Riassunto: soluzione equ. differenziali di primo grado

Problema soluzione numerica dell'equazione differenziale:

$$x^{'}(t) = f(x(t), t)$$

dato $x_0 = x(t_0)$.

Soluzione iterativa utilizzando l'equazione differenziale e lo sviluppo in serie di Taylor: relazione tra posizioni a distanza temporale h:
$x_{n+1} \leftrightarrow x_n$ (dove $x_{n+1} \equiv x(t_n + h)$ e $x_n \equiv x(t_n)$)

Riassunto: soluzione equ. differenziali di primo grado

Problema soluzione numerica dell'equazione differenziale:

$$x^{'}(t) = f(x(t), t)$$

dato $x_0 = x(t_0)$.

Soluzione iterativa utilizzando l'equazione differenziale e lo sviluppo in serie di Taylor: relazione tra posizioni a distanza temporale h: $x_{n+1} \leftrightarrow x_n$ (dove $x_{n+1} \equiv x(t_n + h)$ e $x_n \equiv x(t_n)$)

Metodo di Eulero

$$k_1 = hf(x_n, t_n) \qquad \textit{derivata nel punto iniziale}$$

$$x_{n+1} = x_n + k_1 + \mathcal{O}(h^2) \qquad \textit{estrapolazione a } t_n + h$$

Riassunto: soluzione equ. differenziali di primo grado

Problema soluzione numerica dell'equazione differenziale:

$$x^{'}(t) = f(x(t), t)$$

dato $x_0 = x(t_0)$.

Soluzione iterativa utilizzando l'equazione differenziale e lo sviluppo in serie di Taylor: relazione tra posizioni a distanza temporale h:
$x_{n+1} \leftrightarrow x_n$ (dove $x_{n+1} \equiv x(t_n + h)$ e $x_n \equiv x(t_n)$)

Metodo di Runge Kutta al second'ordine

$$k_1 = hf(x_n, t_n) \qquad \textit{derivata nel punto iniziale}$$

$$k_2 = hf(x_n + \frac{k_1}{2}, t_n + \frac{h}{2}) \qquad \textit{estrap. al punto intermedio e derivata}$$

$$x_{n+1} = x_n + k_2 + \mathcal{O}(h^3) \qquad \textit{estrapolazione a } t_n + h$$

Riassunto: soluzione equ. differenziali di primo grado

Problema soluzione numerica dell'equazione differenziale:

$$x^{'}(t) = f(x(t), t)$$

dato $x_0 = x(t_0)$.

Soluzione iterativa utilizzando l'equazione differenziale e lo sviluppo in serie di Taylor: relazione tra posizioni a distanza temporale h:
$x_{n+1} \leftrightarrow x_n$ (dove $x_{n+1} \equiv x(t_n + h)$ e $x_n \equiv x(t_n)$)

Metodo di Runge Kutta al quart'ordine

$$k_1 = hf(x_n, t_n) \qquad k_2 = hf\left(x_n + \frac{k_1}{2}, t_n + \frac{h}{2}\right)$$

$$k_3 = hf\left(x_n + \frac{k_2}{2}, t_n + \frac{h}{2}\right) \qquad k_4 = hf(x_n + k_3, t_n + h)$$

$$x_{n+1} = x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathcal{O}(h^5)$$

Riassunto: soluzione equ. diff. di ordini superiori

Una generica equazione differenziale del secondo ordine è del tipo

$$x''(t) = f(t, x(t), x'(t))$$

Si può trasformare in un sistema di due equazioni del primo ordine ponendo

$$x'(t) = v(t)$$

si ottiene

$$\begin{cases} x'(t) = v(t) \\ v'(t) = a(t, x(t), v(t)) \end{cases}$$

che è un sistema del primo ordine nelle variabili $x(t)$ e $v(t)$. Ovviamente lo stesso "trucco" funziona per tutti gli ordini.

Riassunto: soluzione equ. diff. per N corpi in D dimensioni

La soluzione di un sistema di equazioni non presenta particolari problemi.
Lo sviluppo riportato di seguito (per RK al secondo ordine) richiede che le operazioni siano rigidamente sequenziali: si deve quindi fare attenzione ad eseguire ogni passo del calcolo su tutte le equazioni prima di passare al passo successivo.

$$\text{I} \begin{cases} \vec{k_{1i}} = h\vec{v_i}(t_n) \\ \vec{w_{1i}} = h\vec{f_i}(\vec{v_1}(t_n), ..., \vec{x_1}(t_n), ..., t_n) \end{cases}$$

$$\text{II} \begin{cases} \vec{k_{2i}} = h\left(\vec{v_i}(t_n) + \frac{\vec{w_{1i}}}{2}\right) \\ \vec{w_{2i}} = h\vec{f_i}(\vec{v_1}(t_n) + \frac{\vec{w_{11}}}{2}, ..., \vec{x_1}(t_n) + \frac{\vec{k_{11}}}{2}, ..., t_n + \frac{h}{2}) \end{cases}$$

$$\text{III} \begin{cases} \vec{x_i}(t_n + h) = \vec{x_i}(t_n) + \vec{k_{2i}} + \mathcal{O}(h^3) \\ \vec{v_i}(t_n + h) = \vec{v_i}(t_n) + \vec{w_{2i}} + \mathcal{O}(h^3) \end{cases}$$

Riassunto: soluzione equ. diff. per N corpi in D dimensioni

La soluzione di un sistema di equazioni non presenta particolari problemi.
Lo sviluppo riportato di seguito (per RK al secondo ordine) richiede che le operazioni siano rigidamente sequenziali: si deve quindi fare attenzione ad eseguire ogni passo del calcolo su tutte le equazioni prima di passare al passo successivo.

$$\text{I} \begin{cases} \vec{k_{1i}} = h\vec{v_i}(t_n) \\ \vec{w_{1i}} = h\vec{f_i}(\vec{v_1}(t_n), ..., \vec{x_1}(t_n), ..., t_n) \end{cases}$$

$$\text{II} \begin{cases} \vec{k_{2i}} = h\left(\vec{v_i}(t_n) + \frac{\vec{w_{1i}}}{2}\right) \\ \vec{w_{2i}} = h\vec{f_i}(\vec{v_1}(t_n) + \frac{\vec{w_{11}}}{2}, ..., \vec{x_1}(t_n) + \frac{\vec{k_{11}}}{2}, ..., t_n + \frac{h}{2}) \end{cases}$$

$$\text{III} \begin{cases} \vec{x_i}(t_n + h) = \vec{x_i}(t_n) + \vec{k_{2i}} + \mathcal{O}(h^3) \\ \vec{v_i}(t_n + h) = \vec{v_i}(t_n) + \vec{w_{2i}} + \mathcal{O}(h^3) \end{cases}$$

I: calcolo di $\vec{x_i'}(t_n)$ (banale: $\vec{x_i'}(t_n) = \vec{v}(t_n)$)
calcolo di $\vec{v_i'}(t_n)$ usando l'equazione differenziale

Riassunto: soluzione equ. diff. per N corpi in D dimensioni

La soluzione di un sistema di equazioni non presenta particolari problemi.
Lo sviluppo riportato di seguito (per RK al secondo ordine) richiede che le operazioni siano rigidamente sequenziali: si deve quindi fare attenzione ad eseguire ogni passo del calcolo su tutte le equazioni prima di passare al passo successivo.

$$\text{I} \begin{cases} \vec{k_{1i}} = h\vec{v_i}(t_n) \\ \vec{w_{1i}} = h\vec{f_i}(\vec{v_1}(t_n), ..., \vec{x_1}(t_n), ..., t_n) \end{cases}$$

$$\text{II} \begin{cases} \vec{k_{2i}} = h\left(\vec{v_i}(t_n) + \frac{\vec{w_{1i}}}{2}\right) \\ \vec{w_{2i}} = h\vec{f_i}(\vec{v_1}(t_n) + \frac{\vec{w_{11}}}{2}, ..., \vec{x_1}(t_n) + \frac{\vec{k_{11}}}{2}, ..., t_n + \frac{h}{2}) \end{cases}$$

$$\text{III} \begin{cases} \vec{x_i}(t_n + h) = \vec{x_i}(t_n) + \vec{k_{2i}} + \mathcal{O}(h^3) \\ \vec{v_i}(t_n + h) = \vec{v_i}(t_n) + \vec{w_{2i}} + \mathcal{O}(h^3) \end{cases}$$

II: si estrapolano $\vec{x_i}$ e $\vec{v_i}$ al primo ordine a $t_n + h/2$ e si calcolano $\vec{x_i'}(t_n + h/2)$ e $\vec{v_i'}(t_n + h/2)$.

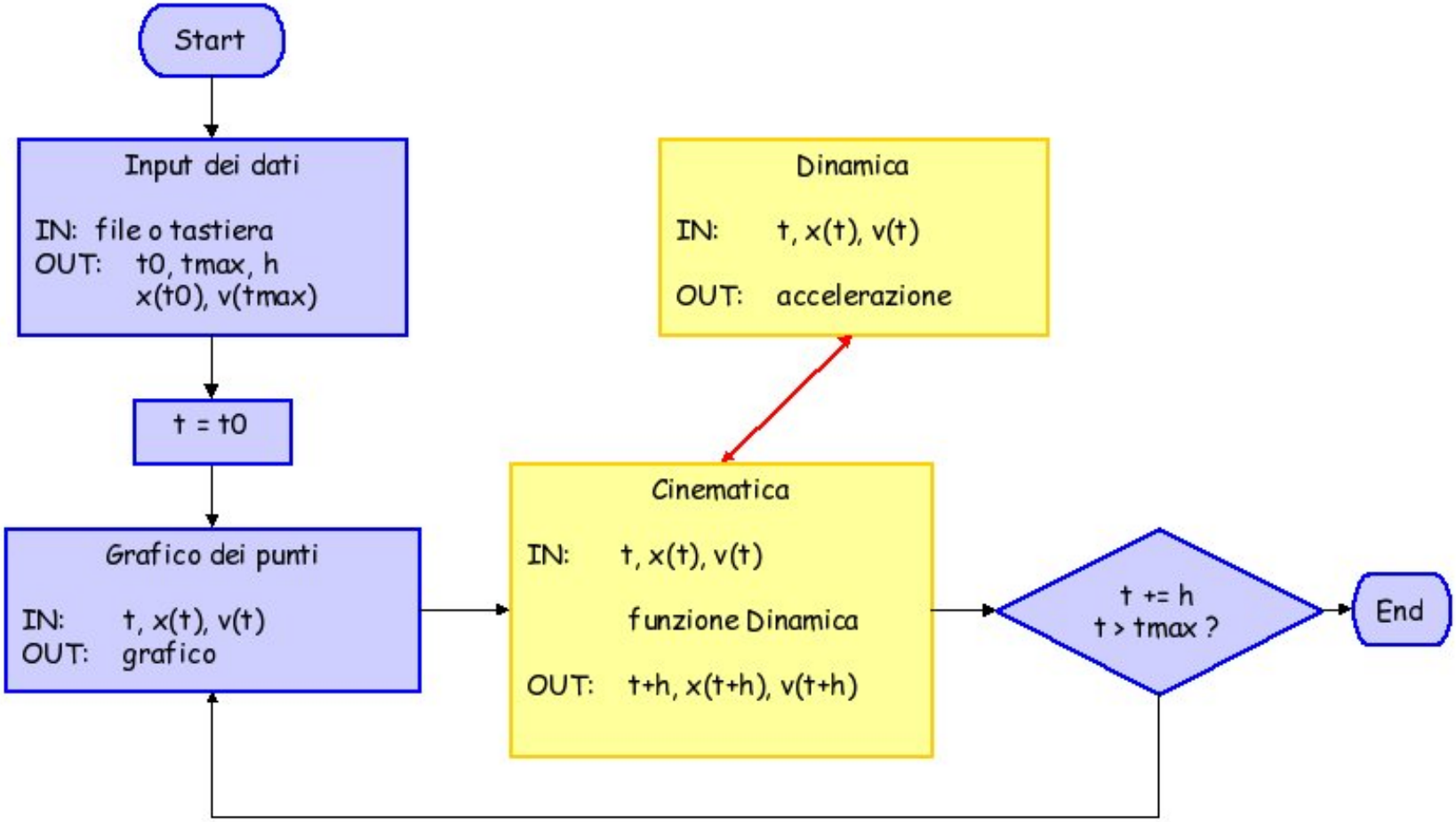
Riassunto: soluzione equ. diff. per N corpi in D dimensioni

La soluzione di un sistema di equazioni non presenta particolari problemi.
Lo sviluppo riportato di seguito (per RK al secondo ordine) richiede che le operazioni siano rigidamente sequenziali: si deve quindi fare attenzione ad eseguire ogni passo del calcolo su tutte le equazioni prima di passare al passo successivo.

$$
\begin{array}{ll}
\text{I} & \begin{cases} \vec{k_{1i}} = h\vec{v_i}(t_n) \\ \vec{w_{1i}} = h\vec{f_i}(\vec{v_1}(t_n), ..., \vec{x_1}(t_n), ..., t_n) \end{cases} \\
\text{II} & \begin{cases} \vec{k_{2i}} = h\left(\vec{v_i}(t_n) + \frac{\vec{w_{1i}}}{2}\right) \\ \vec{w_{2i}} = h\vec{f_i}(\vec{v_1}(t_n) + \frac{\vec{w_{11}}}{2}, ..., \vec{x_1}(t_n) + \frac{\vec{k_{11}}}{2}, ..., t_n + \frac{h}{2}) \end{cases} \\
\text{III} & \begin{cases} \vec{x_i}(t_n + h) = \vec{x_i}(t_n) + \vec{k_{2i}} + \mathcal{O}(h^3) \\ \vec{v_i}(t_n + h) = \vec{v_i}(t_n) + \vec{w_{2i}} + \mathcal{O}(h^3) \end{cases}
\end{array}
$$

III: si calcolano $\vec{x_i}(t_n + h)$ e $\vec{v_i}(t_n + h)$

Schema (un corpo una dimensione) (I)



Schema (un corpo una dimensione) (II)


```
void cinematica(double &t, double h, double &x,
double &v);
```

contenuto:

$$k_1 = hv(t_n)$$
$$w_1 = hf(t_n, x(t_n), v(t_n))$$
$$k_2 = h(v(t_n) + w_1/2)$$
$$w_2 = hf(t_n + h/2, x(t_n) + k_1/2, v(t_n) + w_1/2)$$
$$x(t_n + h) = x(t_n) + k_2 + \mathcal{O}(h^3)$$
$$v(t_n + h) = v(t_n) + w_2 + \mathcal{O}(h^3)$$

```
double dinamica(double t, double x, double v);
```

restituisce: $f(t_n, x(t_n), v(t_n))$ (*accelerazione*)

Schema (un corpo una dimensione) (III)

Come disegnare i punti ? Siccome i punti vengono aggiunti uno alla volta è utile utilizzare il costruttore default per `TGraph`:

```
TGraph::TGraph()
```

e poi, quando sono disponibili, aggiungere ad uno ad uno i punti con

```
void TGraph::SetPoint(int i, double x, double y)
```

Ad esempio:

```
TGraph *gr = new TGraph;
...
gr->SetPoint(0,x0,y0);
gr->Draw(...);
for (...){
  ...
  gr->SetPoint(i++,x,y);
  ...
  gPad->Modified();
  gPad->Update();
}
```

Disegna una serie di punti

```
TGraph *gr = new TGraph;
...
gr->SetPoint(0,x0,y0);
gr->Draw(...);
for (...){
  ...
  gr->SetPoint(0,x,y);
  ...
  gPad->Modified();
  gPad->Update();
}
```

Aggiorna le coordinate del punto ($\rightarrow$ punto in movimento, animazione)

Esercitazione in laboratorio

1. Risolvere numericamente con il metodo di RK al second'ordine

$$\frac{d^2x}{dt^2} + \beta\frac{dx}{dt} + \omega^2 x = 0$$

confrontandola con la soluzione analitica

$$x(t) = Ae^{-\frac{\beta}{2}t}cos(\omega_1 t + \phi)$$

dove $\omega_1^2 = \omega^2 - \beta^2/4$. Provate i casi $\beta = 0$ (oscillatore armonico) e $\beta = 1$ (oscillatore smorzaro) entrambi con $\omega = 4$.

2. Impostate il metodo di RK al quarto ordine e ripetete il punto 1
3. Affrontate uno di questi due problemi (facoltativo):
   - applicate un metodo per il controllo dell'errore
   - affrontate il problema della dinamica di molti corpi applicando il metodo RK al sistema Terra-Sole.

Valutazione dell'errore (I)

È chiaro dalla trattazione generale sulla soluzione numerica delle equazioni differenziali, che la precisione è tanto migliore quanto più piccolo è il passo h.

Il problema che si pone è quello di scegliere un valore sensato per l'incremento temporale.

Conservazione dell'energia

Una tecnica possibile consiste nel calcolare, ad ogni passo, l'energia del sistema. Tale energia dovrebbe rimanere costante. Se si individuano delle variazioni (al di là di una certa soglia predeterminata) significa che le precisione è insufficiente.
In questo caso si può intervenire riducendo il passo.

Valutazione dell'errore (II)

Raddoppio del passo

Metodo generale (vale per sistemi conservativi e non).
Assumiamo di utilizzare un metodo di Runge-Kutta di ordine n.
Per ogni passo dell'iterazione si fa un passo $2h$ e, indipentemente, due passi di lunghezza h. Denotiamo con $y(t+2h)$ la soluzione esatta e con y_1 e y_2 le soluzioni approssimate ottenute rispettivamente con un solo passo pari a $2h$ o con due passi pari ciascuno a h.

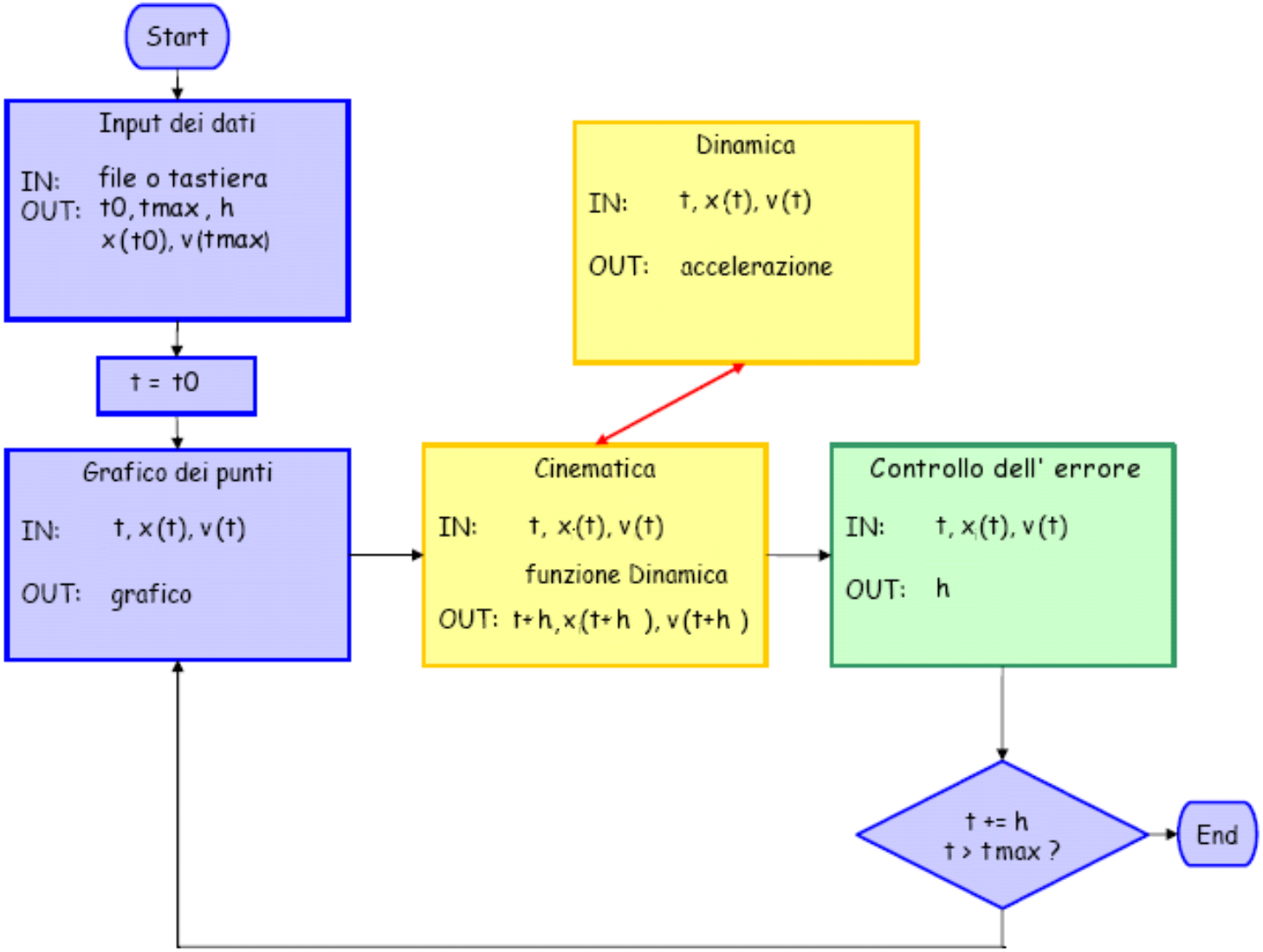
$$y(t+h) = y_1 + (2h)^{n+1}\alpha + \mathcal{O}(h^{n+2})$$
$$y(t+h) = y_2 + 2(h)^{n+1}\alpha + \mathcal{O}(h^{n+2})$$

(dove α è un numero dell'ordine di $y^{(n+1)}/(n+1)!$)
Sostituendo e indicando $\Delta = y_2 - y_1$

$$y(t+h) = y_2 + \frac{\Delta}{2^n - 1} + \mathcal{O}(h^{n+2})$$

si ottiene il valore "esatto" dell'errore fino all'ordine $\mathcal{O}(h^{n+1})$.
Tale errore può essere mantenuto al di sotto di un certa soglia prefissata diminuendo h.

Schema (con controllo dell'errore)



Sistemi di equazioni differenziali (I)

Lo studio della dinamica di N corpi in D dimensioni comporta la soluzione di un sistema di N×D equazioni (×2 per la separazione delle variabili x e v).

Due le soluzioni possibili

- vettore 2×N×D
  Poco naturale: appesantisce la funzione dinamica ma rende cinematica assolutamente generale (facilmente estendibile a gradi superiori al secondo)
- 2 matrici N×D
  Molto "fisico": rende ben leggibile dinamica ma complica cinematica (il comportamente di x e v non è simmetrico).

Sistemi di equazioni differenziali (II)

Allocazione:

```
double *x;
x = new double[N*D*2];
```

Loop:

```
for (int n=0;n<N;n++){
  for (int d=0;d<D;d++){
    x[2*n*D+d    ] = ... // pos.
    x[(2*n+1)*D+d] = ... // vel.
  }
}
```

```
x[0]  comp. x  corpo 1
x[1]  comp. y  corpo 1
x[2]  comp. z  corpo 1
x[3]  comp. vx corpo 1
x[4]  comp. vy corpo 1
x[5]  comp. vz corpo 1
x[6]  comp. x  corpo 2
x[7]  comp. y  corpo 2
x[8]  comp. z  corpo 2
....
```

Prototipi

```
void dinamica(double t, double *x, double *dxdt);
```

(il vettore `dxdt` contiene le derivate di `x`)

```
void cinematica(double &t, double h, double *x);
```

Codice cinematica (runge-kutta al second'ordine):

$$\vec{k_1} = h\vec{f}(\vec{x}(t_n), t_n) \qquad \vec{k_2} = h\vec{f}(\vec{x}(t_n) + \frac{\vec{k_1}}{2}, t_n + \frac{h}{2})$$

$$\vec{x}(t_n + h) = \vec{x}(t_n) + \vec{k_2} + \mathcal{O}(h^3)$$

Assumo qui e nel seguito, per semplicità di notazione, che `N`, `D` e il vettore delle masse `mass[N]` siano variabili globali.

Sistemi di equazioni differenziali (III)

Allocazione:

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

```
x[0][0]  comp. x  corpo 1
x[0][1]  comp. y  corpo 1
x[0][2]  comp. z  corpo 1
v[0][0]  comp. vx corpo 1
v[0][1]  comp. vy corpo 1
v[0][2]  comp. vz corpo 1
x[1][0]  comp. x  corpo 2
x[1][1]  comp. y  corpo 2
x[1][2]  comp. z  corpo 2
....
```

Loop:

```
for (int n=0;n<N;n++){
  for (int d=0;d<D;d++){
    x[n][d] = ... // pos.
    v[n][d] = ... // vel.
  }
}
```

Prototipi

```
double dinamica(int i, int d, double t, double **x, double **v);
```

(dinamica ritorna la componente d-esima dell'accelerazione del corpo i-esimo)

```
void cinematica(double &t, double h, double **x, double **v);
```

Codice cinematica (runge-kutta al second'ordine):

$$\vec{k_{1i}} = h\vec{v_i}(t_n) \qquad \vec{w_{1i}} = h\vec{f_i}(\vec{v_1}(t_n), \ldots, \vec{x_1}(t_n), \ldots, t_n)$$

$$\vec{k_{2i}} = h\left(\vec{v_i}(t_n) + \frac{\vec{w_{1i}}}{2}\right) \qquad \vec{w_{2i}} = h\vec{f_i}(\vec{v_1}(t_n) + \frac{\vec{w_{11}}}{2}, \ldots, \vec{x_1}(t_n) + \frac{\vec{k_{11}}}{2}, \ldots, t_n + \frac{h}{2})$$

$$\vec{x_i}(t_n + h) = \vec{x_i}(t_n) + \vec{k_{2i}} + \mathcal{O}(h^3) \qquad \vec{v_i}(t_n + h) = \vec{v_i}(t_n) + \vec{w_{2i}} + \mathcal{O}(h^3)$$

Suggerimenti

Cinematica

Cinematica deve, all'inizio, allocare dinamicamente i vettori necessari per il metodo di runge-kutta (k, w e gli eventuali vettori di coordinate e velocità temporanee) e distruggerli alla fine (il metodo non è ottimale per la velocità di esecuzione ma evita "pasticci" con le variabili globali).

Dinamica

È utile schematizzare la funzione dinamica come somma delle interazione con gli N-1 corpi più un eventuale contributo di forza esterna.

```
double dinamica(int i, int d, double t, double **x, double **v);
double fInter(int i, int j, int d, double t, double **x, double **v);
double fExter(int i, int d, double t, double **x, double **v);

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

Disegno 3D

Assi:

```
TView::TView();
void TView::Draw();
```

Grafico

```
TPolyMarker3D::TPolyMarker3D();
void TPolyMarker3D::SetPoint(int n, double x, double y, double z)
```

(analogo a `TGraph`)

Struttura a classe

Pratico per controllare creazione, distruzione e generalizzarne l'utilizzo a problemi diversi.

```
class RungeKutta{
  public:
    RungeKutta(int N, int D, int rkord);
                  // costruttore che alloca i vettori e definisce l'ordine di RK
    ~RungeKutta(); // distruttore (dealloca i vettori);
    void   Cinematica();
    void   SetMass(int n, double val);
    double GetMass(int n);
    void   SetPos(int n, int d, double val);
    double GetPos(int n, int d);
    void   SetVel(int n, int d, double val);
    double GetVel(int n, int d);
    void   SetT(double t0);
    double GetT();
    void   SetStep(double h);
    double GetStep();
    void   SetInter(double (*fInter)(..)){m_fInter = fInter;}
    void   SetExter(double (*fExter)(..)){m_fExter = fExter;}
    ...
  private:
    int m_N,m_D;
    double m_t,m_h;
    double **m_x,**m_v,*m_mass;
    double **m_k1,**m_w1,...;
    double (*m_fInter)(...);
    double (*m_fExter)(...);
    double m_dinamica(...);
    ...
};
```