hw12

April 7, 2024

[1]:
```
import numpy as np
import numpy.linalg as la
```

[4]:
```
r = { "student01": {"% exercises":  20, "score": 55},
      "student02": {"% exercises": 100, "score": 100},
      "student03": {"% exercises":  90, "score": 100},
      "student04": {"% exercises":  70, "score": 70},
      "student05": {"% exercises":  50, "score": 75},
      "student06": {"% exercises":  10, "score": 25},
      "student07": {"% exercises":  30, "score": 60}
    }
```

[7]:
```
M = np.array([[r[s]["% exercises"], r[s]["score"]] for s in r.keys()])
M
```

[7]:
```
array([[ 20,  55],
       [100, 100],
       [ 90, 100],
       [ 70,  70],
       [ 50,  75],
       [ 10,  25],
       [ 30,  60]])
```

[85]:
```
from itertools import chain,combinations
def nset(n,iterable):
    s = list(iterable)
    return list(combinations(s,n))
```

[86]:
```
nset(3,['a','b','c','d','e'])
```

[86]:
```
[('a', 'b', 'c'),
 ('a', 'b', 'd'),
 ('a', 'b', 'e'),
 ('a', 'c', 'd'),
 ('a', 'c', 'e'),
 ('a', 'd', 'e'),
 ('b', 'c', 'd'),
 ('b', 'c', 'e'),
```

```
 ('b', 'd', 'e'),
 ('c', 'd', 'e')]
```

[89]:
```
correct = { 'a': 150,
            'b': 125,
            'c': 275,
            'd': 80,
            'e': 300
          }

def mkestimates(correct):
    keys = correct.keys()
    return { (x,y,z): np.round(correct[x] + correct[y] + correct[z] + rng.
 ↪uniform(-2,2),2)
              for (x,y,z) in nset(3,keys) }
```

[91]:
```
mass_estimates = mkestimates(correct)

mass_estimates
```

[91]:
```
{('a', 'b', 'c'): 551.03,
 ('a', 'b', 'd'): 353.19,
 ('a', 'b', 'e'): 574.36,
 ('a', 'c', 'd'): 506.1,
 ('a', 'c', 'e'): 724.92,
 ('a', 'd', 'e'): 531.9,
 ('b', 'c', 'd'): 478.21,
 ('b', 'c', 'e'): 701.98,
 ('b', 'd', 'e'): 504.75,
 ('c', 'd', 'e'): 653.07}
```

[93]:
```
# as a vector...
b=[mass_estimates[k] for k in mass_estimates.keys() ]
b
```

[93]:
```
[551.03, 353.19, 574.36, 506.1, 724.92, 531.9, 478.21, 701.98, 504.75, 653.07]
```

[95]:
```
def sbv(i,n):
    return np.array([1 if j ==i else 0 for j in range(n)])

def sbvalpha(elem,ls):
    return sbv(list(ls).index(elem),len(ls))
```

```
M = np.array([sbvalpha(x,correct.keys()) + sbvalpha(y,correct.keys()) +␣
↪sbvalpha(z,correct.keys()) for (x,y,z) in mass_estimates])
b = np.array([mass_estimates[x] for x in mass_estimates.keys()])
(M,b)
```

[95]:
```
(array([[1, 1, 1, 0, 0],
        [1, 1, 0, 1, 0],
        [1, 1, 0, 0, 1],
        [1, 0, 1, 1, 0],
        [1, 0, 1, 0, 1],
        [1, 0, 0, 1, 1],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 0, 1],
        [0, 1, 0, 1, 1],
        [0, 0, 1, 1, 1]]),
 array([551.03, 353.19, 574.36, 506.1 , 724.92, 531.9 , 478.21, 701.98,
       504.75, 653.07]))
```

[97]:
```
(M,b)
```

[97]:
```
(array([[1, 1, 1, 0, 0],
        [1, 1, 0, 1, 0],
        [1, 1, 0, 0, 1],
        [1, 0, 1, 1, 0],
        [1, 0, 1, 0, 1],
        [1, 0, 0, 1, 1],
        [0, 1, 1, 1, 0],
        [0, 1, 1, 0, 1],
        [0, 1, 0, 1, 1],
        [0, 0, 1, 1, 1]]),
 array([551.03, 353.19, 574.36, 506.1 , 724.92, 531.9 , 478.21, 701.98,
       504.75, 653.07]))
```

[102]:
```
M.transpose() @ M
```

[102]:
```
array([[6, 3, 3, 3, 3],
       [3, 6, 3, 3, 3],
       [3, 3, 6, 3, 3],
       [3, 3, 3, 6, 3],
       [3, 3, 3, 3, 6]])
```

[74]:
```
aa=la.lstsq(M,b,rcond=None)
aa
```

[74]:
```
(array([149.6235    , 125.3465    , 275.0235    ,  79.79183333,
        300.5365    ]),
 array([8.07883383]),
```

 5,
 array([4.24264069, 1.73205081, 1.73205081, 1.73205081, 1.73205081]))

[117]:
```
def f(x,v0,h0):
    return -9.8*x**2/2 + v0*x + h0

def ff(x):
    return f(x,10,200)

[ (x,ff(x)) for x in np.arange(0,7.55,.05)]
```

[117]:
```
[(0.0, 200.0),
 (0.05, 200.48775),
 (0.1, 200.951),
 (0.15000000000000002, 201.38975),
 (0.2, 201.804),
 (0.25, 202.19375),
 (0.30000000000000004, 202.559),
 (0.35000000000000003, 202.89975),
 (0.4, 203.216),
 (0.45, 203.50775),
 (0.5, 203.775),
 (0.55, 204.01775),
 (0.6000000000000001, 204.236),
 (0.65, 204.42975),
 (0.7000000000000001, 204.599),
 (0.75, 204.74375),
 (0.8, 204.864),
 (0.8500000000000001, 204.95974999999999),
 (0.9, 205.031),
 (0.9500000000000001, 205.07775),
 (1.0, 205.1),
 (1.05, 205.09775),
 (1.1, 205.071),
 (1.1500000000000001, 205.01975),
 (1.2000000000000002, 204.944),
 (1.25, 204.84375),
 (1.3, 204.719),
 (1.35, 204.56975),
 (1.4000000000000001, 204.396),
 (1.4500000000000002, 204.19774999999998),
 (1.5, 203.975),
 (1.55, 203.72775),
 (1.6, 203.456),
 (1.6500000000000001, 203.15975),
 (1.7000000000000002, 202.839),
 (1.75, 202.49375),
```

```
(1.8, 202.124),
(1.85, 201.72975),
(1.9000000000000001, 201.311),
(1.9500000000000002, 200.86775),
(2.0, 200.4),
(2.0500000000000003, 199.90775),
(2.1, 199.391),
(2.15, 198.84975),
(2.2, 198.284),
(2.25, 197.69375),
(2.3000000000000003, 197.079),
(2.35, 196.43975),
(2.4000000000000004, 195.77599999999998),
(2.45, 195.08775),
(2.5, 194.375),
(2.5500000000000003, 193.63774999999998),
(2.6, 192.876),
(2.6500000000000004, 192.08974999999998),
(2.7, 191.279),
(2.75, 190.44375),
(2.8000000000000003, 189.584),
(2.85, 188.69975),
(2.9000000000000004, 187.791),
(2.95, 186.85775),
(3.0, 185.9),
(3.0500000000000003, 184.91774999999998),
(3.1, 183.911),
(3.1500000000000004, 182.87974999999997),
(3.2, 181.82399999999998),
(3.25, 180.74375),
(3.3000000000000003, 179.63899999999998),
(3.35, 178.50975),
(3.4000000000000004, 177.356),
(3.45, 176.17775),
(3.5, 174.975),
(3.5500000000000003, 173.74775),
(3.6, 172.49599999999998),
(3.6500000000000004, 171.21974999999998),
(3.7, 169.91899999999998),
(3.75, 168.59375),
(3.8000000000000003, 167.24399999999997),
(3.85, 165.86974999999998),
(3.9000000000000004, 164.47099999999998),
(3.95, 163.04775),
(4.0, 161.6),
(4.05, 160.12775),
(4.1000000000000005, 158.63099999999997),
```

```
(4.15, 157.10974999999996),
(4.2, 155.564),
(4.25, 153.99374999999998),
(4.3, 152.399),
(4.350000000000005, 150.77974999999998),
(4.4, 149.13599999999997),
(4.45, 147.46774999999997),
(4.5, 145.77499999999998),
(4.55, 144.05775),
(4.600000000000005, 142.31599999999997),
(4.65, 140.54974999999996),
(4.7, 138.75899999999996),
(4.75, 136.94375),
(4.800000000000001, 135.10399999999996),
(4.850000000000005, 133.23975),
(4.9, 131.35099999999997),
(4.95, 129.43775),
(5.0, 127.49999999999999),
(5.050000000000001, 125.53774999999996),
(5.100000000000005, 123.55099999999996),
(5.15, 121.53974999999997),
(5.2, 119.50399999999999),
(5.25, 117.44375),
(5.300000000000001, 115.35899999999995),
(5.350000000000005, 113.24974999999995),
(5.4, 111.11599999999999),
(5.45, 108.95774999999998),
(5.5, 106.77499999999998),
(5.550000000000001, 104.56774999999993),
(5.600000000000005, 102.33599999999996),
(5.65, 100.07974999999996),
(5.7, 97.79899999999998),
(5.75, 95.49374999999998),
(5.800000000000001, 93.16399999999996),
(5.850000000000005, 90.80974999999998),
(5.9, 88.43099999999998),
(5.95, 86.02774999999997),
(6.0, 83.6),
(6.050000000000001, 81.14774999999995),
(6.100000000000005, 78.67099999999994),
(6.15, 76.16974999999996),
(6.2, 73.64399999999995),
(6.25, 71.09375),
(6.300000000000001, 68.51899999999992),
(6.350000000000005, 65.91974999999996),
(6.4, 63.295999999999935),
(6.45, 60.64775),
```

```
(6.5, 57.974999999999994),
(6.550000000000001, 55.27774999999994),
(6.6000000000000005, 52.555999999999926),
(6.65, 49.80974999999998),
(6.7, 47.03899999999999),
(6.75, 44.24374999999998),
(6.800000000000001, 41.42399999999995),
(6.8500000000000005, 38.57974999999996),
(6.9, 35.710999999999956),
(6.95, 32.81774999999996),
(7.0, 29.899999999999977),
(7.050000000000001, 26.957749999999947),
(7.1000000000000005, 23.99099999999993),
(7.15, 20.999749999999977),
(7.2, 17.983999999999952),
(7.25, 14.943749999999966),
(7.300000000000001, 11.878999999999905),
(7.3500000000000005, 8.78974999999997),
(7.4, 5.675999999999931),
(7.45, 2.53774999999996),
(7.5, -0.625)]
```

[107]:
```
x = np.arange(0,20,.5)
x
```

[107]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5,
       11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5, 16. ,
       16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5])
```

[134]:
```
#height_estimates = { x: np.round(ff(x) + rng.uniform(-5,5),2) for x in np.
 ↪arange(0,7.55,.25) }
#height_estimates
```

[134]:
```
{0.0: 199.6,
 0.25: 202.96,
 0.5: 207.23,
 0.75: 208.29,
 1.0: 207.47,
 1.25: 203.96,
 1.5: 199.18,
 1.75: 202.91,
 2.0: 204.29,
 2.25: 196.31,
 2.5: 195.71,
 2.75: 187.89,
 3.0: 187.61,
```

```
3.25: 177.12,
3.5: 171.07,
3.75: 171.89,
4.0: 158.68,
4.25: 152.64,
4.5: 146.7,
4.75: 138.52,
5.0: 127.27,
5.25: 122.38,
5.5: 103.97,
5.75: 96.91,
6.0: 83.08,
6.25: 67.34,
6.5: 55.75,
6.75: 45.42,
7.0: 25.33,
7.25: 14.67,
7.5: -1.45}
```

[136]:

```
height_estimates = {0.0: 199.6,
                    0.25: 202.96,
                    0.5: 207.23,
                    0.75: 208.29,
                    1.0: 207.47,
                    1.25: 203.96,
                    1.5: 199.18,
                    1.75: 202.91,
                    2.0: 204.29,
                    2.25: 196.31,
                    2.5: 195.71,
                    2.75: 187.89,
                    3.0: 187.61,
                    3.25: 177.12,
                    3.5: 171.07,
                    3.75: 171.89,
                    4.0: 158.68,
                    4.25: 152.64,
                    4.5: 146.7,
                    4.75: 138.52,
                    5.0: 127.27,
                    5.25: 122.38,
                    5.5: 103.97,
                    5.75: 96.91,
                    6.0: 83.08,
                    6.25: 67.34,
                    6.5: 55.75,
                    6.75: 45.42,
```

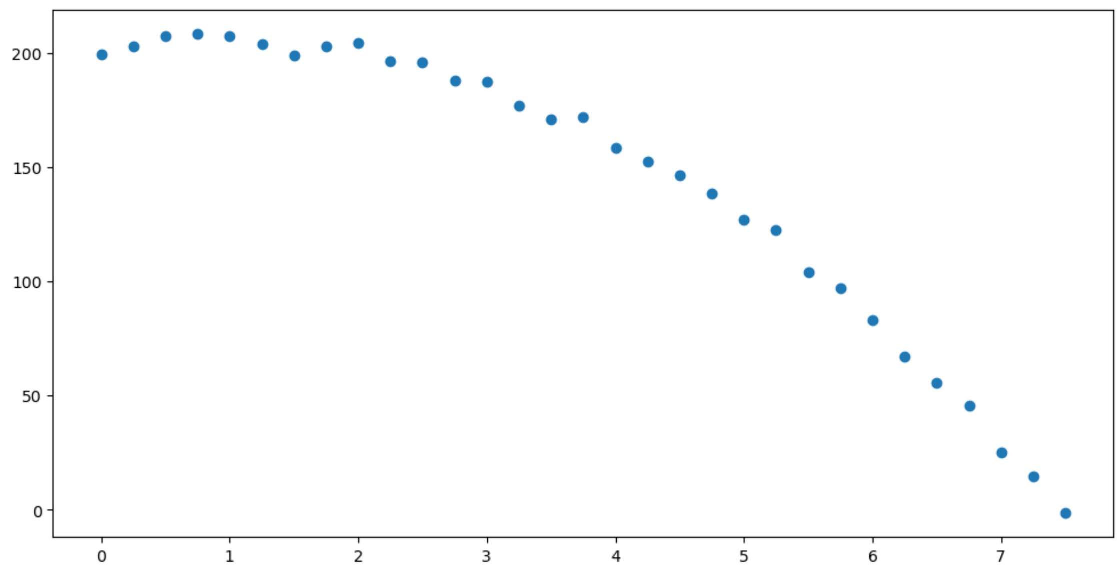
```
                        7.0: 25.33,
                        7.25: 14.67,
                        7.5: -1.45}
```

[197]:
```
import matplotlib.pyplot as plt

## lists x,y have been populated; lets plot the points
def plot_data(x,y):
    fig, ax = plt.subplots(figsize=(12,6))
    return ax.plot(x,y,"o")

x1 = np.array(list(height_estimates.keys()))
y1 = np.array(list(height_estimates.values()))
plot_data(x1,y1)
```

[197]:
```
[<matplotlib.lines.Line2D at 0x7f8f4f957fd0>]
```



[144]:
```
Mf = np.array([[t**2, t, 1] for t in height_estimates.keys() ])
Mb = np.array([height_estimates[k] for k in height_estimates.keys()])

coeffs = la.lstsq(Mf,Mb,rcond=None)[0]
coeffs
```

[144]:
```
array([ -4.87577207,   9.52224214, 201.12954545])
```

[153]:
```
MMf = np.array([[t, 1] for t in height_estimates.keys() ])
MMb = np.array([height_estimates[k] + (9.8/2)*k**2 for k in height_estimates.
 ↪keys()])
```

```
coeffs_alt = la.lstsq(MMf,MMb,rcond=None)[0]
coeffs_alt
```

[153]:
```
array([  9.70395161, 200.90997984])
```

[201]:
```
def mk_func(cc):

    # get the coefficients and report them
    if len(cc) == 3:
      alpha,beta,gamma = cc
    else:
      alpha = -9.8/2
      beta,gamma = cc
    print(f"f(t) = {alpha:.04}*t^2 + {beta:.04}*t + {gamma:.04}")

    # return the linear function determined by these coefficients
    return lambda t:alpha*t**2 + beta*t + gamma

fa = mk_func(coeffs)
fb = mk_func(coeffs_alt)
```

```
f(t) = -4.876*t^2 + 9.522*t + 201.1
f(t) = -4.9*t^2 + 9.704*t + 200.9
```

[191]:
```
aaaa = np.arange(5)
fa(aaaa)
(aaaa,x1)
```

[191]:
```
(array([0, 1, 2, 3, 4]),
 [0.0,
  0.25,
  0.5,
  0.75,
  1.0,
  1.25,
  1.5,
  1.75,
  2.0,
  2.25,
  2.5,
  2.75,
  3.0,
  3.25,
  3.5,
  3.75,
  4.0,
  4.25,
```

```
 4.5,
 4.75,
 5.0,
 5.25,
 5.5,
 5.75,
 6.0,
 6.25,
 6.5,
 6.75,
 7.0,
 7.25,
 7.5])
```

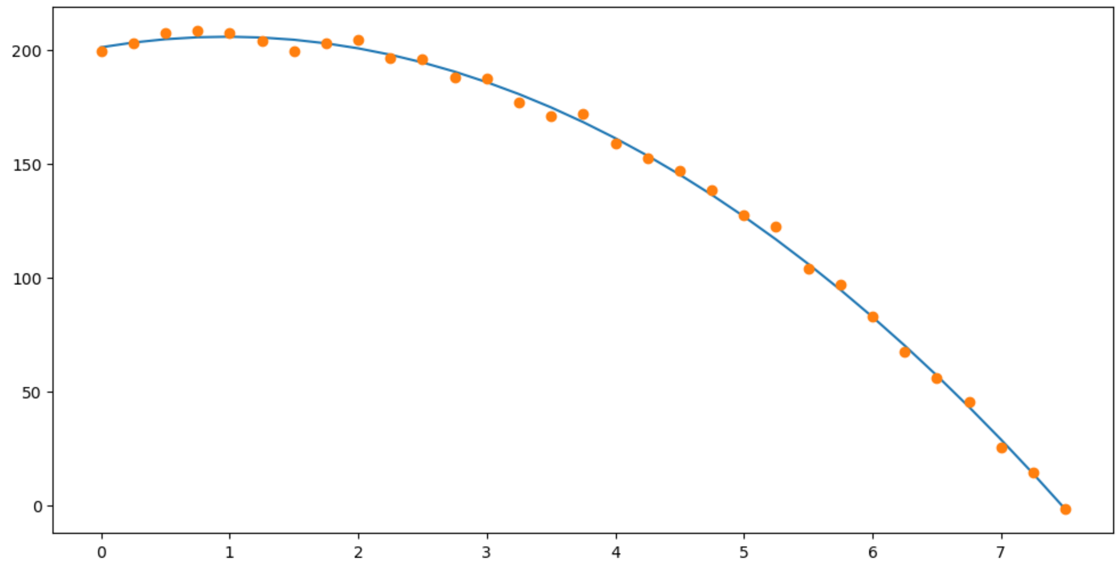
[174]:
```
def plot_curve_fit(x0,f,x,y):
    # graph the line with slope alpha and y-intercept beta, and plot the data␣
 ↪points
    #
    fig,ax = plt.subplots(figsize=(12,6))
    ax.plot(x0,f(x0))
    ax.plot(x,y,'o')

#    ax.plot(x0,f(x0))

    return fig,ax
```

[202]:
```
plot_curve_fit(x1,fa,x1,y1)
```

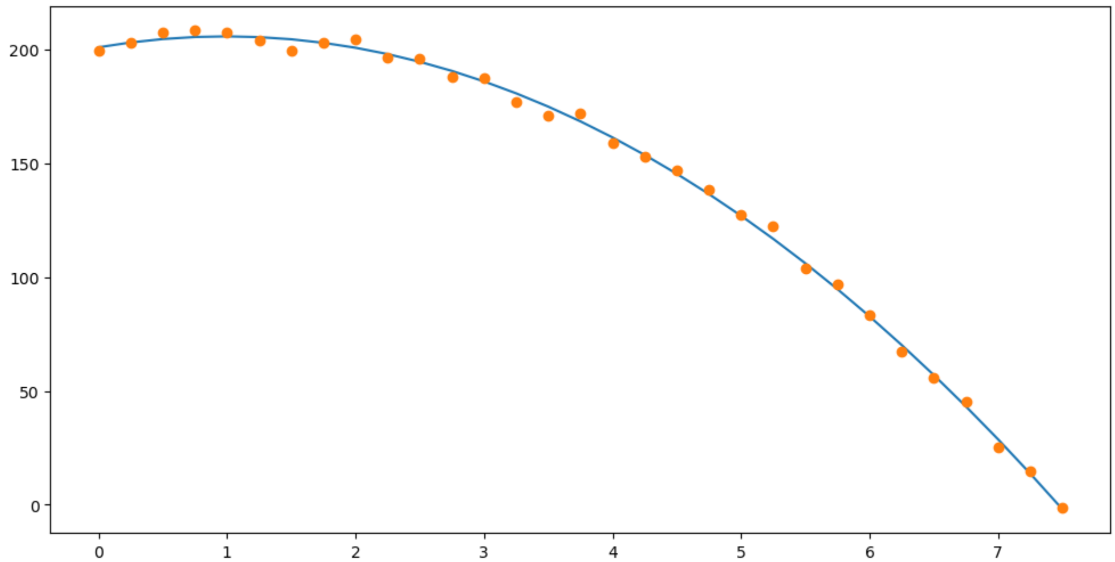
[202]:
```
(<Figure size 1200x600 with 1 Axes>, <Axes: >)
```



[203]:

```
plot_curve_fit(x1,fb,x1,y1)
```

[203]: (<Figure size 1200x600 with 1 Axes>, <Axes: >)



[ ]:

```
```