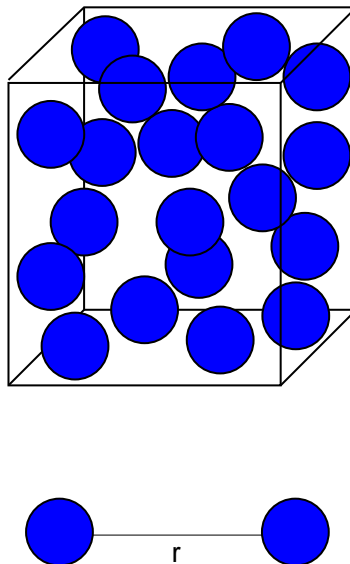


## 8 Molekulardynamik

### 8.1 Modell

Integration der Newtonschen Bewegungsgleichungen, d.h. keine Quanteneffekte. Beispiele: Simulation von Biomolekülen oder von Galaxien. Hier: System von Gasatomen in einem Kasten.



Lennard-Jones Energie: 2 Atome  $\underline{r}_i, \underline{r}_j$  mit Abstand  $r := |\underline{r}| := |\underline{r}_i - \underline{r}_j|$

$$V(r) = 4\epsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^6 \right] \quad (64)$$

$\epsilon$ : Stärke der Wechselwirkung.

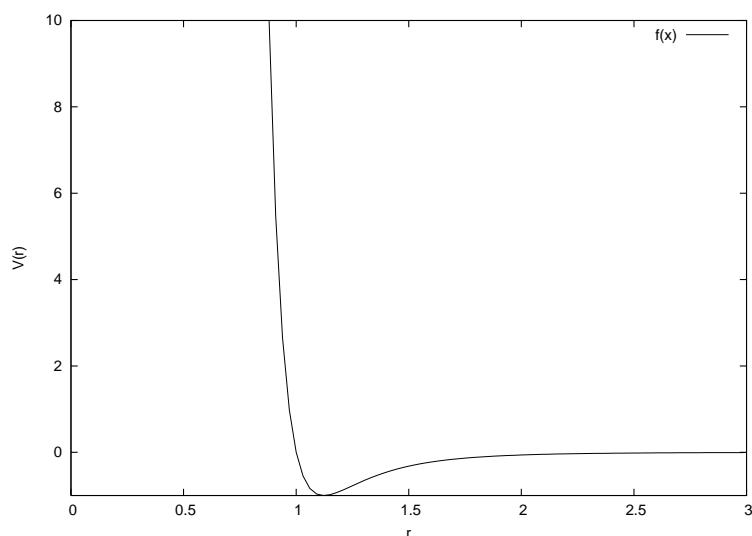
$\sigma$ : “Größe” der Atome.

$r^{-6}$  Term: van der Waals Wechselwirkung (anziehend), durch Ladungsfluktuationen, die Dipolmomente erzeugen (Dipolfeld  $\sim 1/r^3$ ).

$r^{-12}$  Term: “hard-core” Abstoßung, heuristisch (bequem, da  $r^{-12} = (r^{-6})^2$ ).

Minimum bei  $r_{\min} = 2^{1/6}\sigma \approx 1.125\sigma$ ,  $V(r_{\min}) = -\epsilon$ .

Bei Wechselwirkung von verschiedenen Atomsorten  $A, B$ . Heuristik:

Figure 19: Lennard-Jones Potential für  $\epsilon = \sigma = 1$ .

$$\sigma_{AB} = \frac{\sigma_A + \sigma_B}{2}, \quad \epsilon_{AB} = \sqrt{\epsilon_A \epsilon_B} \quad (65)$$

Gesamtenergie  $N$  Atome:  $V_{\text{ges}}(\underline{r}_1, \dots, \underline{r}_N) = \sum_{i < j} V(|\underline{r}_i - \underline{r}_j|)$

Um Randeffekte zu minimieren und “großes” System vorzugucken: *periodische Randbedingungen*. Man stellt sich vor, dass der Kasten in alle Richtungen unendlich oft wiederholt wird  $\rightarrow$  bei der Abstandsberechnung muß die *Minimum Image Konvention* angewendet werden: der Abstand zweier Teilchen ist das Minimum zwischen den Abständen aller wiederholten Teilchen. .

## 8.2 Datenstrukturen

---

[Selbsttest]

---

Überlegen Sie: Welche Daten muss man pro Atom speichern ?

Lesen Sie erst weiter, wenn Sie sich selber Gedanken gemacht haben

---

Für jedes Atom braucht man verschiedene Daten:

```
/* stores data of one atom: */
typedef struct
{
    double      m;          /* mass of atom */
    double      sigma;      /* 'size' of atom for LJ */
}
```

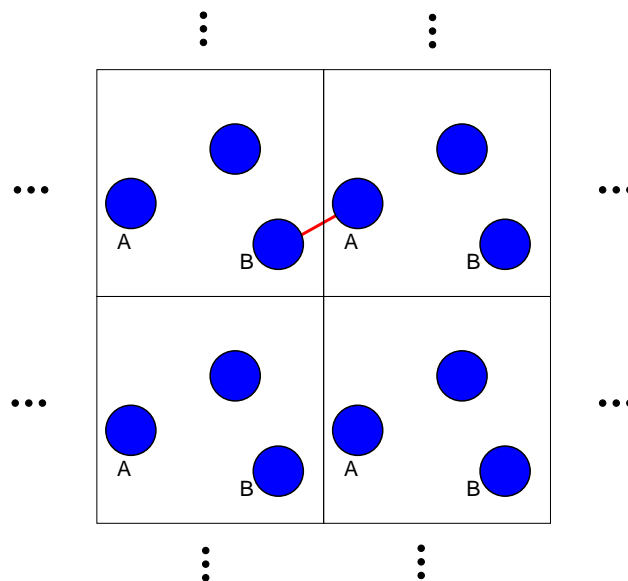


Figure 20: Periodische Randbedingungen: das System wiederholt sich in allen Richtungen. Minimum Image Konvention: der kürzeste Abstand zwischen den Teilchen A;B ist durch eine Linie dargestellt.

```
double  epsilon;      /* LJ energy parameter */
double   *x;          /* position of atom */
double   *v;          /* velocity of atom */
double   *f;          /* force on atom */
} glas_atom_t;
```

---

[Selbsttest]

---

Überlegen Sie: Welche Daten braucht man für das Gesamtsystem  
Lesen Sie erst weiter, wenn Sie sich selber Gedanken gemacht haben

---

Globale System-Datenstrukturen. Zusammenfassung in einer Struktur, damit bei Übergeben an Unterprogramme nur ein Zeiger übergeben werden muß.

```
/* stores all global system data: */
typedef struct
{
    int          dim;      /* dimension of system */
    double       *l;       /* sizes of system */
    double       *lh;      /* half sizes of system */
    int          N;        /* total number of particles */
    double       tau;      /* integration step size */
}
```

```

    double          tau2;          /* integration step size^2 */
    double          T;             /* temperature */
} glas_system_t;

```

Die Daten aller Atom werden als ein Array von Elementen des Typs `glas_atom_t` gespeichert. Dieses Array wird durch folgendes Unterprogramm erzeugt und initialisiert. Atome werden zufällig verteilt und mit Geschwindigkeit 0 gestartet.

```

/***** glas_setup() *****/
/** Creates and initialises atom data          */
/** PARAMETERS: (*)= return-paramter          */
/**      system: global system parameters      */
/** RETURNS:                                   */
/**      array of atom data                    */
/*****
glas_atom_t *glas_setup(glas_system_t *system)
{
    glas_atom_t *atom;          /* here all atom data is stored */
    int t, t2, d;               /* loop counters */

    atom = (glas_atom_t *) malloc(system->N*sizeof(glas_atom_t));
    for(t=0; t<system->N; t++)   /* initialize atoms */
    {
        atom[t].m = 1;
        atom[t].sigma = 1;
        atom[t].epsilon = 1;
        atom[t].x = (double *) malloc(system->dim*sizeof(double));
        atom[t].v = (double *) malloc(system->dim*sizeof(double));
        atom[t].f = (double *) malloc(system->dim*sizeof(double));
        for(d=0; d<system->dim; d++) /* put atom randomly */
        {
            atom[t].x[d] = drand48()*system->l[d];
            atom[t].v[d] = 0;
        }
    }
    return(atom);
}

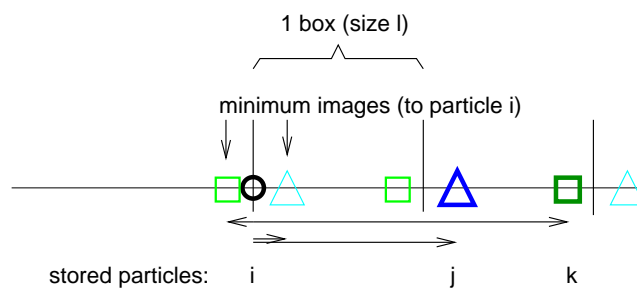
```

### 8.3 Kräfte

Berechnen Sie durch Gradientenbildung:  
Kraft von Teilchen  $j$  auf Teilchen  $i$ :

---

Hinweis: Teilchen können beliebig weit wandern  $\rightarrow$  minimum Image Bedingung aufwendiger:  
Man muss ganzzahlige Vielfache der Systemgröße abziehen (oder dazuaddieren):



Unterprogramm für Kraft- und Energieberechnung (siehe `md_fragent2.c`):

```

/***** glas_energy_forces() *****/
/** Calculates potential energy and forces of system **/
/** PARAMETERS: (*)= return-paramter **/
/** system: global system parameters **/
/** atom: atom data **/
/** RETURNS: **/
/** energy **/
/*****/
double glas_energy_forces(glas_system_t *system, glas_atom_t *atom)
{
    double energy = 0.0, force;
    double epsilon4, sigma2; /* interaction parameters */
    int t, t2, d; /* loop counter */
    double *r; /* difference vector between two atoms */
    double r2, rm6; /* distance^2, distance^-6 */

    r = (double *) malloc(system->dim*sizeof(double));
    for(t=0; t<system->N; t++) /* loop over all atoms */
        for(d=0; d<system->dim; d++)
            atom[t].f[d] = 0; /* initialise force */

    for(t=0; t<system->N; t++) /* loop over all pairs of atoms */
        for(t2=t+1; t2<system->N; t2++)
        {
            r2 = 0.0; d = 0; /* calculate distance */
            while(d<system->dim)
            {
                r[d] = atom[t].x[d] - atom[t2].x[d];
                if(r[d] < -system->lh[d]) /* minimum image convention */
                    r[d] += floor(-r[d]/system->l[d]+0.5)*system->l[d];
                if(r[d] > system->lh[d])
                    r[d] -= floor(r[d]/system->l[d]+0.5)*system->l[d];
                r2 += r[d]*r[d];
                d++;
            }
            sigma2 = 0.5*(atom[t].sigma+atom[t2].sigma);
            sigma2 = sigma2*sigma2;
            epsilon4 = 4.0*sqrt(atom[t].epsilon * atom[t2].epsilon);
            r2 /= sigma2; rm6 = 1.0/(r2*r2*r2);
            energy += epsilon4*(rm6*(rm6-1.0)); /* calculate energy */
            for(d=0; d<system->dim; d++)
            {
                force = 6*epsilon4*(rm6*(2*rm6-1.0))*r[d]/(r2*sigma2);
                atom[t].f[d] += force; /* and forces */
                atom[t2].f[d] -= force;
            }
        }
    free(r); return(energy);
}

```

## 8.4 Integration der Bewegungsgleichung

Methode: endliche Differenzen mit Schrittweite  $\tau$ . Ziel: Integration mit Fehler  $O(\tau^3)$ . (Höhere Ordnung nicht nötig, da Geschwindigkeiten für  $T = \text{const}$  reskaliert werden.)

Ausgangspunkt:  $\dot{x} = v$ ,  $\ddot{x} = F/m$ . Taylorentwicklung

$$x(t + \tau) = x(t) + v(t)\tau + \frac{1}{2} \frac{F(t)}{m} \tau^2 + O(\tau^3) \quad (66)$$

Gesucht: Gleichung für  $v(t)$ . Taylorentwicklung:

$$v(t + \tau) = v(t) + \frac{F(t)}{m} \tau + \frac{1}{2} \ddot{v}(t) \tau^2 + O(\tau^3) \quad (67)$$

Term  $\ddot{v}(t)$  unbekannt, daher Entwicklung um  $t + \tau$ :

$$v(t) = v(t + \tau) - \tau \frac{F(t + \tau)}{m} + \frac{1}{2} \ddot{v}(t + \tau) \tau^2 + O(\tau^3) \quad (68)$$

Berechne (67)-(68)

$$v(t + \tau) - v(t) = v(t) - v(t + \tau) + \left( \frac{F(t)}{m} + \frac{F(t + \tau)}{m} \right) \tau + \frac{1}{2} (\ddot{v}(t) - \ddot{v}(t + \tau)) \tau^2 + O(\tau^3) \quad (69)$$

Taylorentwicklung von  $\ddot{v}(t + \tau) = \ddot{v}(t) + \frac{d}{dt} \ddot{v}(t) \tau + O(\tau^2)$  zeigt: letzter Term ist  $O(\tau^3)$ , also ist

$$v(t + \tau) = v(t) + \frac{1}{2} \left( \frac{F(t)}{m} + \frac{F(t + \tau)}{m} \right) \tau + O(\tau^3) \quad (70)$$

Gleichungen (66) und (70) heißen das *Velocity-Verlet* Verfahren.

## 8.5 Temperatur

Gleichverteilungssatz: im Mittel  $\frac{1}{2} k_B T$  kinetische Energie pro Freiheitsgrad, d.h.

$$\sum_i \frac{1}{2} m_i \langle \dot{\underline{r}}_i^2 \rangle = \frac{3N}{2} k_B T \quad (71)$$

Problem: Integration der Bewegungsgleichungen lässt Gesamtenergie konstant.

Einfachste Lösung: Reskalierung der Geschwindigkeiten, so dass (71) erfüllt ist, d.h. man multipliziert jede Geschwindigkeit mit

$$\lambda = \sqrt{\frac{3N k_B T}{\sum_i m_i \dot{\underline{r}}_i^2}} \quad (72)$$

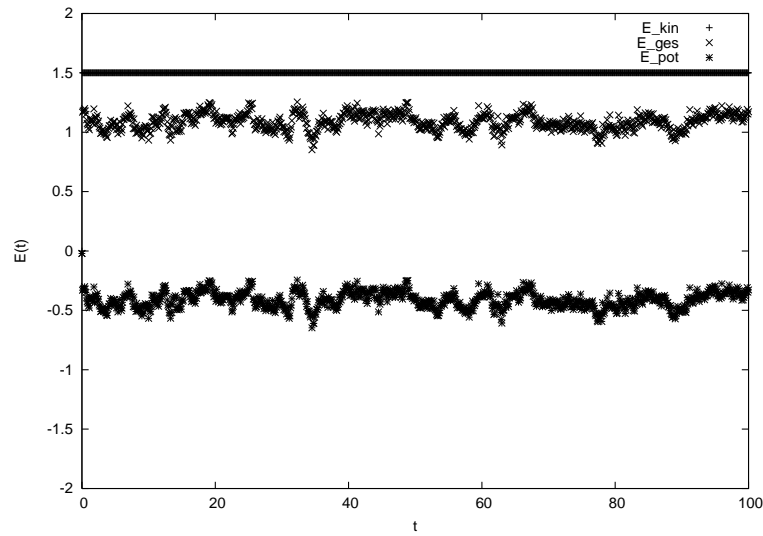


Figure 21: Kinetische, Gesamt- und potentielle Energie eines LJ Systems mit Reskalierung auf  $k_B T = 1$  ( $L = 10, 50$  Teilchen,  $m = 1$ ,  $\epsilon = 1$ ,  $\sigma = 1$ ,  $\tau = 0.001$ ).

Nachteil: Kinetische Energie = konstant.  $\rightarrow$  Nosé-Hoover Thermostat. Grundidee: Einführung eines “Temperatur-Bad” Teilchens, das mit allen Atomen wechselwirkt und falls  $\sum_i m_i \underline{v}_i^2 < 3Nk_B T$  Energie langsam zuführt (bzw abführt).