curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

MCS 507 Lecture 15
Mathematical, Statistical and Scientific Software
Jan Verschelde, 30 September 2019

curve plotting and animations

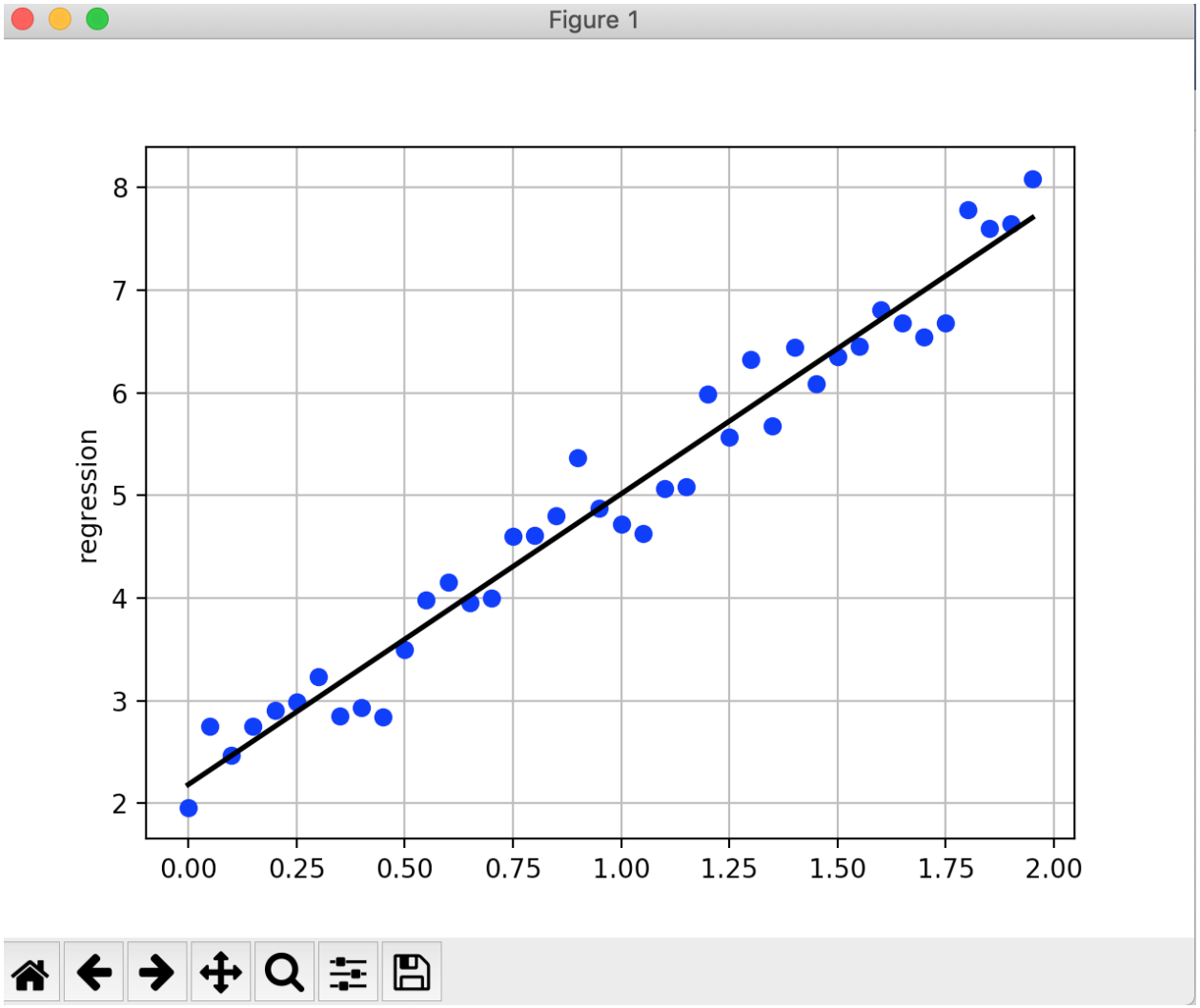
1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

IPython

IPython is an enhanced python, enhanced for scientific computing.

```
$ ipython --pylab
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
Type 'copyright', 'credits' or 'license' for more informati
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' f
Using matplotlib backend: MacOSX

In [1]:
```

linear regression



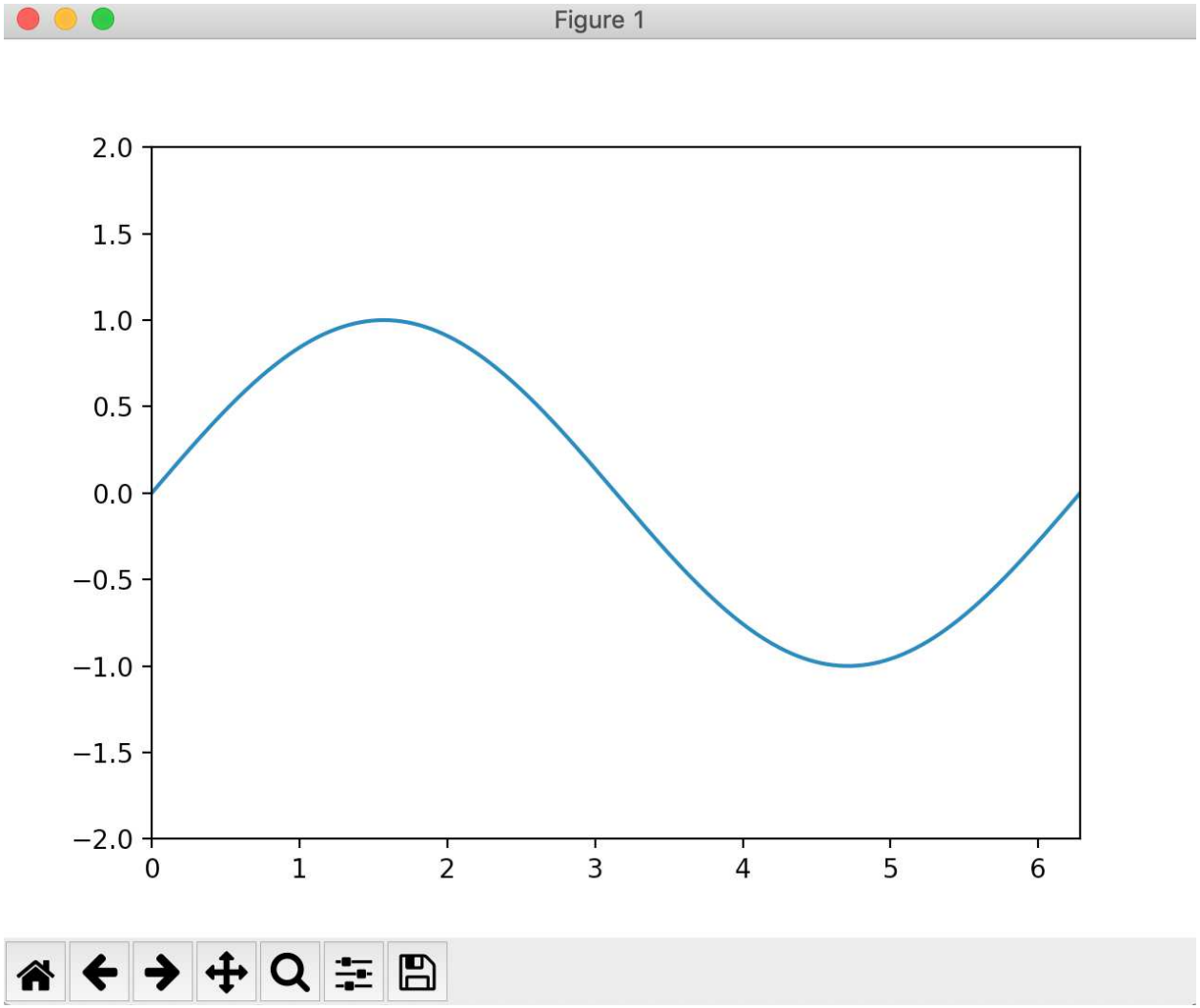
ipython --pylab

```
In [1]: x = arange(0.0, 2.0, 0.05)

In [2]: noise = 0.3*randn(len(x))

In [3]: y = 2 + 3*x + noise

In [4]: m, b = polyfit(x, y, 1)

In [5]: plot(x, y, 'bo', x, m*x+b, '-k', linewidth=2)
Out[5]:
[<matplotlib.lines.Line2D at 0x1236f8da0>,
 <matplotlib.lines.Line2D at 0x1236f8f28>]

In [6]: ylabel('regression')
Out[6]: Text(111.69444444444443, 0.5, 'regression')

In [7]: grid(True)
```

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

plotting sin(x)



the script `pylabsinplot.py`

We can use `pylab` in a noninteractive session,
typing `python pylabsinplot.py` at the command prompt.

The code for `pylabsinplot.py` is listed below.

```
from pylab import arange, sin, pi
from pylab import plot, axis, show

X = arange(0, 2*pi, 0.01)
Y = sin(X)
plot(X, Y)
axis([0, 2*pi, -2, 2])
show()
```

writing a plot to file

Instead of `show`, we can write the plot to file, with `savefig` of `matplotlib.pyplot`.

```
import matplotlib.pyplot as plt
from numpy import linspace, sin, pi

X = linspace(0.0, 2*pi, 100)
Y = sin(X)
plt.plot(X, Y)
plt.axis([0, 2*pi, -2, 2])
# plt.show()
plt.savefig('pyplotsinplot.png')
```

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

making movies

We plot sine functions of increasing frequencies, applying the `FuncAnimation` of the `animation` module of `matplotllib`.

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

def animate(freq):
    """
    Makes a plot of the sine function
    for the given frequence in freq.
    """
    x = np.linspace(0, 2*np.pi, 100)
    y = np.sin((freq+1)*x) # first call is with 0
    plt.clf() # clears the current plot
    axs = plt.axes(xlim=(0, 2*np.pi), ylim=(-1.5, 1.5))
    plt.plot(x, y)
```

defining and saving the animation

```
fig = plt.figure()

anim = animation.FuncAnimation(fig, animate, frames=5)
anim.save('animatedsine.gif', writer='imagemagick')

plt.show()
```

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

a moving billiard ball

Consider a billiard ball rolling over a pool table, bouncing against the edges of the table.

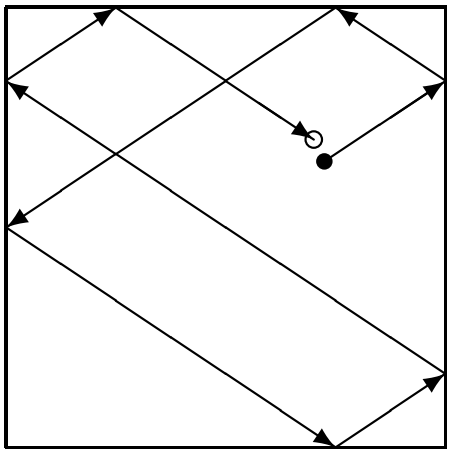
We will develop first a very basic GUI and then later add extra features.

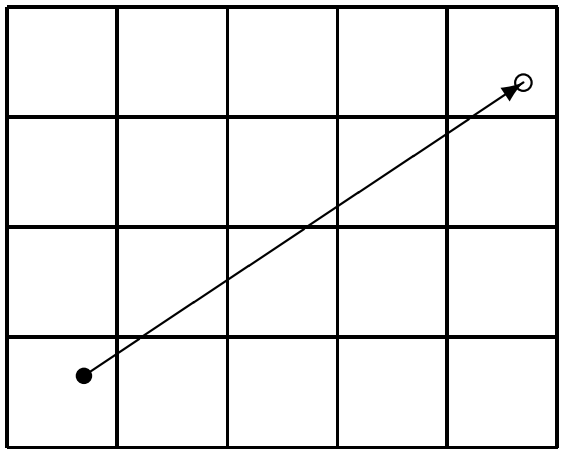
Basic elements of the GUI:

1. drawing on canvas
2. buttons call for action
3. animation of the rolling ball

Key idea: trajectory of ball is a straight line running over multiple copies of the pool table.

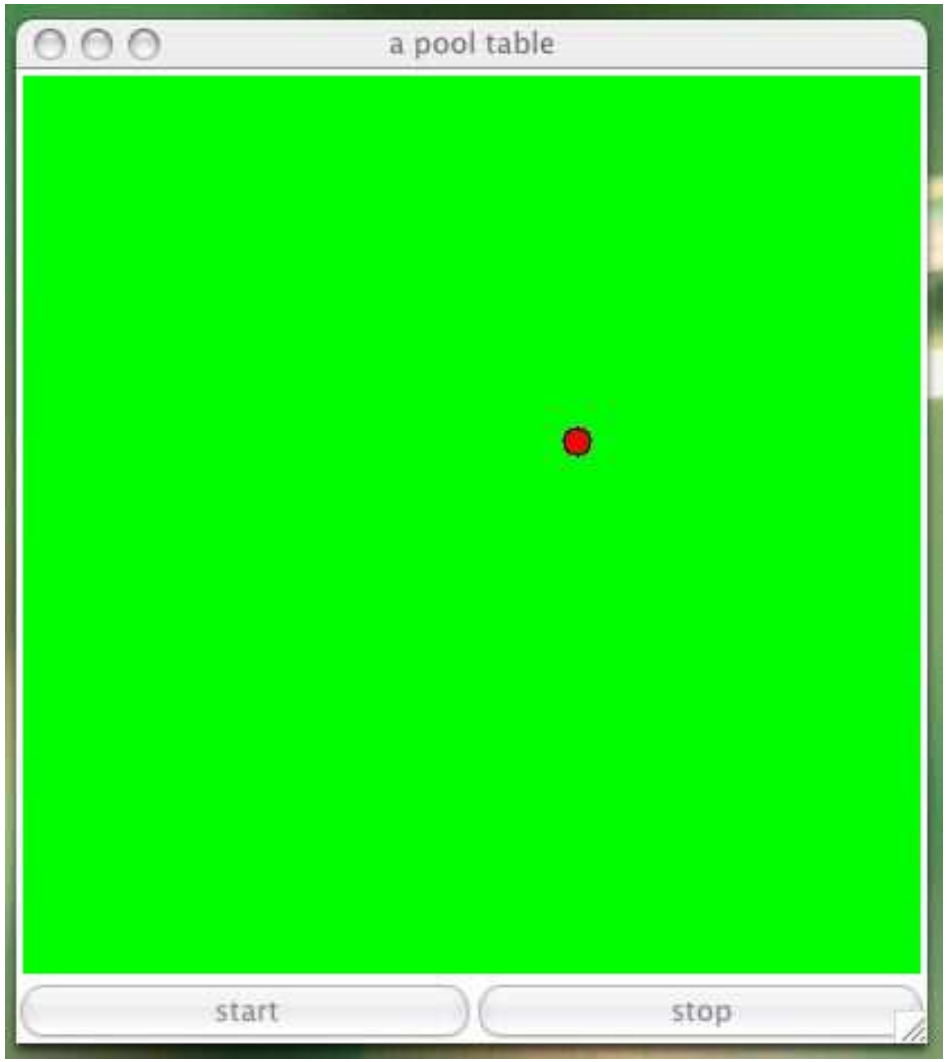
copying and folding tables





For any position on the straight vector on the right,
we map its coordinates into the pool table at the left.

layout of a basic GUI



specifications of the GUI

A GUI consists of several components (called *widgets*).

Our first basic layout uses

1. a canvas to draw a ball
2. a button to start the animation
3. a button to stop the animation

Actions triggered by buttons:

1. start:
   1. the initial position of the ball is random
   2. the ball start rolling in a random direction, bouncing off against the edges of the table

   After a stop, the balls rolls from the previous position, but in a different random direction.
2. stop: the animation stops

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

object oriented design

```
from Tkinter import Tk, Canvas, Button, W, E
from math import cos, sin, pi
from random import randint, uniform

class BilliardBall(object):
    """
    GUI to simulate billiard ball movement.
    """
    def __init__(self, wdw, dimension, increment, delay):
        """
        determines the layout of the GUI
        """
    def animate(self):
        "performs the animation"
    def start(self):
        "starts the animation"
    def stop(self):
        "stops the animation"
```

the main program

```
def main():
    """
    launches the GUI
    """
    top = Tk()
    dimension = 400 # dimension of pool table
    increment = 10  # increment for coordinates
    delay = 60      # how much sleep before update
    BilliardBall(top, dimension, increment, delay)
    top.mainloop()

if __name__ == "__main__":
    main()
```

The code for the class is considered as a module.

the constructor `__init__`

The object data attributes are

1. constants: dimension, increment, and delay are the parameters of the GUI
2. variables:
   1. `togo`: animation on or off
   2. `xpos`: x coordinate of the ball
   3. `ypos`: y coordinate of the ball
3. widgets:
   1. canvas spans two columns, in row 0
   2. start button on row 1, column 0
   3. stop button on row 1, column 1

code for __init__

```
def __init__(self, wdw, dimension, increment, delay):
    """
    determines the layout of the GUI
    """
    wdw.title('a pool table')
    self.dim = dimension # dimension of the canvas
    self.inc = increment
    self.dly = delay
    self.togo = False # state of animation
    # initial coordinates of the ball
    self.xpos = randint(10, self.dim-10)
    self.ypos = randint(10, self.dim-10)
```

canvas and buttons

```
self.cnv = Canvas(wdw, width=self.dim, \
    height=self.dim, bg ='green')
self.cnv.grid(row=0, column=0, columnspan=2)
self.startbut = Button(wdw, text='start', \
    command = self.start)
self.startbut.grid(row=1, column=0, sticky=W+E)
self.stopbut = Button(wdw, text='stop', \
    command = self.stop)
self.stopbut.grid(row=1, column=1, sticky=W+E)
```

The buttons trigger actions `start()` and `stop()` which are methods of the `BilliardBall` class.

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

methods `start()` and `stop()`

```
def start(self):
    "starts the animation"
    self.togo = True
    self.animate()

def stop(self):
    "stops the animation"
    self.togo = False
```

The `animate` method will

1. draw the ball on canvas
2. generate a random direction
3. move the ball in the direction using increment, as long as `togo == True`
4. update the canvas after a delay

the method `animate()`

```
def animate(self):
    """
    performs the animation
    """
    self.draw_ball()
    angle = uniform(0, 2*pi)
    xdir = cos(angle)
    ydir = sin(angle)
    while self.togo:
        xvl = self.xpos
        yvl = self.ypos
        self.xpos = xvl + xdir*self.inc
        self.ypos = yvl + ydir*self.inc
        self.cnv.after(self.dly)
        self.draw_ball()
        self.cnv.update()
```

the method `draw_ball()`

The ball is represented as a red circle,
with a radius of 6 pixels:

```
def draw_ball(self):
    """
    draws the ball on the pool table
    """
    xvl = self.map_to_table(self.xpos)
    yvl = self.map_to_table(self.ypos)
    self.cnv.delete('dot')
    self.cnv.create_oval(xvl-6, yvl-6, xvl+6, yvl+6, \
        width=1, outline='black', fill='red', \
        tags='dot')
```

As the ball moves, the values of the coordinates
may exceed the canvas dimensions.

With `map_to_table()` we get the bouncing effect.

the method `map_to_table()`

On canvas: (0,0) is at the topleft corner.

`self.dim` stores the number of pixel rows and columns on canvas.

```
def map_to_table(self, pos):
    """
    keeps the ball on the pool table
    """
    if pos < 0:
        (quot, remd) = divmod(-pos, self.dim)
    else:
        (quot, remd) = divmod(pos, self.dim)
    if quot % 2 == 1:
        remd = self.dim - remd
    return remd
```

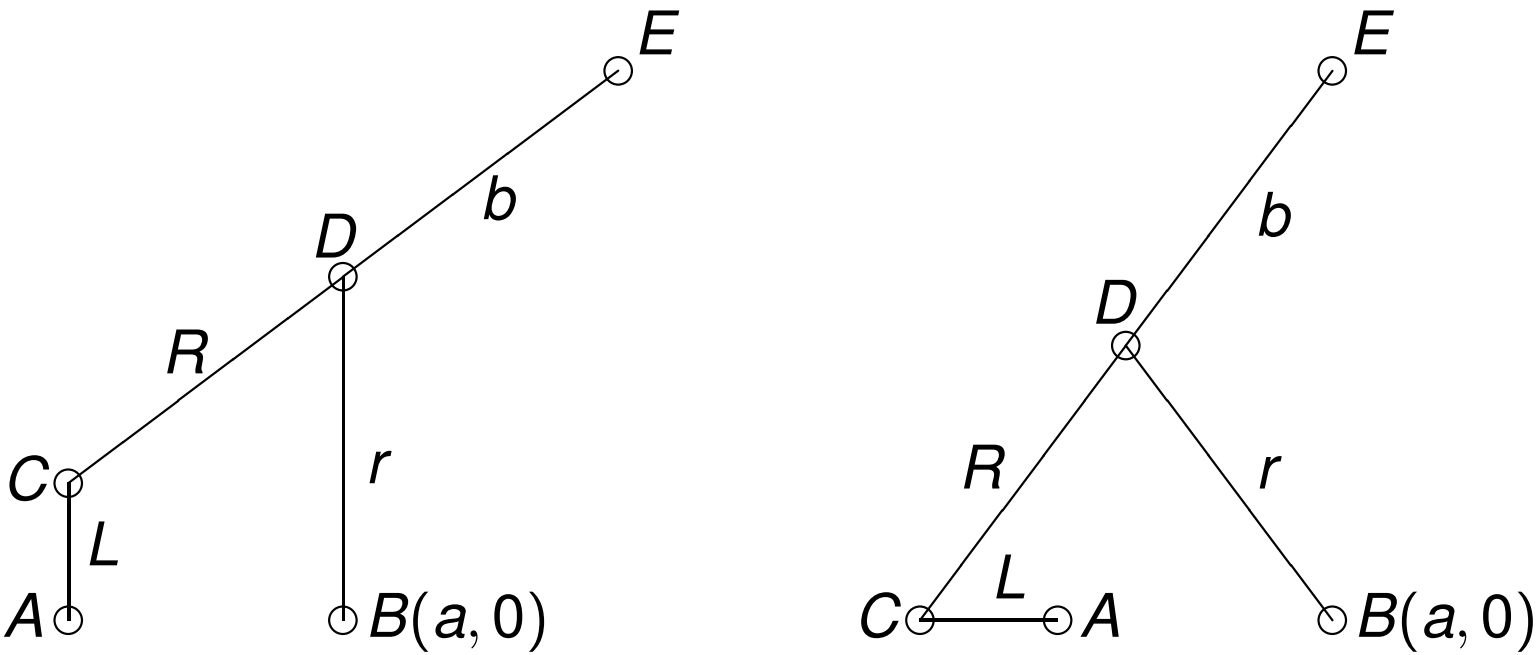
Imagine a succession of pool tables next to each other.
When the quotient `quot` is odd, we reflect,
otherwise we copy remainder `remd`.

curve plotting and animations

1. Curve Plotting with Matplotlib
   - linear regression
   - using pylab in a Python script
   - saving plots as frames in a movie
2. Animations with Tkinter
   - animating a bouncing ball
   - defining the layout of a basic GUI
   - methods stop, start, animate
3. Modeling a 4-Bar Mechanism
   - computing the trajectory of the coupler point

a 4-bar mechanism

We have 4 bars and 4 joints, labeled A, B, C, and D:



t is the angle between AB and crank AC, above $t = \frac{\pi}{2}$ and $t = \pi$. As the crank turns, the point E traces a curve.

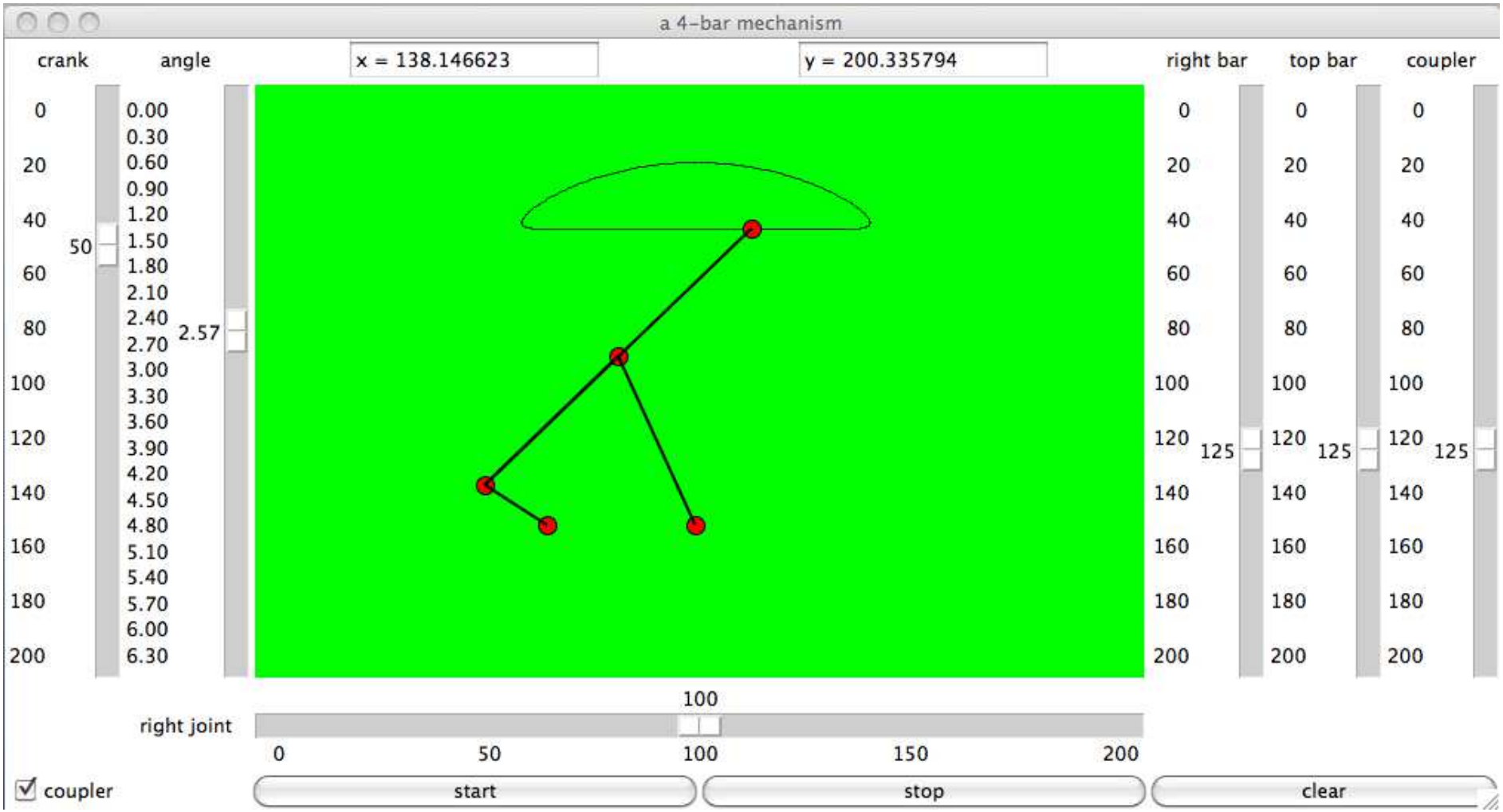
the Chebyshev mechanism

During the motion, the 4 bars are rigid. If the origin $(0, 0)$ is at A, then the mechanism is determined by 5 parameters:

1. L, the length of the crank, the bar between A and C;
2. a, the position of B;
3. R, the length of the bar between C and D;
4. r, the length of the bar between B and D;
5. b, the length of the bar between D and E.

Observe that the bar between D and E has always the same direction as the bar between C and D.
For $a = 100$, $L = 50$, $R = 125$, $r = 125$, and $b = 125$,
E traces a line and the mechanism translates circular motion into linear motion: the Chebyshev mechanism.

a GUI to model a 4-bar mechanism



the layout of the GUI

The central object is the canvas, to draw the mechanism.

The other widgets are

- 6 scales to enter the parameters of the mechanism:
  length of the crank, value of the angle, position of the right joint,
  length of the right, the top, and coupler bar.
- one check box to draw the coupler curve
- two entries to display the coordinates of the moving coupler point
- 7 labels to document widgets
- 2 buttons to start and stop the animation,
  one button to clear the canvas.

The animation moves the values of the angle,
which makes the crank turn and the coupler point move.

design of the code

The code consists of two parts:

1. `fourbargui.py` defines the GUI;
2. `fourbar.py` is an object-oriented model to compute the coordinates of the coupler point, given the angle, for some fixed configuration of the mechanism.

```
$ python fourbar.py
give number of samples = 4
1.57 (0.0,0.0)(100.0,0.0)(  0.0, 50.0)(100.0,125.0)(200.0,200.0)
3.14 (0.0,0.0)(100.0,0.0)(-50.0,  0.0)( 25.0,100.0)(100.0,200.0)
4.71 (0.0,0.0)(100.0,0.0)( -0.0,-50.0)( -0.0, 75.0)( -0.0,200.0)
6.28 (0.0,0.0)(100.0,0.0)( 50.0, -0.0)( 75.0,122.5)(100.0,244.9)
```

With the aid of computer algebra, we find symbolic expressions for the (x, y) coordinates of the coupler point.

the coordinates of the coupler point

For an angle t, the coordinates of the crank C are $(L\cos(t), L\sin(t))$.

Abbreviating $\cos(t)$ by c and $\sin(t)$ by s,
we have to solve the system

$$\begin{cases} (x-a)^2 + y^2 - r^2 = 0 \\ (x-Lc)^2 + (y-Ls)^2 - R^2 = 0 \\ c^2 + s^2 - 1 = 0 \end{cases}$$

The first equation expresses that the point D with coordinates (x, y) is at distance r from the point B with coordinates $(a, 0)$.

The second equation expresses that the point D with coordinates (x, y) is at distance R from the crank C.

The third equation originates from the substitution $c = \cos(t)$ and $s = \sin(t)$, needed to have an equivalent algebraic system.

a symbolic solution

Computing a lexicographic Gröbner basis yields a linear equation in y that depends only on the parameters: $y = \dfrac{z_2 \pm \sqrt{d}}{2z_1}$, where

$$z_1 = L^2 + a^2 - 2aLc, \quad \text{and} \quad z_2 = -LR^2s + L^3s + Lsr^2 + a^2Ls - 2L^2sca.$$

And x depends on y:

$$x = \frac{-a^2 + r^2 + L^2 - R^2 - 2yLs}{2(-a + Lc)}.$$

Only if the discriminant $d \geq 0$ will we find real values for y.

Because the coupler curve is closed, we take only one of the two solutions for y.

the discriminant

The discriminant d equals

$$
\begin{aligned}
&-2L^2R^2s^2a^2 - 2L^2R^2s^2r^2 + 2a^4R^2 - a^2r^4 + L^6s^2 - L^2R^4 + 2L^4R^2 \\
&+2L^4r^2 - L^2r^4 - 7L^4a^2 - 7L^2a^4 - L^6 - a^6 + 6L^4s^2a^2 + 4L^3R^2s^2ca \\
&-4L^5s^2ca - 2L^2s^2r^2a^2 - 2L^4R^2s^2 + L^2s^2r^4 + 5a^4L^2s^2 + 2L^2r^2R^2 \\
&+6L^5ac + 4L^2a^2R^2 + 4L^2r^2a^2 + 20L^3ca^3 + 2a^2r^2R^2 + 4L^3s^2r^2ca \\
&-12a^3L^3s^2c + 4L^4s^2c^2a^2 - 8L^3aR^2c - 8L^3ar^2c - 8a^3R^2Lc - 8a^3r^2Lc \\
&-a^2R^4 + 2a^4r^2 + 6a^5Lc - 8a^2L^4c^2 - 8a^4L^2c^2 + 2aLcR^4 + 2aLcr^4 \\
&+8a^2L^2c^2R^2 + 8a^2L^2c^2r^2 - 2L^4s^2r^2 + L^2R^4s^2 - 4aLcr^2R^2.
\end{aligned}
$$

Once coordinates for C and D are known, given a value for b computing the coordinates for E is straightforward.

Recall that the bar between D and E has the same direction as the bar between C and D.

Summary + Exercises

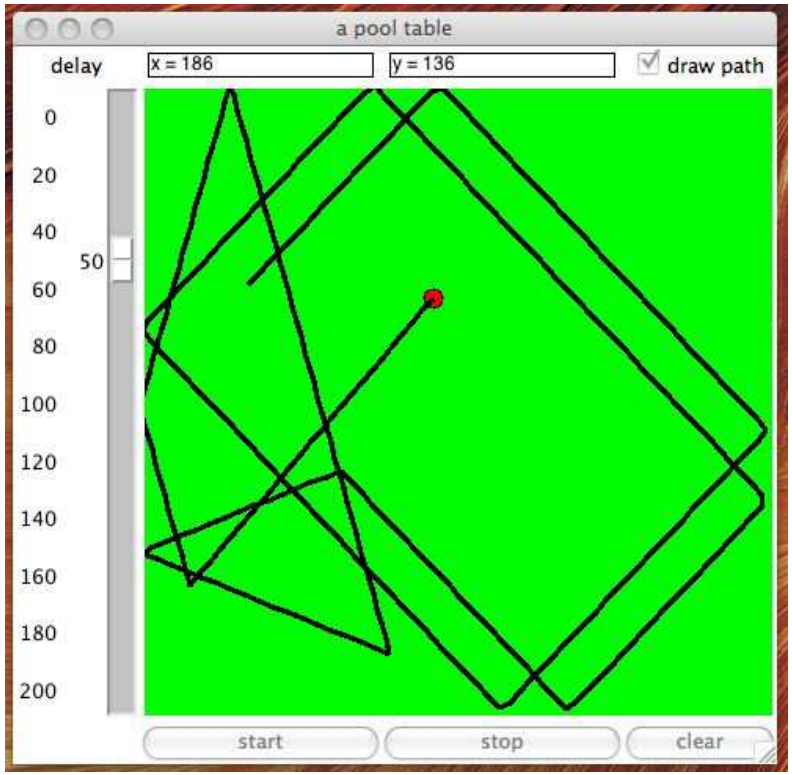
See `http://www.matplotlib.org` for Matplotlib 3.1.1.
A manual of Tkinter is at `https://wiki.python.org/moin/TkInter`.

Exercises:

1. The animation of the sine with increasing frequency produced plots which became less smooth as the frequency increased. Modify the making of the movie with a proper adjustment of the step size used for sampling. Justify your choice of the step size.
2. Describe the design of a fully automatic, adaptive step size selection for plotting. Write a small script to demonstrate your design and illustrate your step selection on sine functions of increasing frequencies.

GUI exercises

3. Extend our basic GUI for billiards into



and more exercises

The remaining exercises concern the billiard ball GUI:

4. Adjust the GUI to work for rectangular pool tables.
5. Modify the GUI to use the Entry widgets to allow the user to enter the initial position of the ball.
6. Add an extra check button to give the user the option to either enter the initial position of the ball, or to let the computer generate random coordinates.