soluzioni esercizi fisica applicata Lezione 1

November 16, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
## see https://matplotlib.org/users/customizing.html to customize
import matplotlib
matplotlib.rc('text', usetex=True)
matplotlib.rc('font', family='serif')
matplotlib.rc('figure',figsize=( 9.6, 6.8))
matplotlib.rc('xtick',labelsize=20)
matplotlib.rc('ytick',labelsize=20)
matplotlib.rc('legend',fontsize='x-large')
matplotlib.rc('xtick',direction='in')
matplotlib.rc('xtick',labelsize=24)
matplotlib.rc('xtick.major',size=7)
matplotlib.rc('xtick.minor',size=4)
matplotlib.rc('xtick.major',width=1.25)
matplotlib.rc('xtick.minor',width=1.25)
matplotlib.rc('ytick',direction='in')
matplotlib.rc('ytick',labelsize=24)
matplotlib.rc('ytick.major',size=7)
matplotlib.rc('ytick.minor',size=4)
matplotlib.rc('ytick.major',width=1.25)
matplotlib.rc('ytick.minor',width=1.25)
matplotlib.rc('patch',edgecolor='k' )
# matplotlib.rc('patch',force_edgecolor=True)
# matplotlib.rc('scatter',markeredgecolor='k')
matplotlib.rc('legend',markerscale=1.5)
matplotlib.rc('errorbar',capsize = 3)
matplotlib.rc('axes',labelsize='x-large')
matplotlib.rc('axes',linewidth=1.25)
```

Esercizio 1.1

a) 1 μ secolo = (10^{-6} secoli) $\frac{100\text{anni}}{1\text{secolo}}\frac{365\text{giorni}}{1\text{anno}}\frac{24\text{h}}{1\text{giorno}}\frac{60\text{minuti}}{1\text{h}} = 52.6$ min

b) la differenza percentuale e' quindi:

$\frac{52.6-50}{52.6} = 4.9\%$

Esercizio 1.2

[35]:
```
# Dati
x01=0. # km
x02=350. # km
v01=100. # km/h
v02=-80. # km/h
t01=0. # h
t02=2. # h
```

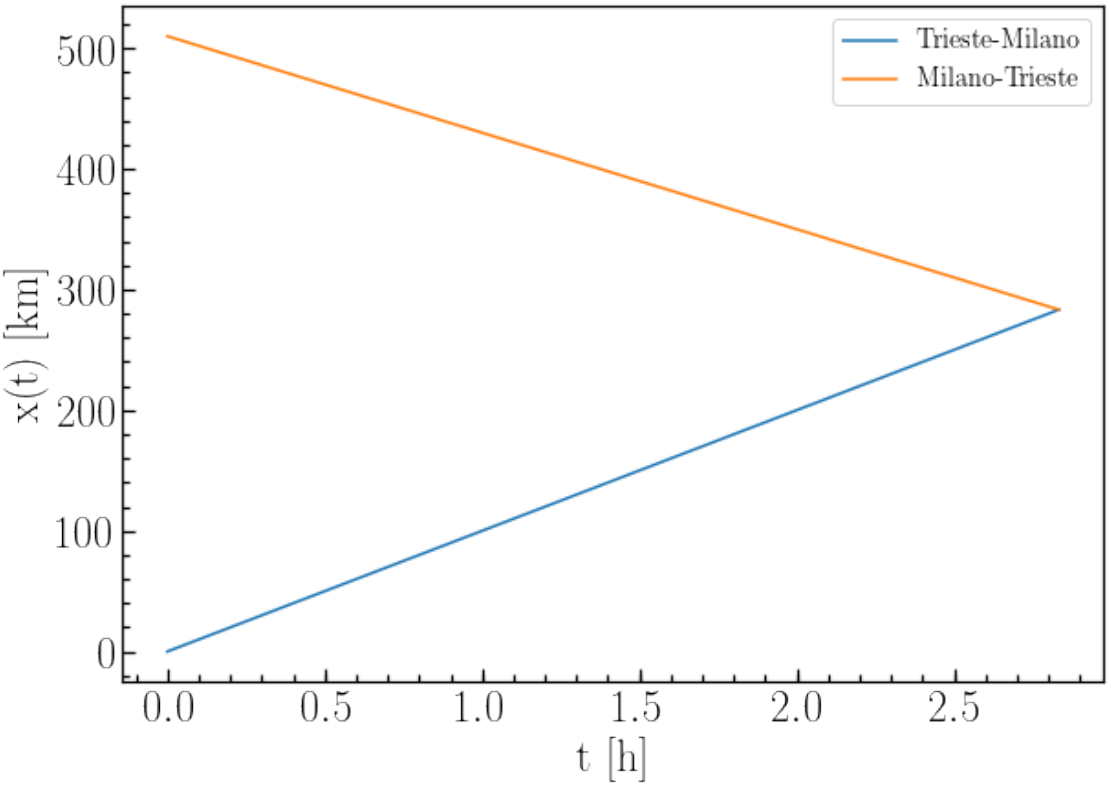
[60]:
```
# ugualiamo le due leggi orarie del moto rettilineo uniforme per trovare il␣
 ↪momento in cui i due treni si incrociano
# x1= x01 + v01*(t-t01)
# x2= x02 + v02*(t-t02)

t_in = (x02 - v02*t02) / (v01 - v02)

print('tempo trascorso dalla partenza del primo treno in cui si␣
 ↪incontrano',t_in,'h')
print('orario di incontro',14+t_in//1,':',t_in%1*60)

# distanza da Trieste in cui si incontrano i treni (assumendo che trieste si␣
 ↪trovi al km 0 e milano al km 350):
x1_tin = v01*t_in
print('distanza da Trieste in cui si incontrano i treni:', x1_tin,'km')

x2_tin = x02 + v02*(t_in-t02)
print('distanza da Trieste in cui si incontrano i treni:', x2_tin,'km')
```

```
('tempo trascorso dalla partenza del primo treno in cui si incontrano',
2.8333333333333335, 'h')
('orario di incontro', 16.0, ':', 50.00000000000001)
('distanza da Trieste in cui si incontrano i treni:', 283.33333333333337, 'km')
('distanza da Trieste in cui si incontrano i treni:', 283.3333333333333, 'km')
```

[46]:
```
# plot moto dei due treni
def x1(t):
    return x01 + v01*(t-t01)

def x2(t):
    return x02 + v02*(t-t02)


t_grid=np.linspace(0.,t_in,100)

plt.plot(t_grid,x1(t_grid),label='Trieste-Milano')
plt.plot(t_grid,x2(t_grid),label='Milano-Trieste')
plt.ylabel('x(t) [km]',fontsize=24)
plt.xlabel('t [h]',fontsize=24)
```

```
plt.minorticks_on()
plt.legend()
```

[46]: <matplotlib.legend.Legend at 0x7fb14eafb450>



Esercizio 1.3

[48]:
```
# dati
x1=0.
v1=161./3.6 # [m/s]
x2=676. # [m]
v2=29./3.6 # [m/s]
a1=0.95 # m/s^2 (modulo dell'accelerazione, il meno lo metto dentro le
 ↪equazioni)
d=x2-x1
```

[49]:
```
# Se non voglio che tamponino il treno deve raggiungere la locomotiva con al
 ↪piu' la stessa velocita
# (cosi che la raggiunge con velocita' relativa nulla)
# Per trovare il tempo in cui le due locomotive  hanno la stessa velocita
 ↪risolvo
# v1 - at = v2 -> t = (v1 - v2) /a
```

```
# Nell'istante che il treno raggiunge la locomotiva vale l'ugualianza:
# x1 + v1t - 0.5at^2 = x2 + v2t -> x2-x1 = (v1-v2)t - 0.5 at^2
# sostituendo t nell'equazione ed esplicitando l'accelerazione trovo

# decellerazione minima per non tamponare
a_min=1./2./(x2-x1)*(v1-v2)**2.
print('Accelerazione minima per non tamponare ',a_min,'m/s^2')
# a1=-0.95 non e' sufficiente a prevenire il tamponamento
```

```
('Accelerazione minima per non tamponare ', 0.9944115713346482, 'm/s^2')
```

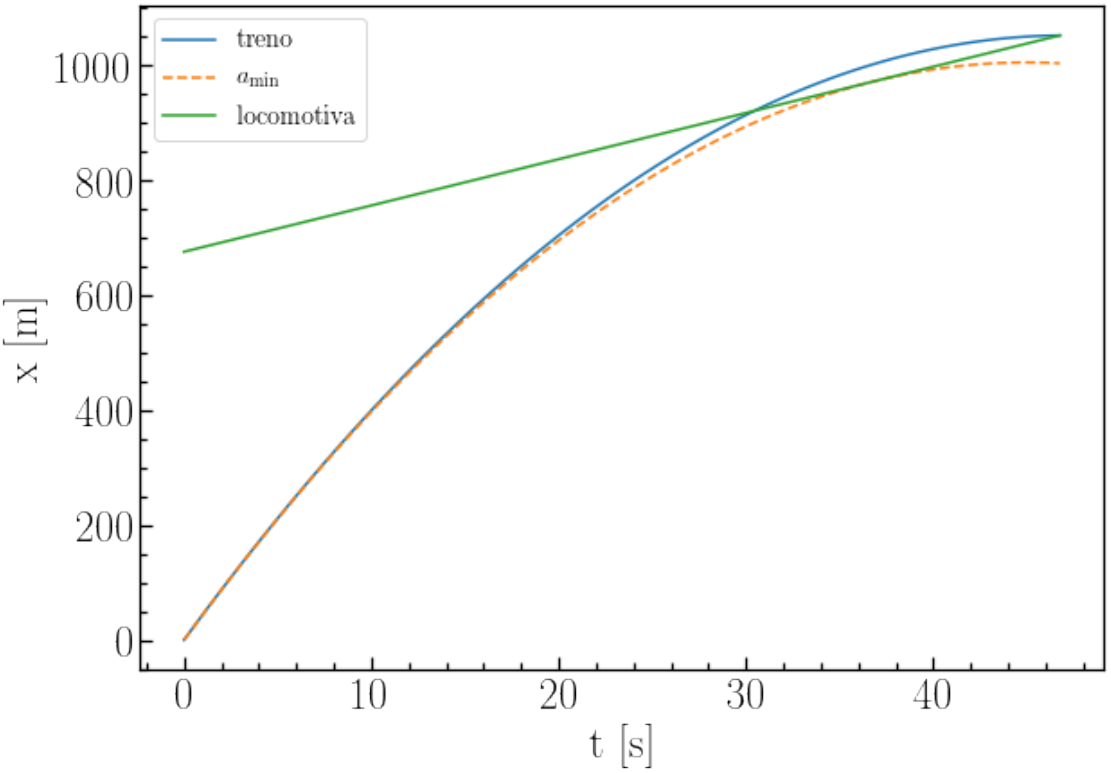
[50]:
```
# La stessa cosa la potevo notare dal fatto che risolvendo x1 + v1t - 0.5at^2 =
 ↪x2 + v2t
# rispetto al tempo, trovo due soluzioni reali (correspondenti al "sorpasso"
 ↪della locomotiva da parte del treno
# ed il successivo sorpasso del terno da parte dalla locomotiva man mano che il
 ↪primo decellera)

t1=( (v1-v2) - np.sqrt((v2-v1)**2.-2*a1*d))/a1

print('Istante in cui il treno tampona la locomotiva',t1,'s')

t2=( (v1-v2) + np.sqrt((v2-v1)**2.-2*a1*d))/a1

print('istante in cui la locomotiva superebbe il treno se si trovassero su due
 ↪binari paralleli',t2,'s')

# la condizione a_min >= 1./2./(x2-x1)*(v1-v2)**2. corrisponde e chiedere che
 ↪il termine sotto radice sia <=0
```

```
('Istante in cui il treno tampona la locomotiva', 30.439822767132465, 's')
('istante in cui la locomotiva superebbe il treno se si trovassero su due binari
paralleli', 46.75315968900788, 's')
```

[51]:
```
def x(t,x0,v0,a,t0=0):
    return x0+v0*(t-t0)-0.5*a*(t-t0)**2.

t_grid=np.linspace(0.,t2,100)

x0=x1
v0=v1
a=a1
plt.plot(t_grid,x(t_grid,x0,v0,a),label='treno')

x0=x1
v0=v1
a=a_min
```

```
plt.plot(t_grid,x(t_grid,x0,v0,a),'--',label='$a_{\\rm min}$')

x0=x2
v0=v2
a=0
plt.plot(t_grid,x(t_grid,x0,v0,a),label='locomotiva')
plt.xlabel('t [s]',fontsize=24)
plt.ylabel('x [m]',fontsize=24)
plt.minorticks_on()
plt.legend()
```

[51]: <matplotlib.legend.Legend at 0x7fb14df52e90>



Esercizio 1.4

[52]:
```
# dati
t1=1. # [s]
h=20. # [m]
g=9.81 # [m/s^2]

# tempo caduta prima pallina
t_fin = np.sqrt(2*h/g)
print('tempo caduta prima pallina', t_fin,'s')
```

```
('tempo caduta prima pallina', 2.019275109384609, 's')
```

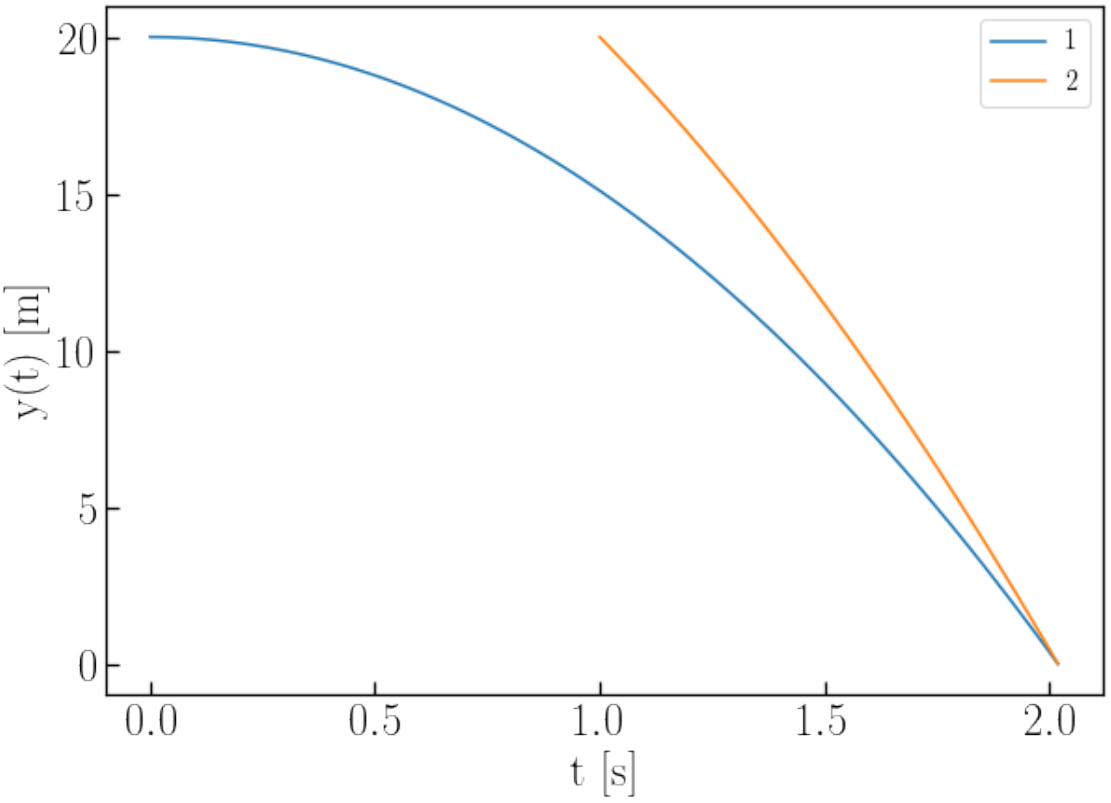
[53]:
```
# vogliamo trovare la velocita' della pallina 2 perche' tocchi terra nello␣
 ↪stesso istante;
# ovvero y2(t_fin) = 0 = h - v0*(t_fin - t1) - 1/2*g*(t_fin-t1)^2; esplicitando␣
 ↪v0:

v0 = (h - 0.5*g*(t_fin-t1)**2.)/(t_fin-t1)

print('velocita iniziale', v0,'m/s, ovvero', v0*3.6,'km/h')

v_fin1 = -g*(t_fin)

print('velocita prima pallina all arrivo al suolo', v_fin1,'m/s, ovvero',␣
 ↪v_fin1*3.6,'km/h' )

v_fin2 = -v0 -g*(t_fin-t1)

print('velocita seconda pallina all arrivo al suolo', v_fin2,'m/s, ovvero',␣
 ↪v_fin2*3.6,'km/h' )
```

```
('velocita iniziale', 14.622243480527462, 'm/s, ovvero', 52.64007652989886,
'km/h')
('velocita prima pallina all arrivo al suolo', -19.809088823063014, 'm/s,
ovvero', -71.31271976302685, 'km/h')
('velocita seconda pallina all arrivo al suolo', -24.621332303590478, 'm/s,
ovvero', -88.63679629292572, 'km/h')
```

[57]:
```
# legge oraria della prima pallina
def y1(t):
    return h - 0.5*g*(t)**2.

# legge oraria della seconda pallina
def y2(t):
    return h - v0*(t-t1) - 0.5*g*(t-t1)**2.

def v1(t):
    return -g*t

def v2(t):
    return -v0-g*(t-t1)

t_grid=np.linspace(0.,t_fin,100)
plt.plot(t_grid,y1(t_grid),label='1')
t_grid=np.linspace(t1,t_fin,100)
plt.plot(t_grid,y2(t_grid),label='2')
plt.xlabel('t [s]',fontsize=24)
```

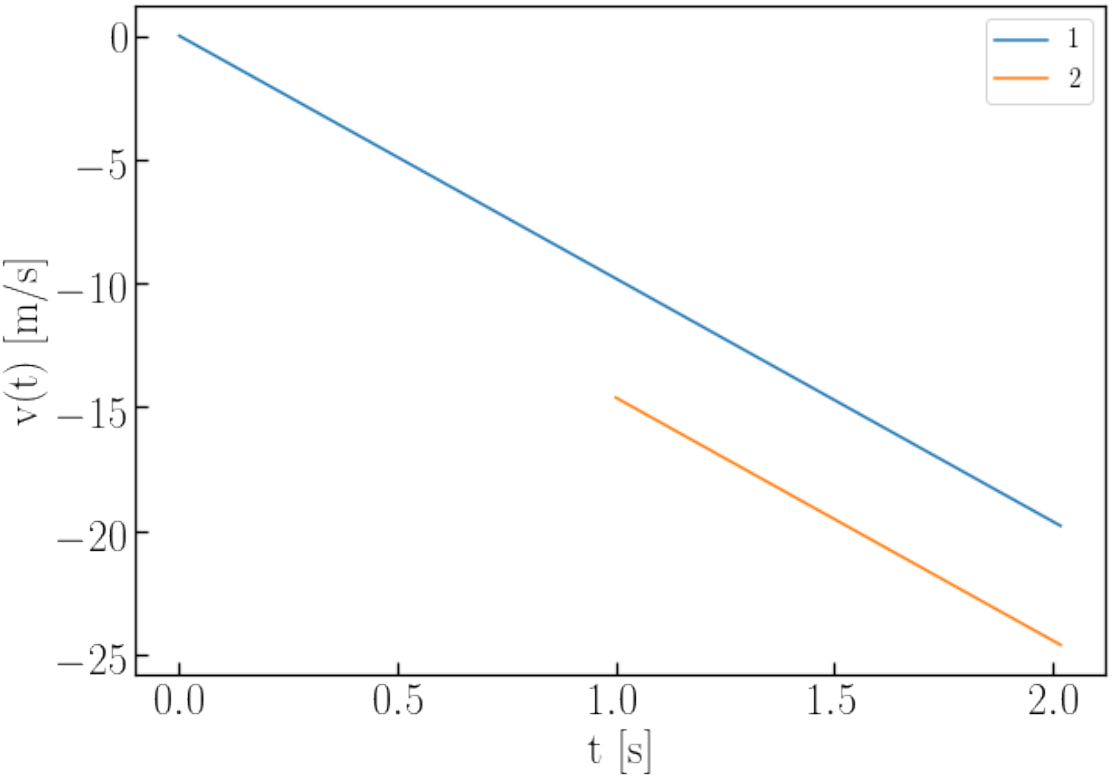
```
plt.ylabel('y(t) [m]',fontsize=24)
plt.legend()
```

[57]: <matplotlib.legend.Legend at 0x7fb14f098d90>



[58]:
```
t_grid=np.linspace(0.,t_fin,100)
plt.plot(t_grid,v1(t_grid),label='1')
t_grid=np.linspace(t1,t_fin,100)
plt.plot(t_grid,v2(t_grid),label='2')
plt.xlabel('t [s]',fontsize=24)
plt.ylabel('v(t) [m/s]',fontsize=24)
plt.legend()
```

[58]: <matplotlib.legend.Legend at 0x7fb14e1b7e50>


0
−5
−10
−15
−20
−25
v(t) [m/s]
0.0
0.5
1.0
1.5
2.0
t [s]
1
2

[ ]: