04.02 models

aside from linear and polynomial models, consider periodic data and linearization.

1 periodic data

periodic models for periodic data, lets say.

example 06

fit temperature log to a periodic model. below is the data for dc at the millenium rollover.

TOD	Δt	T (°C)
12mn	0	-2.2
3am	$\frac{1}{8}$	-2.8
6am	$\frac{1}{4}$	-6.1
9am	$\frac{3}{8}$	-3.9
12nn	$\frac{1}{2}$	0.0
3pm	$\frac{5}{8}$	1.1
6pm	$\frac{3}{4}$	-0.6
9pm	$\frac{7}{8}$	-1.1

choose model $y = c_1 + c_2 \ cos \ 2\pi t + c_3 \ sin \ 2\pi t$ with period 24h.

$$
\begin{aligned}
c_1 + c_2\ cos\ 2\pi(0) + c_3\ sin\ 2\pi(0) &= -2.2 \\
c_1 + c_2\ cos\ 2\pi(\frac{1}{8}) + c_3\ sin\ 2\pi(\frac{1}{8}) &= -2.8 \\
c_1 + c_2\ cos\ 2\pi(\frac{1}{4}) + c_3\ sin\ 2\pi(\frac{1}{4}) &= -6.1 \\
c_1 + c_2\ cos\ 2\pi(\frac{3}{8}) + c_3\ sin\ 2\pi(\frac{3}{8}) &= -3.9 \\
c_1 + c_2\ cos\ 2\pi(\frac{1}{2}) + c_3\ sin\ 2\pi(\frac{1}{2}) &= 0.0 \\
c_1 + c_2\ cos\ 2\pi(\frac{5}{8}) + c_3\ sin\ 2\pi(\frac{5}{8}) &= 1.1 \\
c_1 + c_2\ cos\ 2\pi(\frac{3}{4}) + c_3\ sin\ 2\pi(\frac{3}{4}) &= -0.6 \\
c_1 + c_2\ cos\ 2\pi(\frac{7}{8}) + c_3\ sin\ 2\pi(\frac{7}{8}) &= -1.1
\end{aligned}
$$

$$\Downarrow$$

$$
A = \begin{bmatrix} 1 & cos 0 & sin 0 \\ 1 & cos\frac{\pi}{4} & sin\frac{\pi}{4} \\ 1 & cos\frac{\pi}{2} & sin\frac{\pi}{2} \\ 1 & cos\frac{3\pi}{4} & sin\frac{3\pi}{4} \\ 1 & cos\pi & sin\pi \\ 1 & cos\frac{5\pi}{4} & sin\frac{5\pi}{4} \\ 1 & cos\frac{3\pi}{2} & sin\frac{3\pi}{2} \\ 1 & cos\frac{7\pi}{4} & sin\frac{7\pi}{4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 1 & 0 & 1 \\ 1 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 1 & -1 & 0 \\ 1 & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 1 & 0 & -1 \\ 1 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, b = \begin{bmatrix} -2.2 \\ -2.8 \\ -6.1 \\ -3.9 \\ 0.0 \\ 1.1 \\ -0.6 \\ -1.1 \end{bmatrix}
$$

$$\Downarrow$$

$$
A^T A c = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -15.6 \\ -2.9778 \\ -10.2376 \end{bmatrix}
$$

$$\Downarrow$$

$$
c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -1.95 \\ -0.7445 \\ -2.5594 \end{bmatrix}
$$

$$\Downarrow$$

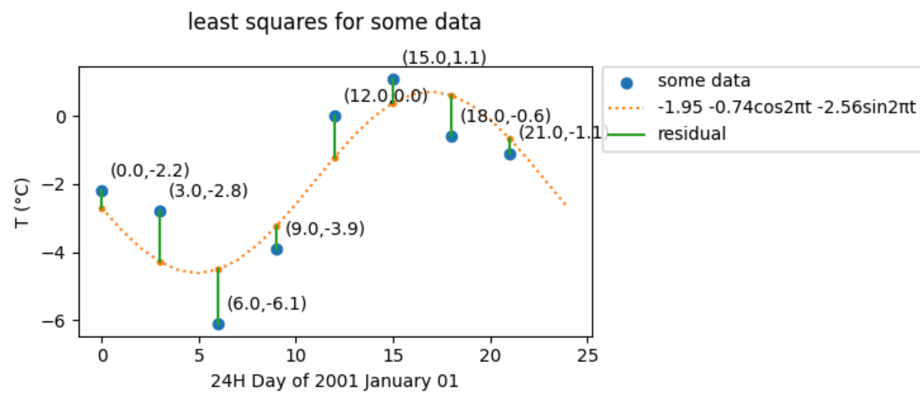
$$
\begin{aligned}
y = -0.19500 - 0.7445\ cos\ 2\pi t - 2.5594\ sin\ 2\pi t, \\
RMSE \approx 1.063.
\end{aligned}
$$

code

```
# example 06: uses basic least squares

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def main():
  p_per = lambda cs,t: cs[0] + cs[1]*np.cos(2*np.pi*t) + cs[2]*np.sin(2*np.pi*t)
  s_per = lambda cs: f"{cs[0]:.2f} {cs[1]:+.2f}cos2πt {cs[2]:+.2f}sin2πt"

```

```
LHS:
[[ 8.00000000e+00 -5.55111512e-16 -2.22044605e-16]
 [-5.55111512e-16  4.00000000e+00  3.43213444e-16]
 [-2.22044605e-16  3.43213444e-16  4.00000000e+00]]

RHS:  [-15.6         -2.97781746 -10.23761543]

[c]:  [-1.95       -0.74445436 -2.55940386]

SE: 9.040958351402141
RMSE: 1.0630709261028954
```

	t	y	model	error
0	0.0	-2.2	-2.694454	0.4945
1	3.0	-2.8	-4.286181	1.4862
2	6.0	-6.1	-4.509404	-1.5906
3	9.0	-3.9	-3.233363	-0.6666
4	12.0	0.0	-1.205546	1.2055
5	15.0	1.1	0.386181	0.7138
6	18.0	-0.6	0.609404	-1.2094
7	21.0	-1.1	-0.666637	-0.4334



example 07

example 06 continued with better model. $\longrightarrow$

choose model $y = c_1 + c_2\ cos\ 2\pi t + c_3\ sin\ 2\pi t + c_4\ cos\ 4\pi t$ with period 24h.

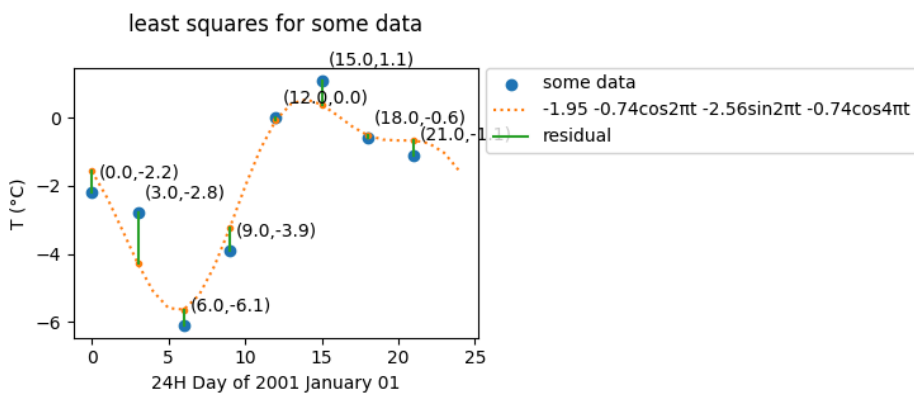
code

```
 1 # during lecture, ok?
 2
 3 #  # data for example 07
 4 #  p_per = lambda cs,t: cs[0] + cs[1]*np.cos(2*np.pi*t) + \
 5 #            cs[2]*np.sin(2*np.pi*t) + cs[3]*np.cos(4*np.pi*t)
 6 #  s_per = lambda cs: f"{cs[0]:.2f} {cs[1]:+.2f}cos2πt {cs[2]:+.2f}sin2πt {cs[1]:+.2f}cos4πt"
 7
 8 # 01 copy-pasted previous code cell for example 06
 9 # 02 renamed pasted code cell to example 07
10 # 03 replaced lines 08,09 with p_per,s_per above
11 # 04 updated line 16→17 for 4th term: "nu = 3" to "nu = 4"
12 # 05 added 4th term to first for-loop: "a[i,3] = np.cos(4*np.pi*xs[i])"
13 #
14 # nothing else needs to change - no labels, no anything - bc only model changed
15
```

```
 1 # example 07: uses basic least squares
 2
 3 import matplotlib.pyplot as plt
 4 import numpy as np
 5 import pandas as pd
 6
 7 def main():
 8   p_per = lambda cs,t: cs[0] + cs[1]*np.cos(2*np.pi*t) + \
 9             cs[2]*np.sin(2*np.pi*t) + cs[3]*np.cos(4*np.pi*t)
10   s_per = lambda cs: f"{cs[0]:.2f} {cs[1]:+.2f}cos2πt {cs[2]:+.2f}sin2πt {cs[1]:+.2f}cos4πt"
```

```
LHS:
[[ 8.00000000e+00 -5.55111512e-16 -2.22044605e-16 -4.28626380e-16]
 [-5.55111512e-16  4.00000000e+00  3.43213444e-16 -3.35876614e-16]
 [-2.22044605e-16  3.43213444e-16  4.00000000e+00  8.10400148e-17]
 [-4.28626380e-16 -3.35876614e-16  8.10400148e-17  4.00000000e+00]]

RHS:  [-15.6         -2.97781746 -10.23761543   4.5       ]

[c]:  [-1.95       -0.74445436 -2.55940386  1.125     ]

SE: 3.9784583514021388
RMSE: 0.7052001800377445
```

	t	y	model	error
0	0.0	-2.2	-1.569454	-0.6305
1	3.0	-2.8	-4.286181	1.4862
2	6.0	-6.1	-5.634404	-0.4656
3	9.0	-3.9	-3.233363	-0.6666
4	12.0	0.0	-0.080546	0.0805
5	15.0	1.1	0.386181	0.7138
6	18.0	-0.6	-0.515596	-0.0844
7	21.0	-1.1	-0.666637	-0.4334



2 data linearization

so about the lemming situation. the first part looks something like this:

$$y = c_1 e^{c_2 t}.$$

(the second part looks like $y_{\text{cliff}} = 2$, but lets not scare small children. today.)

least squares doesnt look like it will be ok with that but its nothing natural logs cant flatten.

$$ln\ y = ln(c_1 e^{c_2 t}) = ln\ c_1 + c_2 t.$$

with $k = ln\ c_1$,

$$k + c_2 t, \quad c_1 = e^k.$$

is that ok? its ok-ish = its ok enough.

if youre the supreme court and afraid of someone (with money) suing (instead of sharing with) you (or whatever it is they think theyre doing), you expect least squares find c_1, c_2 to minimize

$$(c_1 e^{c_2 t_1} - y_1)^2 + \cdots + (c_1 e^{c_2 t_m} - y_m)^2, \quad i = 1, \ldots, m.$$

and the translated problem minimizes

$$(ln\ c_1 + c_2 t_1 - y_1)^2 + \cdots + (ln\ c_1 + c_2 t_m - y_m)^2, \quad i = 1, \ldots, m$$

which results in a different c_1, c_2.

example 08

use model linearization to find best least squares exponential fit to the following world automobile supply data.

year	cars (M)
1950	53.05
1955	73.04
1960	98.31
1965	139.78
1970	193.48
1975	260.20
1980	320.39

$$k + c2t = ln\ y \quad \Rightarrow \quad \begin{bmatrix} 1 & 1950 - 1950 = 0 \\ \vdots & \vdots \\ 1 & 1980 - 1950 = 30 \end{bmatrix} \begin{bmatrix} k = ln\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} ln(53.05) \\ \vdots \\ ln(320.39) \end{bmatrix},$$

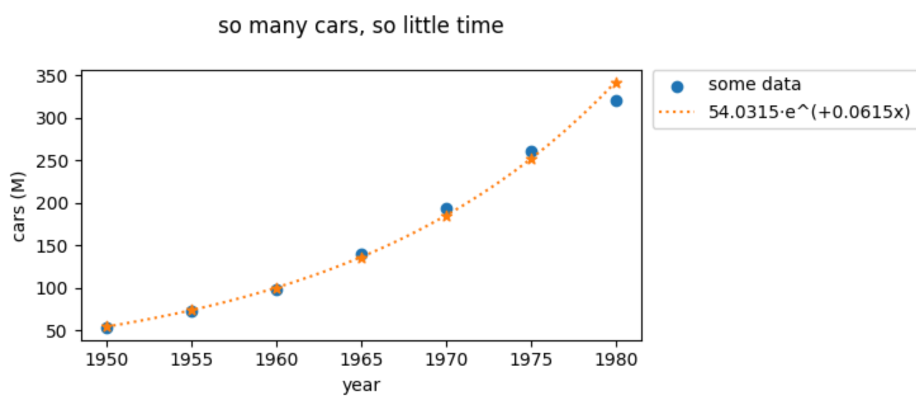
where $t \equiv \Delta t$ from 1950.

code

```
# example 08: uses polyfit

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def main():
  y_lin = lambda cs,x: cs[0]*np.exp(cs[1]*x)
  s_lin = lambda cs: f"{cs[0]:.4f}·e^({cs[1]:+.4f}x)"
  s_ys = "cars"
```

coeffs: [c,k] = 3.9896,0.0615 → y(x) = 54.0315·e^(0.0615x)

	year	cars	model	error
0	1950	53.05	54.031463	-0.9815
1	1955	73.04	73.491388	-0.4514
2	1960	98.31	99.959982	-1.6500
3	1965	139.78	135.961481	3.8185
4	1970	193.48	184.929249	8.5508
5	1975	260.20	251.533205	8.6668
6	1980	320.39	342.125184	-21.7352



example 09 that one law

fit model $y = c_1 e^{c_2 t}$ to transistor count on intel CPUs.

CPU	year	transistors
4004	1971	2250
8008	1972	2500
8080	1974	5000
8086	1978	29000
286	1982	120000
386	1985	275000
486	1989	1180000
Pentium	1993	3100000
Pentium II	1997	7500000
Pentium III	1999	24000000
Pentium 4	2000	42000000
Itanium	2002	220000000
Itanium 2	2003	410000000

$$k + c2t = ln\ y \quad \Rightarrow \quad \begin{bmatrix} 1 & 1971 - 1970 = 1 \\ \vdots & \vdots \\ 1 & 2003 - 1970 = 33 \end{bmatrix} \begin{bmatrix} k = ln\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} ln(2250) \\ \vdots \\ ln(410000000) \end{bmatrix},$$

where $t \equiv \Delta$ year from 1970.

$$A^T A x = A^T b \quad \Rightarrow \quad \begin{bmatrix} 13 & 235 \\ 235 & 5927 \end{bmatrix} \begin{bmatrix} k \\ c_2 \end{bmatrix} = \begin{bmatrix} 176.90 \\ 3793.23 \end{bmatrix},$$

which has solution $k \approx 7.197, c_2 \approx 0.3546 \Rightarrow y = 1335.3\ e^{0.3546t}$. gordon c moore, cofounder of intel, predicted the doubling law, $\Delta\ t = \frac{ln\ 2}{c_2} \approx 1.95$ years, tho it seems to have accelerated after you were born.

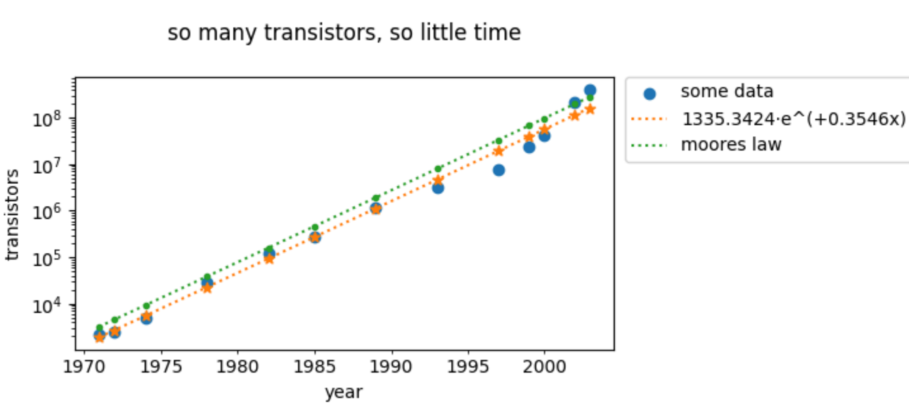
code

```
1   # during lecture, ok?
2
3   #  xs_yy_offset = 1970
4   #  h = 1
5   #  xs = np.array([1,2,4,8,12,15,19,23,27,29,30,32,33])
6   #  ys = np.array([2250,2500,5000,29000,120000,275000,1180000,3100000, \
7   #                  7500000,24000000,42000000,220000000,410000000]) # b
8
9   # 01 copy-pasted previous code cell for example 08
10  # 02 renamed pasted code cell to example 09
11  # 03 replaced lines 11-14 with lines 04-08 above
12  #
13  # 04 switch to log scale
14  #    add import statement: "import matplotlib.ticker as mticker"
15  #    before "plt.show()", add lines
16  #      plt.loglog()
17  #      ax = plt.gca()
18  #      ax.xaxis.set_minor_formatter(mticker.ScalarFormatter())
19  #
20  # 05 update labels from cars to (cpu) transistors
21  #
22  # 06 added moores law: moore = lambda y,t: y*pow(2,t/2)
23
```

```
# example 09: uses polyfit

import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import numpy as np
import pandas as pd

def main():⋯

if __name__ == "__main__":⋯
```

```
coeffs: [c,k] = 7.1969,0.3546 → y(x) = 1335.3424·e^(0.3546x)
```

	year	transistors	model	error	moores	err2
0	1971	2250.0	1.903754e+03	346.2461	3.210383e+03	-960.3832
1	1972	2500.0	2.714120e+03	-214.1197	4.580693e+03	-2080.6935
2	1974	5000.0	5.516522e+03	-516.5220	9.325668e+03	-4325.6679
3	1978	29000.0	2.278967e+04	6210.3281	3.865248e+04	-9652.4807
4	1982	120000.0	9.414793e+04	25852.0665	1.602045e+05	-40204.5326
5	1985	275000.0	2.728132e+05	2186.8303	4.653694e+05	-190369.4495
6	1989	1180000.0	1.127037e+06	52963.3194	1.928836e+06	-748835.9699
7	1993	3100000.0	4.655976e+06	-1555976.3992	7.994526e+06	-4894526.0755
8	1997	7500000.0	1.923461e+07	-11734614.6333	3.313524e+07	-25635242.2759
9	1999	24000000.0	3.909488e+07	-15094876.8385	6.745884e+07	-43458839.2157
10	2000	42000000.0	5.573629e+07	-13736286.3996	9.625277e+07	-54252766.4487
11	2002	220000000.0	1.132855e+08	106714483.5614	1.959575e+08	24042480.1523
12	2003	410000000.0	1.615075e+08	248492546.8566	2.795994e+08	130400564.0912



example 11 drugs

the time course of drug concentration y in the bloodstream is well described by

$$y = c_1 t e^{c_2 t},$$

where t denotes time after administration of drug. the characteristics of the model are a quick rise as the drug enters the bloodstream, followed by an exponential decay. so, yeah, drugs have a half-life.

$$\begin{aligned} ln\ y &= ln\ c_1 + ln\ t + ln\ c_2 t \\ k + c_2 t &= ln\ y - ln\ t, \quad k = ln\ c_1 \end{aligned}$$

$$\Downarrow$$

$$A = \begin{bmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}, b = \begin{bmatrix} ln\ y_1 - ln\ t_1 \\ \vdots \\ ln\ y_m - ln\ t_m \end{bmatrix}.$$

fit model $y = c_1 t e^{c_2 t}$ to measured level of the drug norfluoxetine in a patients bloodstream.

hour	concentration (ng/ml)
1	8.0
2	12.3
3	15.5
4	16.8
5	17.1
6	15.8
7	15.2
8	14.0

solution: $k \approx 2.28, c_2 \approx -0.215 \Rightarrow y = 9.77t \cdot e^{-0.215t}$.

code

```
 1 # during lecture
 2
 3 #  xs = np.array([1,2,3,4,5,6,7,8])
 4 #  ys = np.array([8.0,12.3,15.5,16.8,17.1,15.8,15.2,14.0])
 5 #  s_lin = lambda cs: f"{cs[0]:.4f}t·e^({cs[1]:+.4f}x)"
 6
 7 # 01 copy-pasted previous code cell for example 09
 8 # 02 renamed pasted code cell to example 11
 9 # 03 replaced data: lines 14-15 with lines 03-04 above
10 # 04 removed offset, line 12
```