Tir parabòlic

Pol Tibau
Universitat Politècnica de Catalunya

December 18, 2023

0.1 Introducció

En aquest projecte posarem en pràctica els mètodes vistos a classe per resoldre EDOs. Estudiarem el llançament d'un projectil en dues dimensions, del qual podem descriure la seva posició i velocitat respecte el temps com un sistema de 4 equacions diferencials. Les equacions donades són les següents:

$$\begin{cases} \frac{d\mathbf{x}}{dt} = \mathbf{v} \\ \frac{d\mathbf{v}}{dt} = -R\|\mathbf{v}\|\mathbf{v} + \mathbf{g} \end{cases} \tag{1}$$

On $\mathbf{g}$ és el vector representant de l'acceleració de la gravetat ($\mathbf{g} = (0, -9.8)^T m/s^2$) i R correspon al coeficient de fregament, que depèn principalment de l'àrea projectada de l'objecte llançat i de la densistat de l'aire. En aquest cas prendrem $R = 0.00132m^{-1}$.

Perquè la nostra solució sigui única, es donen les següents condicions inicials:

$$\begin{cases} \mathbf{x}(0) = (0, 0)^T \\ \mathbf{v}(0) = v_0(cos\theta, sin\theta)^T \end{cases} \tag{2}$$

Durant tot l'estudi suposarem que $v_0 = 100$, i d'entrada prendrem $\theta = \frac{\pi}{4}$.

0.2 Resolució del sistema usant el métode d'Euler i Runge-Kutta explícit de quart ordre

Començarem resolent el problema de valor inicial amb dos mètodes diferents, en un interval de temps de 10 segons i 20 subintervals. Acabat representarem en un gràfic les trajectòries aproximades pels dos mètodes. Abans veiem com calcular la solució amb els mètodes:

```
def Euler(f, xSpan, IC, nOfSteps):
    h = (xSpan[1] - xSpan[0]) / nOfSteps
    x = np.linspace(xSpan[0], xSpan[1], nOfSteps+1)
    y = np.zeros( (len(IC), nOfSteps + 1) )
    y[:, 0] = IC
    for i in np.arange(0, nOfSteps):
        y[:, i + 1] = y[:, i] + h * f(x[i], y[:, i])
    return x, y

def RK4(f, xSpan, IC, nOfSteps):
    h = (xSpan[1] - xSpan[0]) / nOfSteps
    x = np.linspace(xSpan[0], xSpan[1], nOfSteps+1)
    y = np.zeros( (len(IC), nOfSteps + 1) )
    y[:, 0] = IC
    for i in np.arange(0, nOfSteps):
        k1 = f(x[i], y[:,i])
        k2 = f(x[i]+h/2,y[:,i]+h*k1/2)
```

```
        k3 = f(x[i]+h/2,y[:,i]+h*k2/2)
        k4 = f(x[i]+h,y[:,i]+h*k3)
        y[:, i + 1] = y[:, i] + h*(k1 + 2*k2 + 2*k3 + k4)/6
    return x, y
```

Sobre els dos mètodes, cal dir que **xSpan** representa l'interval de temps on estem solucionant el sistema, **IC** és la condició inicial (en aquest cas $IC = (0, 0, 100 * cos(\frac{\pi}{4},)$, i **nOfSteps** designa la quantitat de subintervals que definim. Finalment **f** representa la funció de la EDO tal que $\frac{dy}{dx} = f(x, y)$, que en el nostre cas, ja que podem reduir el nostre sistema a:

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dy}{dt} = v_y \\ \frac{dv_x}{dt} = -R\|v_x\|v_x \\ \frac{dv_y}{dt} = -R\|v_y\|v_y - 9.8 \end{cases} \tag{3}$$

Podem concloure que la nostra funció resulta:

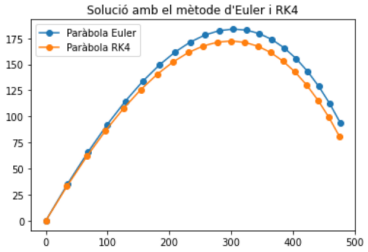
$$f(t, [x, y, v_x, v_y]) = [v_x, v_y, -R\|v_x\|v_x, -R\|v_y\|v_y - 9.8] \tag{4}$$

I per tant la podem representar al codi com:

```
def f(t,x):
v = np.array([x[2],x[3]])
R = 0.00132
g = np.array([0,-9.8])
return np.array([v[0],v[1],-R*np.linalg.norm(v[0])*v[0]+g[0],-R*np.linalg.no
```

Observem que el procediment dels mètodes és molt similar, només difereixen en el càlcul de y[:, i+1]. Per una banda, Euler ho fa directament multiplicant el pas h per l'evualuació del sistema en la posició anterior i la seva aproximació corresponent. En canvi, Runge-Kutta explícit de quart ordre calcula prèviament 4 k-evaluacions de la funció amb petites variacions depenents del paràmetre h i les k-evaluacions anteriors per determinar finalment a partir d'una combinació lineal d'aquestes y[:, i+1].

Vegem el gràfic obtingut de la representació de la trajectòria de cada mètode:



Finalment si ens preguntem en quina posició es troben els projectils per t=10 pels dos mètodes, trobem que per Euler tenim les coordenades [476.8266, 93.8524] i per RK4 [475.6536, 80.6268]

0.3 Estimació de l'error pels mètodes usats

A partir de les següents fórmules, aproximarem l'error que apareix per cada un dels mètodes:

$$\begin{cases} E = \|X_m - X_{2m}\| \\ r = \frac{E}{\|X_{2m}\|} \end{cases} \tag{5}$$

Aplicant-ho doncs per m=20, ja que és el nombre d'intervals que teníem a l'apartat 1, sols cal calcular la solució pels mètodes de Euler i RK4 per 40 subintervals i obtenim (arrodonint al sisè decimal):

$$\begin{cases} E_{RK4} = \frac{\|Yrk4_{20} - Yerk4_{40}\|}{\|Yrk4_{40}} \| = 3.976915e-07 \\ E_{Euler} = \frac{\|Yeuler_{20} - Yeuler_{40}\|}{\|Yeuler_{40}} \| = 0.015697 \end{cases} \tag{6}$$

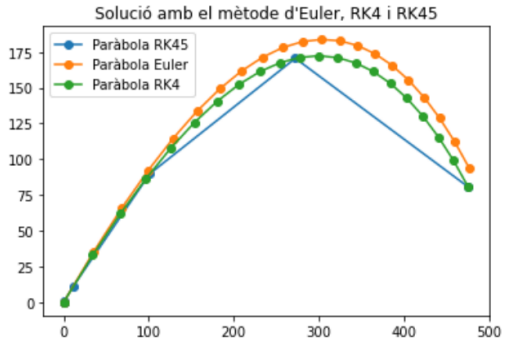
Així doncs podem assegurar que tenim una xifra significativa correcta pel mètode de Euler amb 20 intervals, i cinc xifres significatives pel mètode de RK4, ja que $7.912797e-07 < 0.5e-05$ i $0.013718 < 0.5e-01$.

0.4 Implementació d'un mètode de pas variable: RK45

Utilitzant la llibreria "*Scipy.integrate*" , podem usar la funció "$solve_ivp()$" que com diu el seu nom permet resoldre problemes de valor inicial. Donada la funció associada a la EDO, un interval de temps i la condició inicial,sense donar indicacions extres obtindrem la solució del nostre problema mitjançant l'ús del mètode RK45.

El mètode RK45 no té definit un pas h que serà constant durant l'execució del codi. El pas h serà modificat convenientment per assegurar que l'error local queda fitat per una tolerància donada. Si no ho especifiquem a l'usar el codi, la tolerància per defecte es defineix com 10^{-3} per l'error relatiu i 10^{-6} per l'error absolut.

Si en comparem la solució obtinguda amb els mètodes anteriors, deixant la tolerància d'error relatiu local per defecte, observem la següent gràfica:



No hem de fixar-nos en la paràbola sinó els punts, que són els que realment mostren la precisió del model. Per comparar l'error entre els mètodes, usarem la fòrmula per estimar l'error a l'apartat anterior. Degut a que no definim un nombre de subintervals, podem veure que per $rtol = 10^{-5}$ tenim 10 avaluacions de la solució, i $rtol = 10^{-8}$ si elevem al quadrat en fem 20, per tant podem aproximar X_{2m} imposant que volem una tolerància per l'error relatiu com 10^{-8}. Amb la fòrmula de l'apartat anterior obtenim (arrodonint al quart decimal):

$$E_{RK45} = \frac{\|Yrk45_{rtol=10^{-5}} - Yrk45_{rtol=10^{-8}}\|}{\|Yrk45_{rtol=10^{-8}}\|} = 7.3237e-07 \qquad (7)$$

Podem concloure doncs que amb el mètode de pas variable l'estimació de l'error és menor però similar al mètode RK4.

0.5 Implementació de l'argument *events*

Una de les ventatges que ens ofereix la funció $solve_ivp$ és la possiblitat de definir l'argument $events = f(...)$. Donada una funció això ens permet que a part de resoldre el problema la funció ens aporti alguna dada que ens interessa d'aquest. Per exemple, si volem saber quina distància ha recorregut el projectil a l'impactar contra el terra, podem definir la funció:

```
def Terra(t,x):
    return x[1]
```

I si a l'hora de definir la funció $solve_ivp$ ho fem de la següent manera:

```
SolRK45_2 = sc.solve_ivp(f,[0,20],angle(phi),method='RK45',events=Terra)
```

A l'imprimir la nostra solució podrem veure la distància que recorre el projectil durant la trajectòria fins impactar de nou contra el terra, i també amb quin temps ho fa. En el cas inicial, per un angle de $\frac{\pi}{4}$ obervem que el projectil arriba al terra a t = 11.8066 i la distància horitzontal recorreguda és de x = 532.9744.

0.6 Trobem l'angle indicat perquè el projectil arribi a un objectiu donat

A partir de l'argument *events* usat en aquest darrer apartat, construint unes funcions auxiliars podem dissenyar un mètode per anar aproximant l'angle (en poques iteracions) perquè el nostre projectil s'estavelli contra un objectiu desitjat.

Usarem el mètode de Newton. Volem trobar el zero d'una funció que depèn de l'angle del llançament tal que $f(\phi) = 500$. Definirem:

```
def diferencia(phi, D, tol):
return (sc.solve_ivp(f, [0,20], angle(phi), method='RK45',rtol=tol,
        events=Terra).y_events[0][1][0]-D)
```

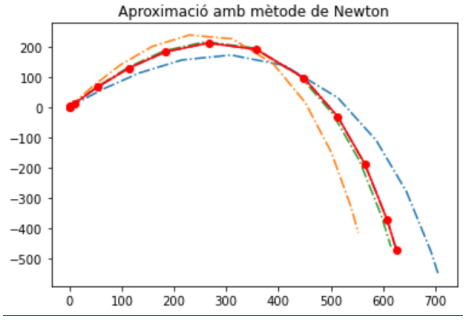
```
def derivada(phi, D, h=1e-5):
    return (diferencia(phi+h,D,tol)-diferencia(phi,D,tol))/h

def Terra(t,x):
    return x[1]

def tir(distancia, phi_ini, tol, max_it):
    phi = phi_ini
    it = 0
    while (abs(diferencia(phi, distancia,tol)) > tol and it < max_it):
        sol = sc.solve_ivp(f, [0,20], angle(phi), method='RK45', rtol=tol,
                events=Terra)
        plt.plot(sol.y[0],sol.y[1],'-.')
        phi -= diferencia(phi, distancia,tol) / derivada(phi, distancia)
        it += 1
    return phi, it
```

En primer lloc, la funció *diferencia* ens retorna simplement com de lluny estem de l'objectiu desitjat amb l'angle en què hem llançat el nostre projectil. *phi* és l'angle de llançament, D és la distància on es troba l'objectiu i *tol* és la tolerància que fita la precisió que desitgem.

En segon lloc, la funció *derivada* aproxima numèricament la derivada que usem per completar la fòrmula del mètode de Newton i poder actualitzar l'angle cada iteració amb una millora assegurada.

Finalment, la funció *tir*, a partir d'un angle inicial usa fins obtenir una precisió mínima escollida el mètode de Newton. A l' acabar retorna l'angle que busquem amb el nombre d'iteracions que es reqereixen. Degut a que el mètode de Newton convergeix, generalment, en poques iteracions, observem que tenim establert un màxim d'iteracions. Això ho fem ja que és possible que per algun cas el mètode no convergeixi, i d'aquesta manera provoquem que al cap de 1000 iteracions el codi s'aturi, encara que sigui sense trobar cap resultat.

Si fem una prova imposant $D = 500$, l'algorisme ens troba amb un total de 5 iteracions i una precisió mínima de $1e{-}5$ que l'angle ideal és $52.753290182863360^{\underline{o}}$ (el projectil arriba a 499.9999999958924 metres de distància).



0.7 (Extra) Trobem l'angle indicat perquè el projectil arribi a un objectiu donat

Tot i que és un mètode molt menys eficient, també podem trobar l'angle ideal a partir d'una cerca dicotòmica. És a dir, iniciem suposant que l'angle es troba entre $0^{\underline{o}}$ i $90^{\underline{o}}$, i a partir d'aqui prenem el punt mig del nostre interval i mirem si amb tal angle quedem curts o ens passem del nostre objectiu. En cas que ens passem de llarg, per la següent iteració el punt mig passarà a ser l'extrem dret, mentre que l'extrem esquerre quedarà intacte. Si en canvi fem curt, deixarem l'extrem dret intacte i el punt mig serà l'esquerre.

No obstant el més interessant d'usar aquesta alternativa, és que arribem a un angle diferent amb el que també s'arriba a l'objectiu desitjat. Mentre que per $D = 500$, amb l'algorisme anterior trobem que l'angle ideal és de $52^{\underline{o}}$, amb aquest mètode arribem al resultat de $27.94921875^{\underline{o}}$. El procediment utilitzat és el següent:

```
tol = 1e-1
hi2 = np.pi/2
phi1 = 0
E = 100
while(E >= tol):
    SolRK45_2 = sc.solve_ivp(f, [0,20], angle((phi2+phi1)/2), rtol=1e-10,
            events=Terra)
    plt.plot(SolRK45_2.y[0],SolRK45_2.y[1], 'o')
    E = abs(SolRK45_2.y_events[0][1][0]-500)
    if(E < tol):
        print("L'angle trobat s : ", (phi1+phi2)/2 * 180/np.pi,
                " (en graus)")
    elif(SolRK45_2.y_events[0][1][0]-500 > 0):
        phi2 = (phi2+phi1)/2
    else:
        phi1 = (phi2+phi1)/2
plt.show()
```

Podem observar també que fent una petita modificació, si definim que l'interval inicial va de $45^{\underline{o}}$ a $90^{\underline{o}}$, i invertim l'actualització de $phi1$ i $phi2$, podem trobar el mateix angle que amb el mètode de Newton. Invertim l'actualització dels extrems de l'interval ja que arribat per angles majors a 45 graus, si ens quedem curts vol dir que l'angle és massa gran, el projectil va molt amunt però recorre pocs metres:

```
tol = 1e-1
phi2 = np.pi/2
phi1 = np.pi/4
E = 100
while(E >= tol):
    SolRK45_2 = sc.solve_ivp(f, [0,20], angle((phi2+phi1)/2), rtol=1e-10,
```

```
                events=Terra)
        plt.plot(SolRK45_2.y[0],SolRK45_2.y[1], '-o')
        E = abs(SolRK45_2.y_events[0][1][0]-500)
        if(E < tol):
            print("L'angle trobat s : ", (phi1+phi2)/2 * 180/np.pi,
                  " (en graus)")
        elif(SolRK45_2.y_events[0][1][0]-500 > 0):
            phi1 = (phi2+phi1)/2
        else:
            phi2 = (phi2+phi1)/2
    plt.show()
```

Els gràfics obtinguts són els següents: