Il metodo di Runge-Kutta

Riassumendo possiamo dire che il metodo di Runge-Kutta di ordine due consiste nell'eseguire una estrapolazione del primo ordine da $x(0)$ a $x(\Delta t/2)$, nel valutare la derivata $x'(\Delta t/2)$ e nell'utilizzarla per ottenere una stima di $x(\Delta t)$ esatta al secondo ordine.

Vediamo la sequenza di calcolo:

$$k_1 = \Delta t f(x(0),0)$$

Calcolo della derivata nel punto iniziale

$$k_2 = \Delta t f\left(x(0) + \frac{k_1}{2}, \frac{\Delta t}{2}\right)$$

Estrapolazione al primo ordine in $\Delta t/2$ e calcolo della derivata

$$x(\Delta t) = x(0) + k_2 + O(\Delta t^3)$$

Estrapolazione al secondo ordine in Δt

Runge-Kutta di ordine quattro

Esiste un metodo di Runge-Kutta esatto al quart'ordine, del quale riportiamo solo la sequenza di calcolo, ma che è sempre conveniente usare:

$$k_1 = \Delta t f(x(0),0)$$

$$k_2 = \Delta t f\left(x(0) + \frac{k_1}{2}, \frac{\Delta t}{2}\right)$$

$$k_3 = \Delta t f\left(x(0) + \frac{k_2}{2}, \frac{\Delta t}{2}\right)$$

$$k_4 = \Delta t f(x(0) + k_3, \Delta t)$$

$$x(\Delta t) = x(0) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5)$$

Il calcolo della traiettoria

Il problema che dobbiamo risolvere é:

$$\vec{x}_i''(t) = \frac{1}{m_i}\vec{F}_i\left(\vec{x}_1'(t),\ldots,\vec{x}_N'(t),\vec{x}_1(t),\ldots,\vec{x}_N(t),t\right)$$

che, come abbiamo visto, va riscritto nella forma:

$$\begin{cases}\vec{x}_i'(t) = \vec{v}_i(t)\\ \vec{v}_i'(t) = \vec{f}_i\left(\vec{v}_1(t),\ldots,\vec{v}_N(t),\vec{x}_1(t),\ldots,\vec{x}_N(t),t\right)\end{cases}$$

Se abbiamo N corpi ($i=1\ldots N$) e risolviamo il problema in tre dimensioni si tratta di un sistema di $2x3xN$ equazioni del primo ordine.

Dobbiamo osservare che la quasi totalità dell' interdipendenza tra le equazioni del sistema risiede nel termine di forza: la componente j-ma della forza che agisce sul punto i è funzione di tutte le componenti delle posizioni e delle velocità di tutti gli N corpi.

Soluzione del sistema

Visto che siamo capaci di risolvere una singola equazione differenziale, la soluzione di un sistema di molte equazioni non presenta particolari problemi. Di fatto il metodo che usiamo è rigidamente sequenziale, per cui si deve solo fare attenzione ad eseguire ogni passo del calcolo su tutte le equazioni prima di passare al passo successivo. Proviamo con il metodo di Runge-Kutta di ordine due.

$$\vec{k}_{1i} = \Delta t \vec{v}_i(0) \qquad \vec{w}_{1i} = \Delta t \vec{f}_i(\vec{v}_1(0),\ldots,\vec{v}_N(0),\vec{x}_1(0),\ldots,\vec{x}_N(0),0)$$

$$\vec{k}_{2i} = \Delta t\left(\vec{v}_i(0) + \frac{\vec{w}_{1i}}{2}\right) \qquad \vec{w}_{2i} = \Delta t \vec{f}_i\left(\vec{v}_1(0) + \frac{\vec{w}_{11}}{2},\ldots,\vec{x}_1(0) + \frac{\vec{k}_{11}}{2},\ldots,\frac{\Delta t}{2}\right)$$

$$\vec{x}_i(\Delta t) = \vec{x}_i(0) + \vec{k}_{2i} + O(\Delta t^3) \qquad \vec{v}_i(\Delta t) = \vec{v}_i(0) + \vec{w}_{2i} + O(\Delta t^3)$$

Manipolazione dei vettori

Il programma deve manipolare vettori di DxN componenti; sia D che N possono variare, per cui è necessario allocare dinamicamente. Ci sono due soluzioni possibili:

Soluzione 2: matrice [N][D]

Allocazione

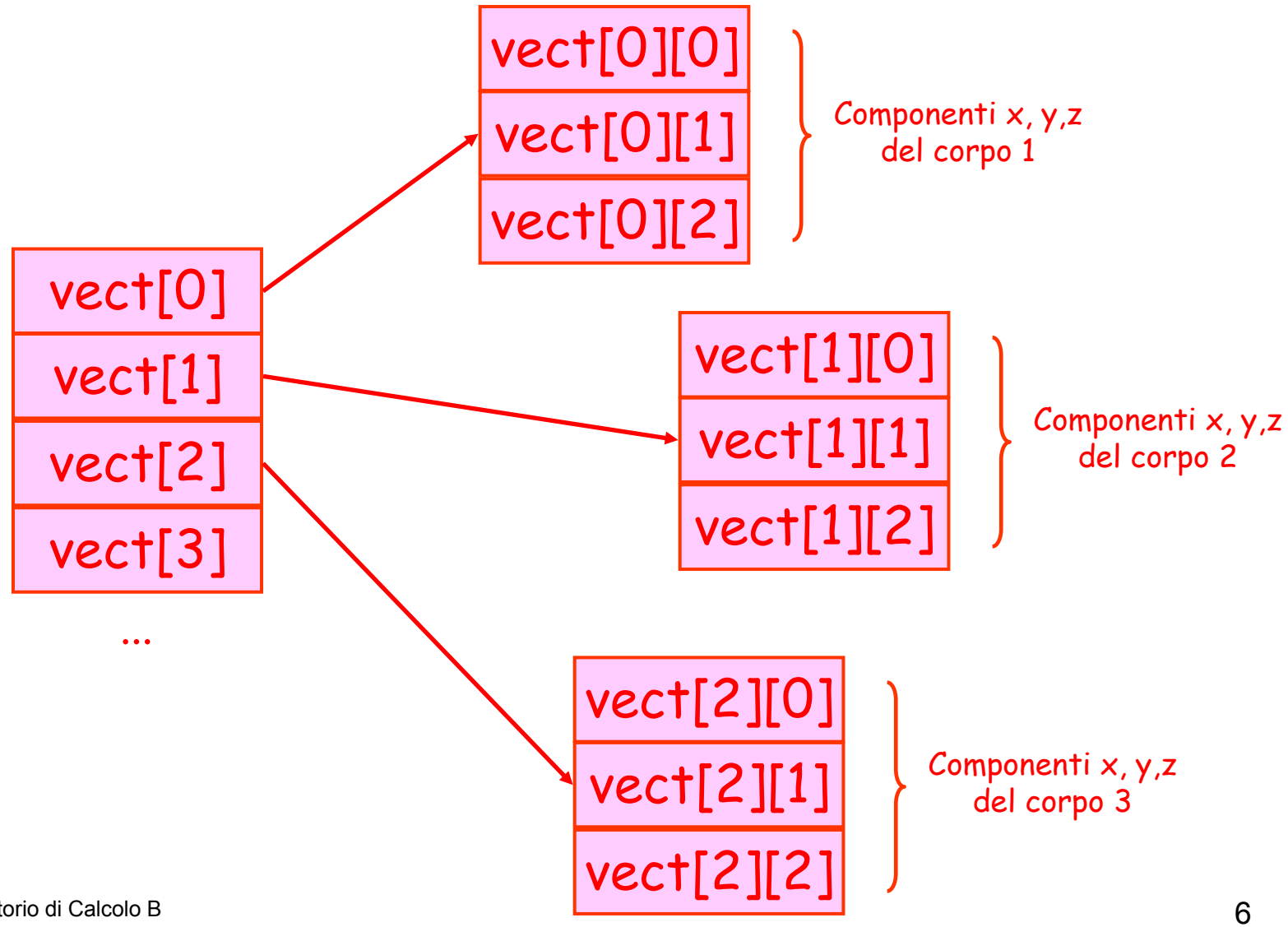
```
double **vect;
int n;
vect = new double*[N];
for (n=0; n<N; n++) {
   vect[n] = new double[D];
}
```

Loop

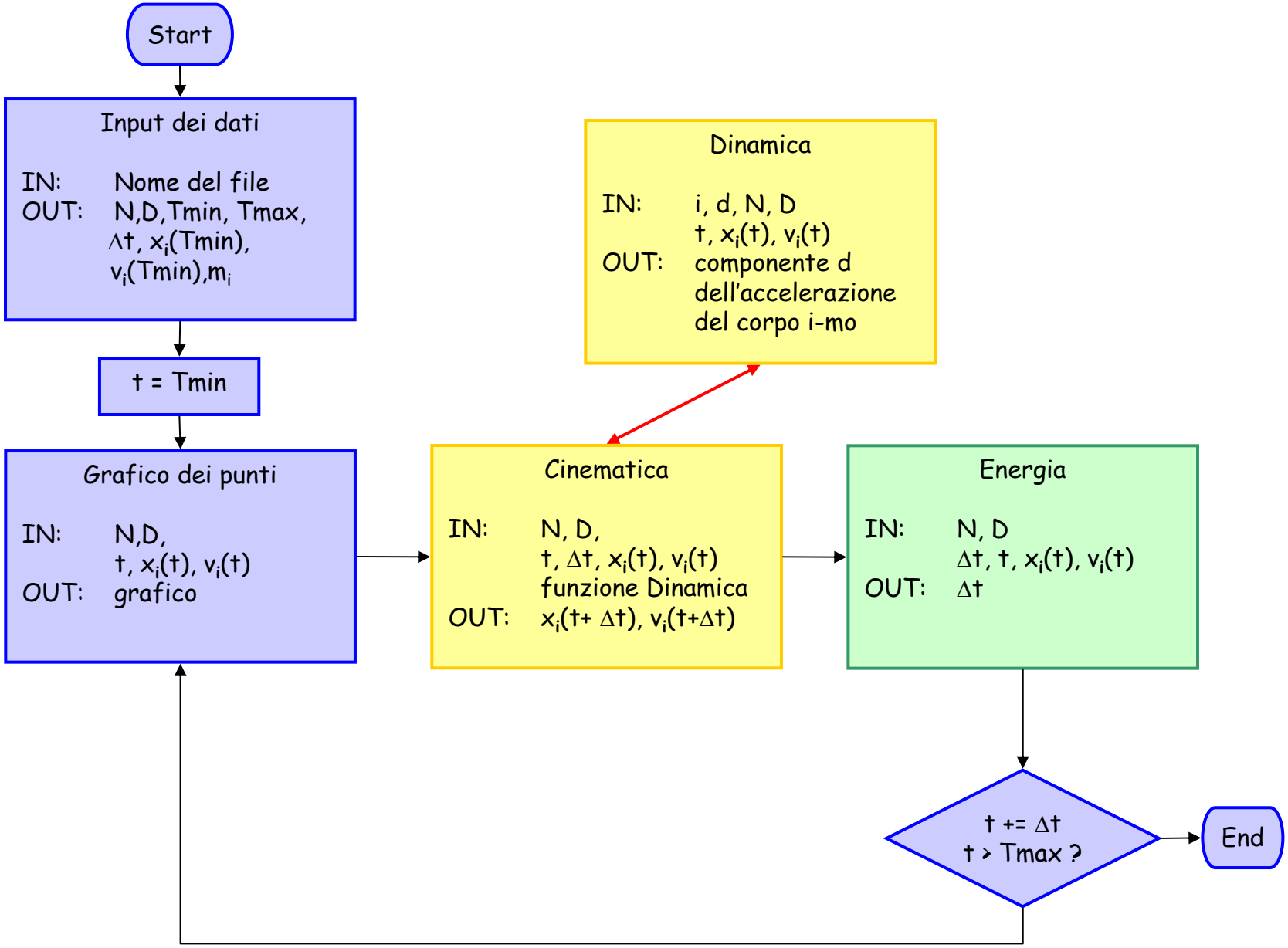
```
int n,d;
for (n=0; n<N; n++) {
    for (d=0; d<D; d++) {
          vect[n] [d] = ....
    }
}
```

Manipolazione dei vettori

Soluzione 2: matrice [N][D]



Schema a blocchi completo



Suggerimenti: input

Qui non ci sono particolari difficoltà. La funzione deve:

- Leggere N e D
- Fabbricare (con allocazione dinamica) i vettori che servono e che, verosimilmente, saranno variabili globali (x, v, m, w, k…)
- Leggere i valori iniziali di x e v e le masse m

Siccome non ci sono novità importanti il consiglio è di scrivere una function che fabbrichi i vettori e metta dei valori iniziali particolari, e poi scrivere la lettura da file alla fine.

Suggerimenti: cinematica

La funzione deve eseguire uno step con il metodo di Runge-Kutta di ordine quattro, utilizzando una funzione esterna per il calcolo della forza.

A fianco trovate un esempio del calcolo dei primi due passi del metodo di ordine due.

Alla fine delle operazioni il vettore x, che all'inizio contiene x(t), deve contenere x(t+dt) e analogamente per v.

Se implementate il controllo dell'energia totale vi conviene però lasciare il risultato in xtmp e vtmp ed eseguire la copia nella reoutine di controllo. In questo modo, se necessario, potete rifare un passo di cinematica con un dt diverso se necessario.

```
// Passo 1
for (i=0; i<N; i++) {
  for (d=0; d<D; d++) {
    k1[i][d] = dt*v[i][d];
    w1[i][d] = accel(i,d,x,v,t);
  }
}
// Passo 2
for (i=0; i<N; i++) {
  for (d=0; d<D; d++) {
    vtmp[i][d] = v[i][d]+w1[i][d]/2;
    xtmp[i][d] = x[i][d]+k1[i][d]/2;
    k2[i][d] = dt*vtmp[i][d];
    w2[i][d] = …
  }
}
```

Suggerimenti: dinamica

La funzione deve calcolare la componente d-ma della forza agente sulla particella i-ma.

In un problema a N corpi questa forza è la somma delle interazioni con gli N-1 corpi più un eventuale contributo di un campo esterno.

Converrà quindi creare funzioni esplicite per le interazioni e per il campo esterno, che andranno poi modificate di volta in volta a seconda del problema specifico.

Di seguito trovate una traccia della scomposizione in funzioni

```
// Definizioni
double accel(int i, int d, double **x, double **v, double t);
double fInter(int i, int j, int d, double **x, double **v, double t);
double fExt(int i, int d, double **x, double **v, double t);
// Traccia per accel
double accel(int i, int d, double **x, double **v, double t) {
  double ftot = 0.0;
  for (j=0; j<N; j++) {
    if (i != j) ftot += fInter(i,j,d,x,v,t);
  }
  ftot += fExt(i,d,x,v,t);
  return ftot/mass[i];
}
```