Επίλυση Συνήθων Διαφορικών Εξισώσεων - ODE

Μέθοδος του Euler – Επίλυση διαφορικής εξίσωσης

Ξέρουμε ότι η ελεύθερη πτώση σωμάτων περιγράφεται από δύο διαφορικές εξισώσεις:

$$\upsilon = \frac{dy}{dt} \qquad \text{και} \qquad -g = \frac{d\upsilon}{dt}$$

Η αναλυτική λύση των εξισώσεων αυτών προκύπτει μετά από ολοκλήρωση, στη μορφή:

$$y(t) = y_0 + \upsilon_0 t - \frac{1}{2} g t^2 \quad \text{και} \quad \upsilon(t) = \upsilon_0 - gt$$

Οι προηγούμενοι ορισμοί εμπεριέχουν τον ορισμό της παραγώγου όπου εξετάζεται η μεταβολή του $\Delta y/\Delta t$ στο όριο που το χρονικό διάστημα είναι $\Delta t \to 0$

Ο ορισμός αυτός δεν μπορεί να χρησιμοποιηθεί σε Η/Υ. Ωστόσο μπορούμε να χρησιμοποιήσουμε διαφορετικές προσεγγίσεις.

Έστω ότι γνωρίζουμε την παράγωγο μιας συνάρτησης ως προς την ανεξάρτητη μεταβλητή (έστω για ευκολία αυτή είναι ο χρόνος t)

Ξέρουμε επομένως ότι: $\frac{dy}{dt} = f(y,t)$

Στην προηγούμενη εξίσωση, y είναι η λύση που ψάχνουμε να βρούμε και $f(y,t)$ είναι μια καλά συμπεριφερόμενη τυχαία συνάρτηση.

Μέθοδος του Euler – Επίλυση διαφορικής εξίσωσης

Το πρόβλημα που έχουμε να λύσουμε είναι να βρούμε την *y,* για οποιαδήποτε τιμή της ανεξάρτητης μεταβλητής *t* όταν ξέρουμε την αρχική τιμή της *y,* (έστω y_0) σε κάποια τιμή της ανεξάρτητης μεταβλητής (έστω t=0)

Θα θέλαμε επομένως να βρούμε την τιμή της *y* για τιμή της ανεξάρτητης μεταβλητής *t+Δt* όταν ξέρουμε την τιμή της *y* για *t.*

Αλλά αυτή η τιμή μπορεί να προσεγγισθεί από το … ανάπτυγμα Taylor της συνάρτησης:

$$y(t+\Delta t) = y(t) + \frac{dy}{dt}\Delta t + \frac{1}{2!}\frac{d^2y}{d^2t}\Delta t^2 + \frac{1}{3!}\frac{d^3y}{d^3t}\Delta t^3 + \cdots$$

Το ευκολότερο που θα μπορούσαμε να κάνουμε στην προσέγγιση είναι να αγνοήσουμε όρους οι οποίοι είναι $2^{ης}$ και μεγαλύτερης τάξης σε *Δt* (αν το *Δt* έχει επιλεγεί αρκετά μικρό τότε οι όροι Δt^{η} θα είναι πολύ πολύ πιο μικροί).

Μπορούμε να γράψουμε επομένως: $y(t+\Delta t) = y(t) + \frac{dy}{dt}\Delta t + O(\Delta t^2)$

Ο τελευταίος όρος, αποτελεί την μέγιστη συνεισφορά στο σφάλμα της προσέγγισης που κάνουμε και είναι της τάξης του $\frac{1}{2!}\frac{d^2y}{d^2t}\Delta t^2$

Η παραπάνω προσέγγιση για την λύση μιας διαφορικής εξίσωσης ονομάζεται μέθοδος του Euler και είναι ακριβής σε τάξη $(\Delta t)^2$. Προφανώς για γραμμικές συναρτήσεις η 2^{η} παράγωγος είναι μηδέν και επομένως το σφάλμα μηδενίζεται

Μέθοδος του Euler – Επίλυση διαφορικής εξίσωσης

Με βάση τον προηγούμενο φορμαλισμό μπορούμε να γράψουμε τα βήματα της μεθόδου *του Euler*

$$y(t+\Delta t) = y(t) + \frac{dy}{dt}\Delta t = y(t) + f(y,t)\Delta t \quad \text{όπου χρησιμοποιήσαμε:} \quad \frac{dy}{dt} = f(y,t)$$

Θα πρέπει να υπολογίσουμε την τιμή του $\frac{dy}{dt} = f(y,t)$

στην τιμή *t* της ανεξάρτητης μεταβλητής

Θεωρώντας ότι *y*(t=0) = 0 και ότι Δ*t*=*h* μπορούμε να γράψουμε την μέθοδο Euler ως:

Έστω ότι: $t_n = \eta h$ και $y(t_n)=y_n$ για n=0, 1, 2, …

$$y_1 = y_0 + f(y_0,t)h$$
$$y_2 = y_1 + f(y_1,t)h$$

$$y_{n+1} = y_n + f(y_n,t)h$$

Για την περίπτωση της ελεύθερης πτώσης και για τον υπολογισμό της ταχύτητας έχουμε:

$$\frac{dv}{dt} = f(v,t) = -g \qquad \frac{dy}{dt} = f(y,t) = v$$

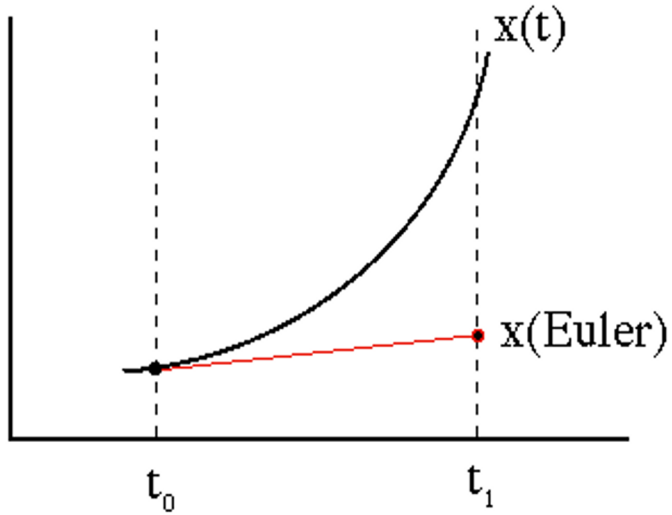
Ο αλγόριθμος του Euler

- Η εφαρμογή της μεθόδου του Euler στον υπολογιστή, εμπεριέχει τα ακόλουθα βήματα
  - Εισάγουμε την αρχική συνθήκη, το μέγεθος και αριθμό των βημάτων (Δt ή Δx) που ορίζονται μέσα στο διάστημα $[x_0,x_n]$ ή $[t_0,t_n]$
  - Υπολογίζουμε τη τιμή της συνάρτησης και της παραγώγου στην αρχή κάθε διαστήματος
  - Υπολογίζουμε τη τιμή της συνάρτησης στο τέλος του διαστήματος
  - Επαναλαμβάνουμε n φορές τα βήματα 2 και 3, όσα είναι τα βήματα που έχουμε καθορίσει βάσει του μεγέθους του βήματος και του ολικού διαστήματος

Για να καθορίσουμε την ευστάθεια του αλγόριθμου θα πρέπει να τρέξουμε το πρόγραμμά μας πολλές φορές για διαφορετικές τιμές για το μέγεθος του βήματος (Δx ή Δt) και να δούμε για ποιες τιμές τα αποτελέσματα του αλγορίθμου παραμένουν σταθερά.

Αυτό γιατί ο αλγόριθμος δίνει αποτελέσματα που συμφωνούν με την αναλυτική λύση για μικρές τιμές των διακριτοποιημένων διαστημάτων ενώ για μεγάλες τιμές αποκλίνει του σωστού αποτελέσματος

Γραφική Αναπαράσταση – Μέθοδος Euler



Ουσιαστικά αυτό που κάνουμε με τη μέθοδο του Euler είναι να ακολουθήσουμε την εφαπτομένη της καμπύλης της λύσης για το διάστημα (t_1-t_0)

Κατόπιν υπολογίζουμε και πάλι την κλίση της καμπύλης στο νέο σημείο και προχωρούμε με βάση τη κλίση αυτή στο επόμενο διάστημα

Αν θυμηθούμε, από το ανάπτυγμα Taylor έχουμε:

$$x(t) = x(t_0) + x'(t)\delta t + \frac{1}{2}x''(t)\delta t^2 + \cdots$$

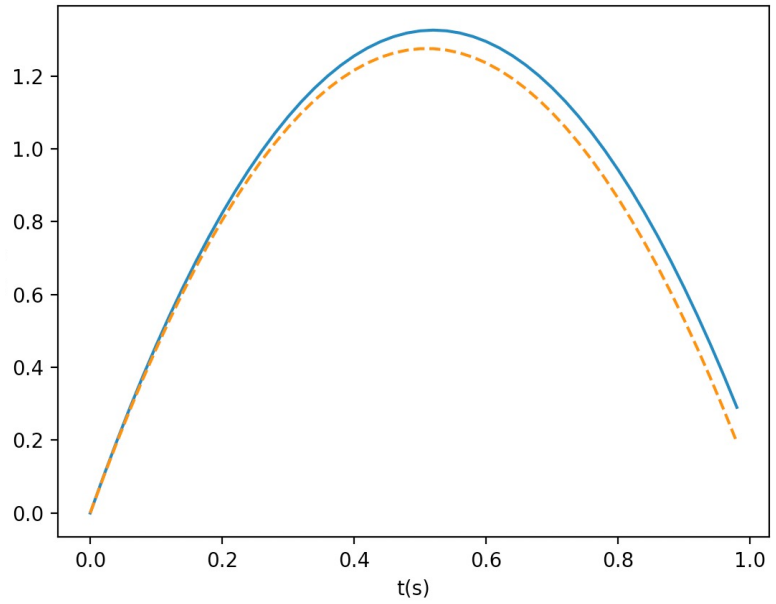
Χρησιμοποιώντας την κλίση σε κάθε διάστημα ουσιαστικά χρησιμοποιούμε τη 1η παράγωγο και αγνοούμε τους ανώτερους όρους τάξης $O(\Delta t^2)$

Το σφάλμα επομένως σε κάθε βήμα είναι της τάξης $O(\Delta t^2)$ και για N βήματα σε ένα συγκεκριμένο διάστημα αυξάνει σχεδόν γραμμικά.

Επειδή ο αριθμός των βημάτων είναι ανάλογος του διαστήματος $1/\Delta t$ ουσιαστικά το σφάλμα είναι γραμμικό με Δt

Ελεύθερη πτώση σωμάτων – Μέθοδος Euler

```
#/usr/bin/python3

import numpy as np
import matplotlib.pyplot as plt

g,dt = 9.8, 0.02    # g and time step

y,v0 = 0.0, 5.0    # arxiki thesi toy swmatos kai taxytita

ta, ya = [],[]

t, yb = 0.0, []    # yb is the theory curve

v = v0             #
while t < 1.0:     # ekseliksi tou sustimatos gia 1 sec
    ta.append(t)
    ya.append(y)
    yb.append(v0*t - g * t * t/2.0)    # theory

    y = y + v*dt       # Euler vima gia ti thesi
    v = v - g*dt       # Euler vima gia tin taxytita
    t = t + dt

plt.figure()
plt.plot(ta,ya,ta,yb,'--')

plt.xlabel('t(s)')
plt.ylabel('y(m)')
plt.show()
```



Η λύση για την ταχύτητα είναι ίδια με την αναλυτική.

Η λύση για την θέση διαφέρει από την αναλυτική. ($y=y + vdt - gdt^2$)

Η ενέργεια δεν διατηρείται:

Αρχικά : $E_0 = mgy_0 + mv_0^2/2$

Μετά από 1 βήμα Euler:

$E_0 = mgy_0 + mv_0^2/2 + mg^2dt^2/2$

Ανάγκη βελτίωσης της μεθόδου του Euler

- Η μέθοδος κάνει γραμμική εξέλιξη ως προς την ανεξάρτητη μεταβλητή και επομένως το σφάλμα αυξάνει ως h^2

  Οι εξισώσεις αντιστοιχούν στο πρώτο όρο του αναπτύγματος γύρω από το σημείο t_0 ή x_0 και επομένως είναι τάξης h^2, $O(h^2)$

  → Η μέθοδος Euler δεν είναι ακριβής.

- Ο φορμαλισμός της μεθόδου του Euler είναι ασύμμετρος.

  Προχωρά τη λύση μέσω του διαστήματος h αλλά χρησιμοποιεί την πληροφορία σχετικά με την παράγωγο μόνο στην αρχή του διαστήματος

- Αν έχουμε τις εξισώσεις κίνησης ενός σώματος γράφουμε: $\frac{d\vec{v}}{dt} = \vec{a}(\vec{r},\vec{v}); \quad \frac{d\vec{r}}{dt} = \vec{v}$

- Με τη μέθοδο του Euler θα γράψουμε: $\vec{v}_{n+1} = \vec{v}_n + h\vec{a}_n \qquad \vec{r}_{n+1} = \vec{r}_n + h\vec{v}_n$

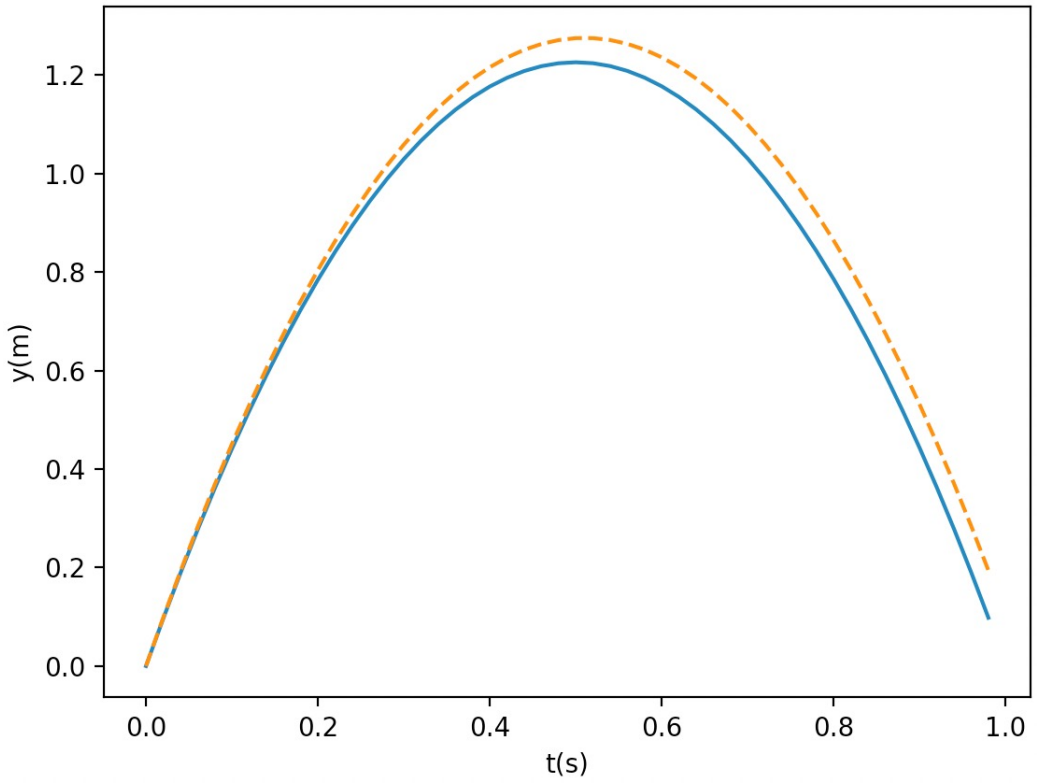
- Μια απλή παραλλαγή της μεθόδου είναι η χρησιμοποίηση των εξισώσεων:

  $\vec{v}_{n+1} = \vec{v}_n + h\vec{a}_n \qquad \vec{r}_{n+1} = \vec{r}_n + h\vec{v}_{n+1}$ ← Euler-Cromer (χρησιμοποίηση της νέας ταχύτητας)

  - Δεν κερδίζουμε όμως τίποτα σε ακρίβεια. Απλά η μέθοδος αυτή δίνει σωστότερα αποτελέσματα για μια κατηγορία προβλημάτων

Ελεύθερη πτώση σωμάτων – Μέθοδος Euler - Cromer

```
#/usr/bin/python3

import numpy as np
import matplotlib.pyplot as plt

g,dt = 9.8, 0.02    # g and time step

y,v0 = 0.0, 5.0     # arxiki thesi toy swmatos
                    # kai taxytita

ta, ya = [],[]

t, yb = 0.0, []     # yb is the theory curve

v = v0              #
while t < 1.0:      # ekseliksi tou
                    # sustimatos gia 1 sec
    ta.append(t)
    ya.append(y)
    yb.append(v0*t - g * t * t/2.0)    # theory

                    # Sti methodo toy Euler-Crommer
                    # prwta upologizetai i taxutita
                    # (i paragwgos tis thesis) sti
                    # xroniki stigmi t kai katopin
                    # xrisimopoieitai i paragwgos
                    # ayti gia ton upologismo tis
                    # thesis tou swmatos,
    v = v - g*dt    # Euler vima gia tin taxytita
    y = y + v*dt    # Euler-Crommer vima gia ti thesi
    t = t + dt

plt.figure()
plt.plot(ta,ya,ta,yb,'--')

plt.xlabel('t(s)')
plt.ylabel('y(m)')
plt.show()
```



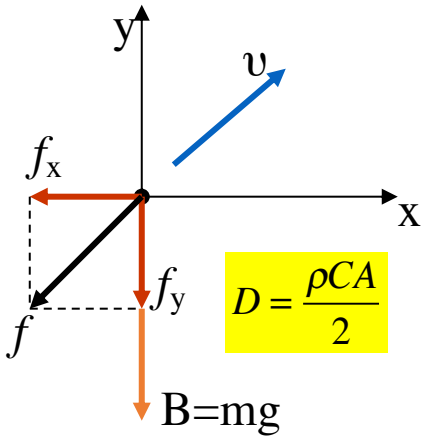
Κίνηση σώματος σε 2 διαστάσεις με αντίσταση ρευστού

Θεωρούμε ότι έχουμε ένα βλήμα το οποίο κινείται με αρχική ταχύτητα υ σε ένα μέσο

Έστω ότι η δύναμη της αντίστασης του μέσου είναι: $f = -Dv^2$

Η διεύθυνση της *F* είναι αντίθετη αυτής της υ και το μέτρο της είναι:

$$f_x = -Dvv_x \qquad f_y = -Dvv_y \qquad f = \sqrt{f_x^2 + f_y^2} \quad \text{και} \quad v = \sqrt{v_x^2 + v_y^2}$$



Επομένως μπορούμε να γράψουμε:

$$\sum F_x = f_x = -Dvv_x = ma_x \qquad \sum F_y = f_y = -(mg + Dvv_y) = ma_y$$

$$a_x = -(D/m)vv_x \qquad a_y = -g - (D/m)vv_y$$

Η σταθερά D εξαρτάται από την πυκνότητα του αέρα, ρ, το σχήμα Α του σώματος (επιφάνεια όπως φαίνεται από εμπρός) και από μια αδιάσταση σταθερά C που ονομάζεται σταθερά αντίστασης

Κίνηση σώματος σε 2 διαστάσεις με αντίσταση ρευστού

- Για ένα αρκετά μικρό χρονικό διάστημα Δt μπορούμε να τη θεωρήσουμε σταθερή

- Στο χρονικό διάστημα Δt η μέση x(y)-συνιστώσα της επιτάχυνσης είναι:

$$a_x = \frac{\Delta \upsilon_x}{\Delta t} \quad \text{και} \quad a_y = \frac{\Delta \upsilon_y}{\Delta t}$$

ενώ η ταχύτητα αλλάζει κατά: $\Delta \upsilon_x = a_x \Delta t$ και $\Delta \upsilon_y = a_y \Delta t$

- Επομένως στο τέλος του διαστήματος Δt η ταχύτητα έχει τιμή

$$\upsilon_x + \Delta \upsilon_x = \upsilon_x + a_x \Delta t, \quad \upsilon_y + \Delta \upsilon_y = \upsilon_y + a_y \Delta t$$

- Ανάλογα αλλάζουν οι συντεταγμένες του σώματος (χρησιμοποιούμε τη μέση αλλαγή της ταχύτητας στο Δt)

$$\Delta x = (\upsilon_x + \frac{\Delta \upsilon_x}{2})\Delta t = \upsilon_x \Delta t + \frac{\Delta \upsilon_x}{2} \Delta t = \upsilon_x \Delta t + \frac{1}{2} a_x (\Delta t)^2 \qquad x + \Delta x = x + \upsilon_x \Delta t + \frac{1}{2} a_x (\Delta t)^2$$

Κίνηση σώματος σε 2 διαστάσεις με αντίσταση ρευστού

```
import numpy as np
import matplotlib.pyplot as plt
dt=0.25
v0 = 700
theta = 55*np.pi/180
A_m = 4E-5
x,y = [],[]
x.append(0)
y.append(0)
xpos = 0.0
ypos = 0.0
vx = v0 * np.cos(theta)
vy = v0 * np.sin(theta)
istep = 0
doit = True
while doit:
    f = A_m * np.sqrt(vx**2 + vy**2)     # drag force
    vy = vy - 9.8 * dt - f * vy * dt
    vx = vx - f * vx * dt
    xpos = xpos + vx*dt
    ypos = ypos + vy*dt
    x.append(xpos)      # Euler-Cromer
    y.append(ypos)
    istep = istep + 1
    if y[istep] <=  0. :
        doit = False            # vlima ktipise sto edafos
```

```
nstep = istep
slope = (y[nstep] – y[nstep-1])/(x[nstep] – x[nstep-1])
x[nstep] = x[nstep-1] – y[nstep-1]/slope
y[nstep] = 0

plt.plot(x,y)
plt.show()
```

Παράδειγμα: Αρμονικός ταλαντωτής

- Οριζόντιο ελατήριο με μάζα m στο ένα άκρο του. Η εξίσωση της κίνησης είναι:

$$m\frac{d^2x}{dt^2} = -kx$$ όπου x η απομάκρυνση από τη θέση ισορροπίας

- Θέτουμε τις αρχικές συνθήκες x(0) και υ(0) και μετατρέπουμε την αρχική εξίσωση σε σύστημα δύο εξισώσεων πρώτου βαθμού

$$\upsilon = \frac{dx}{dt} \qquad \frac{d\upsilon}{dt} = -\frac{k}{m}x$$

- Η πρακτική ερώτηση που παρουσιάζεται τώρα είναι πως μπορούμε να ελέγξουμε τις διάφορες μεθόδους αν δίνουν τα σωστά αποτελέσματα. Μια πολύ ισχυρή μέθοδος είναι αυτή των νόμων διατήρησης. Στη μηχανική όταν δεν έχουμε τριβές η ολική ενέργεια διατηρείται και μπορούμε να θέσουμε το ερώτημα κατά πόσο οι προσεγγιστικές μέθοδοι ικανοποιούν αυτή τη διατήρηση.
- Οι παραπάνω εξισώσεις κίνησης της μάζας εξαρτώμενης από ελατήριο δίνουν ενέργεια $E = m\upsilon^2/2 + kx^2/2$. Από τη μέθοδο του Euler οι εξισώσεις είναι

$$\left.\begin{aligned} x_{n+1} &= x_n + \upsilon_n h \\ \upsilon_{n+1} &= \upsilon_n + \left[(-k/m)x_n\right]h \end{aligned}\right\} \Rightarrow E_{n+1} = \frac{m\upsilon_{n+1}^2}{2} + \frac{kx_{n+1}^2}{2} = (1+\delta)E_n \qquad \delta = h^2\frac{k}{m}$$

Άρα $$E_{n+1} = (1+\delta)^n E_0 = (1 + n\delta + O(n^2\delta^2))E_0$$ Η μέθοδος Euler δεν διατηρεί τη ενέργεια

Σφάλμα – μη διατήρηση ενέργειας

- Μπορεί όμως να υποθέσουμε ότι η διαφορά για κάθε βήμα είναι μικρή αφού είναι της ίδιας τάξης μεγέθους με το σφάλμα της μεθόδου $O(h^2)$
- Το σφάλμα όμως είναι προσθετικό και μετά από n βήματα γίνεται nδ δηλαδή πρώτης τάξης στο δ. Αν το nδ γίνει συγκρίσιμο με τη μονάδα, η προσεγγιστική μέθοδος δεν έχει έννοια.
- Το σφάλμα, όντας προσθετικό, θέτει όρια για τον αριθμό των βημάτων που μπορούμε να χρησιμοποιήσουμε στη μέθοδο του Euler.

Για $n_{max}\delta \approx 1$ $\quad n_{max} = \frac{1}{\delta} = \frac{m}{kh^2}$

Βλέπουμε ότι ο μέγιστος αριθμός των βημάτων είναι ανάλογος της μάζας και αντιστρόφως ανάλογος της σταθεράς του ελατηρίου και του βήματος. Βλέπουμε επίσης ότι η ποσότητα $n_{max}h$ έχει διαστάσεις χρόνου και επειδή η περίοδος είναι ανάλογη με το $\sqrt{m/k}$ μας ενδιαφέρει $n_{max}h$ να είναι τουλάχιστον όσο η περίοδος.

Μέθοδος του Euler – Ραδιενεργός διάσπαση

Πολλοί πυρήνες είναι ασταθείς π.χ. U^{235} διασπάται σε δύο πυρήνες εκπέμποντας ακτινοβολία. Η σχάση γίνεται με τυχαίο τρόπο και για αυτό μιλάμε για την πιθανότητα διάσπασης των πυρήνων ή για την μέση ζωή τους, δηλαδή το μέσο χρονικό διάστημα για να διασπαστεί περίπου το 60% της αρχικής ποσότητα του πυρήνα

Αν $N_U(t)$ είναι ο αριθμός των πυρήνων U^{235} τη χρονική στιγμή, t, τότε ο ρυθμός διάσπασής του δίδεται από τη σχέση:

$$\tau \frac{\mathrm{dN}}{dt} = -N_U$$ Η λύση της εξίσωσης αυτής είναι $$\frac{\mathrm{d}N}{N_U} = -\frac{dt}{\tau} \Rightarrow N_U(t) = N(t=0)e^{-\frac{t}{\tau}}$$

Η σταθερά **τ** είναι μια σταθερά που καθορίζει το μέσο χρόνο ζωής του U^{235}.

Όταν $t=\tau$ τότε η ποσότητα που δεν έχει διασπαστεί είναι $e^{-1} \sim 0.37$

- Η παραπάνω εξίσωση της ραδιενεργούς διάσπασης αποτελεί μια διαφορική εξίσωση πρώτης τάξης
- Η εξίσωση αυτή λύνεται αναλυτικά αλλά θα δοκιμάσουμε μια αριθμητική λύση ώστε να ελέγξουμε αν ο αλγόριθμος που αναπτύσσουμε δίνει τη σωστή απάντηση ώστε να τον χρησιμοποιήσουμε σε προβλήματα που δεν έχουμε αναλυτική λύση

Ραδιενεργός διάσπαση - python

```
import numpy as np
import math
import matplotlib.pyplot as plt
# set parameters
k = 0.05
N0 = 1000.
dt = 2.
tfinal = 100.
# initialize array of time t
t = np.arange(0.,tfinal+dt,dt)
npoints = len(t)
# initialize arrays for solution
N = np.zeros(npoints)
# compute exact solution for comparison
NExact = N0 * np.exp(-k*t)
# Euler Solution
N[0] = N0
for i in range(npoints-1):
    N[i+1] = N[i] - k * N[i] * dt
```

```
# Plot results
plt.ion()
plt.figure(1)
plt.plot(t,N,'mo',label='Euler')
plt.plot(t,NExact,'k',label='Exact')
plt.xlabel('t')
plt.ylabel('N')
plt.legend()
plt.axis([-5,105,-10,1010])
plt.grid('on')
plt.figure(2)
plt.plot(t,N-NExact,'bo')
plt.xlabel('t')
plt.ylabel('Euler - Exact')
plt.axis([-5,105,-20,1])
plt.grid('on')
plt.show()
```

Η μέθοδος του ενδιάμεσου σημείου

Θα μπορούσαμε να χρησιμοποιήσουμε τη μέθοδο του μέσου σημείου όπου οι εξισώσεις που χρησιμοποιούνται τώρα είναι

$$\vec{v}_{n+1} = \vec{v}_n + h\,\vec{a}_n \qquad \vec{r}_{n+1} = \vec{r}_n + h\,\frac{\vec{v}_{n+1} + \vec{v}_n}{2}$$

Στην περίπτωση αυτή χρησιμοποιούμε το μέσο όρο των δύο ταχυτήτων. Αντικαθιστώντας την έκφραση της ταχύτητας από την πρώτη εξίσωση στη δεύτερη θα έχουμε

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{v}_n + \frac{1}{2}\vec{a}_n h^2$$

Η μορφή αυτή είναι ενδιαφέρουσα. Το σφάλμα αποκοπής είναι ακόμα τάξης h^2 στην εξίσωση της ταχύτητας αλλά για την εξίσωση θέσης, το σφάλμα αποκοπής είναι τώρα h^3. Στην πραγματικότητα, η μέθοδος αυτή δίνει πολύ πιο σωστά αποτελέσματα από τις δύο προηγούμενες για προβλήματα που εμπλέκουν κίνηση βλημάτων αλλά και πάλι για άλλη κατηγορία προβλημάτων δεν δίνει σωστά αποτελέσματα

- Πως μπορούμε ωστόσο να αυξήσουμε την ακρίβεια της μεθόδου ώστε το σφάλμα να είναι $O(h^3)$ τουλάχιστο

Η μέθοδος του ενδιάμεσου σημείου – Runge–Kutta 2ου βαθμού

Ο λόγος για τον οποίο η μέθοδος είναι ενδιαφέρουσα φαίνεται από το γεγονός ότι αν εφαρμόσουμε την μέθοδο του Euler χρησιμοποιώντας τη παράγωγο στο ενδιάμεσο σημείο θα έχουμε:

$$x(t) + f\left(t + \frac{1}{2}\delta t\right)\delta t = x(t) + \left[f(t) + \frac{1}{2}f'(t)\delta t\right]\delta t + O(\delta t)$$

Αυτή είναι η σωστή έκφραση δεύτερης τάξης. Έχουμε κερδίσει μια τάξη μεγέθους στο σφάλμα με το να υπολογίζουμε την παράγωγο σε κάποια άλλη χρονική στιγμή

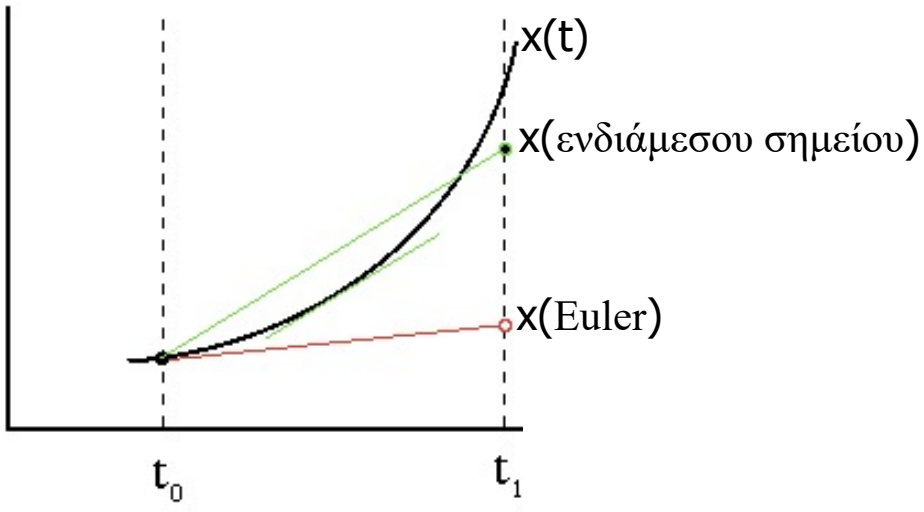
Γεωμετρικά αυτό σημαίνει το εξής:

Πως βρίσκουμε την παράγωγο στο ενδιάμεσο σημείο?

Χρησιμοποιώντας τη μέθοδο του Euler:

$$\delta x = f(x_i, t_i)\delta t$$

$$x_{i+1} = x_i + f\left(x_i + \frac{1}{2}\delta x, t_i + \frac{1}{2}\delta t\right)\delta t$$



Παράδειγμα

- Έστω η συνάρτηση $x'(t) = -x(t)$

  Η μέθοδος του Euler θα μας δώσει:

$$\frac{dx}{dt} = f(x(t), t) = -x(t) \qquad h = \Delta t$$

$$x(t+h) = x(t) + hf(x(t), t) = x(t) + h(-x(t)) = x(t) - hx(t) = x(t)(1-h)$$

  Η μέθοδος του ενδιάμεσου σημείου βάζοντας βήμα h/2 θα δώσει:

$$x(t+h) = x(t) + hf\left(x\left(t+\frac{h}{2}\right), t+\frac{h}{2}\right)$$

$$= x(t) + hf\left(\left[x(t) + \frac{h}{2}f(x(t), t)\right], t+\frac{h}{2}\right)$$

$$= x(t) + hf\left(x(t) + \frac{h}{2}\left(-x(t)\right), t+\frac{h}{2}\right)$$

$$= x(t) + h\left(-x(t) + \frac{h}{2}x(t)\right) = x(t)\left(1 - h + \frac{h^2}{2}\right)$$

Όπου στη 2η ισότητα αντικαταστήσαμε το όρισμα $x\left(t+\frac{h}{2}\right)$ της f με το ανάπτυγμά του $x\left(t+\frac{h}{2}\right) = x(t) + \left(\frac{h}{2}\right)f(x(t), t)$, ενώ στη 3η και 4η ισότητα αντικαταστήσαμε $f(x) = -x$

Μέθοδος ενδιάμεσου σημείου – Runge-Kutta 2ου βαθμού

```
# Ptosi mpalas
import numpy as np
import matplotlib.pyplot as plt
# set parameters
g = 9.8
X0 = 0.
V0 = 0.
dt  = 2.
tfinal = 100.
# initialize arrays
t = np.arange(0.,tfinal+dt,dt)
npoints = len(t)
X = np.zeros(npoints)
V = np.zeros(npoints)
XExact = X0 + V0*t - 0.5*g*t**2
# Lysi RUNGE_KUTTA 2ou BATHMOU
X[0] = X0
V[0] = V0
for i in range(npoints-1):
    # compute midpoint velocity
    Vmid = V[i] - 0.5*g*dt
    # use midpoint velocity to advance position
    X[i+1] = X[i] + Vmid*dt
    V[i+1] = V[i] - g*dt
```

```
# plot the results
plt.figure()
plt.plot(t,X,'mo',label='Euler')
plt.plot(t,XExact,'k',label='Exact')
plt.xlabel('t (s)')
plt.ylabel('X (m)')
plt.legend()
```

Προσοχή: Στην περίπτωση του βαρυτικού πεδίου η δύναμη/επιτάχυνση είναι σταθερές οπότε για την ταχύτητα φαίνεται σαν να μην χρησιμοποιείται η παράγωγος στο ενδιάμεσο σημείο.

Σε περιπτώσεις που η δύναμη/επιτάχυνση μεταβάλλονται, η έκφραση για την ταχύτητα θα είναι:

$$v_{i+1} = v_i + \frac{F(x_{i+1/2}, v_{i+1/2})}{m} dt$$

Βήματα για την μέθοδο Runge-Kutta 2ου βαθμού

- Υπολογισμός της ταχύτητας V και θέσης X στο μέσο του χρονικού διαστήματος:

$$v_{i+1/2} = v_i + \frac{F(x_i, v_i)}{m}\frac{dt}{2} \qquad x_{i+1} = x_i + v_i\frac{dt}{2}$$

- Χρησιμοποίηση των τιμών της ταχύτητας και επιτάχυνσης στο ενδιάμεσο σημείο για να φθάσουμε στο τέλος του χρονικού διαστήματος:

$$v_{i+1} = v_i + \frac{F(x_{i+1/2}, v_{i+1/2})}{m}dt \qquad x_{i+1} = x_i + v_{i+1/2}dt$$

- Διατήρηση ενέργειας? Πτώση σώματος στο βαρυτικό πεδίο:

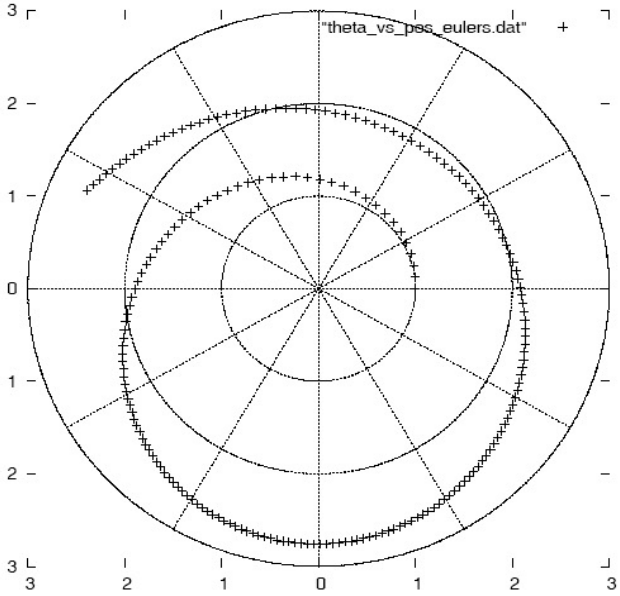
$$v_{mid} = v_i - \frac{gdt}{2} \qquad x(dt) = v_0 + v_{mid}dt \;\Rightarrow x(dt) = v_0 + v_0 dt - \frac{gdt^2}{2}$$

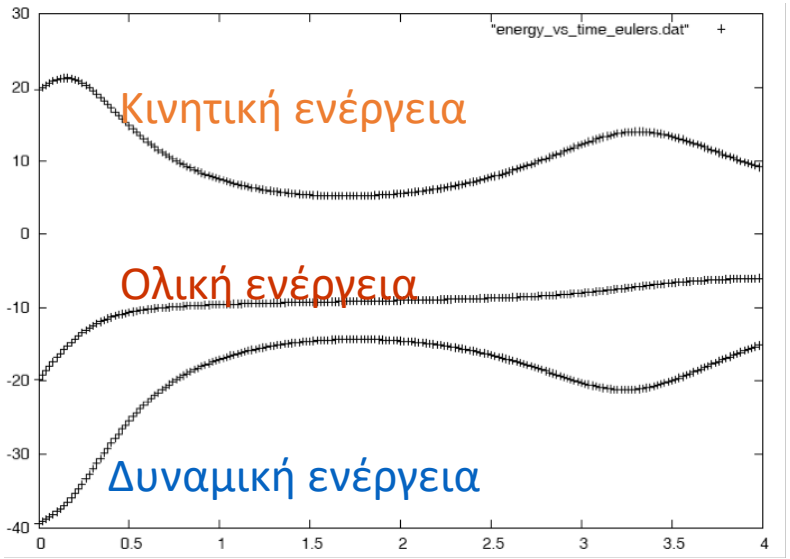
$$E(dt) = mg(x_0 + v_0 dt - \frac{1}{2}gdt^2) + \frac{1}{2}m(v_0 - gdt)^2 \Rightarrow E(dt) = mgx_0 + \frac{1}{2}mv_0^2$$

$$\Rightarrow E(dt) = E_0$$ **Διατήρηση ενέργειας**

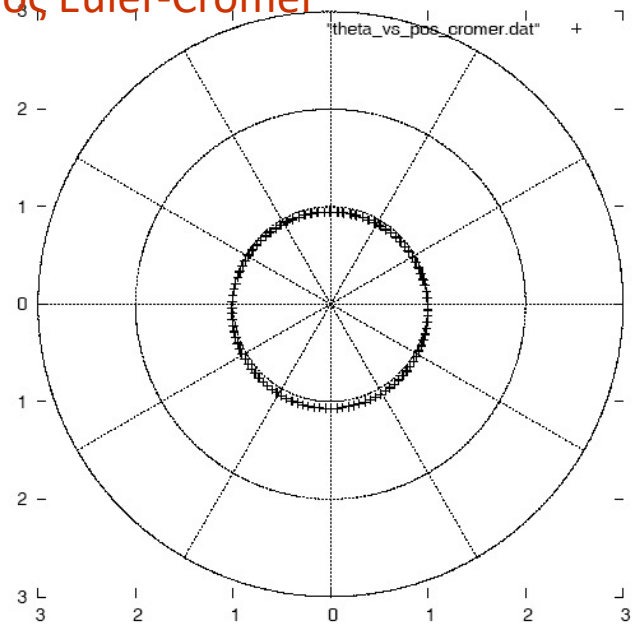
Κίνηση πλανητών

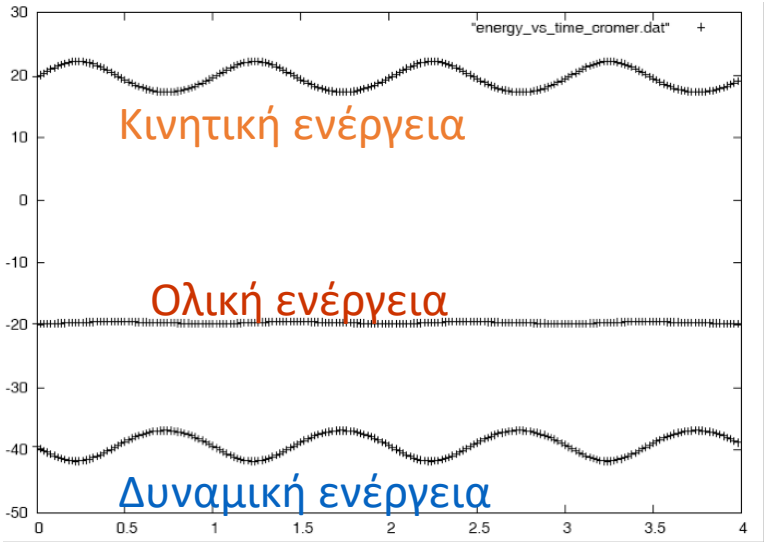
Μέθοδος Euler





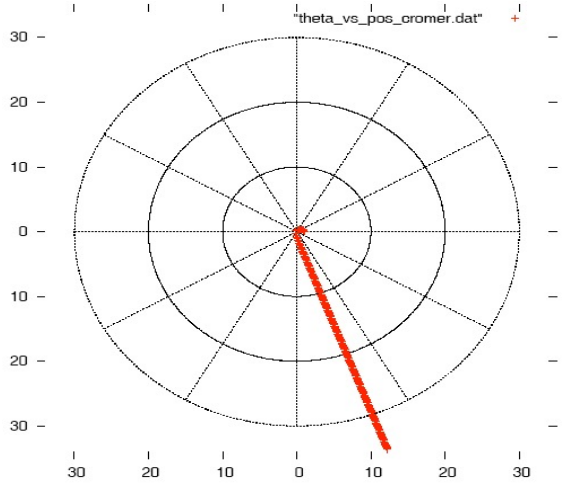
Μέθοδος Euler-Cromer

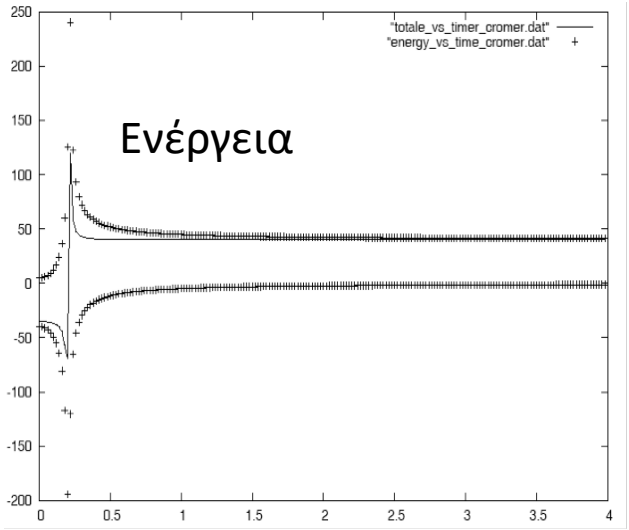


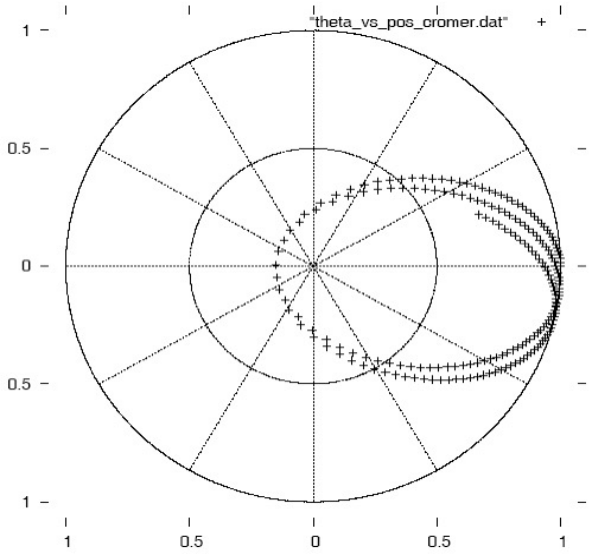


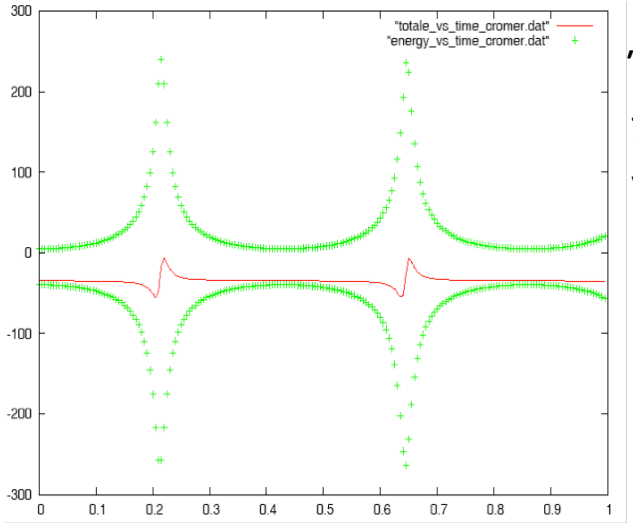
Κίνηση πλανητών

Μέθοδος Euler-Cromer για τροχιά με μεγάλη εκκεντρότητα (αρχική ταχύτητα π AU/yr, βήμα 0.02yr και 200 βήματα). Λόγω αριθμητικού σφάλματος ο πλανήτης φαίνεται να αποκτά ταχύτητα διαφυγής









Ίδιες συνθήκες όπως προηγουμένως αλλά για βήμα 0.005

Η μέθοδος του Verlet - εισαγωγικά

Η μέθοδος του Euler στηρίζεται στον ορισμό της δεξιάς παραγώγου.
Ένας ισοδύναμος ορισμός είναι

$$f'(t) = \lim_{h\to 0} \frac{f(t+h) - f(t-h)}{2h}$$

Αυτή η εξίσωση λέμε ότι είναι ζυγισμένη ως προς t.
Χρησιμοποιώντας το ανάπτυγμα Taylor μπορούμε να γράψουμε:

$$f(t+h) = f(t) + hf'(t) + \frac{1}{2!}h^2 f''(t) + \frac{1}{3!}h^3 f'''(\zeta_+)$$

$$f(t-h) = f(t) - hf'(t) + \frac{1}{2!}h^2 f''(t) - \frac{1}{3!}h^3 f'''(\zeta_-)$$

όπου $\zeta_\pm$ είναι μεταξύ t και $t \pm h$

Επομένως αφαιρώντας γράφουμε:

σφάλμα αποκοπής τάξης h^2

$$f'(t) = \frac{f(t+h) - f(t-h)}{2h} - \frac{1}{3!}h^2 f'''(\zeta) \quad \text{οπου } t-h \le \zeta \le t+h$$

- Χρησιμοποιώντας το ανάπτυγμα Taylor για *f*(t+h) και *f*(t-h) μπορούμε να γράψουμε τη δεύτερη παράγωγο της *f* με τη μορφή

$$f''(t) = \frac{f(t+h) + f(t-h) - 2f(t)}{h^2} - \frac{1}{4!}h^2 f^{(4)}(\zeta) \quad \text{οπου } t-h \le \zeta \le t+h$$

Υπολογισμός παραγώγου

```
def derivative(f,a,method='central',h=0.01):
 '''Compute the difference formula for f'(a)  with step size h.
    Parameters ---------- f : function
    a : number Compute derivative at x = a
    method : string
             Difference formula: 'forward', 'backward' or 'central'
    h : number Step size in difference formula
    Returns -------
    float Difference formula:
    central: f(a+h) - f(a-h))/2h
    forward: f(a+h) - f(a))/h
    backward: f(a) - f(a-h))/h '''
if method == 'central':
   return (f(a + h) - f(a - h))/(2*h)
elif method == 'forward':
   return (f(a + h) - f(a))/h
elif method == 'backward':
   return (f(a) - f(a - h))/h
else:
   raise ValueError("Method must be 'central', 'forward' or 'backward'.")
```

Η Μέθοδος Verlet

- Παίρνοντας τις εξισώσεις κίνησης $\frac{d\vec{r}}{dt} = \vec{v}(t); \quad \frac{d^2\vec{r}}{dt^2} = \vec{a}(\vec{r})$

  και χρησιμοποιώντας τις σχέσεις
  για την πρώτη και δεύτερη παράγωγο που δείξαμε πριν θα έχουμε

$$\frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2h} + O(h^2) = \vec{v}_n \qquad \frac{\vec{r}_{n+1} + \vec{r}_{n-1} - 2\vec{r}_n}{h^2} + O(h^2) = \vec{a}_n \quad \text{οπου} \quad \vec{a}_n = \vec{a}(\vec{r}_n)$$

❑ Από τις εξισώσεις αυτές παίρνουμε:

$$\vec{v}_n = \frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2h} + O(h^2)$$

$$\vec{r}_{n+1} = 2\vec{r}_n - \vec{r}_{n-1} + h^2\vec{a}_n + O(h^4)$$

❑ Γνωρίζοντας r_0 και r_1 μπορούμε να υπολογίσουμε r_2.
Γνωρίζοντας r_2 και r_1 μπορούμε να υπολογίσουμε r_3 και άρα να πάρουμε $υ_2$

❑ Το πρόβλημα της μεθόδου είναι ότι δεν είναι εύκολο να την ξεκινήσουμε.
Οι αρχικές συνθήκες που μας δίνονται συνήθως είναι $r_1 = r(t = 0)$ και
$υ_1 = υ(t = 0)$ αλλά όχι $r_0 = r(t = -h)$

➢ Επιπλέον προβλήματα στρογγυλοποίησης στον τύπο της ταχύτητας
λόγω αφαίρεσης σχεδόν ίσων και μεγάλων μεγεθών

Η Μέθοδος Verlet – ταχύτητας

Από τις σχέσεις: $\vec{\upsilon}_n = \frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2h} + O(h^2)$ (1)

$\vec{r}_{n+1} = 2\vec{r}_n - \vec{r}_{n-1} + h^2\vec{a}_n + O(h^4)$ (2)

Λύνοντας την (1) ως προς r_{n-1} έχουμε: $\vec{r}_{n-1} = \vec{r}_{n+1} - 2h\vec{\upsilon}_n$ (3)

Αντικαθιστώντας στην (2): $\vec{r}_{n+1} = \vec{r}_n + h\vec{\upsilon}_n + \frac{h^2}{2}\vec{a}_n$ (4)

Από την (1) έχουμε ακόμα: $\vec{\upsilon}_{n+1} = \frac{\vec{r}_{n+2} - \vec{r}_n}{2h}$ (5)

Αλλά σύμφωνα με την (2), r_{n+2} δίνεται από: $\vec{r}_{n+2} = 2\vec{r}_{n+1} - \vec{r}_n + h^2\vec{a}_{n+1}$ (6)

Αντικατάσταση της (6) στη (5) δίνει: $\vec{\upsilon}_{n+1} = \frac{\vec{r}_{n+1} - \vec{r}_n}{h} + \frac{h}{2}\vec{a}_{n+1}$ (7)

Χρησιμοποιώντας την (4) για αντικατάσταση του $r_{n+1} - r_n$ στην (7) έχουμε:

$$\vec{\upsilon}_{n+1} = \frac{h\vec{\upsilon}_n + \frac{h^2}{2}\vec{a}_n}{h} + \frac{h}{2}\vec{a}_{n+1} \Rightarrow \boxed{\vec{\upsilon}_{n+1} = \vec{\upsilon}_n + \frac{h}{2}(\vec{a}_n + \vec{a}_{n+1})}$$

$$\boxed{\vec{r}_{n+1} = \vec{r}_n + h\vec{\upsilon}_n + \frac{h^2}{2}\vec{a}_n}$$

Μέθοδος Verlet ταχύτητας

- Η μέθοδος αυτή είναι μαθηματικά ισοδύναμη με την γενική μορφή του Verlet αλλά δεν παρουσιάζει προβλήματα σφαλμάτων στογγυλοποίησης και επίσης ξεκινά εύκολα από τις αρχικές συνθήκες και μόνο.

Η Μέθοδος Verlet – ταχύτητας

Σχηματικά η μέθοδος ταχύτητας Verlet μπορεί να χρησιμοποιηθεί ως εξής:

Δεδομένων των αρχικών συνθηκών x_k και υ_k και της συνάρτησης της δύναμης ή επιτάχυνσης ενός σώματος F(x):

Βήμα 1: Υπολογίζουμε $\upsilon_{k+1/2} = \upsilon_k + \frac{h}{2m} F(x_k)$

Βήμα 2: Υπολογίζουμε $x_{k+1} = x_k + \upsilon_{k+1/2} h$

Βήμα 3: Υπολογίζουμε $\upsilon_{k+1} = \upsilon_{k+1/2} + \frac{h}{2m} F(x_{k+1})$

Runge-Kutta 4ου βαθμού – RK4

Η μέθοδος Runge-Kutta 4ου βαθμού είναι η πλέον χρησιμοποιούμενη μέθοδος για την λύση κανονικών διαφορικών εξισώσεων

Η μέθοδος δουλεύει πολύ καλά ακόμα και με όχι τόσο μικρό βήμα ως προς την ανεξάρτητη μεταβλητή

Ο αλγόριθμος στηρίζεται στον παρακάτω τύπο:

$$u_{n+1} = u_n + \frac{1}{6}(k_1 + 2\,k_2 + 2k_3 + k_4)\,\Delta t$$

Runge-Kutta 4ου βαθμού

όπου: $k_1 = f(u_n, t_n)$ ← τιμή στην αρχή του διαστήματος

$$k_2 = f\left(u_n + \frac{1}{2}k_1\Delta t, t_{n+1/2}\right)$$

$$k_3 = f\left(u_n + \frac{1}{2}k_2\Delta t, t_{n+1/2}\right)$$

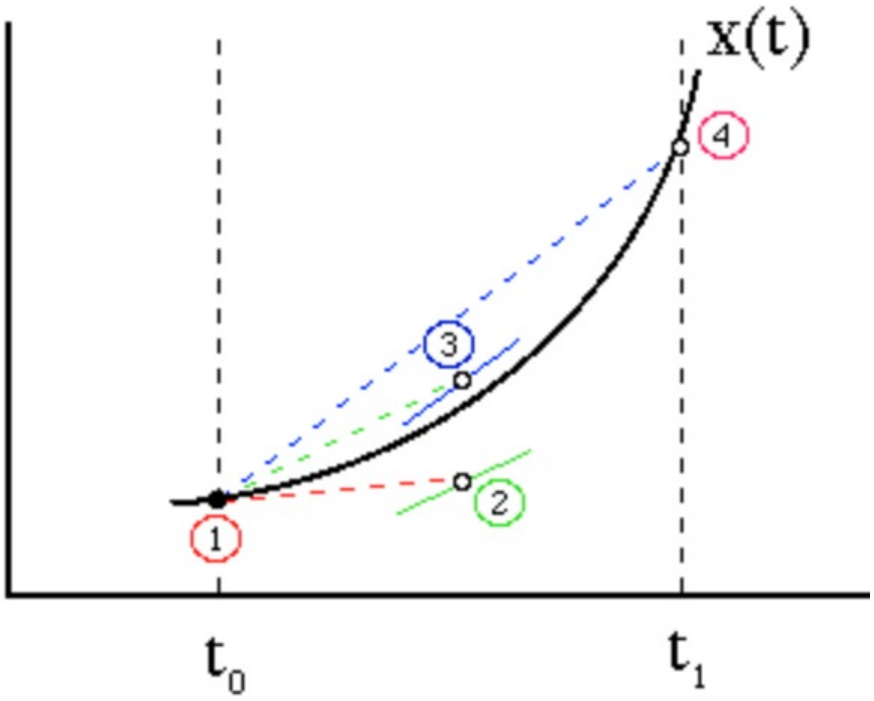
← 2 εκτιμήσεις τιμών στο μέσο του διαστήματος

$$k_4 = f\left(u_n + k_3\Delta t, t_{n+1/2}\right)$$ ← εκτίμηση τιμής στο τέλος του διαστήματος

και $\frac{du}{dt} = u' = f(u_n, t_n)$

Το αριθμητικό σφάλμα της μεθόδου είναι $C\Delta t^4$ για σταθερό C. Επομένως το σφάλμα πηγαίνει γρήγορα στο 0 καθώς το βήμα γίνεται μικρότερο, σε αντίθεση με την μέθοδο Euler, (σφάλμα ~Δt), Euler-Cromer και Runge-Kutta 2ου βαθμού (σφάλμα ~Δt^2)

Runge-Kutta 4ου βαθμού



RK4- παράδειγμα

```
# Gia tin diaforiki eksiswsi: dy / dx = (x - y)/2
def dydx(x, y):
    return ((x - y)/2)

#Eyresi tis timis tou y gia dedomeno x kai vima h
# Arxiki sunthiki y0 sto x0
def rungeKutta(x0, y0, x, h):
    # Euresi tou arithmou twn vimatwn
    # step height h
    n = (int)((x - x0)/h)
    # Epanalipsi gia arithmo vimatwn
    y = y0
    for i in range(1, n + 1):
        #Efarmogi tis RK4 gia euresi tis neas timis tou y
        k1 = h * dydx(x0, y)
        k2 = h * dydx(x0 + 0.5 * h, y + 0.5 * k1)
        k3 = h * dydx(x0 + 0.5 * h, y + 0.5 * k2)
        k4 = h * dydx(x0 + h, y + k3)
        # Nea timi tou y
        y = y + (1.0 / 6.0)*(k1 + 2 * k2 + 2 * k3 + k4)
        # Nea timi tou x
        x0 = x0 + h
    return y
# Main
x0 = 0
y = 1
x = 2
h = 0.2
print('The value of y at x is:', rungeKutta(x0, y, x, h))
```