Apéndice C

Programación del potencial

A la hora de implementar el potencial Finnis-Sinclair, se utilizan dos archivos, cada uno de ellos compuesto por varias funciones. El primer archivo sirve para obtener los valores relativos a la interaciones átomo-átomo del potencial (E_P) y el segundo a las interacciones derivadas de la densidad (E_N).

C.1. Interacción entre átomos

```
static pthread-once-t eam-once=PTHREAD-ONCE-INIT;
static const struct material-t *material=NULL;
static int init=0;
static double cf-0;
static double cf-1;
static double cf-2;
static double cf-3;
static double cf-4;

/* fit a+b*r+c*r^2+d*r^3 for r less than r-c */

static double r-c;
static double a;
static double b;
static double c;
static double d;


/* Finis-Sinaclair init routines */
```

```
static void eam-init(void)
{

        double fi-r-i;
        double r-i;

        /* V=cf-0+cf-1*r+cf-2*r^2+cf-3*r^3+cf-4*r^4 */

        cf-0=material->beta*material->r-cutoff*material->r-cutoff;
        cf-1=material->gama*material->r-cutoff*material->r-cutoff-
2*material->beta*material->r-cutoff;
        cf-2=material->beta+material->alpha*material->r-cutoff*material->r-cutoff-
2*material->gama*material->r-cutoff;
        cf-3=material->gama-2*material->r-cutoff*material->alpha;
        cf-4=material->alpha;

        r-c=2;
        r-i=0.1*material->a*M-SQRT1-2;

        fi-r-i=cf-0+cf-1*r-i+cf-2*r-i*r-i+cf-3*r-i*r-i*r-i;

        a=((cf-0-cf-4*r-i*r-c*r-c*r-c)+r-c/((r-c-r-i)*(r-c-r-i)*(r-c-r-i))*(1000-fi-r-i));
        b=(cf-1-3*r-c*r-c/((r-c-r-i)*(r-c-r-i)*(r-c-r-i))*(1000-fi-r-i)-
cf-4*r-c*r-c/((r-c-r-i)*(r-c-r-i)*(r-c-r-i))* (3*r-i*r-i*r-i*r-i+6*r-i*r-i*r-c*r-c-r-c*r-c*r-c*r-c-8*r-
i*r-i*r-i*r-c));
        c=((cf-2-3*r-c*cf-4*(r-c+r-i))+3*r-c/((r-c-r-i)*(r-c-r-i)*(r-c-r-i))*(1000-fi-r-i));
        d=((cf-3+cf-4*(3*r-c+r-i))-(1000-fi-r-i)/((r-c-r-i)*(r-c-r-i)*(r-c-r-i)));

        init=1;

        return;
}


/* Finnis-Sinclair */

void eam(double *energy,
double *force,
const double r,
const struct material-t *P-material)
{

        if(material==NULL)
        material=P-material;
```

```
        if(init==0) pthread-once( &eam-once, eam-init );

        if( r <= material->r-cutoff )

                if( r <= r-c ){
                        *energy = a+b*r+c*r*r+d*r*r*r;
                        *force = b+2*c*r+3*d*r*r;
                        return;
                }

                else{
                        *energy = cf-0+cf-1*r+cf-2*r*r+cf-3*r*r*r+cf-4*r*r*r*r;
                        *force = cf-1+2*cf-2*r+3*cf-3*r*r+4*cf-4*r*r*r;
                        return;
                }
        }

        else{
                *energy=*force=0.0;
                return;
        }

}


/**
* compute V"[r]
* input:
* r-separation
* P-material-material data
* output:
* V"[r]
*/

double eam-force-const(const double r,
const struct material-t *P-material)
{

        if(r >material->r-cutoff) return(0.0);
        else (2*cf-2+6*cf-3*r+12*cf-4*r*r);

}
```

C.2. Interacciones relativas a la densidad

```
static pthread-once-t density-once=PTHREAD-ONCE-INIT;
static const struct material-t *material=NULL;
static double r-i;
static int n2=100;
static double n=0.15;
static double a;
static double b;
static double c;
static double d;
static int i-init=0;

/* initialize static data */

static void density-init(void)
{
        r-i=0.1*material->a*M-SQRT1-2;

        /* spline coeffs */

        a=(n2-1)*(n-material->ec)*(n-material->ec)*(material->ec-r-i)*
(material->ec-r-i)*(material->ec-r-i)/((n-r-i)*(n-r-i)*(n-r-i));
        b=3*(n2-1)*(material->ec-n)*(material->ec-n)*(material->ec-r-i)*
(material->ec-r-i)/((n-r-i)*(n-r-i)*(n-r-i));
        c=1+3*(n2-1)*(material->ec-n)*(material->ec-n)*(material->ec-r-i)/
((n-r-i)*(n-r-i)*(n-r-i));
        d=(n2-1)*(material->ec-n)*(material->ec-n)/((n-r-i)*(n-r-i)*(n-r-i));

        i-init=1;

        return; }

/* calculate density weight function and its derivative */

void density(double *w,
double *w-prime,
const double r,
const struct material-t *P-material)
{

        double exponent;
        double dr;
```

```
        if(material == NULL) material=P-material;

        if(i-init==0) pthread-once( &density-once, density-init );

        if( r <= P-material->ec ){

            if( r <= r-i ){
                *w=a+b*(r-P-material->ec)+c*(r-P-material->ec)*
(r-P-material->ec)+d*(r-P-material->ec)*(r-P-material->ec)*(r-P-material->ec);
                *w-prime=b*(r-P-material->ec)+2*c*(r-P-material->ec)+
3*d*(r-P-material->ec)*(r-P-material->ec);
                return;
            }

            else {
                *w=(r-P-material->ec)*(r-P-material->ec);
                *w-prime=2*(r-P-material->ec);
                return;
            }
        }

        else {
            *w=*w-prime=0.0;
            return;
        }
    }


    /* compute embeding function */

    void embeding-function(double *f,
    double *f-p,
    const double rho)
    {

        /* compute embedding function */

        *f=-material->fe*sqrt(rho);
        if (rho == 0) *f-p = 0;
        else *f-p=-material->fe*0.5/(sqrt(rho));
        return;

    }
```

```
/* compute the 2nd and 3rd derivative of the density weight function */

void density-derivatives(double *w-2,
double *w-3,
const double r,
const struct material-t *P-material)
{

        double exponent;

        if(i-init==0) pthread-once( &density-once, density-init );

        if(r <= P-material->ec){
                *w-2 = 2;
                *w-3 = 0;
                return;
        }

        else {
                *w-2 = *w-3 = 0.0;
                return;                 }

}


void embeding-function-derivatives(double *f-2,
double *f-3,
const double rho)
{

        if (rho == 0){
                *f-2=0;
                *f-3=0;
        }

        else{
                *f-2 =material->fe*0.25*pow(rho,-1.5);
                *f-3 =-material->fe*0.375*pow(rho,-2.5);
        }

        return;             }
```