Full Penn Approximation RafaCastle

Preambles and Linhard dielectric function

Values

Here we import the energy loss function data (can’t share those)

```
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
necesita
LaunchKernels[];
lanza kernels
ParallelEvaluate[Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]];
evalúa en paralelo    necesita
kIDnums = ParallelEvaluate[$KernelID];
          evalúa en paralelo    identificador de kernel
```

```
In[ ]:= NombreELF1 = "Al2O3 ELF.dat";
NombreELF2 = "CaF2 ELF.dat";
NombreELF3 = "LiF ELF.dat";
NombreELF4 = "H2O ELF.dat";
NombreC1 = "Al2O3 Cond.dat";
NombreC2 = "CaF2 Cond.dat";
NombreC3 = "LiF Cond.dat";
NombreC4 = "H2O Cond.dat";

Compuesto = ChoiceDialog["Escoge el compuesto", {Al2O3 → 1, CaF2 → 2, LiF → 3, Agua → 4}];
            diálogo de elección
Do[If[i == Compuesto, {temporal = Import[NombreELFi], Cond = Import[NombreCi]}], {i, 4}];
r··· si                          importa                      importa
{bandgap , wmin , BVal, densidad , elfinicial} =
  Table[Cond[[i]][[1]], {i, 1, Length[Cond]}];
  tabla                      longitud
ELFData = temporal[[elfinicial ;; All]];
                                    todo
Do[temporal[[i, 1]] = temporal[[i, 1]] * QuantityMagnitude[UnitConvert["eV", "Hartrees"]],
repite                                   magnitud de cantidad    convierte unidad
 {i, 1, Length[temporal]}]
        longitud
ELFData = temporal[[1 ;; All]];
                        todo
```

Defining physical constants and converting to Hartree system:

```
c = 137;
bandgap = bandgap * QuantityMagnitude[UnitConvert["eV", "Hartrees"]];
                    magnitud de cantidad   convierte unidad
BVal = BVal * QuantityMagnitude[UnitConvert["eV", "Hartrees"]];
              magnitud de cantidad   convierte unidad
wmin = wmin * QuantityMagnitude[UnitConvert["eV", "Hartrees"]];
              magnitud de cantidad   convierte unidad
inicial = elfinicial;
```

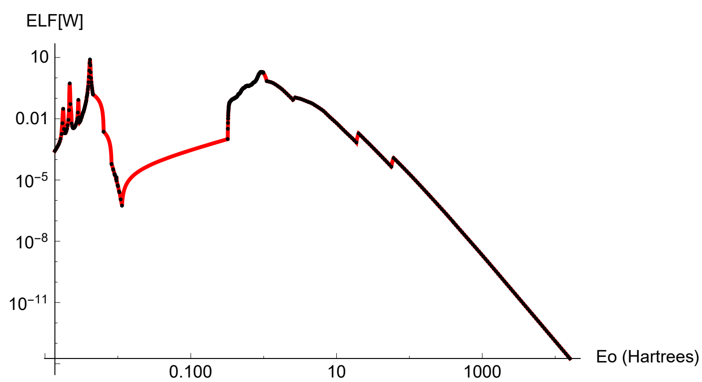
Interpolations

Plot of the aproximate energy loss function

```
In[]:= ELF = Interpolation[Join[{{0, 0}}, ELFData], InterpolationOrder → 1];
             interpolación  junta                  orden de interpolación

In[]:= LogLogPlot[ELF[W], {W, First[ELFData][[1]], Last[ELFData][[1]]}, PlotRange → All,
       representación log log   primero           último             rango de rep···  todo
        PlotStyle → {Thick, Red}, Epilog → {PointSize[Small], Point[Log /@ELFData]},
        estilo de repre···  grueso  rojo  epílogo  tamaño de···  tamaño···  punto  logaritmo
        AxesLabel → {"Eo (Hartrees)", "ELF[W]"}, PlotRange → All]
        etiqueta de ejes                         rango de rep···  todo
       g[w_] := (2 / (Pi * w)) * ELF[w]
                      número pi
```

Out[]=



Equations

```
In[]:= kf[wp_] := ((3 * Pi / 4) ^ (1 / 3)) * (wp) ^ (2 / 3);
                       número pi
       Ef[wp_] := (kf[wp] ^2) / 2
       x[w_, wp_] := w / Ef[wp]
       z[q_, wp_] := q / (2 * kf[wp])
       Ym[q_, w_, wp_] := z[q, wp] - (1 / 4) (x[w, wp] / z[q, wp])
       Yp[q_, w_, wp_] := z[q, wp] + (1 / 4) (x[w, wp] / z[q, wp])
       logsm[q_, w_, wp_] := Log[Abs[(Ym[q, w, wp] + 1) / (Ym[q, w, wp] - 1)]] +
                             lo···  valor absoluto
         Log[Abs[(Yp[q, w, wp] + 1) / (Yp[q, w, wp] - 1)]]
         lo···  valor absoluto
```

```
In[∘]:= a[q_, w_, wp_] := z[q, wp] / x[w, wp]
A[q_, w_, wp_] :=
 - (64 / 3) z[q, wp] * ((a[q, w, wp]) ^2) (3 + 48 (1 + (z[q, wp]) ^2) ((a[q, w, wp]) ^2) +
    256 (3 + (z[q, wp]) ^2) (1 + 3 * (z[q, wp]) ^2) (a[q, w, wp]) ^4)
b[q_, w_, wp_] := x[w, wp] / (z[q, wp] (((z[q, wp]) ^2) - 1))
B[q_, w_, wp_] := Log[((z[q, wp] + 1) / (z[q, wp] - 1)) ^2] +
                  logaritmo
  4 * z[q, wp] * ((b[q, w, wp]) ^2) * (1 + (1 + (z[q, wp]) ^2) ((b[q, w, wp]) ^2) +
     (1 / 3) (3 + (z[q, wp]) ^2) (1 + 3 (z[q, wp]) ^2) ((b[q, w, wp]) ^4))
```

Regiones

```
In[∘]:= logs[q_, w_, wp_] := Piecewise[{{A[q, w, wp], z[q, wp] / x[w, wp] < 0.01},
                             función a trozos
   {B[q, w, wp], z[q, wp] / x[w, wp] > 100}}, logsm[q, w, wp]]
pel[q_, w_, wp_] := (1 / (3 * Pi * wp * q * ((z[q, wp]) ^2))) (logs[q, w, wp])
                              número pi

In[∘]:= F[t_] := (1 - t^2) Log[Abs[(t + 1) / (t - 1)]]
                          lo⋯ valor absoluto
el1m[q_, w_, wp_] := 1 + (1 / (Pi * kf[wp] * (z[q, wp]) ^2)) (1 / 2 + (1 / (8 * z[q, wp]))
                               número pi
      (F[z[q, wp] - x[w, wp] / (4 z[q, wp])] + F[z[q, wp] + x[w, wp] / (4 z[q, wp])]))
el2m[q_, w_, wp_] := (1 / (8 * kf[wp] * (z[q, wp]) ^3)) *
  Piecewise[{{x[w, wp], 0 < x[w, wp] < 4 * z[q, wp] (1 - z[q, wp])},
  función a trozos
    {1 - (z[q, wp] - x[w, wp] / (4 * z[q, wp])) ^2,
     Abs[4 * z[q, wp] (1 - z[q, wp])] < x[w, wp] < 4 * z[q, wp] (1 + z[q, wp])}}, 0]
     valor absoluto
```

Corrections (Shinotsuka 2015)

```
In[∘]:= u[q_, w_, wp_] := (w / (kf[wp] * q))
el1es[q_, w_, wp_] := 1 - ((wp / w) ^2) (1 + (((z[q, wp]) ^2) + 3 / 5) (1 / ((u[q, w, wp]) ^2)))
el1ei[q_, w_, wp_] :=
 1 + (2 / (Pi * q * z[q, wp])) (1 / 2 + 1 / (4 * z[q, wp]) ((1 - (z[q, wp]) ^2 - (u[q, w, wp]) ^2)
          número pi
          Log[Abs[(z[q, wp] + 1) / (z[q, wp] - 1)]] + ((z[q, wp]) ^2 - (u[q, w, wp]) ^2 - 1)
          lo⋯ valor absoluto
          (2 * (u[q, w, wp]) ^2 * z[q, wp] / ((((z[q, wp]) ^2) - 1) ^2))))
el2es[q_, w_, wp_] := 0
el2ei[q_, w_, wp_] := u[q, w, wp] / (q * z[q, wp])

In[∘]:= el1[q_, w_, wp_] := Piecewise[{{el1ei[q, w, wp], u[q, w, wp] < 0.01},
                            función a trozos
   {el1es[q, w, wp], u[q, w, wp] / (z[q, wp] + 1) > 100}}, el1m[q, w, wp]]
el2[q_, w_, wp_] := Piecewise[{{el2ei[q, w, wp], u[q, w, wp] < 0.01},
                            función a trozos
   {el2es[q, w, wp], u[q, w, wp] / (z[q, wp] + 1) > 100}}, el2m[q, w, wp]]
```

Pl and SE components for the FPA energy loss function

Preambles

Indicent electron energy values

```
In[∘]:= coords1 = First[InterpolatingFunctionCoordinates[ELF]];
                  primero
final = Length[coords1];
        longitud
```

Integration limits for momentum transfer

```
In[∘]:= T[Ei_] := Ei;
Tp[Ei_] := T[Ei] - bandgap;
qm[T_, w_] := Sqrt[Tp[T] (2 + Tp[T] / (c^2))] - Sqrt[(Tp[T] - w) (2 + (Tp[T] - w) / (c^2))]
              raíz cuadrada                        raíz cuadrada
qp[T_, w_] := Sqrt[Tp[T] (2 + Tp[T] / (c^2))] + Sqrt[(Tp[T] - w) (2 + (Tp[T] - w) / (c^2))]
              raíz cuadrada                        raíz cuadrada
```

```
In[∘]:= Fac[T_] := (((1 + Tp[T] / (c^2)) ^2) / (1 + Tp[T] / (2 c^2))) (1 / (Pi * Tp[T]))
                                                                        número pi
qm1[w_, wp_] := -kf[wp] + Sqrt[(kf[wp]) ^2 + 2 * w]
                          raíz cuadrada
qp1[w_, wp_] := kf[wp] + Sqrt[(kf[wp]) ^2 + 2 * w]
                         raíz cuadrada
Imel[q_, w_, wp_] := ((el2[q, w, wp])) / ((el1[q, w, wp]) ^2 + (el2[q, w, wp]) ^2)
```

Plasmon

```
In[∘]:= Imepl[q_?NumericQ, w_?NumericQ, kT_] := g[Intw0[w, q, kT]] *
                ¿expresión nu···  ¿expresión numérica?
  Pi / (Abs[pel[q, w, Intw0[w, q, kT]]]) * UnitStep[qm1[w, Intw0[w, q, kT]] - q]
  nú···  valor absoluto                     función paso unidad
```

Single Electron

```
In[∘]:= Imese[q_?NumericQ, w_?NumericQ] :=
                ¿expresión nu···  ¿expresión numérica?
 NIntegrate[g[wp] * Imel[q, w, wp] * UnitStep[qp1[w, wp] - q] * UnitStep[q - qm1[w, wp]],
 integra numéricamente                 función paso unidad          función paso unidad
  {wp, 0, Infinity}, Method → {"AdaptiveQuasiMonteCarlo"}]
          infinito    método
```

Numeric Method

ω and q variables

```
sq = 100;
sw = 200;
s[v_] := 1/v
kw = 0;
kq = 0;
cs = 0;
ccs = 1;
Do[{pw = Tp[coords1[[kT]]] - BVal - wmin,
repite
   While[wmin + s[sw] * kw * pw ≤ Tp[coords1[[kT]]] - BVal,
   mientras
     {vw_{kT,kw} = wmin + s[sw] * kw * pw, pq = qp[coords1[[kT]], vw_{kT,kw}] - qm[coords1[[kT]], vw_{kT,kw}],
      While[qm[coords1[[kT]], vw_{kT,kw}] + s[sq] * kq * pq ≤ qp[coords1[[kT]], vw_{kT,kw}],
      mientras
        vq_{kT,kw,kq} = qm[coords1[[kT]], vw_{kT,kw}] + s[sq] * kq * pq;
        kq++], qfin_{kT,kw} = kq - 1, kq = 0};
    kw++], wfin_{kT} = kw - 1, kw = 0, If[cs == 370,
                                          si
    {sq = sq + 1, sw = sw + 2, ccs = ccs + 1, cs = 350 + ccs}, cs = cs + 1]}, {kT, inicial, final}]
```

SE ELF

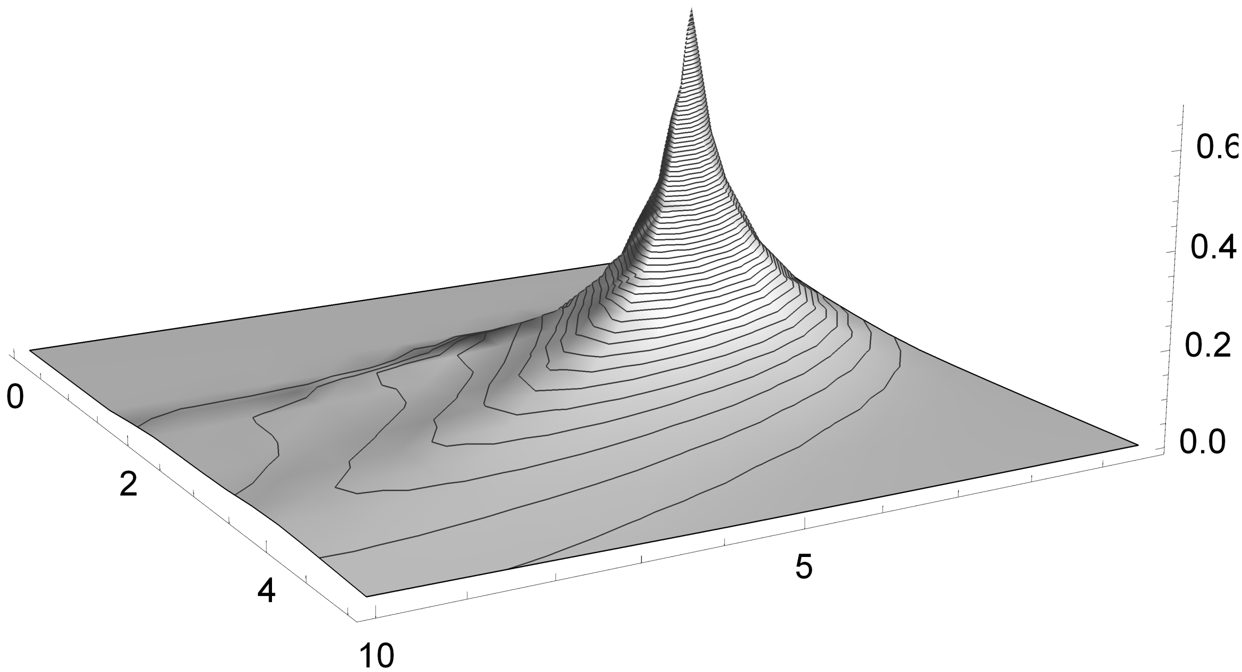
```
In[∘]:= Do[{vSE_{kT,kw,kq} = Imese[vq_{kT,kw,kq}, vw_{kT,kw}],
        repite
           If[kw == wfin_{kT} && kq == qfin_{kT,kw}, Print[{kT, kw, kq}]]},
           si                                       escribe
         {kT, inicial, final}, {kw, 0, wfin_{kT}}, {kq, 0, qfin_{kT,kw}}]

In[∘]:= IMESES1 = Interpolation[Flatten[Table[{{vw_{kT,kw}, vq_{kT,kw,kq}}, vSE_{kT,kw,kq}},
                  interpolación  aplana   tabla
            {kT, inicial, final}, {kw, 0, wfin_{kT}}, {kq, 0, qfin_{kT,kw}}], 2], InterpolationOrder → 1]
                                                                                orden de interpolación
        IMESE[w_, q_] := IMESES1[w, q]

Out[∘]= InterpolatingFunction[ Domain: {{0.317, 1.59×10^4}, {0.00275, 426.}}
                               Output: scalar ]
```

In[ ]:=
```
Plot3D[IMESE[w, q], {w, InterpolatingFunctionDomain[IMESES1][[1, 1]], 10},
 {q, InterpolatingFunctionDomain[IMESES1][[2, 1]], 5}, PlotRange → All,
 LabelStyle → Directive[18], PlotRange → All, Mesh → 70,
 MeshFunctions -> {#3 &}, Boxed → False, AxesEdge → {{1, -1}, {1, -1}, {-1, 1}},
 ColorFunction → (Directive[Opacity[#3 &]]), PlotStyle → Gray,
 ImageSize → 1000, TicksStyle → Directive[Black]]
```
representación gráfica 3D; rango de rep···; todo; estilo de etiqueta; directiva; rango de rep···; todo; malla; funciones de divisiones de malla; rodead···; falso; borde de ejes; función de color; directiva; opacidad; estilo de repr···; gris; tamaño de imagen; estilo de marcas; directiva; negro

Out[ ]=



ω_o

ω_o obtained by numeric methods

In[ ]:=
```
Do[{If[q == 0, wr = 0.9 * vw_{i,w}], w0_{i,w,q} = FindRoot[el1[vq_{i,w,q}, vw_{i,w}, wp] == 0, {wp, wr}][[1, 2]],
  wr = 0.9 * w0_{i,w,q}, If[q == qfin_{i,w} && w == wfin_{i}, Print[{i}]]},
 {i, inicial, final}, {w, 0, wfin_{i}}, {q, 0, qfin_{i,w}}]
```
rep··; si; encuentra raíz; si; escribe

In[ ]:=
```
Intw01 = Interpolation[Flatten[Table[{{vw_{i,w}, vq_{i,w,q}}, w0_{i,w,q}},
    {i, inicial, final}, {w, 0, wfin_{i}}, {q, 0, qfin_{i,w}}], 2], InterpolationOrder → 1]

Intw0[w_, q_, kT_] := Intw01[w, q]
```
interpolación; aplana; tabla; orden de interpolación

PL ELF

In[∘]:=
```
Do[{vPL_kT,kw,kq = Imepl[vq_kT,kw,kq, vw_kT,kw, kT], If[kw == wfin_kT && kq == qfin_kT,kw, Print[kT]]},
repite                                          si                                     escribe
 {kT, inicial, final}, {kw, 0, wfin_kT}, {kq, 0, qfin_kT,kw}]
```

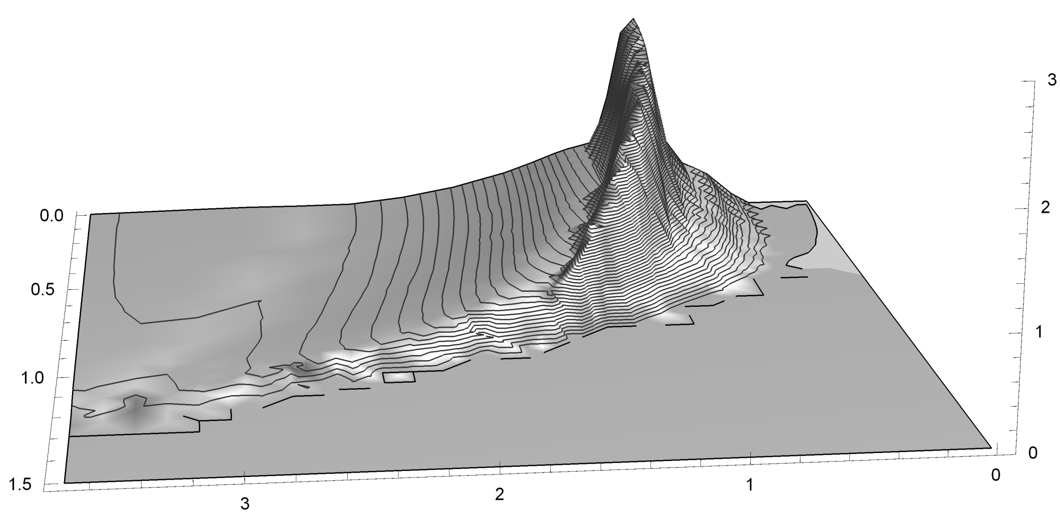
In[∘]:=
```
IMEPLS = Interpolation[Flatten[Table[{{vw_kT,kw, vq_kT,kw,kq}, vPL_kT,kw,kq},
         interpolación  aplana  tabla
    {kT, inicial, final}, {kw, 0, wfin_kT}, {kq, 0, qfin_kT,kw}], 2], InterpolationOrder → 1]
                                                                  orden de interpolación

IMEPL[w_, q_] := IMEPLS[
  w,
  q]
```

Out[∘]= InterpolatingFunction[Domain: {{0.317, 1.59×10^4}, {0.00275, 426.}} Output: scalar]

In[∘]:=
```
Plot3D[IMEPL[w, q], {w, 0, 3.7}, {q, 0.05, 1.5},
representación gráfica 3D
 PlotRange → {0, 3}, Mesh → 70, MeshFunctions -> {#3 &}, Boxed → False,
 rango de representación  malla   funciones de divisiones de malla  rodead···  falso
 AxesEdge → {{1, -1}, {1, -1}, {-1, 1}}, ColorFunction → (Directive[Opacity[#3 &]]),
 borde de ejes                             función de color    directiva  opacidad
 PlotStyle → Gray, ImageSize → 1000, TicksStyle → Directive[Black]]
 estilo de repr···  gris  tamaño de imagen  estilo de marcas  directiva  negro
```

Out[∘]=



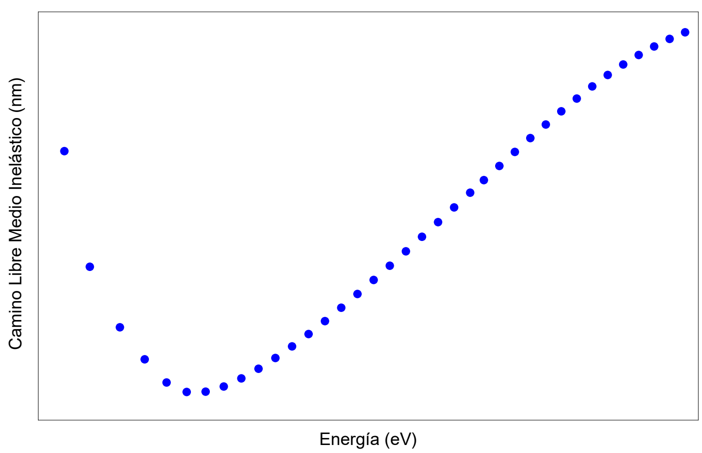
Inelastic mean free path

In[∘]:=
```
CLM1[kT_?NumericQ] :=
          ¿expresión numérica?
 Fac[coords1[[kT]]] * NIntegrate[(1/qq) * (IMESE[ww, qq] + IMEPL[ww, qq]),
                      integra numéricamente
   {ww, wmin, Tp[coords1[[kT]]] - BVal(*coords1[[kT]]-EFermi*)},
   {qq, qm[coords1[[kT]], ww], qp[coords1[[kT]], ww]},
   AccuracyGoal → 20, MinRecursion → 4, MaxRecursion → 100]
   objetivo de exactitud   recursión mínima   máxima recursión
```

In[∘]:=
```
Do[{clm_kT = CLM1[kT], Print[kT]}, {kT, inicial, final, 4}]
repite                  escribe
CLM = Table[
      tabla
   {(coords1[[kT]] - bandgap - BVal) * QuantityMagnitude[UnitConvert["Hartrees", "eV"]],
                                        magnitud de cantidad   convierte unidad
    (1/clm_kT) * QuantityMagnitude[UnitConvert["BohrRadius", "nanometers"]]}, {kT,
                 magnitud de cantidad   convierte unidad
    inicial, final, 4}];
```

In[∘]:=
```
ListLogLogPlot[CLM, PlotStyle → {Blue, Thick}, Frame → True,
representación log log de ··· estilo de repre··· azul grueso marco verdadero
 FrameLabel → {"Energía (eV)", " Camino Libre Medio Inelástico (nm)"}, PlotRange → All]
 etiqueta de marco                                                    rango de rep··· todo
```

Out[∘]=



Stopping Power

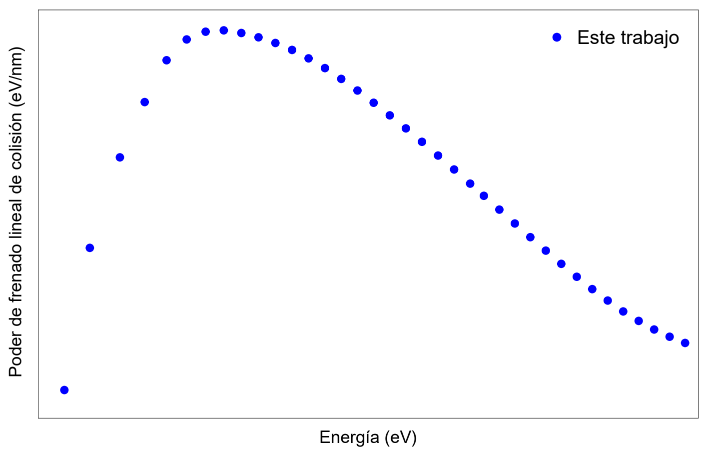
In[∘]:=
```
PF[kT_?NumericQ] :=
      ¿expresión numérica?
 Fac[coords1[[kT]]] * NIntegrate[(1/qq) * ww * (IMESE[ww, qq] + IMEPL[ww, qq]),
                      integra numéricamente
   {ww, wmin, Tp[coords1[[kT]]] - BVal}, {qq, qm[coords1[[kT]], ww],
    qp[coords1[[kT]], ww]}, AccuracyGoal → 20, MinRecursion → 4, MaxRecursion → 100]
                            objetivo de exactitud   recursión mínima   máxima recursión
```

```
Do[{PdFkT = PF[kT], Print[kT]}, {kT, inicial, final}]
repite                escribe

PodFr = Table[
        tabla
   {(coords1[[kT]] - bandgap - BVal) * QuantityMagnitude[UnitConvert["Hartrees", "eV"]],
                                       magnitud de cantidad  convierte unidad
    ((PdFkT * QuantityMagnitude[UnitConvert["Hartrees", "eV"]]) /
              magnitud de cantidad  convierte unidad
      QuantityMagnitude[UnitConvert["BohrRadius", "nanometers"]])}, {kT, 430, final}];
      magnitud de cantidad  convierte unidad
```

In[∘]:=
```
ListLogLogPlot[PodFr, PlotStyle → {Blue, Thick}, Frame → True,
representación log log de lista  estilo de repre···  azul  grueso  marco  verdadero
 FrameLabel → {"Energía (eV)", " Poder de frenado lineal de colisión (eV/nm)"},
 etiqueta de marco
 PlotRange → All, PlotLegends → Placed[{"Este trabajo"}, {Right, Top}]]
 rango de rep···  todo  leyendas de rep···  colocado   derecha  arriba
```

Out[∘]=



CSDA range

In[∘]:=
```
Needs["FunctionApproximations`"]
necesita
```

In[∘]:=
```
PDFA = Table[{PodFr[[i, 1]], PodFr[[i, 2]]}, {i, 1, Length[PodFr]}];
       tabla                                        longitud
PDFAI = Interpolation[Join[PDFA]];
        interpolación  junta
Al[kT_?NumericQ] := NIntegrateInterpolatingFunction[
       ¿expresión numérica?
  1 / PDFAI[EE], {EE, wmin * QuantityMagnitude[UnitConvert["Hartrees", "eV"]],
                              magnitud de cantidad  convierte unidad
    (coords1[[kT]] - bandgap - BVal) * QuantityMagnitude[UnitConvert["Hartrees", "eV"]]}]
                                       magnitud de cantidad  convierte unidad
Do[AlckT = Al[kT], {kT, inicial, final}]
repite
```

```
Alcance = Table[{(coords1[[kT]] - bandgap - BVal) *
                tabla
      QuantityMagnitude[UnitConvert["Hartrees", "eV"]], Alc_kT}, {kT, inicial, final}];
      magnitud de cantidad  convierte unidad
```

```
In[]:= ListLogLogPlot[Alcance, PlotStyle → {Thick}, Frame → True,
       representación log log de lista  estilo de repre···  grueso  marco  verdadero
        FrameLabel → {"Energía (eV)", " Alcance (nm)"}, PlotRange → All]
        etiqueta de marco                                  rango de rep···  todo
```

Out[]=