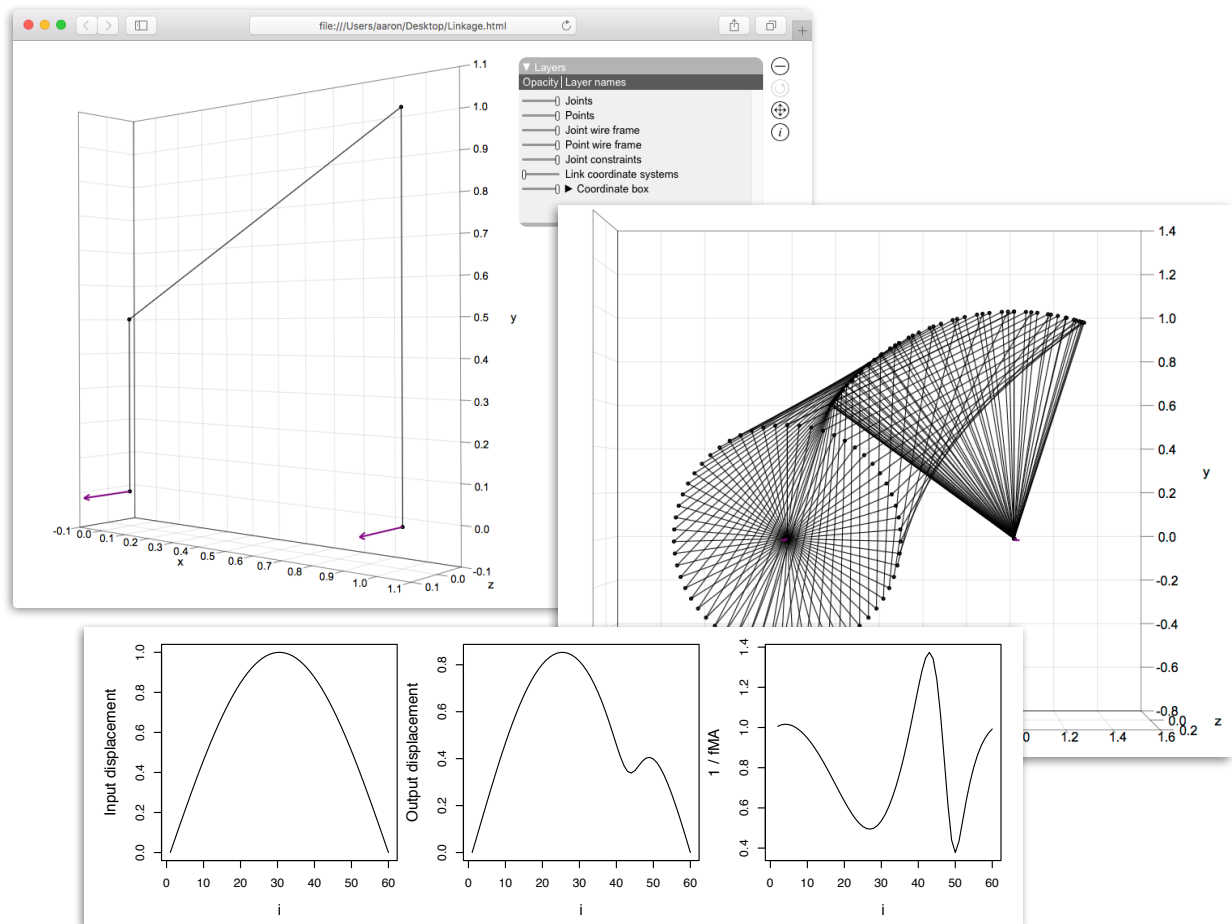


# Modeling linkages in R using linkR

*Simulating two- and three-dimensional linkage mechanisms  
using the R package 'linkR'*



October 2016  
Version 1.1

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Getting started</b>	<b>5</b>
<b>3</b>	<b>Defining a linkage</b>	<b>7</b>
3.1	Joint coordinates . . . . .	7
3.2	Joint types . . . . .	7
3.3	Joint constraint vectors . . . . .	8
3.4	Joint connections . . . . .	9
3.5	Calling ‘defineLinkage’ . . . . .	10
3.6	Adding associated points . . . . .	11
3.7	Connecting associated points into shapes . . . . .	12
<b>4</b>	<b>Simulating linkage motion</b>	<b>14</b>
4.1	Calling ‘animateLinkage’ . . . . .	14
<b>5</b>	<b>Analyzing linkage motion</b>	<b>18</b>
5.1	Joint position over time . . . . .	18
5.2	Point position over time . . . . .	20
5.3	Calling ‘linkageKinematics’ . . . . .	21
5.4	Angular displacement and velocity of links . . . . .	22
5.5	Displacement and velocity of joints . . . . .	23
5.6	Displacement and velocity of points . . . . .	24
5.7	Torque mechanical advantage . . . . .	25
5.8	Force mechanical advantage . . . . .	27
<b>6</b>	<b>Examples</b>	<b>28</b>
6.1	Planar 4-bar (RSSR) . . . . .	28
6.2	3D 4-bar (RSSR) . . . . .	29
6.3	Two coupled sliders (LSSL) . . . . .	30
6.4	Coupled planar and linear sliders (SSPSSL) . . . . .	31
6.5	Three coupled sliders (LSSLSSL) . . . . .	33
6.6	Crank-driving sliding link (RSSL) . . . . .	34
6.7	Slider along rotating link (RLSS) . . . . .	35
6.8	Rotating links in series (RRSS) . . . . .	36
6.9	Coupled linear, planar sliders (LSSLSSPSS) . . . . .	38
6.10	Sliders coupled by rotating link (LSSRSSL) . . . . .	39
6.11	Coupled rotating, linear links (RSSLSSR) . . . . .	40
6.12	Coupled rotating, planar links (RSSRSSPSS) . . . . .	41
6.13	Two 3D 4-bars in series (RSSRSSL) . . . . .	42
6.14	Slider, 3D 4-bar in series (RSSR(SSL)) . . . . .	43
6.15	Slider, 2D 4-bar in series (R(SSL)SSR) . . . . .	44
6.16	Slider in parallel with 4-bar (RS(SSL)SR) . . . . .	45
6.17	Fish cranial joint configuration . . . . .	46

6.18 Bird cranial linkage . . . . .	48
6.19 Fish cranial linkage . . . . .	49
<b>7 Citing linkR</b>	<b>52</b>
<b>8 Acknowledgements</b>	<b>52</b>

# 1 Introduction

During my time as a graduate student I became fascinated by how differences in the internal structure among animals relates to the different ways in which they move. I spent much of that time at the Field Museum of Natural History in Chicago, exploring the skeleton collections and observing the diverse ways in which animal bodies have evolved. There is a long and rich history of researchers using simple mechanical models to simulate the motion of bones and joints in animals. Building off this work, I wanted to develop my own motion simulations using data collected from natural history specimens.

In particular, I wanted to model the motion of interconnected bones as mechanical linkages (e.g. a 4-bar linkage), a type of analysis referred to in engineering as kinematic simulation. There are a number of advanced software tools currently available for multibody simulation and force analysis (e.g. Adams Multibody Dynamics, Abaqus, SimMechanics, SIMPACK MBS software, AnyBody Modeling System). However, these programs were more expensive than I could afford and had capacities far exceeding those that I needed for my linkage simulations. Additionally, I wanted the code that I used to be accessible to anyone who wanted to replicate my results or create simulations of their own. From this the linkR package was born.

Although linkR could have been developed using any basic programming language, I find R packages to be one of the easiest and standardized means of distributing software. Developing linkR on the R platform has the added advantage of integrating linkR with two other R packages I have developed over the past several years: [StereoMorph](#) and [svgViewR](#). StereoMorph provides one approach for collecting shape data that can be used to create linkage simulations while [svgViewR](#) provides an easy-to-use platform for visualizing the resulting 3D, animated simulations.

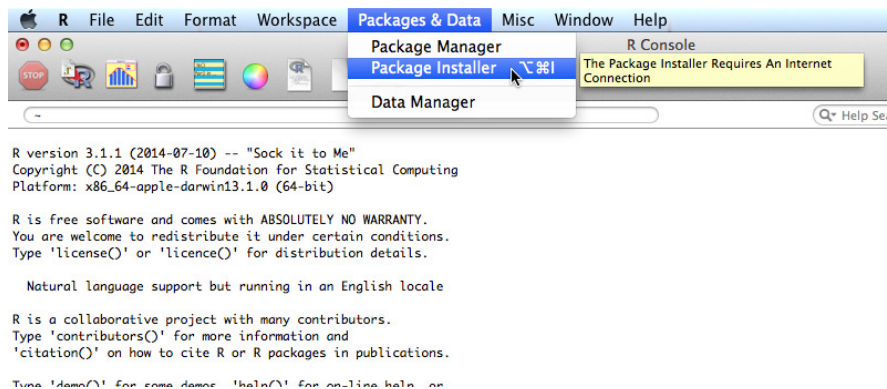
This tutorial will take you through all the steps of using the linkR R package to simulate, analyze, and visualize the motion of basic linkage mechanisms. All of the required code is included within this tutorial so no separate files are required. The last section, [Examples](#), provides commented code for creating many linkages of differing configurations. I hope you find this tutorial helpful and wish you the best with your project!

Aaron Olsen  
October 2016

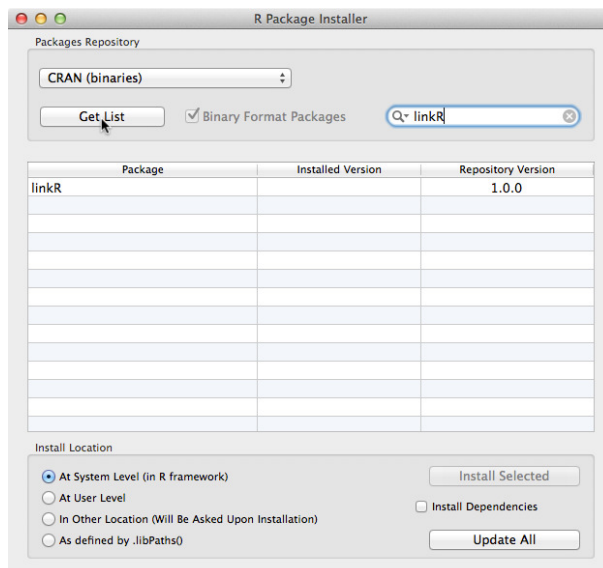
## 2 Getting started

This tutorial will show you how to create 2D and 3D linkage models using the R package [linkR](#). The R project is a computing language and platform that allows users to freely upload and share software packages. The linkR package includes functions for simulating and analyzing the kinematics of 2D and 3D linkage mechanisms. The linkR package uses the R package [svgViewR](#) to create 3D interactive animations that can be visualized in the web browser. The following steps will take you through the process of installing the linkR package.

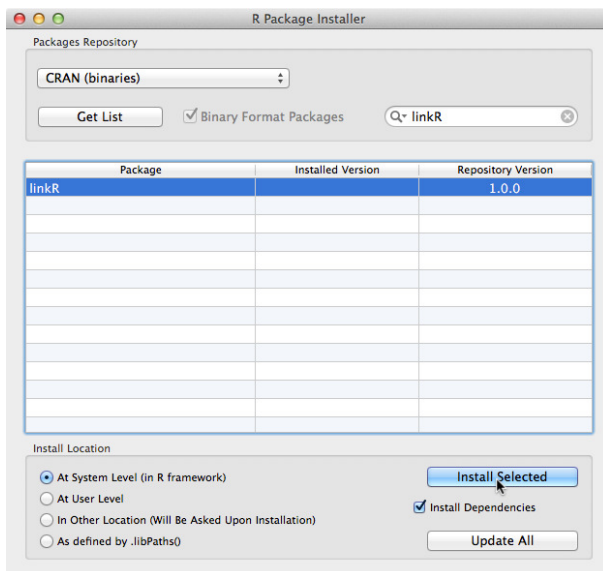
1. If you do not already have R installed on your computer, begin by [installing R](#). R can be installed on Windows, Linux and Mac OS X.
2. Once installed, open R.
3. Go to Packages & Data > Package Installer.



4. Find the linkR package binary by typing 'linkR' into the Package Search box and clicking Get List. (The repository version of linkR will differ from that in the version in the image below).



5. Check the box next to Install Dependencies. This ensures that ‘svgViewR’ is installed with linkR. Then click ‘Install Selected’ to install.



6. Throughout this tutorial I’ve included R code that you can use to reproduce examples and try out linkR for yourself. All R code is in the courier font style on a gray background as in the example below, with comments indicated in orange (and preceded by a ‘#’) and function names in blue.

```
# Print 'Hello world!'  
print('Hello world!')
```

To run each line of code, simply copy and paste the code into the R console.

6. Before calling any linkR functions be sure to load the linkR package into the current R session using ‘library’.

```
# Load linkR  
library(linkR)
```

7. Lastly, you’ll need a compatible web browser to visualize the interactive linkage animations. This includes Safari, Chrome, and Firefox (basically any browser except Internet Explorer).

You are now ready to create linkage models!

### 3 Defining a linkage

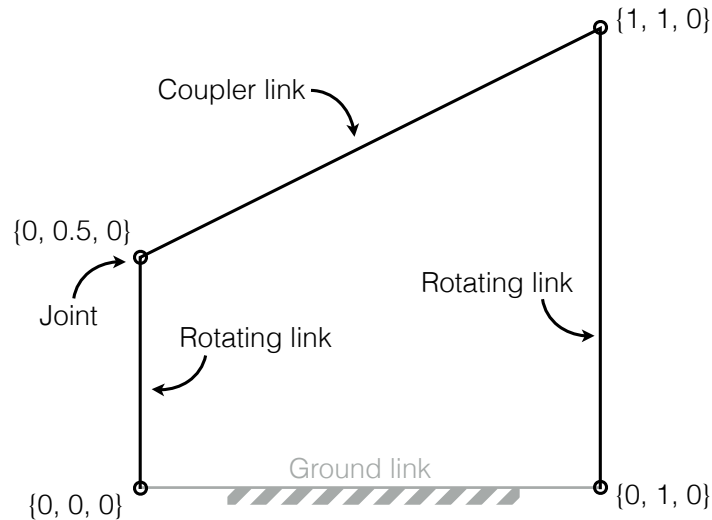
Linkages are represented in linkR as a chain or network of links connected by joints, each of which permits a particular range of motion. These joints and their interconnections are defined by four main input parameters:

1. The joint coordinates
2. The type of motion permitted at each joint
3. A vector describing the permitted motion, if applicable
4. The two links connected by each joint

#### 3.1 Joint coordinates

The joint coordinates are specified as a two- or three-column matrix (for two or three dimensions, respectively) indicating the initial positions of the joints. For example, the joint coordinates of a simple, planar four-bar linkage (three mobile links and one ground link) with the joints arranged such that one rotating link is half the length of the other and both links are vertical :

```
# Define the joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.5,0), c(1,1,0), c(1,0,0))
```



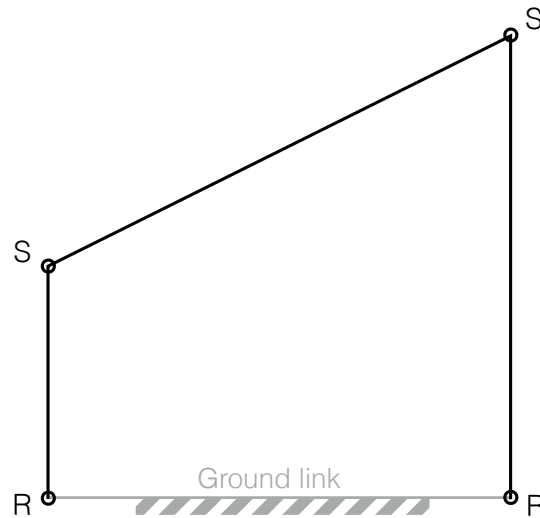
A simple four-bar linkage consisting of four joints and three mobile links. The lower two joints are connected to ground.

#### 3.2 Joint types

The joint type refers to the type of motion (translation and/or rotation) permitted at each joint. linkR currently supports four constraint types:

- **Linear (L)**. Permits translation along a single line or vector
- **Planar (P)**. Permits translation within a plane
- **Rotational (R)**. Permits rotation about a single axis
- **Spherical (S)**. Permits rotation about all three axes (including long-axis rotation)

The joint types are specified as a vector of the above abbreviations ('L', 'P', 'R' or 'S'), having the same number of elements as the number of rows in the joint coordinate matrix and in the same order. For example, a simple four-bar linkage can be defined as having two R-joints connected to ground and two S-joints in between that permit rotation in any direction. These joint constraints give a four-bar mechanism, whether 2D or 3D, a single degree of freedom.



Joint types for a simple four-bar linkage: rotational (R) and spherical (S).

The corresponding vector to define these joint types would then be:

```
# Define the joint types
joint.types <- c('R', 'S', 'S', 'R')
```

### 3.3 Joint constraint vectors

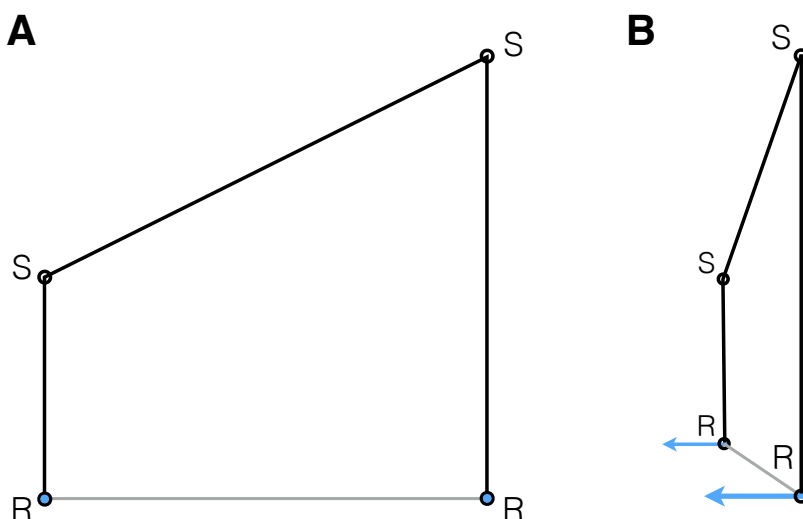
The joint constraint vector refers to a vector that describes the orientation or direction of motion permitted by each joint. For a linear joint, this vector is the line along which the joint moves. For a planar joint, this vector is the vector perpendicular (orthogonal) to the plane in which the joint moves. For a rotational joint, this vector is the axis of rotation. Since a spherical joint permits rotation in all directions, no vector is needed for this joint type. It can be set as NA.

The joint types are specified as a list having the same number of elements as the number of rows in the joint coordinate matrix and in the same order. For example, for a simple



two-dimensional (planar) four-bar linkage, the axes of rotation at the two R-joints are oriented perpendicular to the plane of the linkage. For the previously given coordinates these vectors are oriented parallel to the z-axis.

```
# Define the joint constraint vectors
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1))
```



The joint constraint vectors (blue) for two R-joints in a planar four-bar linkage seen in front (A) and oblique side view (B). The vectors are oriented orthogonal to the linkage plane.

### 3.4 Joint connections

Linkages are defined in linkR as links interconnected by joints, with each joint connecting two links. The two links connected by each joint are specified in a two-column matrix in which each column corresponds to the two links and each row corresponds to each joint. Thus, the matrix should have the same number of rows, and be in the same order, as the joint coordinate matrix.

Each linkage will have a single ground link, which is link 0. The numbering of the remaining links is arbitrary but will be used to associate linkage input motions with the appropriate link. For the simple four-bar linkage, the joint connects form a simple chain: the first joint connects the ground link to link 1, the second joint connects link 1 to link 2, etc. The matrix defining the joint connections would be as follows:

```
# Define the connections among joints
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0))
```

While each joint is allowed to connect only two links, links may be connected by any number of joints. For linkages that have parallel, interconnected sets of links this matrix the 'joint.conn' matrix specifies these branched interconnections among the links.

### 3.5 Calling ‘defineLinkage’

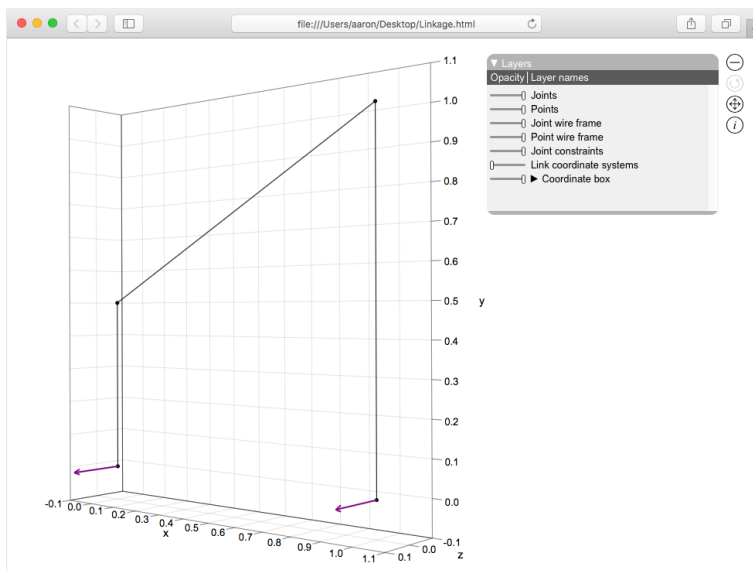
These four inputs are passed to the function ‘defineLinkage’ in order to define the linkage for kinematic simulation:

```
# Create a 'linkage' object
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=
  joint.cons, joint.conn=joint.conn)
```



The returned object (here, ‘linkage’) can be passed to other linkR functions for visualization and kinematic simulation. The linkR package uses a new R package, [svgViewR](#), to create interactive, 3D animated visualizations of linkages that can be viewed in any major web browser. To visualize the linkage in its initial state, use ‘drawLinkage’, specifying the name of the file to be written to the current working directory.

```
# Create a linkage visualization
drawLinkage(linkage, file='Linkage')
```

This will create an ‘html’ animation file using the specified file name.

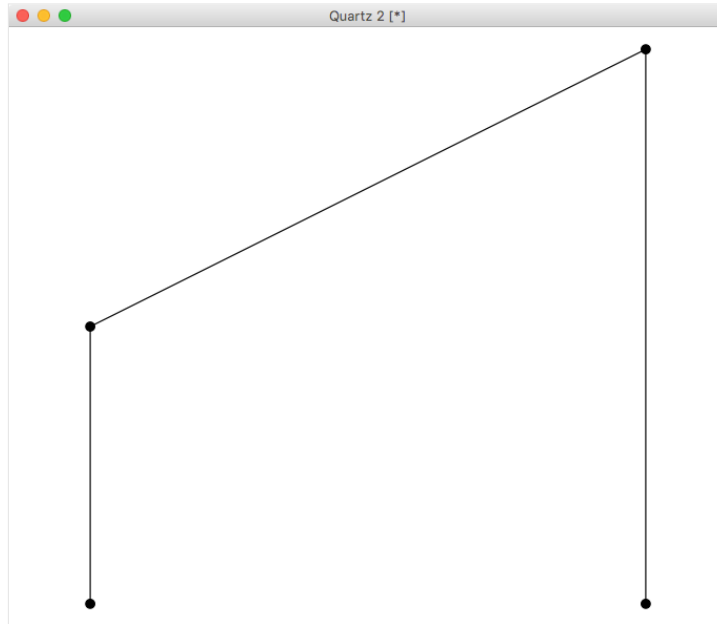


Interactive linkage visualization in a web browser created by drawLinkage.

Once you open this file in a web browser, you can use the mouse to interact with the visualization. In the top right corner of the browser window, click the  icon (or hold down ‘r’) and then click and drag with the mouse to rotate the visualization. Click the  icon to turn on translation and click and drag to move the visualization around the window. Use scroll to zoom in and out. The viewing file is entirely self-contained so it can be moved and viewed on any platform.

You can also visualize the linkage using the native R plot tools by calling drawLinkage without an input for ‘file’. However, this doesn’t provide the animation or interactive features of the [svgViewR](#) visualization.

```
# Plot the linkage using native R plot device (static only)
drawLinkage(linkage)
```



The four-bar linkage visualized in the R plot device.

### 3.6 Adding associated points

Sometimes you'll want to simulate and track the motion of points on or associated with a link rather than the joints or links themselves. First define the coordinates of the points. The points defined below correspond to the midpoint of each link.

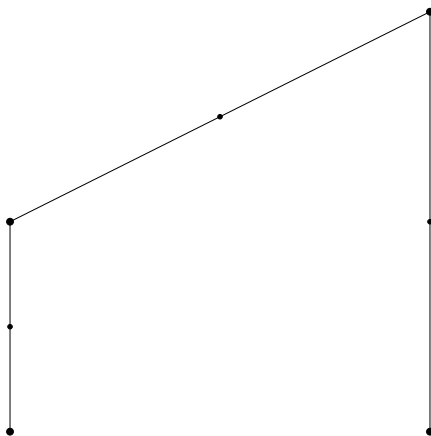
```
# Define the point coordinates
link.points <- rbind(c(0,0.25,0), c(0.5,0.75,0), c(1,0.5,0))
```

Then specify the links with which each point is associated (in the same order as 'link.points').

```
# Set the link associations for each point
link.assoc <- c(1,2,3)
```

Define the linkage, this time adding the input parameters 'link.points' and 'link.assoc'.

```
# Define a linkage with associated points
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=
  joint.cons, joint.conn=joint.conn, link.points=link.points, link.assoc=link.assoc)
```



4-bar with points associated with each link.

When you simulate motion of the linkage, the points will move with their corresponding link allowing you to track motion at point in the linkage.

### 3.7 Connecting associated points into shapes

If you add points not located on the straight-line distance between joints it is easier to visualize how these points are moving relative to one another (and relative to the links) by connecting them with a path. For example, we could use four points to define rectangles around the two rotating links.

```
# Define first rectangle
rec1 <- rbind(c(-0.1,-0.1,0), c(-0.1,0.6,0), c(0.1,0.6,0), c(0.1,-0.1,0))

# Define second rectangle
rec2<- rbind(c(0.9,-0.1,0), c(0.9,1.1,0), c(1.1,1.1,0), c(1.1,-0.1,0))

# Define the point coordinates
link.points <- rbind(rec1, c(0.5,0.75,0), rec2)

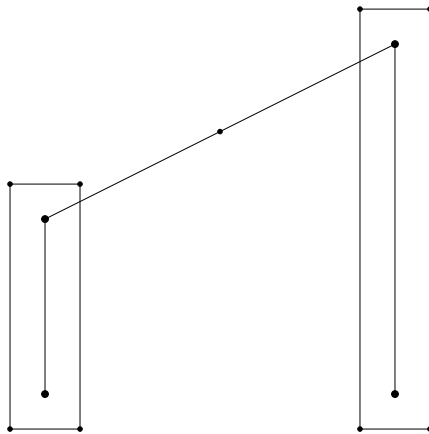
# Set the link associations for each point
link.assoc <- c(1,1,1,1, 2, 3,3,3,3)
```

Specify the points you wish to connect in sequence to draw each path. Specify each path as a separate element of a list.

```
# Define points to connect into paths
path.connect <- list(c(1:4,1), c(6:9,6))
```

Then define the linkage, adding the 'path.connect' parameter.

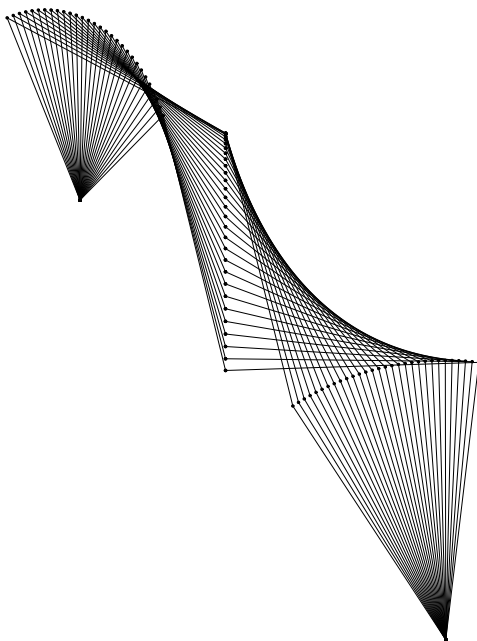
```
# Define the linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types, joint.cons=
  joint.cons, joint.conn=joint.conn, link.points=link.points, link.assoc=link.assoc,
  path.connect=path.connect)
```



4-bar with points connected into rectangles around rotating links.

## 4 Simulating linkage motion

The linkR function ‘animateLinkage’ allows users to simulate linkage motion (also known as ‘kinematic simulation’) given a specified input motion at a joint or joints. The function uses analytical geometry to iteratively solve for the joint positions that satisfy the specified [joint constraints](#) and link lengths. The input motions must equal the linkage’s degrees of freedom (the total number of states or conformation that a linkage can have at a particular time point). Making the number of input motions equal to the degrees of freedom (DOF) ensures a single solution when solving for the linkage motion.



Simulated motion of a 3D 5-bar linkage with an intermediate planar sliding joint (DOF = 2).

### 4.1 Calling ‘animateLinkage’

Once you’ve [defined a linkage](#), use ‘animateLinkage’ to specify the input motion and the joint at which motion is to be input. The input parameter ‘input.param’ is a vector of rotations (for R-joints) or translations (for L- and P-joints) to be applied at the joint(s) specified by the input parameter ‘input.joint’.

A planar (2D) 4-bar linkage has a single DOF (ignoring the rotation of any of the links about their long axis). We simply input a rotation at one of the R-joints and the positions of the other two links can be determined. Begin by using ‘linkR\_examples’ to load the 4-bar introduced in [Defining a Linkage](#). The linkage is referenced here based on the sequence of its joint constraints (‘RSSR’):

```
# Load the planar 4-bar
linkage <- linkR_examples('RSSR')
```

Then use 'animateLinkage', rotating the first link over three angles from 0 to  $\frac{\pi}{4}$  radians (0 to 45 degrees):

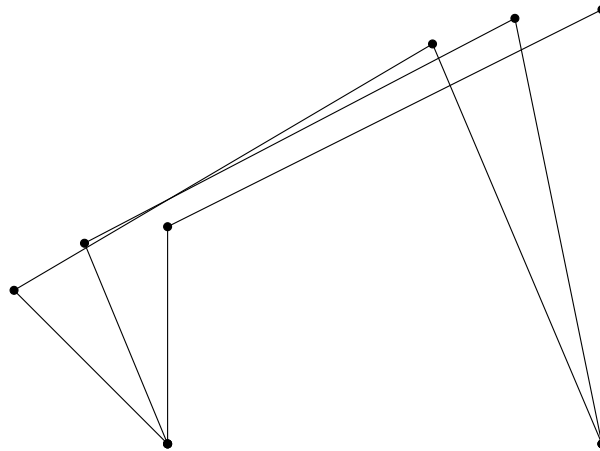
```
# Simulate with input rotations of 0, 22.5, and 45 deg at first R-joint
animate <- animateLinkage(linkage, input.param=c(0,pi/8,pi/4), input.joint=1)
```

Visualize this motion using 'drawLinkage':

```
# Create animated visualization
drawLinkage(animate, file='Animate')
```

The drawLinkage function creates an interactive animation file using the R package [svgViewR](#) that can be opened in any major web browser. You can also create a static visualization, superimposing each frame, by setting the 'animate' parameter to FALSE:

```
# Create static visualization
drawLinkage(animate, file='Static', animate=FALSE)
```



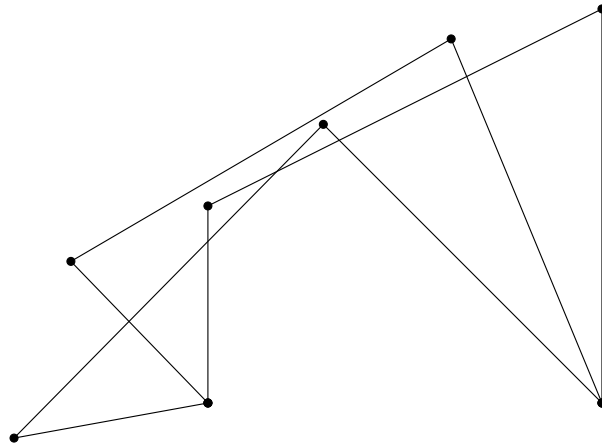
A 4-bar linkage with the first rotating link (left) rotated to 0, 22.5 and 45 degrees (all frames superimposed).

Note that the input rotations follow the right-hand rule. Since the joint constraint vector is positive along the z-axis, a positive angular rotation will rotate the link in a counter-clockwise direction when viewed such that the positive z-axis is pointing toward the viewer (as above).

To specify the input rotations at the other rotating link, change 'input.joint' to '4' since the second R-joint was the fourth joint when [defining the joint types](#).

```
# Simulate with input rotations of 0, 22.5, and 45 deg at second R-joint
animate <- animateLinkage(linkage, input.param=c(0,pi/8,pi/4), input.joint=4)
```

```
# Create animation
drawLinkage(animate, file='Animate_4')
```

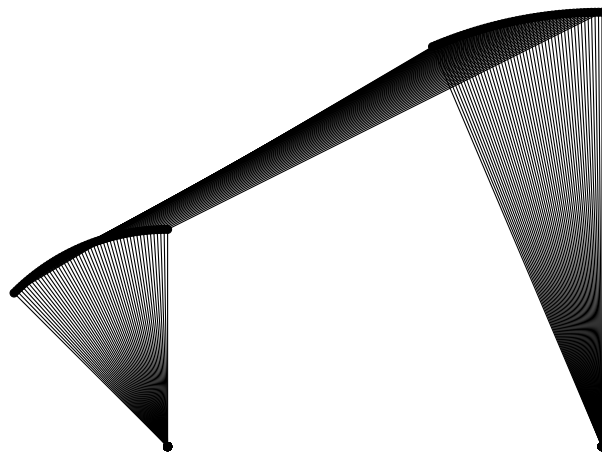


A 4-bar linkage with the second rotating link (right) rotated to 0, 22.5 and 45 degrees.

To simulate motion over a finer scale and create a smoother animation, you can increase the number of angles in 'input.param'.

```
# Simulate with 50 input rotations between 0 and 45 deg
animate <- animateLinkage(linkage, input.param=seq(0,pi/4,length=50), input.joint=1)

# Create animation
drawLinkage(animate, file='Animate_50')
```



A 4-bar linkage with the first rotating link (left) rotated at 50 angles.

You can run the simulation forward and then in reverse to create a looped animation by either adding angles in the reverse direction through 'input.param'

```
# Define forward and reverse input parameters
input.param <- c(seq(0,pi/4,length=50), seq(pi/4,0,length=50))

# Simulate motion
animate <- animateLinkage(linkage, input.param=input.param, input.joint=1)
```



or by setting the 'animate.reverse' argument for drawLinkage to TRUE.

```
# Simulate with 50 input rotations between 0 and 45 deg
animate <- animateLinkage(linkage, input.param=seq(0,pi/4,length=50), input.joint=1)

# Create animation with forward-reverse loop
drawLinkage(animate, file='Animate_loop', animate.reverse=TRUE)
```

Note that if you're performing analysis on the simulated motion, creating a loop using 'animate.reverse' will only affect the visualization file, not the simulation results themselves.

## 5 Analyzing linkage motion

Once you've simulated linkage motion over a range of conformations you can track the relative motion of the links or different points within the linkage to quantify relative motion of linkage elements, calculate transmission metrics (e.g. mechanical advantage), or perform further analyses. The function 'linkageKinematics' takes as input the simulated motion of a linkage and pulls out the motion of the joints, links, and associated points, which can then be used in subsequent analyses.

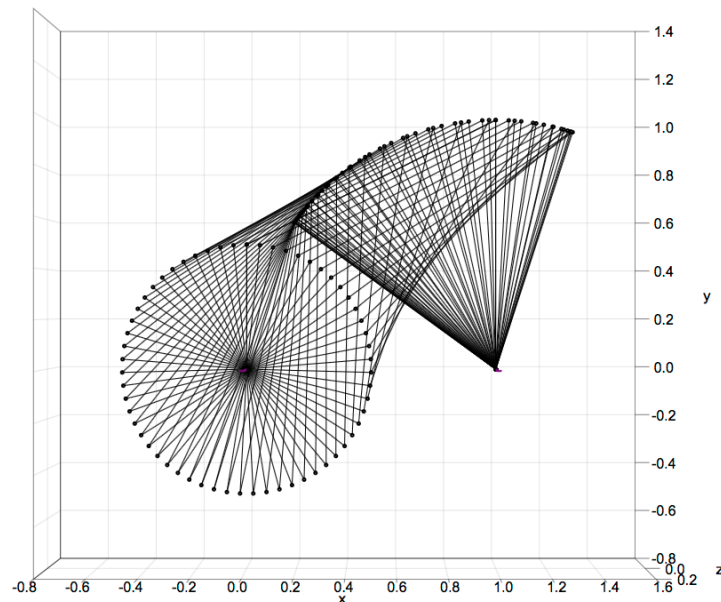
### 5.1 Joint position over time

To demonstrate a few approaches to linkage analysis we'll take the previously introduced 4-bar linkage and rotate the smaller link a full 360 degrees.

```
# Load the tutorial 4-bar linkage with associated points
linkage <- linkR_examples('RSSR_points')

# Input a 360 deg (2*pi rad) rotation of the shorter link
animate <- animateLinkage(linkage, input.param=seq(0,2*pi,length=60), input.joint=1)

# Create animation with forward-reverse loop
drawLinkage(animate, file='Analyze')
```



A 4-bar linkage with the shorter link rotated 360 deg (all frames superimposed). Associated points are omitted for clarity.

Note that because of the linkage configuration a full rotation of the smaller link only results in a partial rotation of the larger link.

To obtain the coordinates of all the joints in the linkage over the simulated iterations, access the ‘joint.coor’ object of the list returned by ‘animateLinkage’ (here, ‘animate’).

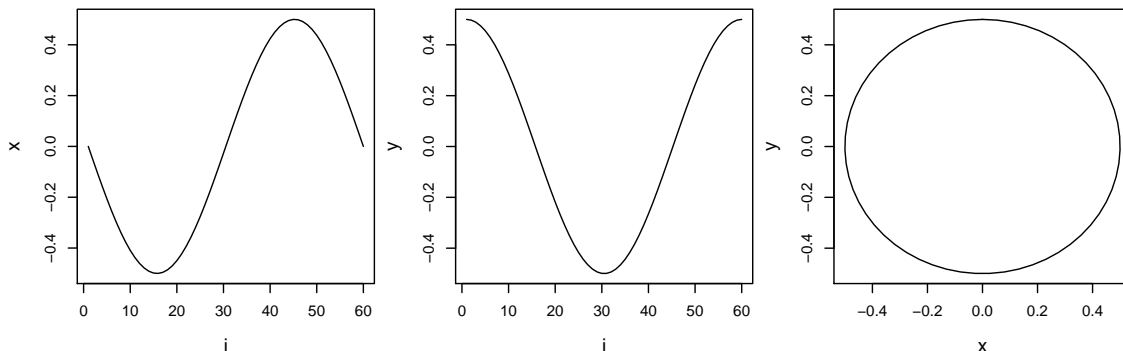
```
# Array of joint coordinates over iterations
joints <- animate$joint.coor
```

‘joint.coor’ is a 3D array in which the first dimension corresponds to the joints, the second corresponds to the xyz-coordinates, and the third corresponds to the animation frame. For example, the following code plots the xy-coordinates of the second joint (the joint between the short rotating link and the coupler link) through the simulation. Since this is a planar linkage, the z-coordinate doesn’t change and the z-vs-t plot can be replaced with a y-vs-x plot.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot xy-coordinates joint over iterations
for(i in 1:2) plot(joints[2, i, ], type='l', ylab=c('x','y')[i], xlab='i')

# Plot xy-coordinates in y-vs-x plot
plot(x=joints[2,1,], y=joints[2,2,], type='l', xlab='x', ylab='y')
```



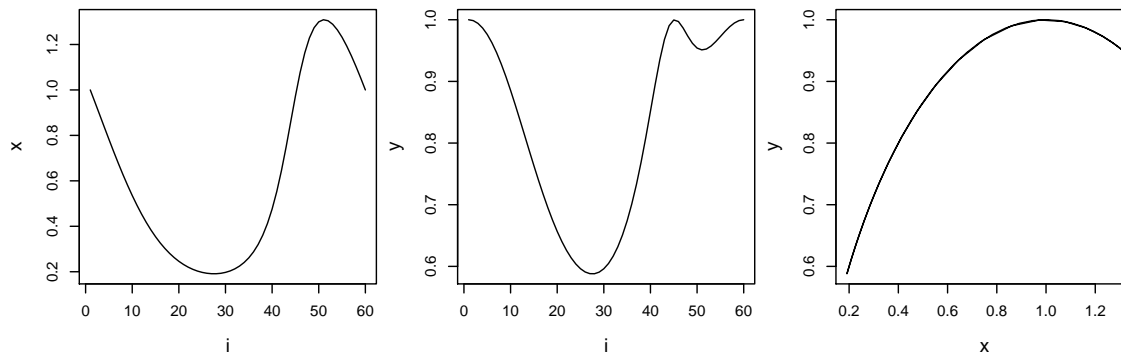
The change in the xy-coordinates of the second joint in an animated 4-bar linkage.

Since this joint is attached to a rotating link that makes a full rotation, its motion describes a circle. The motion of the longer link is more complex, rotating at a non-constant rate over a partial arc. This can be seen by plotting the third joint in the linkage (the joint between the coupler link and the second rotating link).

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot xy-coordinates joint over iterations
for(i in 1:2) plot(joints[3, i, ], type='l', ylab=c('x','y')[i], xlab='i')

# Plot xy-coordinates in y-vs-x plot
plot(x=joints[3,1,], y=joints[3,2,], type='l', xlab='x', ylab='y')
```



The change in the xy-coordinates of the third joint in an animated 4-bar linkage.

## 5.2 Point position over time

If points were associated with links when defining the linkage, their coordinates can be accessed through the ‘link.points’ object of the list returned by ‘animateLinkage’.

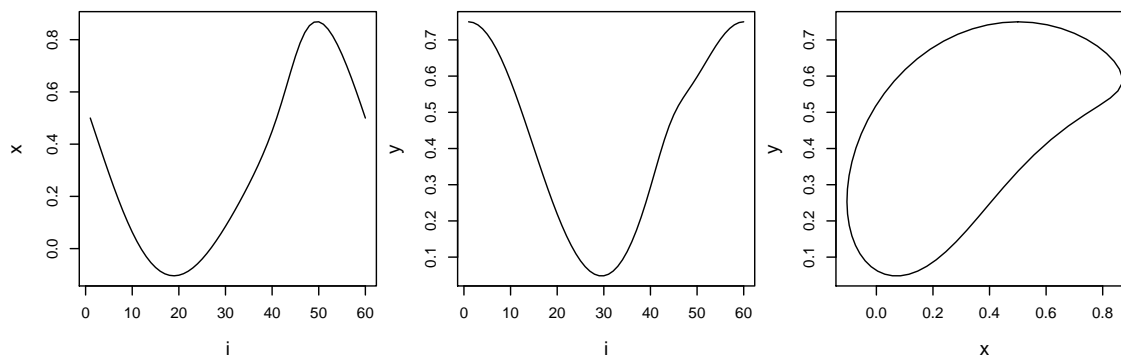
```
# Array of point coordinates over iterations
pts <- animate$link.points
```

Like ‘joint.coor’, ‘link.points’ is a 3D array in which the first dimension corresponds to the joints, the second corresponds to the xyz-coordinates, and the third corresponds to the animation frame. The xy-coordinates of any point can be plotted just as for the joint coordinates. The following code plots the 5th point in ‘link.points’, the point in the middle of the 4-bar coupler link.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot xyz-coordinates joint over iterations
for(i in 1:2) plot(pts[5, i, ], type='l', ylab=c('x','y')[i], xlab='i')

# Plot xy-coordinates in y-vs-x plot
plot(x=pts[5,1,], y=pts[5,2,], type='l', xlab='x', ylab='y')
```



The change in the xy-coordinates of a point in the middle of the 4-bar coupler link.

### 5.3 Calling ‘linkageKinematics’

For certain analyses we might be interested not just in the positions of the joints and associated points but also in their velocity (i.e. derivatives of their position over the animation). Also, we might be interested in the angular displacements (i.e. rotation) of the links and its derivative, angular velocity. The ‘linkageKinematics’ function calculates these quantities to streamline kinematic analysis.

```
# Measure linkage kinematics
kine <- linkageKinematics(animate)
```

The function returns the kinematics of joint translation, link rotation, and associated point translation. For joints and points these kinematic results are accessed by appending the following to ‘joints’ or ‘points’ (e.g. ‘joints.t’, ‘points.tdis.d’):

1. **t**: Displacement of each joint/point in each dimension (3D array)
2. **tdis**: Total displacement of each joint/point (2D matrix)
3. **t.d**: Velocity of each joint/point in each dimension (the derivative of 1, 3D array)
4. **tdis.d**: Total velocity of each joint/point (the derivative of 2, 2D matrix)

The kinematic results for links are similar except that they measure rotation rather than translation (replacing ‘t’ with ‘r’). These can be accessed by appending the following to ‘links’ (e.g. ‘links.r’, ‘links.rdis.d’):

1. **r**: Angular displacement of each link about each axis (3D array)
2. **rdis**: Total angular displacement of each link (2D matrix)
3. **r.d**: Angular velocity of each link about each axis (the derivative of 1, 3D array)
4. **rdis.d**: Total angular velocity of each link (the derivative of 2, 2D matrix)

In the context of rotation, “angular displacements about each axis” refers to the magnitudes of a series of rotations about the z, y, and x-axes, in that order, that reproduces the full rotation, so called ‘Euler angles’. Note that because links can move with both translational and rotational components, different points on a single link can displace to differing extents. Thus, for whole link motion only rotation is quantified while joints and link-associated points provide a means to track translational motion at any point in a linkage.

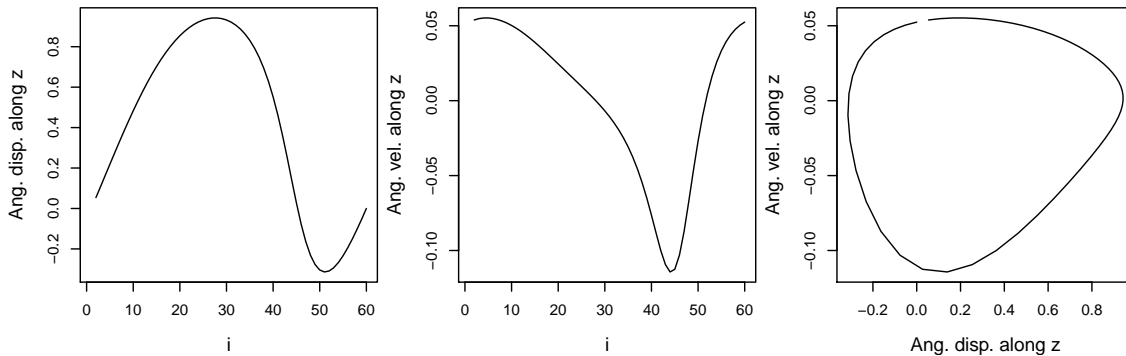
Displacements are measured relative to the first iteration. Also, the “derivatives” are calculated simply by taking the difference between consecutive iterations. For this reason, the first iteration will always be NA. The smoothness or resolution of the derivatives will depend on the motion between consecutive iterations. To increase the smoothness simply increase the number of iterations. The derivatives are calculated with respect to iterations or steps, rather than time.

## 5.4 Angular displacement and velocity of links

To access the Euler components of link angular displacement and velocity, use the objects ‘links.r’ and ‘linkr.r.d’, respectively. For example, the following code plots the angular displacement and velocity of the longer rotating link in the 4-bar linkage about the z-axis. If no link names are provided, link names are automatically assigned as ‘Ground’, ‘Link1’, ‘Link2’, etc. The longer rotating link in the previously introduced 4-bar linkage is the third link or ‘Link3’.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot z-axis angular displacement, velocity, and displacement vs. velocity
plot(kine$links.r['Link3',3,], type='l', xlab='i', ylab='Ang. disp. along z')
plot(kine$links.r.d['Link3',3,], type='l', xlab='i', ylab='Ang. vel. along z')
plot(x=kine$links.r['Link3',3,], y=kine$links.r.d['Link3',3,], type='l',
     xlab='Ang. disp. along z', ylab='Ang. vel. along z')
```



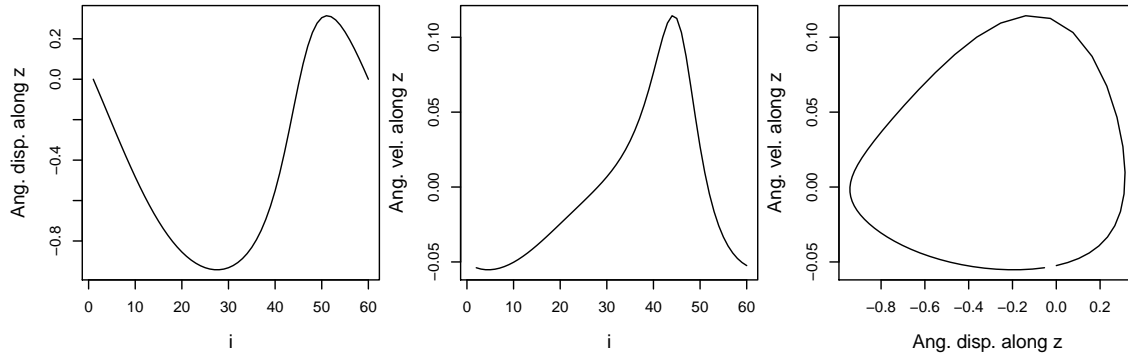
Angular displacement and velocity of the longer rotating link (z-axis component) in the 4-bar linkage.

To access the total angular displacements and velocities of links (i.e. not the rotational components), use the objects ‘links.rdis’ and ‘links.rdis.d’, respectively. For example, the following code plots the total angular displacement and velocity of the longer rotating link in the 4-bar linkage.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot total angular displacement, velocity, and displacement vs. velocity
plot(kine$links.rdis['Link3',], type='l', xlab='i', ylab='Ang. disp. along z')
plot(kine$links.rdis.d['Link3',], type='l', xlab='i', ylab='Ang. vel. along z')
plot(x=kine$links.rdis['Link3',], y=kine$links.rdis.d['Link3',], type='l',
     xlab='Ang. disp. along z', ylab='Ang. vel. along z')
```

Except for the sign of the angular displacements, the total angular displacements are the same as the rotational components along the z-axis. This is because the link is constrained to rotate entirely about the z-axis and therefore has no rotational component about the x- and y-axes.



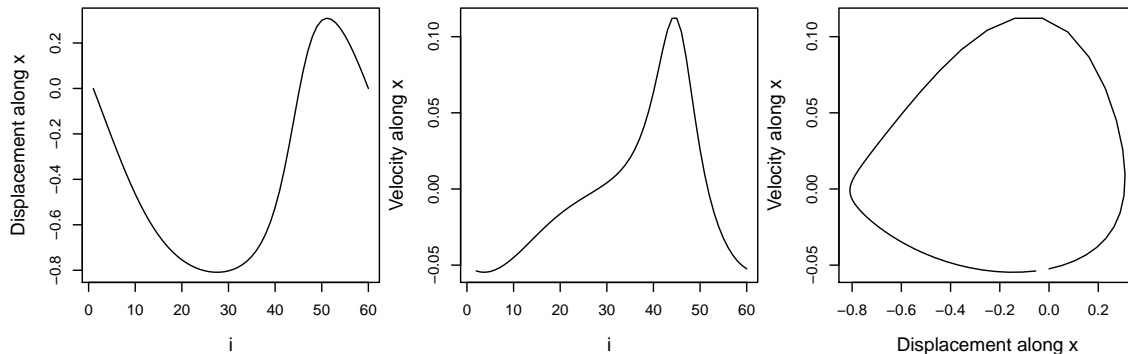
Total angular displacement and velocity of the longer rotating link in the 4-bar linkage.

## 5.5 Displacement and velocity of joints

To access the displacements and velocities of joints along a particular dimension, use the objects ‘joints.t’ and ‘joints.t.d’, respectively. For example, the following code plots the x-axis displacement and velocity of the third joint in the 4-bar linkage.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot x-axis displacement, velocity, and displacement vs. velocity
plot(kine$joints.t[3,1,], type='l', xlab='i', ylab='Displacement along x')
plot(kine$joints.t.d[3,1,], type='l', xlab='i', ylab='Velocity along x')
plot(x=kine$joints.t[3,1,], y=kine$joints.t.d[3,1,], type='l',
     xlab='Displacement along x', ylab='Velocity along x')
```



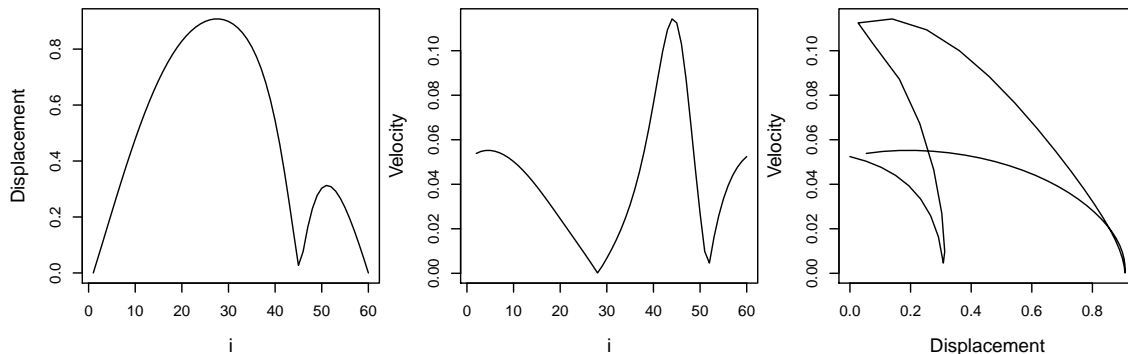
Displacement and velocity along the x-axis of the third joint in the 4-bar linkage.

The displacement plots of the third joint look very similar to the angular displacement plots of the longer rotating link because the joint rotates with the link at a distance of exactly 1 from the points of rotation.

To access the total displacements and velocities of joints, use the objects ‘joints.tdis’ and ‘joints.tdis.d’, respectively. For example, the following code plots the total displacement and velocity of the third joint in the 4-bar linkage.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot total displacement, velocity, and displacement vs. velocity
plot(kine$joints.tdis[3,], type='l', xlab='i', ylab='Displacement')
plot(kine$joints.tdis.d[3,], type='l', xlab='i', ylab='Velocity')
plot(x=kine$joints.tdis[3,], y=kine$joints.tdis.d[3,], type='l',
     xlab='Displacement', ylab='Velocity')
```



Total displacement and velocity along the x-axis of the second joint in the 4-bar linkage.

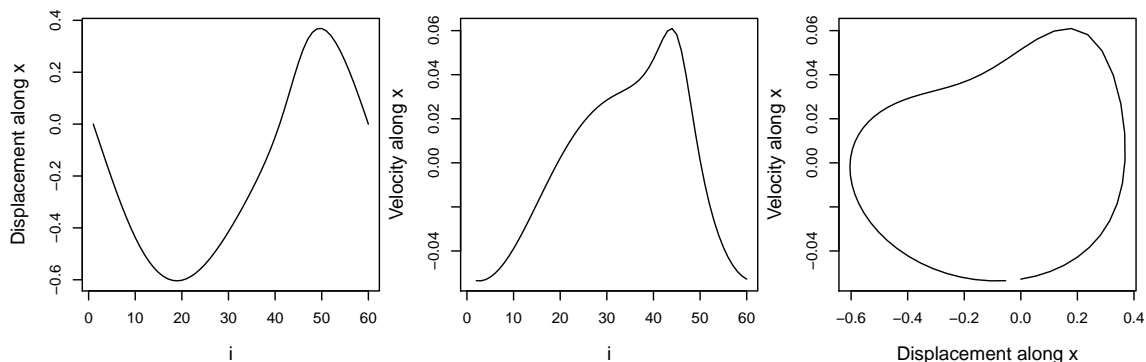
Note that because these are total displacements from the initial position they can only be positive.

## 5.6 Displacement and velocity of points

The displacements and velocities of points associated with links can be accessed in the same manner as joints, simply replacing ‘joints’ with ‘points’ (i.e. ‘points.t’ and ‘points.t.d’). For example, the following code plots the x-axis displacement and velocity of the point attached to the middle of the 4-bar coupler link.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot x-axis displacement, velocity, and displacement vs. velocity
plot(kine$points.t[5,1,], type='l', xlab='i', ylab='Displacement along x')
plot(kine$points.t.d[5,1,], type='l', xlab='i', ylab='Velocity along x')
plot(x=kine$points.t[5,1,], y=kine$points.t.d[5,1,], type='l',
     xlab='Displacement along x', ylab='Velocity along x')
```



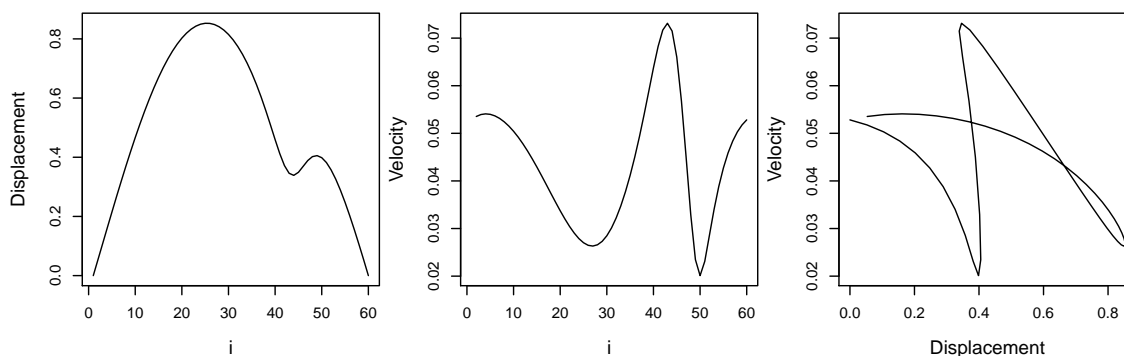


Displacement and velocity along the x-axis of the point in the middle of the 4-bar coupler link.

To access the total displacements and velocities of points, use the objects ‘points.tdis’ and ‘points.tdis.d’, respectively. For example, the following code plots the total displacement and velocity of the point in the middle of the 4-bar coupler link.

```
# Open 3-column, multipanel plot with specified margins
par(mfrow=c(1,3), mar=c(4,4,0.5,0.5))

# Plot total displacement, velocity, and displacement vs. velocity
plot(kine$points.tdis[5,], type='l', xlab='i', ylab='Displacement')
plot(kine$points.tdis.d[5,], type='l', xlab='i', ylab='Velocity')
plot(x=kine$points.tdis[5,], y=kine$points.tdis.d[5,], type='l',
      xlab='Displacement', ylab='Velocity')
```



Total displacement and velocity of the point in the middle of the 4-bar coupler link.

## 5.7 Torque mechanical advantage

The derivatives of displacement can be used to calculate metrics such as mechanical advantage and kinematic transmission. These metrics measure the amplification of relative force (or torque) and velocity through a mechanism (for more details see [Olsen and Westneat 2016](#)).

For rotating links, the derivative of angular displacement can be used to calculate torque mechanical advantage (tMA). Torque MA is equal to the angular velocity of an input link divided by the angular velocity of an output link (the time derivatives of these velocities are absent because they cancel out when taking the ratio). Torque MA is also equal to the ratio of output link torque divided by the input link torque.

$$tMA = \frac{d\theta_i}{d\theta_o} = \frac{d\tau_o}{d\tau_i} \quad (1)$$

A linkage with a torque MA greater than 1 transforms an input torque into a relatively larger output torque. This occurs at the expense of displacement - the output link also rotates over a relatively smaller angle than the input link. Another metric common in

the biomechanical literature is kinematic transmission (KT). This is simply the inverse of torque MA.

$$KT = \frac{1}{tMA} \quad (2)$$

Thus, a linkage with a KT greater than 1 transforms an input rotation into a relatively greater output rotation, but with a relatively lower torque.

Returning to our example 4-bar linkage, if we call the shorter link the input link ('input' and 'output' are arbitrary but it clarifies the discussion), then we can use the output of 'linkageKinematics' to calculate the torque MA of transmission from the input to output link. KT can be calculated by simply dividing the 'rdis.d' values of one link by those of another.

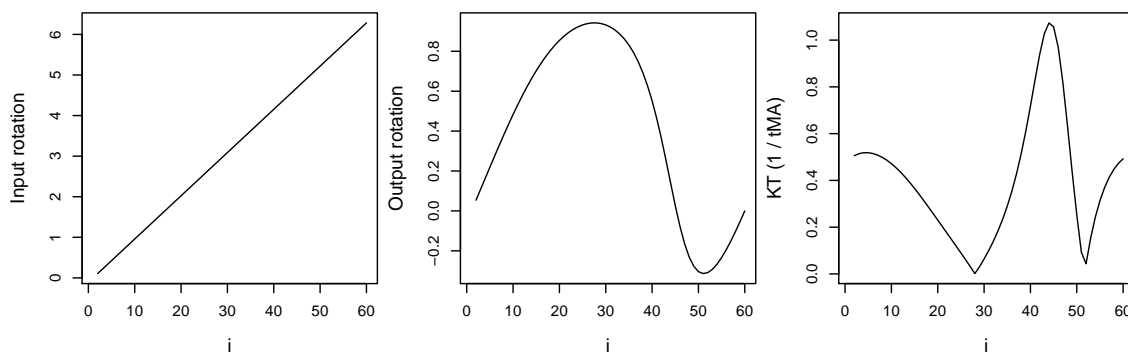
```
# Find KT by dividing instantaneous velocities of output by input
KT <- abs(kine$links.rdis.d['Link3', ] / kine$links.rdis.d['Link1', ])
```

It's important to use 'rdis.d' rather than 'rdis' because 'rdis' is the cumulative angular displacement from the initial iteration, not the instantaneous rate of displacement. Torque MA would simply be the inverse of this ratio. I've chosen to use KT here because the occasional small rotations of the output link cause torque MA to have some very large values, which makes plotting difficult (one could fix this by log-transforming the values).

The following code produces a plot showing how KT changes throughout the linkage's range of motion.

```
# Open 3-column, multipanel plot
par(mfrow=c(1,3), mar=c(4.2,4.1,0.5,0.5))

# Plot total rotation and kinematic transmission (inverse torque MA)
plot(kine$links.r['Link1',3,],type='l',xlab='i',ylab='Input rotation')
plot(kine$links.r['Link3',3,],type='l',xlab='i',ylab='Output rotation')
plot(KT, type='l', xlab='i', ylab='KT (1 / tMA)')
```



The kinematic transmission (KT; right) from the input link (left) to the output link (middle).

Note that for nearly any linkage mechanism, transmission ratios will change as the linkage moves. For this mechanism the output link changes direction twice during one cycle of the input link, causing KT to drop to 0 at these points.

## 5.8 Force mechanical advantage

Torque MA is only valid for rotating links and gives information related to angular velocity and torques. For mechanisms with links that move with some combination of translation and rotation or if you have some predetermined points at which forces are input and output, it is better to use force mechanical advantage (fMA). Force MA is essentially the linear (non-rotational) analogue to torque MA, representing the amplification of output force (relative to input force). Thus, force MA is equal to the ratio of the linear velocities or the ratio of output force to input force.

$$fMA = \frac{dx_i}{dx_o} = \frac{F_o}{F_i} \quad (3)$$

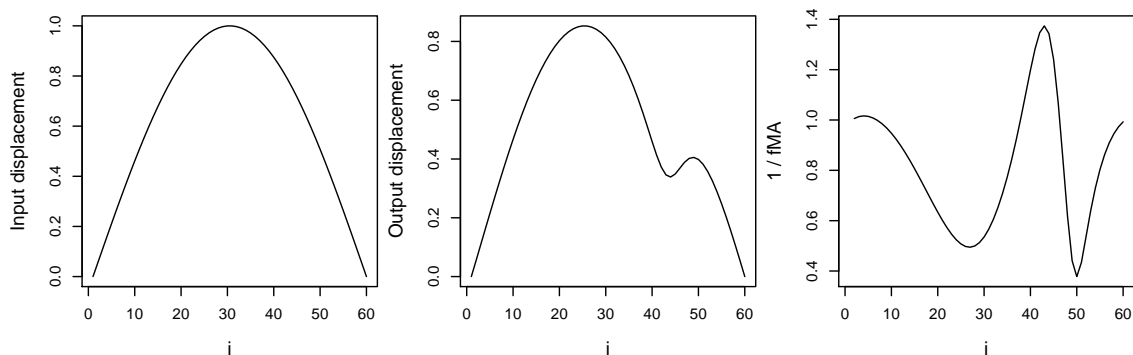
Analogous to KT, force MA can be calculated by simply dividing the ‘tdis.d’ values of one point in the linkage by those of another. The following code uses the joint between the input link and coupler link as the point of input force and the point in the middle of the coupler link as the point of output force.

```
# Find fMA by dividing instantaneous linear velocities of input by output
fMA <- abs(kine$joints.tdis.d[2, ] / kine$points.tdis.d[5, ])
```

The fMA can then be plotted along with the input and output displacements. The inverse of force MA is plotted below because occasional translations of the coupler link cause force MA to have some very large values.

```
# Open 3-column, multipanel plot
par(mfrow=c(1,3), mar=c(4.2,4.1,0.5,0.5))

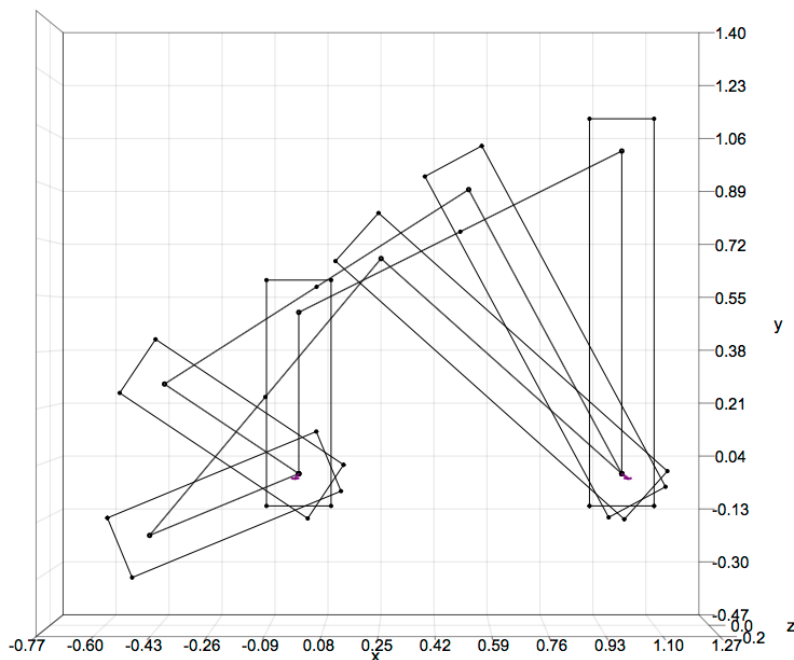
# Plot total rotation and kinematic transmission (inverse torque MA)
plot(kine$joints.tdis[2,],type='l',xlab='i',ylab='Input displacement')
plot(kine$points.tdis[5,],type='l',xlab='i',ylab='Output displacement')
plot(1/fMA, type='l', xlab='i', ylab='1 / fMA')
```



The inverse force MA (right) of transmission from the joint between the input link and coupler link (left) to the point in the middle of the coupler link (middle).

## 6 Examples

### 6.1 Planar 4-bar (RSSR)



Planar 4-bar linkage (DOF=1). Three iterations shown.

```
# Define the joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.5,0), c(1,1,0), c(1,0,0))

# Define the joint types
joint.types <- c('R', 'S', 'S', 'R')

# Define the joint constraint vectors
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1))

# Define the connections among joints
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0))

# Define first rectangle
rec1 <- rbind(c(-0.1,-0.1,0), c(-0.1,0.6,0), c(0.1,0.6,0), c(0.1,-0.1,0))

# Define second rectangle
rec2 <- rbind(c(0.9,-0.1,0), c(0.9,1.1,0), c(1.1,1.1,0), c(1.1,-0.1,0))

# Define the point coordinates
link.points <- rbind(rec1, c(0.5,0.75,0), rec2)

# Set the link associations for each point
link.assoc <- c(1,1,1,1, 2, 3,3,3,3)

# Define points to connect into paths
path.connect <- list(c(1:4,1), c(6:9,6))
```

```

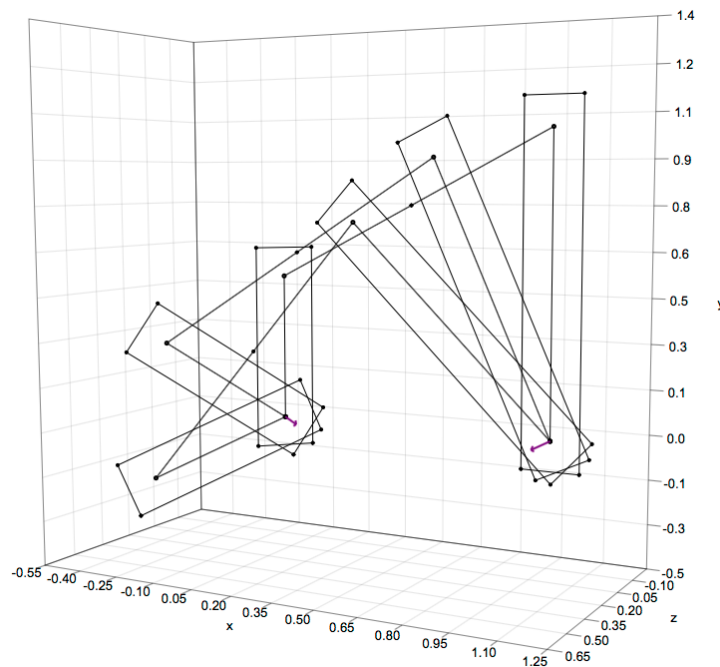
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,2*pi,length=60), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSR.html')

```

## 6.2 3D 4-bar (RSSR)



3D 4-bar linkage (DOF=1). Three iterations shown.

```

# Define the joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.5,0), c(1,1,0), c(1,0,0))

# Define the joint types
joint.types <- c('R', 'S', 'S', 'R')

# Define the joint constraint vectors
joint.cons <- list(c(1,0,1), NA, NA, c(0,0,1))

# Define the connections among joints
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0))

# Define first rectangle
rec1 <- rbind(c(-0.07,-0.1,0.07), c(-0.07,0.6,0.07),
  c(0.07,0.6,-0.07), c(0.07,-0.1,-0.07))

```

```

# Define second rectangle
rec2<- rbind(c(0.9,-0.1,0), c(0.9,1.1,0), c(1.1,1.1,0), c(1.1,-0.1,0))

# Define the point coordinates
link.points <- rbind(rec1, c(0.5,0.75,0), rec2)

# Set the link associations for each point
link.assoc <- c(1,1,1,1, 2, 3,3,3,3)

# Define points to connect into paths
path.connect <- list(c(1:4,1), c(6:9,6))

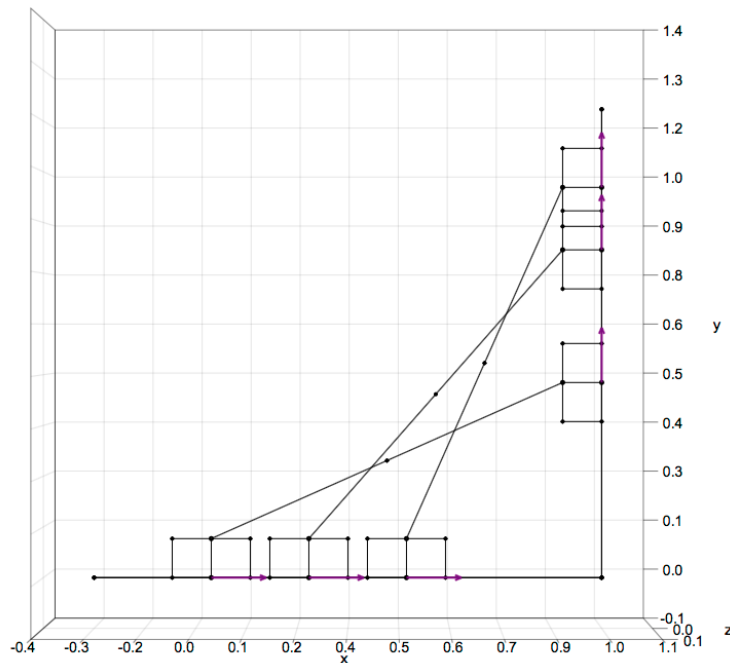
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,2*pi,length=60), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSR_3d.html')

```

## 6.3 Two coupled sliders (LSSL)



Linkage with two coupled linear sliding links (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.1,0), c(0.9,0.5,0), c(1,0.5,0))

```

```

# Define joint types
joint.types <- c("L", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(1,0,0), NA, NA, c(0,1,0))

# Define points associated with links
link.points <- rbind(c(-0.3,0,0), c(1,0,0), c(-0.1,0,0), c(-0.1,0.1,0), c(0.1,0.1,0),
  c(0.1,0,0), c(1,0.4,0), c(0.9,0.4,0), c(0.9,0.6,0), c(1,0.6,0), c(1,0,0), c(1,1.2
    ,0),
  c(0.45, 0.3, 0))

# Define links with which points are associated
link.assoc <- c(0,0,1,1,1,1,3,3,3,3,0,0,2)

# Define lines connecting associated points
path.connect <- list(c(1,2), c(3:6,3), c(7:10,7), c(11,12))

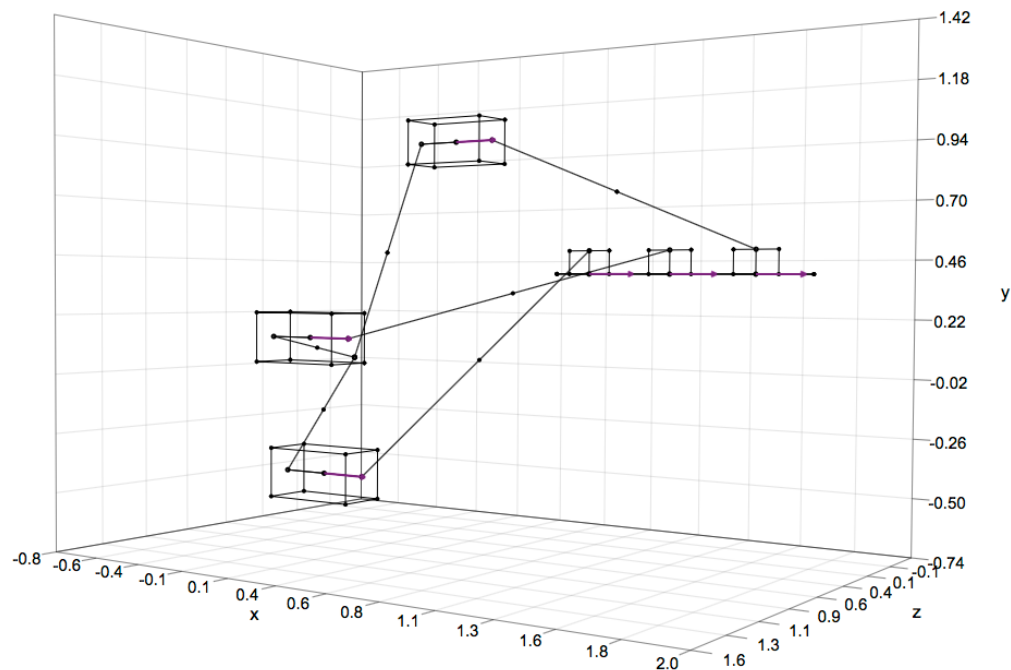
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, link.points=link.points, link.assoc=link.assoc,
  path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,0.5,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='LSSL.html', animate.reverse=TRUE)

```

## 6.4 Coupled planar and linear sliders (SSPSSL)



A planar sliding link coupled to a linear sliding link (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(-0.6,0,0), c(-0.2,1,0), c(0,1,0), c(0.2,1,0), c(1.5,0.5,0), c(1
.5,0.4,0))

# Define joint types
joint.types <- c("S", "S", "P", "S", "S", "L")

# Define joint constraints
joint.cons <- list(NA, NA, c(1,0,0), NA, NA, c(1,0,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,0), c(2,3), c(3,4), c(4,0))

# Define points associated with links
link.points <- rbind(c(-0.4,0.5,0), c(1.4,0.4,0), c(1.4,0.5,0), c(1.6,0.5,0),
c(1.6,0.4,0), c(0.55,0.4,0), c(1.75,0.4,0), c(0.85,0.75,0),
c(-0.2,0.9,0.1), c(0.2,0.9,0.1), c(0.2,0.9,-0.1), c(-0.2,0.9,-0.1),
c(-0.2,1.1,0.1), c(0.2,1.1,0.1), c(0.2,1.1,-0.1), c(-0.2,1.1,-0.1))

# Define links with which points are associated
link.assoc <- c(1,4,4,4,4,0,0,3,rep(2,8))

# Define paths to connect points
path.connect <- list(c(2:5,2), c(6:7), c(9:12,9), c(13:16,13), c(9,13),
c(10,14), c(11,15), c(12,16))

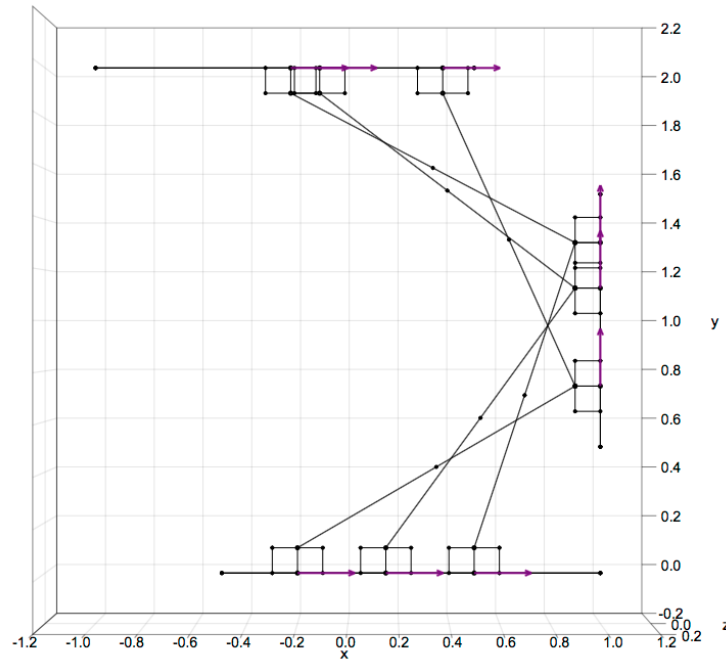
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,-pi/4,length=50), input.joint=6)

# Draw linkage
drawLinkage(anim, file='SSPSSL.html', animate=TRUE, animate.reverse=TRUE)
```



## 6.5 Three coupled sliders (LSSLSSL)



Linkage with three coupled linear sliding links (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(0,0,0),c(0,0.1,0),c(0.9,1,0),c(1,1,0),c(0.9,1,0),c(0,1.9,0),c
(0,2,0))

# Define joint types
joint.types <- c("L", "S", "S", "L", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(1,0,0), NA, NA, c(0,1,0), NA, NA, c(1,0,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Define points associated with links
link.points <- rbind(c(-0.1,0,0), c(-0.1,0.1,0), c(0.1,0.1,0), c(0.1,0,0),
c(1,0.9,0), c(0.9,0.9,0), c(0.9,1.1,0), c(1,1.1,0), c(-0.1,2,0), c(-0.1,1.9,0),
c(0.1,1.9,0), c(0.1,2,0), c(-0.5,0,0), c(1,0,0), c(1,0.5,0), c(1,1.5,0),
c(-1,2,0), c(0.5,2,0), c(0.45,0.55,0), c(0.45,1.45,0))

# Define links with which points are associated
link.assoc <- c(rep(1,4), rep(3,4), rep(5,4), rep(0,6), 2, 4)

# Define paths to connect points
path.connect <- list(c(1:4,1), c(5:8,5), c(9:12,9), c(13,14), c(15,16), c(17,18))

# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
```

```

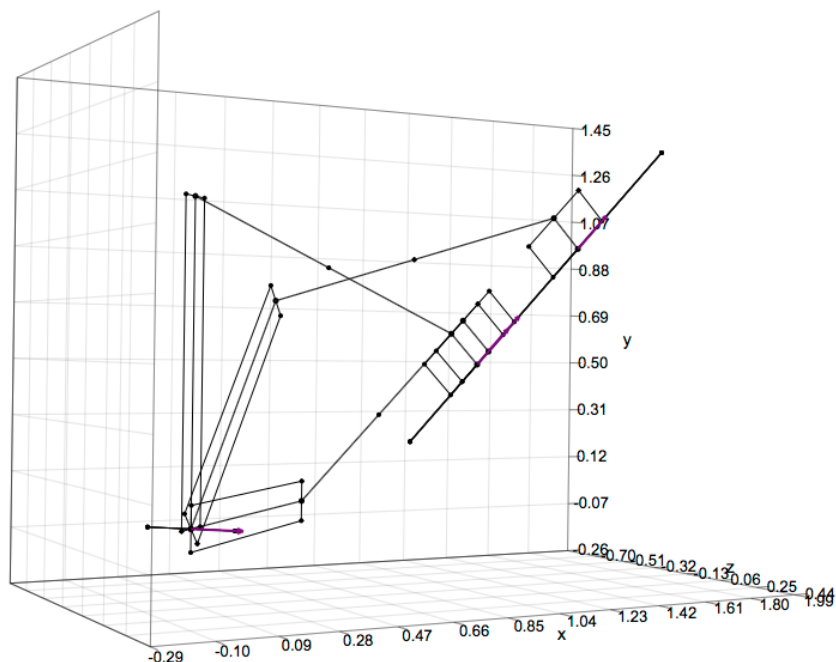
link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(-0.2, 0.5, length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='LSSLSSL.html', animate.reverse=TRUE)

```

## 6.6 Crank-driving sliding link (RSSL)



Rotating link coupled with sliding link (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(0.9,0.6,0), c(1,0.5,0))

# Define joint types
joint.types <- c("R", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(1,0,1), NA, NA, c(1,1,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0))

# Define points associated with links
link.points <- rbind(c(0.75,0.25,0), c(1.75,1.25,0), c(0.9,0.4,0), c(0.8,0.5,0),
  c(1,0.7,0), c(1.1,0.6,0), c(0.1,0,0.1), c(-0.1,0,-0.1), c(-0.05,0,0.05),
  c(-0.05,1,0.05), c(0.05,1,-0.05), c(0.05,0,-0.05), c(0.45,0.8,0))

# Define links with which points are associated
link.assoc <- c(0,0,3,3,3,3,0,0,1,1,1,1,2)

```

```

# Define paths to connect points
path.connect <- list(c(1,2), c(3:6,3), c(7,8), c(9:12,9))

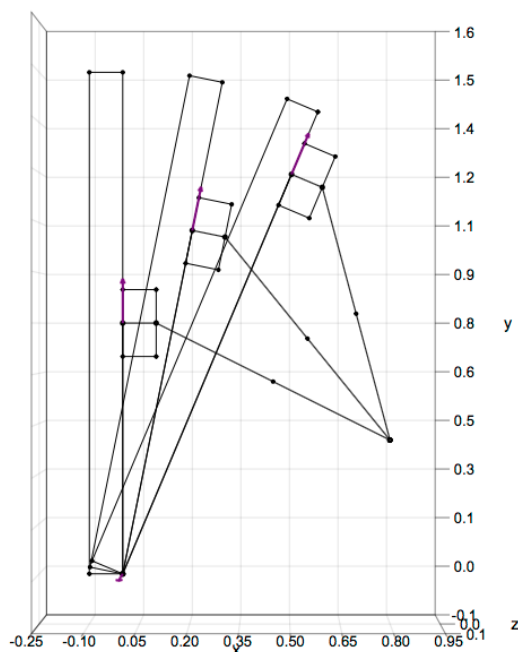
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,-pi/2,length=60), input.joint=1)

# DRAW LINKAGE
drawLinkage(anim, file='RSSL.html', animate.reverse=TRUE)

```

## 6.7 Slider along rotating link (RLSS)



Link sliding along a rotating link (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.75,0), c(0.1,0.75,0), c(0.8,0.4,0))

# Define joint types
joint.types <- c("R", "L", "S", "S")

# Define joint constraints
joint.cons <- list(c(0,0,1), c(0,1,0), NA, NA)

# Define link names
link.names <- c('Ground', 'L1', 'L2', 'L3')

```

```

# Define two links connected by each joint
joint.conn <- rbind(c('Ground', 'L1'), c('L1', 'L2'), c('L2', 'L3'), c('L3', 'Ground'))

# Define points associated with links
link.points <- rbind(c(0,0.65,0), c(0.1,0.65,0), c(0.1,0.85,0), c(0,0.85,0),
  c(0,0,0), c(-0.1,0,0), c(-0.1,1.5,0), c(0,1.5,0), c(0.45, 0.575, 0))

# Define links with which points are associated
link.assoc <- c(2,2,2,2,1,1,1,1,3)

# Define lines connecting associated points
path.connect <- list(c(1:4,1), c(5:8,5))

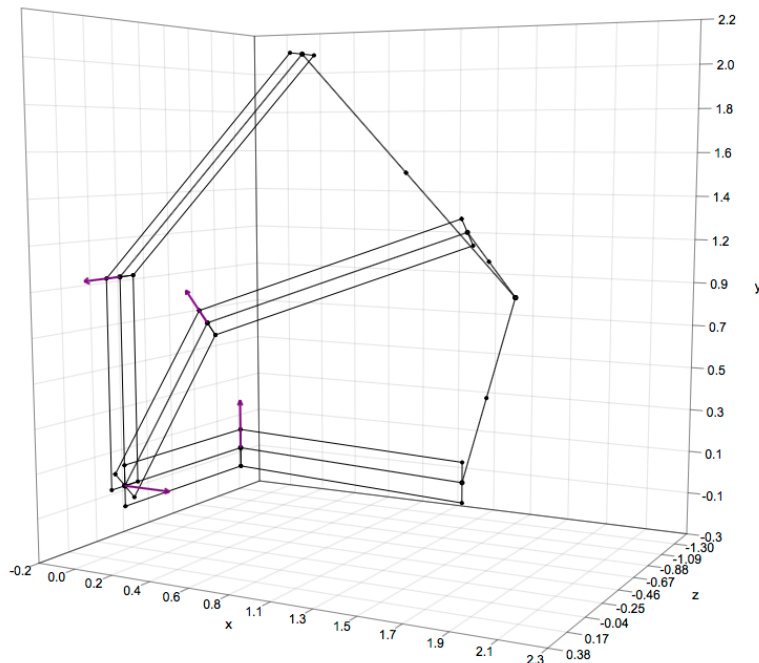
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.names=link.names,
  link.points=link.points, link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,-0.4,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RLSS.html', animate.reverse=TRUE)

```

## 6.8 Rotating links in series (RRSS)



Two rotating links in series with a spherical joint (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,2,0), c(2,1,0))

```

```
# Define joint types
joint.types <- c("R", "R", "S", "S")

# Define joint constraints
joint.cons <- list(c(1,0,0), c(0,0,1), NA, NA)

# Define link names
link.names <- c('Ground', 'L1', 'L2', 'L3')

# Define two links connected by each joint
joint.conn <- rbind(c('Ground', 'L1'), c('L1', 'L2'), c('L2', 'L3'), c('L3', 'Ground'))

# Define points associated with links
link.points <- rbind(c(0,0,0.1), c(0,1,0.1), c(0,1,-0.1), c(0,0,-0.1),
  c(0,1,0.1), c(0,1,-0.1), c(1,2,-0.1), c(1,2,0.1), c(1.5,1.5,0))

# Define links with which points are associated
link.assoc <- c(1,1,1,1,2,2,2,2,3)

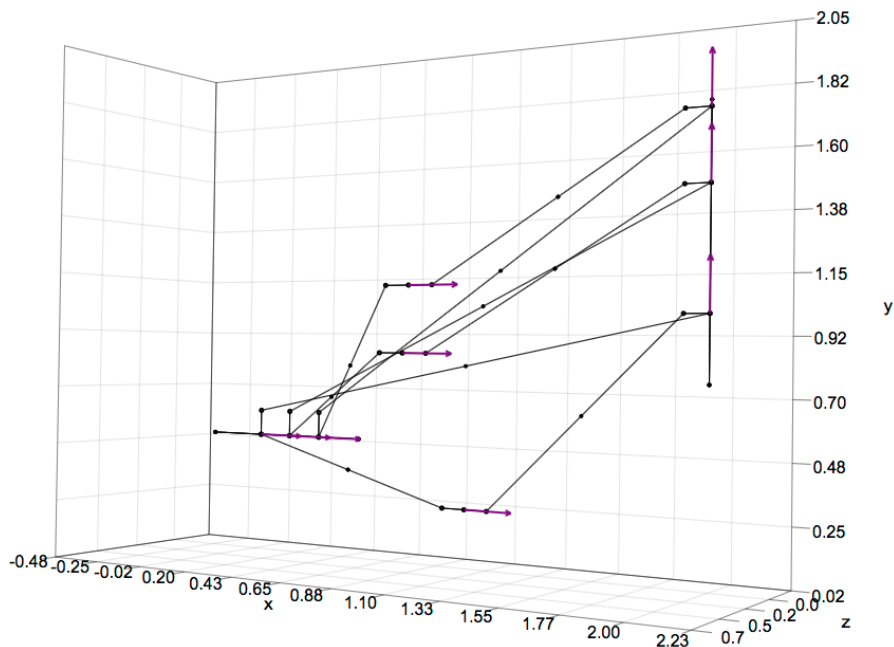
# Define lines connecting associated points
path.connect <- list(c(1:4,1), c(5:8,5))

# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, link.names=link.names, joint.conn=joint.conn,
  link.points=link.points, link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,-pi/2,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RRSS.html', animate.reverse=TRUE)
```

## 6.9 Coupled linear, planar sliders (LSSLSSPSS)



Linkage with two linear sliding links coupled by a planar sliding link (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(0,0.5,0), c(0,0.6,0), c(2,1,0), c(2,1,0), c(0,0.5,0),
  c(0.9,0.25,0), c(1,0.25,0), c(1.1,0.25,0), c(1.9,1,0))

# Define joint types
joint.types <- c("L", "S", "S", "L", "S", "S", "P", "S", "S")

# Define joint constraints
joint.cons <- list(c(1,0,0), NA, NA, c(0,1,0), NA, NA, c(1,0,0), NA, NA)

# Define two links connected by each joint
joint.conn <- rbind(c(0,1),c(1,2),c(2,3),c(3,0),c(1,4),c(4,5),c(5,0),c(5,6),c(6,3))

# Define points associated with links
link.points <- rbind(c(-0.25,0.5,0), c(0.5,0.5,0), c(2,0.75,0), c(2,1.75,0),
  c(1,0.8,0), c(0.45,0.375,0), c(1.5,0.625,0))

# Define links with which points are associated
link.assoc <- c(rep(0,4),2,4,6)

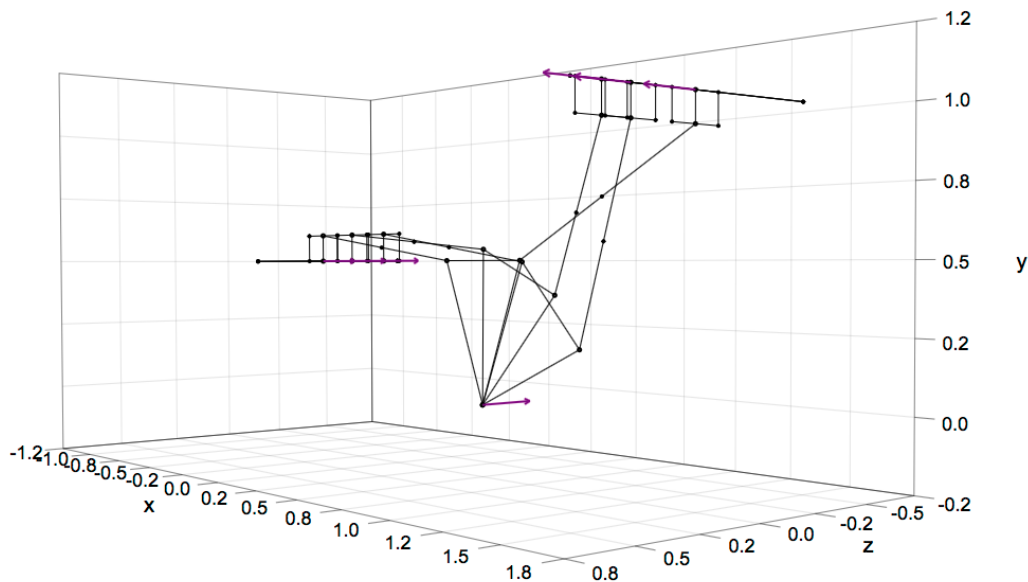
# Define paths to connect points
path.connect <- list(c(1,2), c(3,4))

# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)
```

```
# Animate linkage
anim <- animateLinkage(linkage,input.param=seq(0,0.3,length=50),input.joint=1)

# Draw linkage
drawLinkage(anim, file='LSSLSSPSS.html', animate.reverse=TRUE)
```

## 6.10 Sliders coupled by rotating link (LSSRSSL)



Two linear sliding links coupled by a rotating link (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(-0.5,0.5,0), c(-0.5,0.6,0), c(0.3,0.5,0), c(0.5,0,0),
  c(0.7,0.5,0), c(1.5,0.9,0), c(1.5,1,0))

# Define joint types
joint.types <- c("L", "S", "S", "R", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(1,0,0), NA, NA, c(0,0,-1), NA, NA, c(0,0,1))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Define points associated with links
link.points <- rbind(c(-0.6,0.5,0), c(-0.6,0.6,0), c(-0.4,0.6,0), c(-0.4,0.5,0),
  c(1.5,0.9,-0.1), c(1.5,1,-0.1), c(1.5,1,0.1), c(1.5,0.9,0.1), c(-1,0.5,0),
  c(0,0.5,0), c(1.5,1,-0.5), c(1.5,1,0.5), c(-0.1,0.55,0), c(1.1,0.7,0))

# Define links with which points are associated
link.assoc <- c(rep(1,4), rep(5,4), rep(0,4), 2, 4)

# Define paths to connect points
```

```

path.connect <- list(c(1:4,1), c(5:8,5), c(9,10), c(11,12))

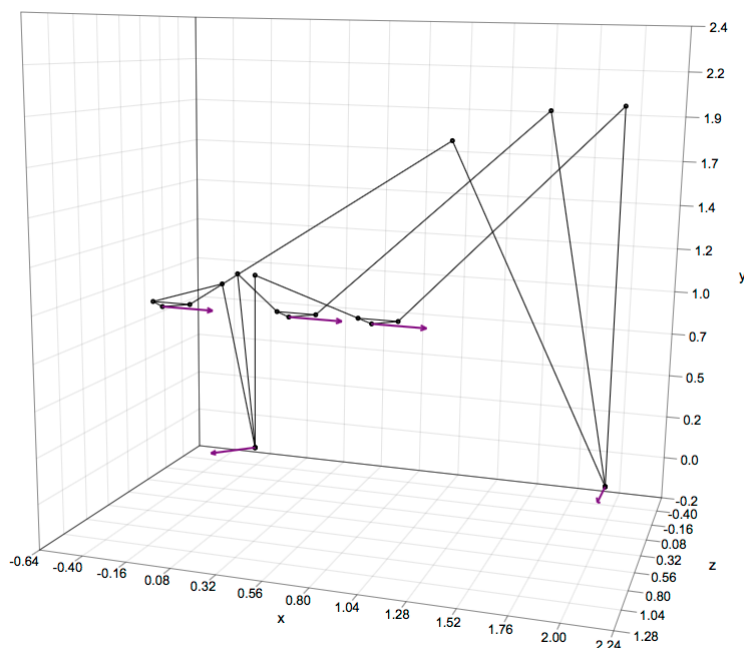
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,pi/4,length=50), input.joint=4)

# Draw linkage
drawLinkage(anim, file='LSSRSSL.html', animate.reverse=TRUE)

```

## 6.11 Coupled rotating, linear links (RSSLSSR)



Two rotating links coupled by a link that slides along a line (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(0.9,1,0.9), c(1,1,1),
  c(1.1,1,0.9), c(2,2,0), c(2,0,0))

# Define joint types
joint.types <- c("R", "S", "S", "L", "S", "S", "R")

# Define joint constraints
joint.cons <- list(c(-1,0,1), NA, NA, c(1,0,0), NA, NA, c(0,0,1))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Define linkage

```

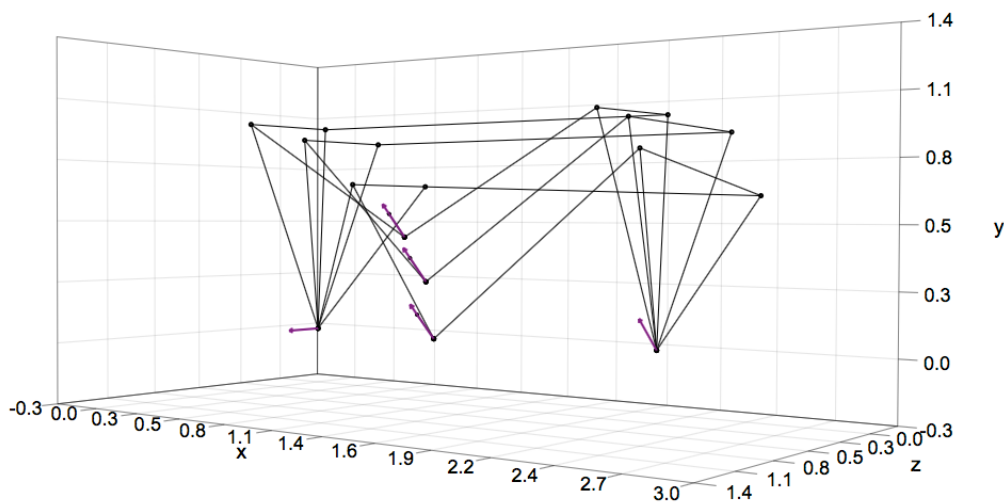


```
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,pi/6,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSLSSR.html', animate.reverse=TRUE)
```

## 6.12 Coupled rotating, planar links (RSSRSPSS)



Two rotating links coupled by link that slides within a plane (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(2,1,0), c(2,0,0),
  c(0,1,0.5), c(1,0.5,0.5), c(1,0.5,0.5), c(1,0.5,0.5), c(2,1,0.5))

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "P", "S", "S")

# Define joint constraints
joint.cons <- list(c(0,0,1), NA, NA, c(0,1,1), NA, NA, c(0,1,1), NA, NA)

# Define two links connected by each joint
joint.conn <- rbind(c(0,1),c(1,2),c(2,3),c(3,0),c(1,4),c(4,5),c(5,0),c(5,6),c(6,3))

# Define points associated with links
link.points <- rbind(c(1,0.5,0.5), c(1,0.6,0.6))

# Define links with which points are associated
link.assoc <- rep(5,2)

# Define lines connecting associated points
path.connect <- list(c(1,2))
```

```

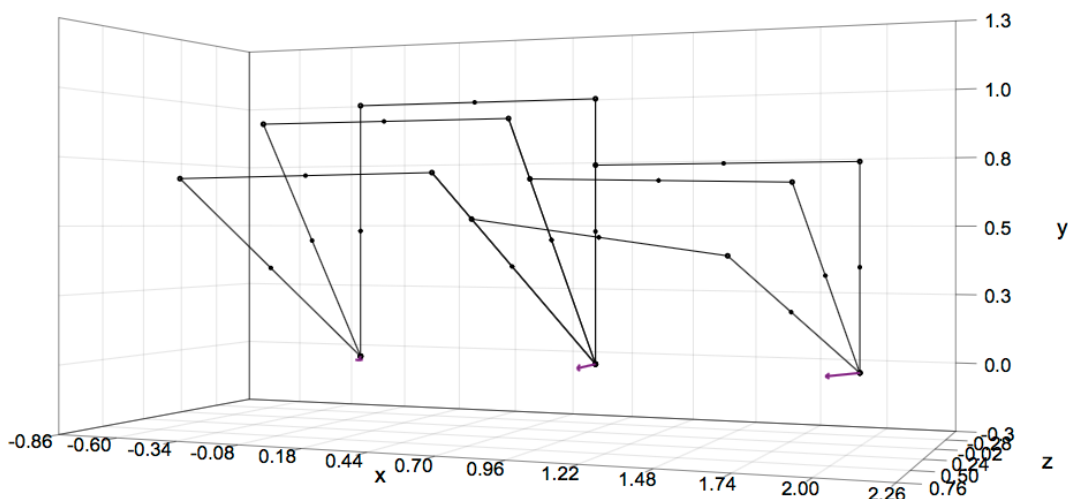
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc, path.connect=path.connect)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(-0.05,-pi/4,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSRSPSS.html', animate.reverse=TRUE)

```

### 6.13 Two 3D 4-bars in series (RSSRSSR))



Two 3D 4-bar linkages coupled via a shared rotating link (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,1,0), c(1,0,0), c(1,0.75,0), c(2,0.75,0),
  c(2,0,0))

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "R")

# Define joint constraints
joint.cons <- list(c(0.5,0,1), NA, NA, c(0,0,1), NA, NA, c(-0.5,0,1))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Define points associated with links
link.points <- rbind(c(0,0.5,0), c(0.5,1,0), c(1,0.5,0), c(1.5,0.75,0), c(2,0.375,0))

# Define links with which points are associated
link.assoc <- c(1,2,3,4,5)

```

```

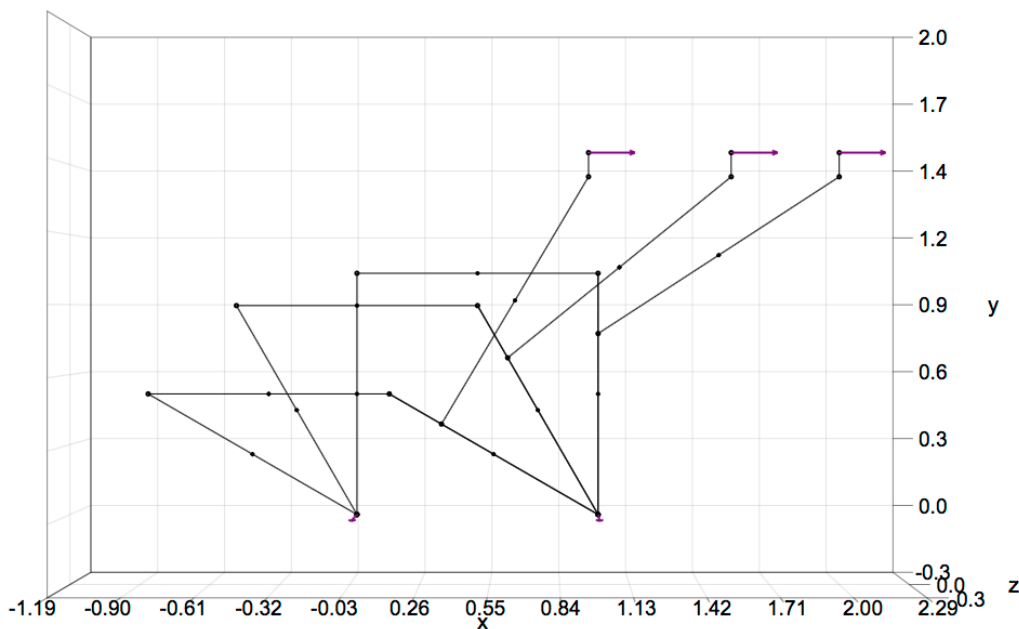
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0, pi/4, length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSRSSR.html', animate=TRUE, animate.reverse=TRUE)

```

## 6.14 Slider, 3D 4-bar in series (RSSR(SSL))



Sliding link in series with a 4-bar linkage (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,1,0), c(1,1,0), c(1,0,0),
  c(1,0.75,0), c(2,1.4,0), c(2,1.5,0))

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1), NA, NA, c(1,0,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(3,4), c(4,5), c(5,0))

# Define points associated with links
link.points <- rbind(c(0,0.5,0), c(0.5,1,0), c(1,0.5,0), c(1.5,1.075,0))

# Define links with which points are associated

```

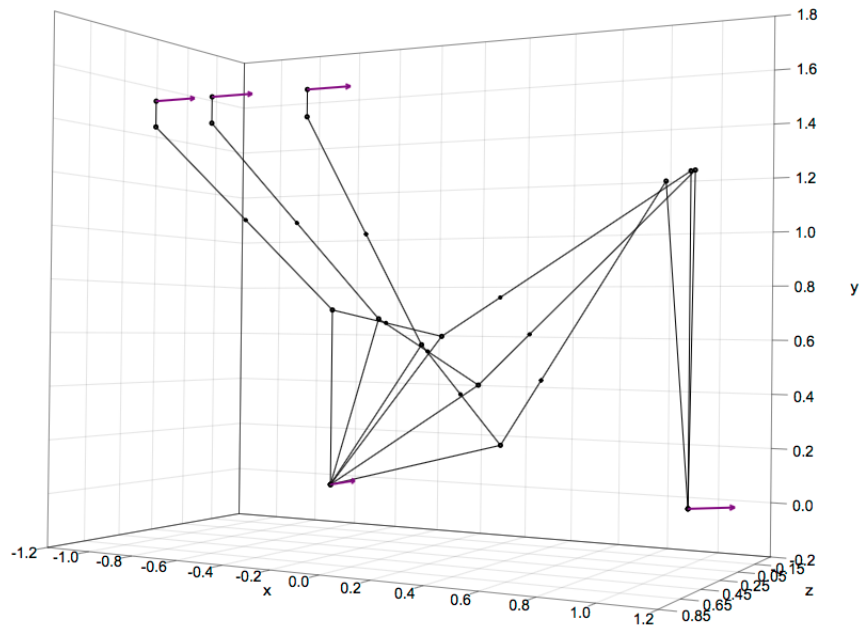
```
link.assoc <- c(1,2,3,4)

# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0, pi/3, length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RSSR(SSL).html', animate.reverse=TRUE)
```

## 6.15 Slider, 2D 4-bar in series (R(SSL)SSR)



Sliding link in series with a 4-bar linkage (DOF=1). Three iterations shown.

```
# Define joint coordinates
joint.coor <- rbind(c(-0.5,0,0), c(0,0.6,0), c(1,1.2,0), c(1,0,0),
  c(-0.5,0.7,0), c(-1,1.4,0.5), c(-1,1.5,0.5))

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(0,0,-1), NA, NA, c(1,0,-1), NA, NA, c(1,0,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(1,4), c(4,5), c(5,0))

# Define points associated with links
link.points <- rbind(c(-0.25,0.65,0), c(0.25,0.75,0), c(-0.75,1.05,0.25))
```

```

# Define links with which points are associated
link.assoc <- c(1,2,4)

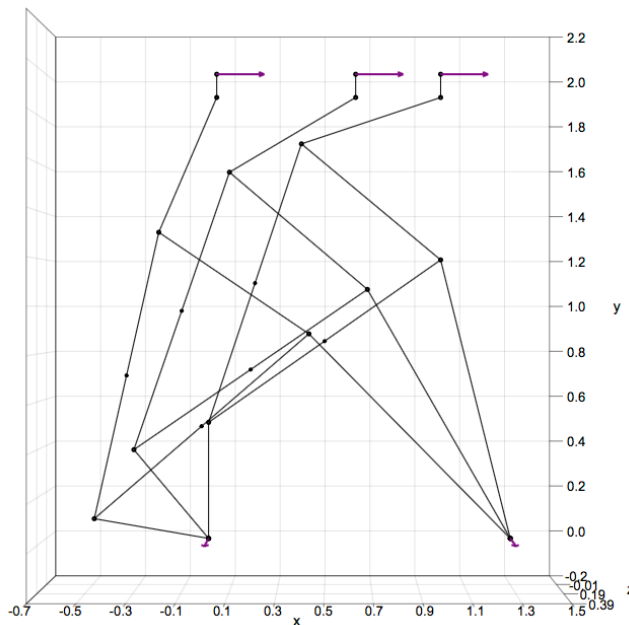
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,pi/5,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='R(SSL)SSR.html', animate.reverse=TRUE)

```

## 6.16 Slider in parallel with 4-bar (RS(SSL)SR)



Sliding link attached to coupler link of 4-bar linkage (DOF=1). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(0,0,0), c(0,0.5,0), c(1,1.2,0), c(1.3,0,0),
  c(0.4,1.7,0), c(1,1.9,0), c(1,2,0))

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "L")

# Define joint constraints
joint.cons <- list(c(0,0,1), NA, NA, c(0,0,1), NA, NA, c(1,0,0))

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(2,3), c(3,0), c(2,4), c(4,5), c(5,0))

```

```

# Define points associated with links
link.points <- rbind(c(0.2,1.1,0), c(0.5,0.85,0))

# Define links with which points are associated
link.assoc <- c(2,2)

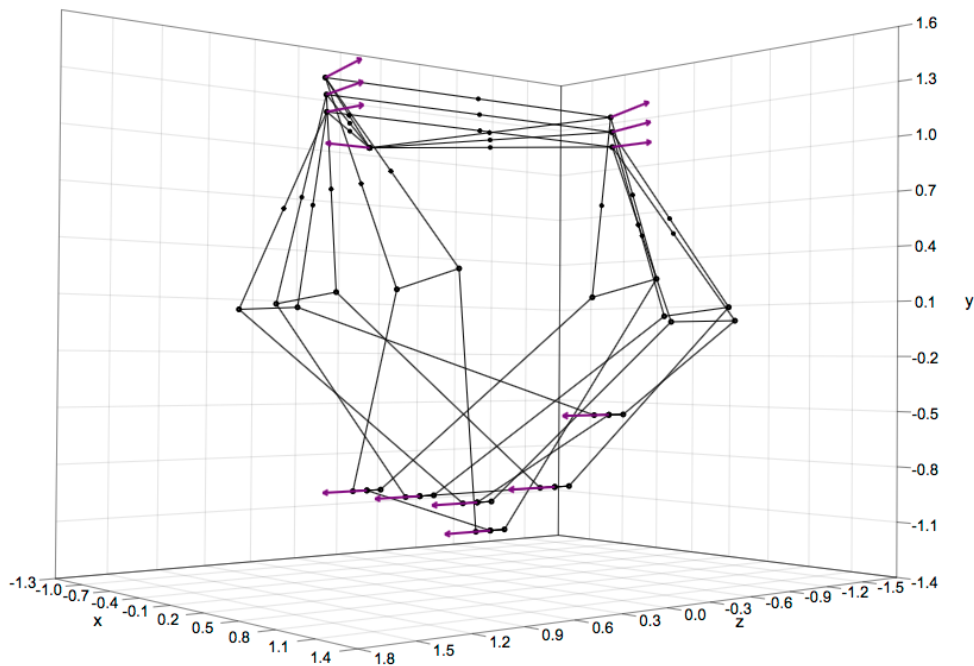
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc)

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(0,1.4,length=50), input.joint=1)

# Draw linkage
drawLinkage(anim, file='RS(SSL)SR.html', animate.reverse=TRUE)

```

## 6.17 Fish cranial joint configuration



A linkage based on the joint configuration (simplified) of a salmon skull (DOF=3). Three iterations shown.

```

# Define joint coordinates
joint.coor <- rbind(c(-1,1,0), c(0,1,1), c(0,1,-1), c(0,0,1.5), c(0,-1,0.1),
  c(0,-1,0), c(0,-1,-0.1), c(0,0,-1.5), c(0.5,0,1.5), c(1,-0.5,0.1), c(1,-0.5,0),
  c(1,-0.5,-0.1), c(0.5,0,-1.5), c(0,-1,0), c(1,-0.5,0))

# Define joint types
joint.types <- c("R", "R", "R", "S", "S", "P", "S", "S", "S", "S", "P", "S", "S", "S", "S")

# Define joint constraints

```

```

joint.cons <- list(c(0,0,1), c(1,0,0), c(1,0,0), NA, NA, c(0,0,1), NA, NA, NA, NA,
  c(0,0,1), NA, NA, NA, NA)

# Define two links connected by each joint
joint.conn <- rbind(c(0,1), c(1,2), c(1,3), c(2,4), c(4,5), c(5,0), c(5,6),
  c(6,3), c(2,7), c(7,9), c(9,0), c(9,8), c(8,3), c(5,10), c(10,9))

# Define points associated with links
link.points <- rbind(c(-0.5,1,0.5), c(0,1,0), c(-0.5,1,-0.5),
  c(0,0.5,1.25), c(0.25,0.5,1.25), c(0,0.5,-1.25), c(0.25,0.5,-1.25))

# Define links with which points are associated
link.assoc <- c(rep(1,3),2,2,3,3)

# Define input parameters
n_iter <- 50
input.param <- list(seq(0*(pi/180), 10*(pi/180), length=n_iter),
  matrix(c(-1,0,0), nrow=n_iter, ncol=3, byrow=TRUE)*matrix(seq(0, 1, length=n_iter),
    nrow=n_iter, ncol=3))

# Define input joint(s)
input.joint <- c(1,6)

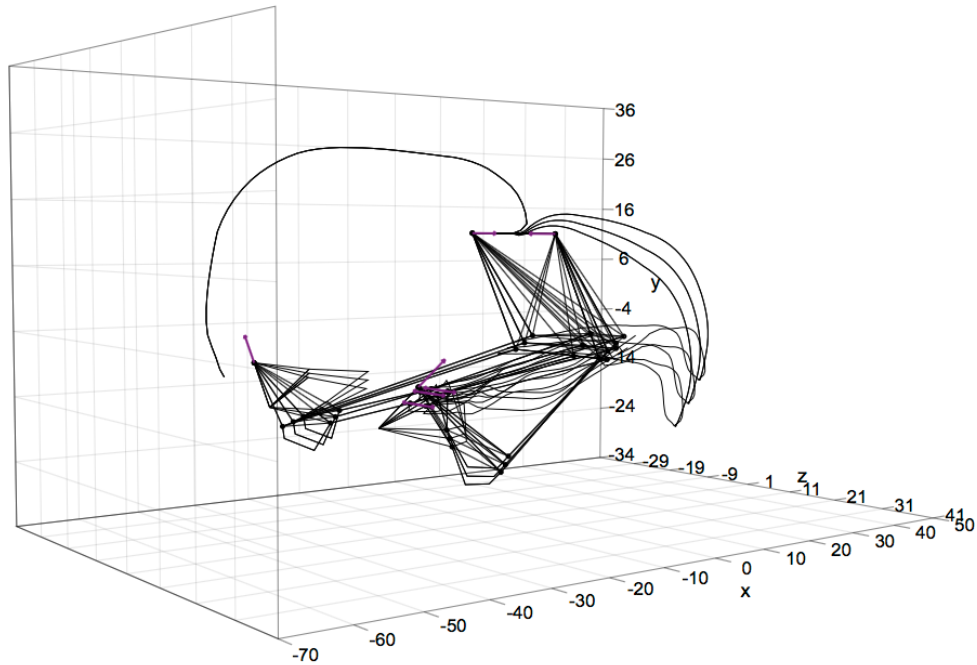
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, joint.conn=joint.conn, link.points=link.points,
  link.assoc=link.assoc)

# Animate linkage
anim <- animateLinkage(linkage, input.param=input.param, input.joint=input.joint)

# Draw linkage
drawLinkage(anim, file='RRRSSPSSSSPSSSS.html', animate.reverse=TRUE)

```

## 6.18 Bird cranial linkage



The kinetic bones of a Great Horned Owl skull (*Bubo virginianus*) modeled as a linkage mechanism (DOF=1). Three iterations shown.

```
# Get specimen data
owl <- linkR_data('owl')

# Copy landmarks for easy reference
lms <- owl$landmarks

# Define joint coordinates
joint.coor <- lms[c('nc_qd_l_R', 'ju_qd_R', 'ju_ub_R', 'nc_ub_R', 'pt_qd_R',
  'pa_pt_R', 'pa_pt_R', 'pa_pt_R', 'pa_ub_R', 'nc_qd_l_L', 'ju_qd_L', 'ju_ub_L',
  'nc_ub_L', 'pt_qd_L', 'pa_pt_L', 'pa_pt_L', 'pa_pt_L', 'pa_ub_L'), ]

# Define joint types
joint.types <- c("R", "S", "S", "R", "S", "S", "P", "S", "S",
  "R", "S", "S", "R", "S", "S", "P", "S", "S")

# Define joint constraints
joint.cons <- list(
  lms['nc_qd_l_R', ]-lms['nc_qd_m_R', ], NA, NA, lms['nc_ub_L', ]-lms['nc_ub_R', ],
  NA, NA, lms['nc_ub_L', ]-lms['nc_ub_R', ], NA, NA,
  lms['nc_qd_l_L', ]-lms['nc_qd_m_L', ], NA, NA, lms['nc_ub_R', ]-lms['nc_ub_L', ],
  NA, NA, lms['nc_ub_R', ]-lms['nc_ub_L', ], NA, NA)

# Define two links connected by each joint
joint.conn <- rbind(
  c('neurocranium', 'quadrate_R'), c('quadrate_R', 'jugal_R'),
  c('jugal_R', 'upperbeak'), c('upperbeak', 'neurocranium'),
  c('quadrate_R', 'pterygoid_R'), c('pterygoid_R', 'pp-slide_R'),
```



```

c('pp-slide_R', 'neurocranium'), c('pp-slide_R', 'palatine_R'),
c('palatine_R', 'upperbeak'), c('neurocranium', 'quadrate_L'),
c('quadrate_L', 'jugal_L'), c('jugal_L', 'upperbeak'),
c('upperbeak', 'neurocranium'), c('quadrate_L', 'pterygoid_L'),
c('pterygoid_L', 'pp-slide_L'), c('pp-slide_L', 'neurocranium'),
c('pp-slide_L', 'palatine_L'), c('palatine_L', 'upperbeak')
)

# Define points associated with links
link.points <- owl$landmarks

# Define links with which points are associated
link.assoc <- owl$lm.assoc

# Define lines connecting associated points
path.connect <- owl$path.connect

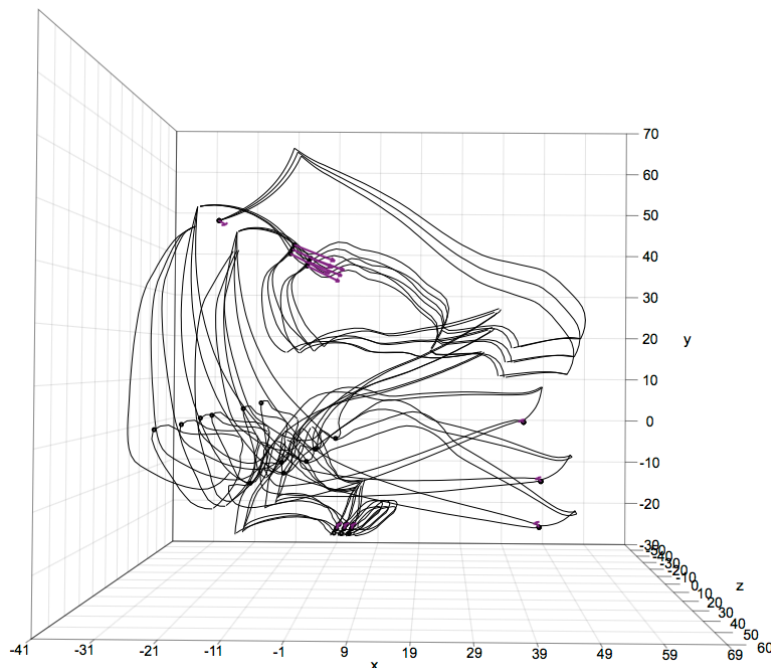
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, link.points=link.points, link.assoc=link.assoc,
  joint.conn=joint.conn, path.connect=path.connect, ground.link='neurocranium')

# Animate linkage
anim <- animateLinkage(linkage, input.param=seq(-0.07,0.17,length=30), input.joint=1)

# Draw linkage
drawLinkage(anim, file='owl.html', animate.reverse=TRUE)

```

## 6.19 Fish cranial linkage



The kinetic bones of an Atlantic salmon skull (*Salmo salar*) modeled as a linkage mechanism

(DOF=3). Three iterations shown.

```
# Get specimen data
salmon <- linkR_data('salmon')

# Copy landmarks for easy reference
lms <- salmon$landmarks

# Define joint coordinates
joint.coor <- lms[c('nc_vc', 'nc_su_a_L', 'pc_su_L', 'ac_hy_L', 'hy_mid',
  'ac_hy_R', 'pc_su_R', 'nc_su_a_R', 'hy_mid', 'lj_sy_inf', 'lj_sy_inf',
  'lj_qd_L', 'lj_sy_inf', 'lj_sy_inf', 'lj_qd_R'), ]

# Define joint types
joint.types <- c("R","R","S","S","P","S","S","R","S","S","P","S","S","S","S")

# Define joint constraints
joint.cons <- list(
  lms['nc_su_p_L', ] - lms['nc_su_p_R', ], lms['nc_su_a_L', ] - lms['nc_su_p_L', ],
  NA, NA, lms['nc_su_a_L', ] - lms['nc_su_a_R', ], NA, NA,
  lms['nc_su_a_R', ] - lms['nc_su_p_R', ], NA, NA,
  lms['nc_su_a_L', ] - lms['nc_su_a_R', ], NA, NA, NA, NA)

# Define two links connected by each joint
joint.conn <- rbind(
  c('vert_column', 'neurocranium'), c('neurocranium', 'suspensorium_L'),
  c('suspensorium_L', 'hyoid_L'), c('hyoid_L', 'hypohyal'),
  c('hypohyal', 'vert_column'), c('hypohyal', 'hyoid_R'),
  c('hyoid_R', 'suspensorium_R'), c('suspensorium_R', 'neurocranium'),
  c('hypohyal', 'hyoid_lowerjaw'), c('hyoid_lowerjaw', 'lowerjaw_symph'),
  c('lowerjaw_symph', 'vert_column'), c('suspensorium_L', 'lowerjaw_L'),
  c('lowerjaw_L', 'lowerjaw_symph'), c('lowerjaw_symph', 'lowerjaw_R'),
  c('lowerjaw_R', 'suspensorium_R'))

# Set long axis rotation constraints
lar.cons <- list(
  list('link'='lowerjaw_L', 'type'='P', 'point'=lms['lj_sy_sup', ],
    'vec'=lms['nc_su_p_L', ] - lms['nc_su_p_R', ]),
  list('link'='lowerjaw_R', 'type'='P', 'point'=lms['lj_sy_sup', ],
    'vec'=lms['nc_su_p_L', ] - lms['nc_su_p_R', ]),
  list('link'='hyoid_R', 'type'='P', 'point'=lms['ac_as_R', ],
    'vec'=lms['nc_su_p_L', ] - lms['nc_su_p_R', ]),
  list('link'='hyoid_L', 'type'='P', 'point'=lms['ac_as_L', ],
    'vec'=lms['nc_su_p_L', ] - lms['nc_su_p_R', ]))

# Define points associated with links
link.points <- salmon$landmarks

# Define links with which points are associated
link.assoc <- salmon$lm.assoc

# Define lines connecting associated points
path.connect <- salmon$path.connect
```

```
# Define linkage
linkage <- defineLinkage(joint.coor=joint.coor, joint.types=joint.types,
  joint.cons=joint.cons, link.points=link.points, link.assoc=link.assoc,
  joint.conn=joint.conn, path.connect=path.connect, ground.link='vert_column',
  lar.cons=lar.cons)

# Set number of animation iterations
anim_len <- 9

# Set input parameters
input.param <- list(seq(0,-0.1,length=anim_len),
  cbind(seq(0.001,-3.001,length=anim_len), rep(0, anim_len), rep(0, anim_len)))

# Animate linkage (this can take some time to run)
anim <- animateLinkage(linkage, input.param=input.param, input.joint=c(1,5))

# Draw linkage
drawLinkage(anim, file='salmon.html', animate.reverse=TRUE)
```

## 7 Citing linkR

The linkR R package are the result of several years of work. I am pleased to offer the software open-source and free of charge and hope that users find it useful and that it brings about interesting, fruitful and fun advances for biologists and non-biologists alike. I only ask that if you use linkR and share your results that you include a citation. For peer-reviewed publications, please cite the [following article](#):

Olsen, A.M. and Westneat, M.W. (2016). Linkage mechanisms in the vertebrate skull: Structure and function of three-dimensional, parallel transmission systems. *Journal of Morphology*. Early View. DOI: 10.1002/jmor.20596.

## 8 Acknowledgements

I would like to thank Mark Westneat, whose previous work modeling cranial kinesis in fishes and birds using linkage mechanisms inspired the development of the linkR package. linkR was developed as we worked together applying three-dimensional linkage models to the kinetic mechanisms of fish and bird skulls. The software benefitted greatly from these collaborations and will continue to benefit from ongoing and future collaborations. I would also like to thank Michael Coates, whose invitation to collaborate in modeling early chondrichthyan jaw linkages using the first version of linkR instigated the application of linkR to modeling buccal expansion in fishes. Thank you to Dave Willard, Ben Marks and Mary Hennen in the Bird Division at the Field Museum of Natural History for accommodating access to fresh and skeletal specimens. The development of linkR was made possible by funding from the US National Science Foundation (DGE-1144082; DGE-0903637; DBI-1612230).