Example of using utilities for 4-Bar linkage solution

This short example walks us through the program **CircCirc4Bar.py**, showing how it uses the functions in **LinkageUtilities.py** to solve the trajectory of a coupler point. The solution of more complex mechanisms (e.g. GearedFiveBar, Klann, Jansen) follows the same basic approach.

Setup

We start by loading `numpy` and `matplotlib` for Matlab-like calculations and plotting. We also check that we are in the right local directory (change this to suit your setup) so we can load in the initial data.

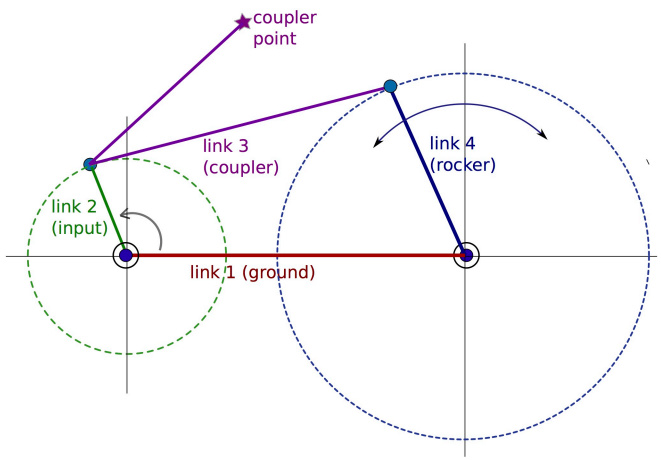
In [4]:
```
import numpy as np
from matplotlib.pyplot import *  #Lazy...
import os

#Make sure we are in the right directory...
# As necessary:
os.chdir('/Volumes/ExTraSpace/Courses/linkages/CircCirc4BarExample')

#Import handy utility functions:
from LinkageUtilities import arcpoints, circcirc, coupler, grashof

# Allow plots and impages to appear in iPython notebook
%matplotlib inline
from IPython.core.display import Image # Import function to display pr
emade figures
Image(filename="FourBarNotation.jpg")
```


Out[4]:



In this image, link1 is along the X axis, but it could be in any orientation.

Create linkage using some initial joint locations

For comparison with the free Linkage animation program (http://blog.rectorsquid.com/linkage-mechanism-designer-and-simulator/), we specify the mechanism in terms of joint locations for a particular configuration. (Note that the *Linkage* program is useful for visualizing the path, but will not give us detailed trajectories of data points which we'll need for velocity, power and force calculations.)

We load the initial configuration assuming the input (crank) joint is first, and then work around the loop in series. The last line gives the coupler location, which is assumed on link3. The input data file looks like this:

```
51.    -25.5
51.    -14.7
76.6    -14.7
76.6    -36.6
70.0    0.0
#First four rows are XY positions of joints 1-4, last row is coupler point
```

In [5]:
```
pointsdata = np.loadtxt('InitialJoints4Bar.txt')
initjoints = pointsdata[0:4,:] #Joints 1-4 in order
initcoupler = pointsdata[4,:]
```

Specify range of angles to rotate input and number steps to take

Note that range can be negative and can be more than one revolution. Angles are in radians, anticlockwise measured from the X axis. Number of steps is whatever you want depending on resolution.

In [6]:
```
thetastart = 0.
thetaend = 1.95*np.pi
#Specify number steps to take
numsteps = 26
```

Find lengths of the links

Although it was convenient to input the linkage in terms of joint locations, we also need the link lengths. Notation: `d12` is an [x,y] vector from first joint to second joint. Then `L2` is the length of link2, and so forth.

numpy.linalg.norm() computes the norm of a matrix; in this case it's just the length of the vector :-)

In [7]:
```
d12 = initjoints[1,:]-initjoints[0,:] #(x,y) distance between joints
L2 = np.linalg.norm(d12)
d23 = initjoints[2,:]-initjoints[1,:]
L3 = np.linalg.norm(d23)
d34 = initjoints[3,:]-initjoints[2,:]
L4 = np.linalg.norm(d34)
dc3 = initcoupler - initjoints[1,:]
Lc = np.linalg.norm(dc3)
d14 = initjoints[0,:]-initjoints[3,:]
L1 = np.linalg.norm(d14)
```

Calculate the angle to the coupler point

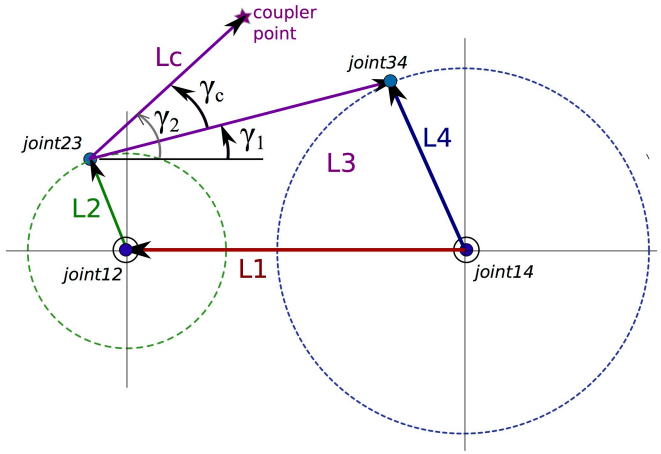
Calculate the angle gamma_c between the coupler link and the coupler point. This angle could be defined from either end of the link; here we do it from the joint between links 2 and 3.

The image below shows the notation of joints, links and gamma_c we have just calculated.

In [8]:
```
gamma1 = np.arctan2(d23[1],d23[0])
gamma2 = np.arctan2(dc3[1],dc3[0])
gammac = -gamma1+gamma2

Image(filename="FourBarVectors.jpg")  #Display image showing the notat
ion
```

Out[8]:



Solve for the current assembly

Although we have loaded in joint locations, the program needs to know the angles so it can compute the right "assembly." (Recall there are two solutions.) We can use CircCirc() to solve the system and see which assembly matches the one we actually have:

In [9]:
```
# Use numpy.allclose() to check if negligible difference.
joint34 = circcirc(initjoints[1,:],L3,initjoints[3,:],L4)
if(np.allclose(joint34[0,:],initjoints[2,:])):
    assembly = 0
elif(np.allclose(joint34[1,:],initjoints[2,:])):
    assembly = 1
else:
    print('Hmmm, neither solution matches the input point...')
```

Compute the crank position for each input angle

The result is `joints23' which is an array of [x,y] positions corresponding to locations of the joint between Link2 and Link3.

In [10]:
```
joints23 = arcpoints(initjoints[0,:],L2,thetastart,thetaend,numsteps)
```

Check Grashof condition

If we want the crank to undergo a full revolution (crank-rocker), we should check if the Grashof condition is satisfied. (Depending on the application, we might not care about this.)

In [11]:
```
if (grashof(L1,L2,L3,L4) == False):
    print('\n Linkage is non-grashof!')
```

Compute remaining joint locations

Finally, we compute the locations of the remaining points for each location of `joint23`. The solutions are stored as `n*2` arrays of (x,y,) coordinates.

In [12]:
```
joints34 = np.zeros((numsteps,2),float)
couplerpts = np.zeros((numsteps,2),float)
for i in range(0,numsteps):
    intersects = circcirc(joints23[i,:],L3,initjoints[3,:],L4)
    joints34[i,:] = intersects[assembly,:]
    couplerpts[i,:] = coupler(joints34[i,:],joints23[i,:],Lc,np.pi+gam
mac)
```

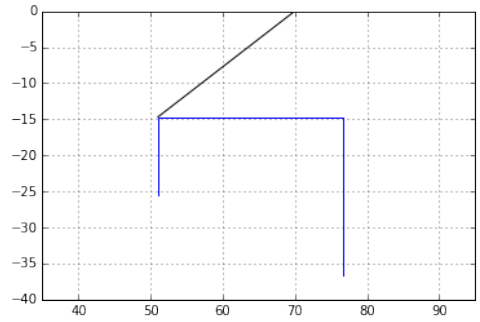
Plot the results

First, plot the initial configuration to make sure it looks as we expected:

In [13]:
```
clf()
grid(True)
axes().set_aspect('equal', 'datalim')  #square and limited by data

# Check: plot initial configuration in blue, with black line to couple
r
plot(initjoints[:,0],initjoints[:,1],'b')
plot((initjoints[1,0],initcoupler[0]),(initjoints[1,1],initcoupler[1])
,'k')
```

Out[13]: `[<matplotlib.lines.Line2D at 0x105888710>]`

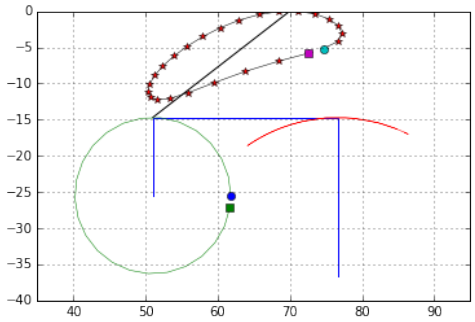


Now plot joint locations for the range of angles specified. For the crank and coupler locations, the plot shows a fat dot at the start angle and a square at the end. The crank trajectory is a green circle. The rocker trajectory is a red arc. The coupler locations for each step are shown as stars, which gives some idea of where the curve is faster and slower.

Also save the plot to a PDF file: CircCirc4Bar.pdf

In [14]:
```
clf()
grid(True)
axes().set_aspect('equal', 'datalim')  #square and limited by data

# Check: plot initial configuration in blue, with black line to couple
r
plot(initjoints[:,0],initjoints[:,1],'b')
plot((initjoints[1,0],initcoupler[0]),(initjoints[1,1],initcoupler[1])
,'k')
plot(joints23[:,0],joints23[:,1],color = 'g',linewidth=0.5)
plot(joints23[0,0],joints23[0,1],'o')
plot(joints23[numsteps-1,0],joints23[numsteps-1,1],'s')
plot(joints34[:,0],joints34[:,1],color = 'r',linewidth=0.5)
plot(couplerpts[:,0],couplerpts[:,1],color = 'k',linewidth=0.5)
plot(couplerpts[:,0],couplerpts[:,1],'*')
plot(couplerpts[0,0],couplerpts[0,1],'o')
plot(couplerpts[numsteps-1,0],couplerpts[numsteps-1,1],'s')
#show()
#Save a PDF of the plot
fig1 = gcf()
fig1.savefig('CircCirc4Bar.pdf')
```



Save trajectory data

Save the various trajectories in a tab-delimited text file for further analysis (e.g. in Excel)

In [15]:
```
Table = np.column_stack((joints23,joints34,couplerpts))
f_handle = file('CircCirc4BarTrajectory.txt', 'w')
headerstring = "joint23(x,y), joint34(x,y),   coupler(x,y)"
np.savetxt(f_handle,Table,header=headerstring,delimiter='\t',newline='
\n',fmt='%4.2f')
f_handle.close()
```

In [ ]:

In [ ]: