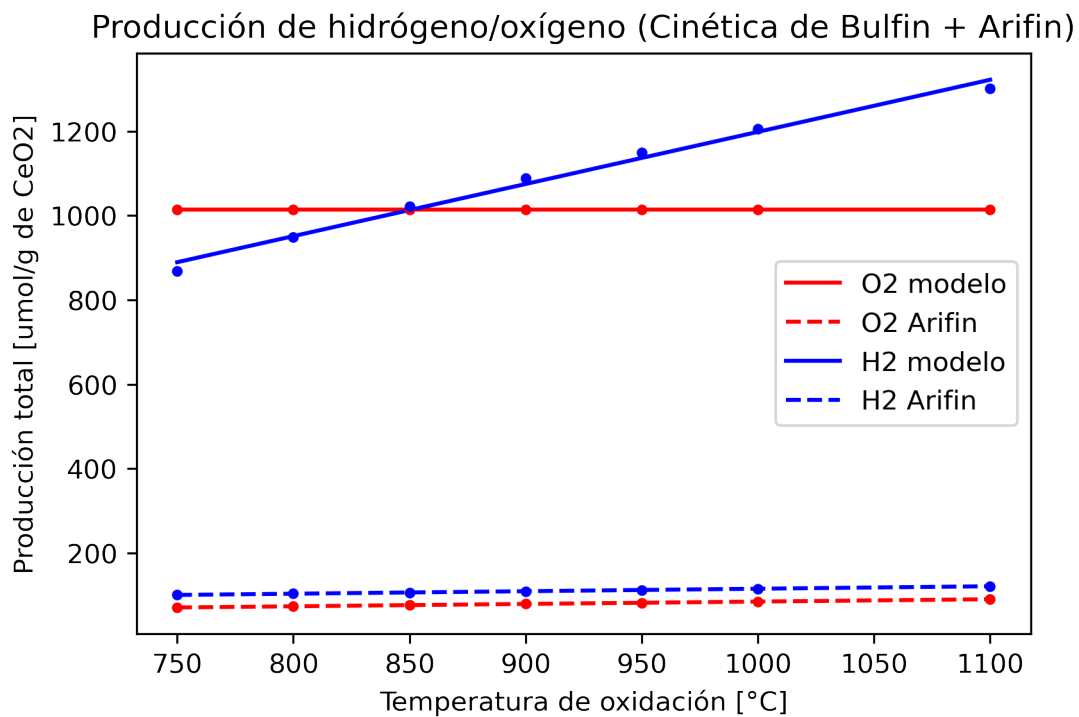
```

plt.plot(T_ox,umol_h2,'b.') #puntos
plt.plot(T_ox,umol_h2_arifin,'b.')#puntos

#Datos extra
plt.title('Producción de hidrógeno/oxígeno (Cinética de Bulfin + Arifin)')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('O2 modelo','O2 Arifin','H2 modelo','H2 Arifin'),
           prop = {'size': 10}, loc='center right')

plt.show()

```



Anexo B6

May 31, 2021

1 Modelo de Comparación: Cinética Arifin (sólo oxidación)

1.1 Concentración de vacancia (delta) inicial de oxidación

```
[1]: import numpy as np
d_arifin=(0.056*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+28.
↪99)*172.115*2*10**-6
d=d_arifin
print(d_arifin)
```

```
[0.02443689 0.02540073 0.02636458 0.02732842 0.02829226 0.02925611
0.0311838 ]
```

1.2 Reacción de Oxidación

```
[2]: dT=17.
Tred=1500.+273.15 #temperatura a la que se fijan los reactantes
Ts=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15 #Este arreglo
↪contiene las distintas Tox que ocupa Arifin
time=240.0 #estos son los segundos que se mantiene a 240°C una vez que alcanza
↪la temperatura de reducción
timer=np.zeros(len(Ts))
dt=0.02 #paso de tiempo
d_max=0.35 #Concentración de vacancias máxima.
R=8.3145*10**(-3)# [kJ /mol K ] Constante universal de los gases
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción
tiempo_total=np.zeros(len(Ts))
y_ox=0*10**-7 #Fracción molar promedio de oxígeno CORREGIR CON CADA ITERACIÓN
Aox=1.0
Eox=29.0
gamma=0.89
kred=np.zeros(len(Ts))
kox=np.zeros(len(Ts))
print_counter=0 #
```

```

#d_final=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]) #Asumimos condición
    ↪ final como oxidación completa de la ceria.
#d_final se obtiene mediante iteración

#Luego de 120 iteraciones se obtiene un d_final de
d_final=np.array([0.01192768, 0.01139412, 0.01089149, 0.01042392, 0.00999278, 0.
    ↪ 00959759, 0.00890827])
#El dox obtenido será la iteración 121 de f(a). Este f(a) solo corresponde a la
    ↪ componente de la oxidación, (esto debido a que se desprecia la reducción por
    ↪ las temperaturas de trabajo.)

dox=d.copy() #'d' corresponde a la distribución de vacancias final de la etapa
    ↪ de reducción
#Luego de correr el programa, dox será la distribución de vacancias final de la
    ↪ etapa de oxidación.

#La oxidación es isotérmica en el experimento de arifin:
Tox=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15

#d_inicial es fijo, son los valores que se obtuvieron al final de la reducción
d_inicial=d.copy()

for i in range(len(Tox)):
    tt=0
    while tt < 600:
        d_n=dox.copy()
        kox[i]=0.3**gamma*(1-(d_inicial[i]-d_n[i])/
    ↪ (d_inicial[i]-d_final[i]))*Aox*np.exp(-Eox/(R*Tox[i]))
            #(d_n[i]**2/d_max)
        if dox[i]>=0.35:
            break
        dox[i]=(d_n[i]/dt)/(1/dt+kox[i]) #como son estáticas no requiere
    ↪ recorrer

    tt+=dt

```

1.3 Resultados segun estequiometría

```
[3]: print('Considerando una reacción estequiométrica:')
      print('-----')

      print('Moles de hidrógeno producidos:')
      umol_h2=(d-dox)/172.115*10**6
      print('MODELO NUMÉRICO - micro-moles de H2 por g de CeO2:',umol_h2)
      umol_h2_arifin=0.059*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+56.
      ↪45
      print('RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2:', umol_h2_arifin)

      print('Diferencia de resultados:', umol_h2-umol_h2_arifin)
```

Considerando una reacción estequiométrica:

```
-----
Moles de hidrógeno producidos:
MODELO NUMÉRICO - micro-moles de H2 por g de CeO2: [ 72.62499041  81.32342947
 89.84276989  98.15888352 106.26369696
 114.15990295 129.36565844]
RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2: [100.7  103.65 106.6
 109.55 112.5  115.45 121.35]
Diferencia de resultados: [-28.07500959 -22.32657053 -16.75723011 -11.39111648
-6.23630304
 -1.29009705   8.01565844]
```

1.4 Función para ajuste lineal

```
[4]: import matplotlib.pyplot as plt
      plt.rcParams['figure.dpi'] = 300
      def b1(x,y):
          t1=x-np.average(x)
          t2=y-np.average(y)
          sxy=sum(t1*t2)
          sxx=sum(t1*t1)
          return sxy/sxx
      def b0(x,y):
          return np.average(y)-b1(x,y)*np.average(x)
      def reg_lin(x,y):
          return b0(x,y)+b1(x,y)*x
```

1.5 Modelo vs Experimento de Arifin: hidrógeno producido

```
[5]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)

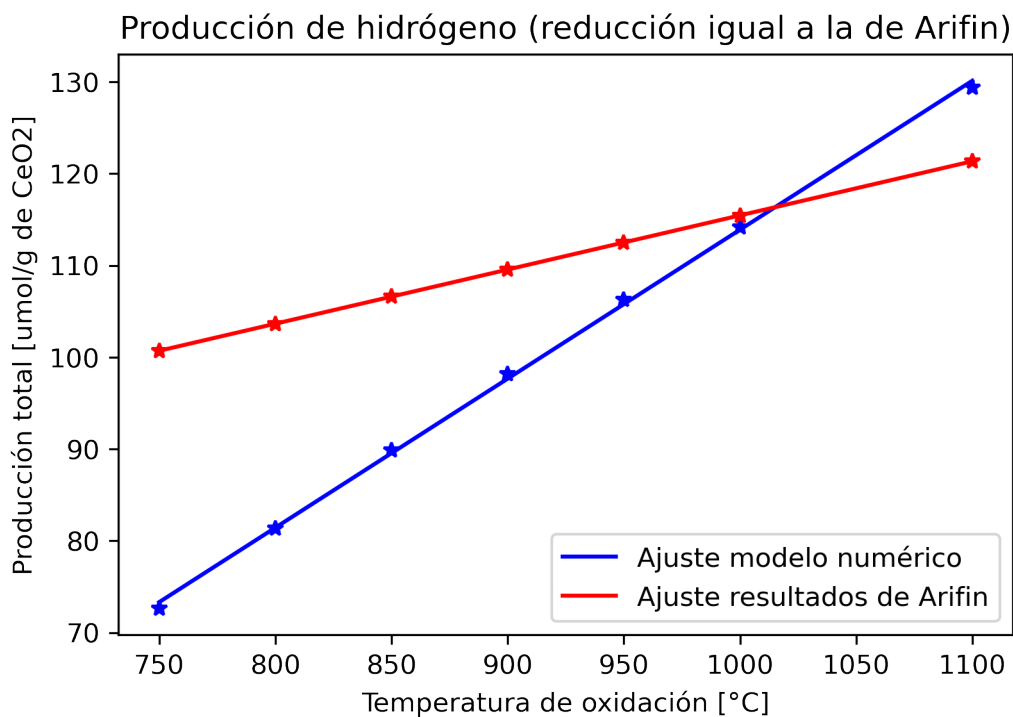
puntosy1=b0(T_ox,umol_h2)+b1(T_ox,umol_h2)*puntosx
puntosy2=b0(T_ox,umol_h2_arifin)+b1(T_ox,umol_h2_arifin)*puntosx

plt.plot(puntosx,puntosy1,'b')#rectas
plt.plot(puntosx,puntosy2,'r') #rectas

plt.plot(T_ox,umol_h2,'b*') #puntos
plt.plot(T_ox,umol_h2_arifin,'r*')#puntos

#Datos extra
plt.title('Producción de hidrógeno (reducción igual a la de Arifin)')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('Ajuste modelo numérico','Ajuste resultados de Arifin'),
           prop = {'size': 10}, loc='lower right')

plt.show()
```



Anexo B7

May 31, 2021

1 Modelo de Comparación: Variación nred=1.65

1.1 Etapa de Reducción

```
[1]: import numpy as np
dT=17.
Tred=1500.+273.15 #temperatura a la que se fijan los reactantes
Ts=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15 #Este arreglo
    ↪ contiene las distintas Tox que ocupa Arifin
time=240.0 #estos son los segundos que se mantiene a 240°C una vez que alcanza
    ↪ la temperatura de reducción
timer=np.zeros(len(Ts))
dt=0.02 #paso de tiempo
d=np.zeros(len(Ts))
d_max=0.35 #Concentración de vacancias máxima.
R=8.3145*10**(-3) # [kJ /mol K ] Constante universal de los gases
Ared= 720000. #+-360000 [1/s] Factor pre-exponencial de reducción
Ered=232. # +-5 [kJ/mol] Energía de activación de reducción
tiempo_total=np.zeros(len(Ts))
y_ox=0 #Fracción molar promedio de oxígeno CORREGIR CON CADA ITERACIÓN
Aox=1.0
Eox=29.0
gamma=0.89
kred=np.zeros(len(Ts))
kox=np.zeros(len(Ts))
print_counter=0 #

#d_final=np.array([0.35, 0.35, 0.35, 0.35, 0.35, 0.35, 0.35]) #para la
    ↪ iteración de f(a) se consideró como
#condición inicial de iteración la reducción completa.

d_final=np.array([0.0298415, 0.02984592, 0.02985032, 0.02985469, 0.02985896, 0.
    ↪ 02986298, 0.02986875])
```

```

#----Al ejecutar este código se obtendrá el resultado de la iteración 1651 de
→f(a) considerando la reducción completa como condicion inicial

#este d_final es obtenido mediante proceso iterativo.

for i in range(len(Ts)):
    tt=0
    while timer[i] < time:
        d_n=d.copy()
        tt+=dt

        if Ts[i] < Tred:
            Ts[i]+=dT*dt

        if Ts[i] >= Tred:

            timer[i]+=dt

        kred[i]=(1-d_n[i]/d_final[i])**1.65*Ared*np.exp(-Ered/(R*Ts[i]))

        kox[i]=y_ox**gamma*(1-d_n[i]/d_final[i])*Aox*np.exp(-Eox/(R*Ts[i]))
        #el término f(a) es en este caso (reducción) d(t)/d_final, puesto
→que d_inicial=0

        d[i]=(d_n[i]/dt+d_max*kred[i])/(kred[i]+kox[i]+1/dt) #como son
→estáticas no requiere recorrer x

        tiempo_total[i]=tt

```

```

[2]: print('d_anterior: ',d_final)
      print('d_actual: ',d)
      print('diferencia: ',d_final-d)
      #iteracion 1

```

```

d_anterior: [0.0298415  0.02984592 0.02985032 0.02985469 0.02985896 0.02986298
 0.02986875]
d_actual: [0.02983156 0.02983598 0.02984038 0.02984475 0.02984902 0.02985304
 0.02985881]
diferencia: [9.93681352e-06 9.93804238e-06 9.93926149e-06 9.94047754e-06
 9.94166677e-06 9.94278363e-06 9.94438025e-06]

```

```

[3]: umol_o2=(d/2)/172.115*10**6
      print('MODELO NUMÉRICO - micro-moles de O2 por g de CeO2:',umol_o2)

```

```

MODELO NUMÉRICO - micro-moles de O2 por g de CeO2: [86.661718  86.67455468

```

```
86.68733329 86.70002476 86.7124258 86.72410079
86.7408582 ]
```

1.2 Cálculos según estequiometría de la reducción

```
[4]: print('Considerando una reacción estequiométrica:')
print('-----')

umol_o2=(d/2)/172.115*10**6
print('MODELO NUMÉRICO - micro-moles de O2 por g de CeO2:',umol_o2)
umol_o2_arifin=0.056*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+28.
    ↪99
print('RESULTADOS DE ARIFIN - micro-moles de O2 por g de CeO2:' ,umol_o2_arifin_
    ↪)
#print('Producción total de O2 en micro moles con 0,2g de CeO2', (d/2)/172.
    ↪115*10**6*0.2)
d_arifin=(0.056*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+28.
    ↪99)*172.115*2*10**-6

#print('-La cantidad de oxígeno obtenida por Arifin corresponderia a una_
    ↪concentración de vacancias de:')
#print(d_arifin)
#print('La concentración de vacancias obtenidas mediante el modelo numérico es:_
    ↪')
#print(d)

#RESULTADOS
```

Considerando una reacción estequiométrica:

```
-----
MODELO NUMÉRICO - micro-moles de O2 por g de CeO2: [86.661718 86.67455468
86.68733329 86.70002476 86.7124258 86.72410079
86.7408582 ]
RESULTADOS DE ARIFIN - micro-moles de O2 por g de CeO2: [70.99 73.79 76.59 79.39
82.19 84.99 90.59]
```

```
[5]: print('Verificación de fracción molar de oxidante:')
print('-----')
moles_producto_o2=(d/2)/172.115*0.2
moles_he=500*tiempo_total/60*0.18/4.0026 #500 standar cm3/min sccm , 0.18 g/cm3_
    ↪4.0026 g/mol
#print('moles de O2 totales:', moles_producto_o2)
#print('moles de helio totales', moles_he)
y_ox_r=moles_producto_o2/(moles_producto_o2+moles_he)
#print('fracción molar de oxidante: ',y_ox_r)
y_ox_prom=np.average(y_ox_r)
print('fracción molar de oxidante promedio:',y_ox_prom )
```

Verificación de fracción molar de oxidante:

fracción molar de oxidante promedio: 1.684385738291143e-07

1.3 Etapa de Oxidación

```
[4]: #d_final=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]) #Asumimos condición
    ↪ final como oxidación completa de la ceria.
    #d_final se obtiene mediante iteración

    #Luego de 140 iteraciones se obtiene un d_final de
    d_final=np.array([0.01476997, 0.01359096, 0.0125308, 0.0115819, 0.01073468, 0.
    ↪ 00997889, 0.0087017 ])
    #El dox obtenido será la iteración 141 de f(a). Este f(a) solo corresponde a la
    ↪ componente de la oxidación, (esto debido a que se desprecia la reducción por
    ↪ las temperaturas de trabajo.)

    dox=d.copy() #'d' corresponde a la distribución de vacancias final de la etapa
    ↪ de reducción
    #Luego de correr el programa, dox será la distribución de vacancias final de la
    ↪ etapa de oxidación.

    #La oxidación es isotérmica en el experimento de arifin:
    Tox=np.array([750., 800., 850., 900., 950., 1000., 1100.])+273.15

    #d_inicial es fijo, son los valores que se obtuvieron al final de la reducción
    d_inicial=d.copy()

    for i in range(len(Tox)):
        tt=0
        while tt < 600:
            d_n=dox.copy()
            kox[i]=0.3**gamma*(1-(d_inicial[i]-d_n[i])/
            ↪ (d_inicial[i]-d_final[i]))*Aox*np.exp(-Eox/(R*Tox[i]))
                #(d_n[i]**2/d_max)
            if dox[i]>=0.35:
                break
            dox[i]=(d_n[i]/dt)/(1/dt+kox[i]) #como son estáticas no requiere
            ↪ recorrer

            tt+=dt
```

```

tiempo: 4.0000000000000003
Temperatura instantánea: [1023.15 1073.15 1123.15 1173.15 1223.15 1273.15
1373.15]
Distribución de vacancias: [0.02856543 0.02983598 0.02984038 0.02984475
0.02984902 0.02985304
0.02985881]
Iteración de caso número 1

```

1.4 Resultados segun estequiometría

```

[7]: print('Considerando una reacción estequiométrica:')
print('-----')

print('Moles de hidrógeno producidos:')
umol_h2=(d-dox)/172.115*10**6
print('MODELO NUMÉRICO - micro-moles de H2 por g de CeO2:',umol_h2)
umol_h2_arifin=0.059*np.array([750., 800., 850., 900., 950., 1000., 1100. ])+56.
↪45
print('RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2:', umol_h2_arifin)
print('Diferencia de resultados:', umol_h2-umol_h2_arifin)

```

```

Considerando una reacción estequiométrica:
-----
Moles de hidrógeno producidos:
MODELO NUMÉRICO - micro-moles de H2 por g de CeO2: [ 87.45368255  94.32990601
100.51603463 106.05593358 111.00471887
115.42096859 122.87857882]
RESULTADOS DE ARIFIN - micro-moles de H2 por g de CeO2: [100.7  103.65 106.6
109.55 112.5  115.45 121.35]
Diferencia de resultados: [-13.24631745 -9.32009399 -6.08396537 -3.49406642
-1.49528113
-0.02903141  1.52857882]

```

```

[8]: #Razon de moles de H2 sobre moles de O2 producidos
print('Razon de moles de H2 sobre moles de O2 producidos:')
print('-----')
#Arifin:
razon_arifin=(umol_h2_arifin)/(umol_o2_arifin)
print('moles de H2/moles de O2 segun arifin:', razon_arifin)

razon_modelo=umol_h2/umol_o2
print('moles de H2/moles de O2 segun el modelo numérico:', razon_modelo)

```

```

Razon de moles de H2 sobre moles de O2 producidos:
-----
moles de H2/moles de O2 segun arifin: [1.41850965 1.40466188 1.39182661
1.37989671 1.36877966 1.35839511

```

```

1.33955183]
moles de H2/moles de O2 segun el modelo numérico: [1.00913857 1.08832294
1.1595239 1.22325148 1.28014777 1.33089842
1.41661705]

```

1.5 Función para ajuste lineal

```

[9]: import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
def b1(x,y):
    t1=x-np.average(x)
    t2=y-np.average(y)
    sxy=sum(t1*t2)
    sxx=sum(t1*t1)
    return sxy/sxx
def b0(x,y):
    return np.average(y)-b1(x,y)*np.average(x)
def reg_lin(x,y):
    return b0(x,y)+b1(x,y)*x

```

1.6 Modelo vs Experimento de Arifin: oxígeno producido

```

[10]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)

puntosy1=b0(T_ox,umol_o2)+b1(T_ox,umol_o2)*puntosx
puntosy2=b0(T_ox,umol_o2_arifin)+b1(T_ox,umol_o2_arifin)*puntosx

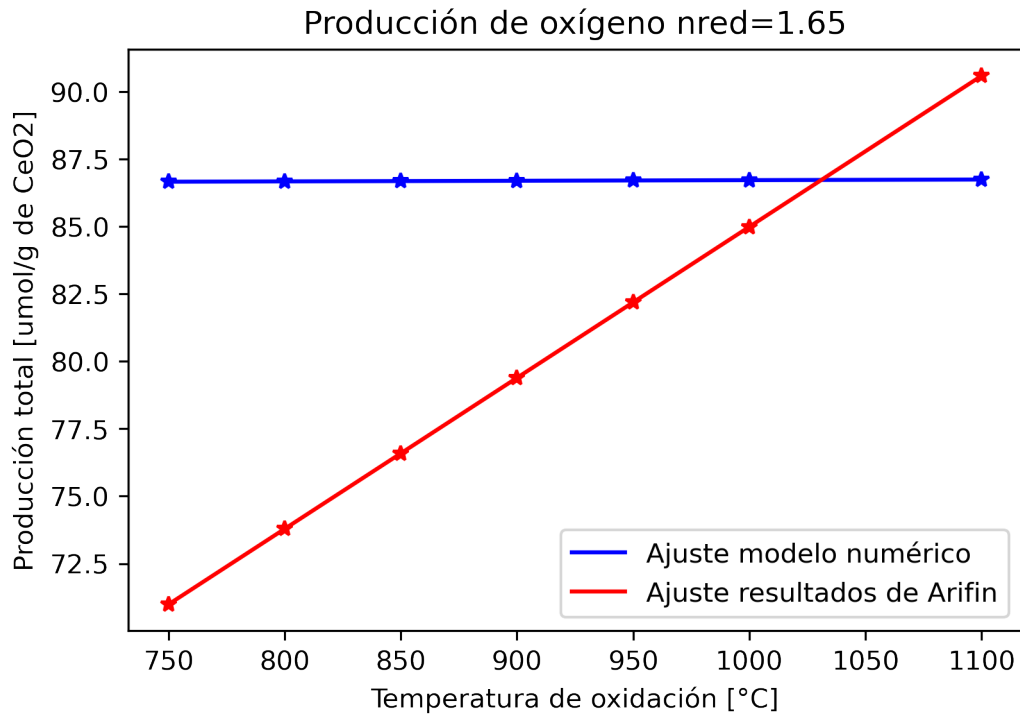
plt.plot(puntosx,puntosy1,'b')#rectas
plt.plot(puntosx,puntosy2,'r') #rectas

plt.plot(T_ox,umol_o2,'b*') #puntos
plt.plot(T_ox,umol_o2_arifin,'r*')#puntos

#Datos extra
plt.title('Producción de oxígeno nred=1.65')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('Ajuste modelo numérico','Ajuste resultados de Arifin'),
          prop = {'size': 10}, loc='lower right')

plt.show()

```



1.7 Modelo vs Experimento de Arifin: hidrógeno producido

```
[11]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)

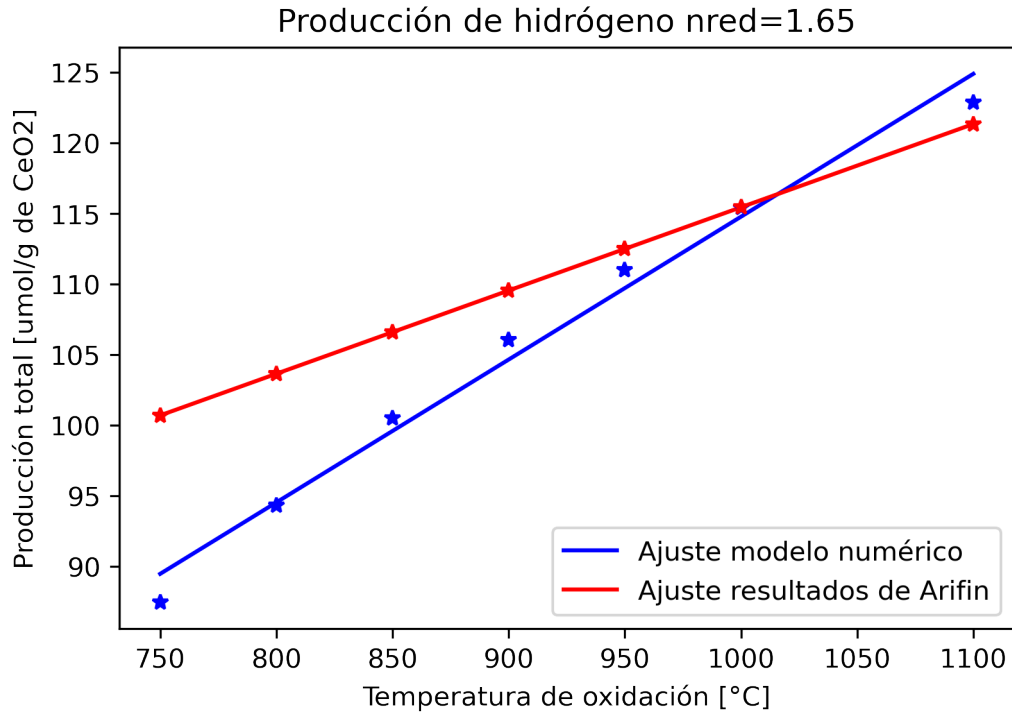
puntosy1=b0(T_ox,umol_h2)+b1(T_ox,umol_h2)*puntosx
puntosy2=b0(T_ox,umol_h2_arifin)+b1(T_ox,umol_h2_arifin)*puntosx

plt.plot(puntosx,puntosy1,'b')#rectas
plt.plot(puntosx,puntosy2,'r') #rectas

plt.plot(T_ox,umol_h2,'b*') #puntos
plt.plot(T_ox,umol_h2_arifin,'r*')#puntos

#Datos extra
plt.title('Producción de hidrógeno nred=1.65')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('Ajuste modelo numérico','Ajuste resultados de Arifin'),
          prop = {'size': 10}, loc='lower right')

plt.show()
```



1.8 Producción de O2 y H2

```
[16]: T_ox=np.array([750., 800., 850., 900., 950., 1000., 1100.])
puntosx=np.linspace(T_ox[0],T_ox[-1],100)
#Oxígeno
puntosy102=b0(T_ox,umol_o2)+b1(T_ox,umol_o2)*puntosx #Ajuste lineal resultados
puntosy202=b0(T_ox,umol_o2_arifin)+b1(T_ox,umol_o2_arifin)*puntosx #Ajuste
↪ lineal arifin
#Hidrógeno
puntosy1h2=b0(T_ox,umol_h2)+b1(T_ox,umol_h2)*puntosx
puntosy2h2=b0(T_ox,umol_h2_arifin)+b1(T_ox,umol_h2_arifin)*puntosx

plt.plot(puntosx,puntosy102,'r') #Modelo O2
plt.plot(puntosx,puntosy202,'r--') #Arifin O2

plt.plot(puntosx,puntosy1h2,'b') #Modelo H2
plt.plot(puntosx,puntosy2h2,'b--') #Arifin H2

plt.plot(T_ox,umol_o2,'r.')
plt.plot(T_ox,umol_o2_arifin,'r.')
```

```

plt.plot(T_ox,umol_h2,'b.') #puntos
plt.plot(T_ox,umol_h2_arifin,'b.')#puntos

#Datos extra
plt.title('Producción de hidrógeno/oxígeno (Cinética de Bulfin + Arifin)')
plt.xlabel('Temperatura de oxidación [°C]')
plt.ylabel('Producción total [umol/g de CeO2]')
plt.legend(('O2 modelo','O2 Arifin','H2 modelo','H2 Arifin'),
           prop = {'size': 10}, loc='upper left')

plt.show()

```

