# skijumpdesign Documentation

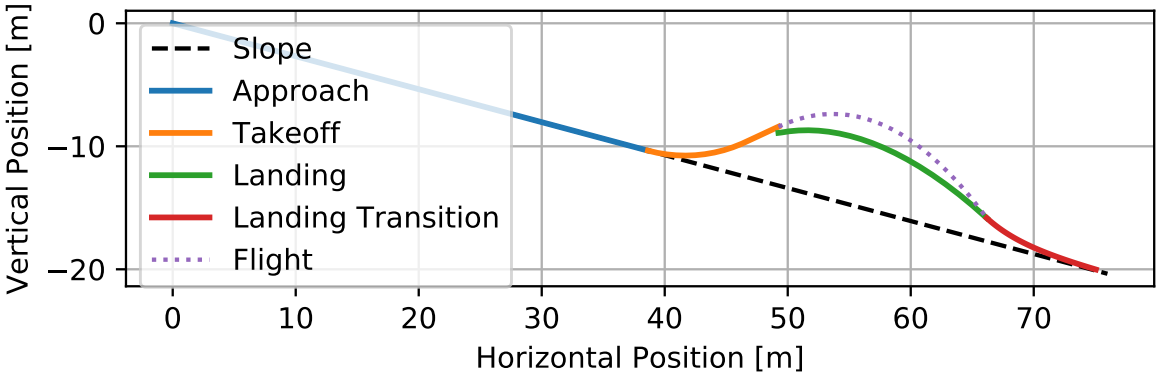
***Release 1.2.0***

**Jason K. Moore, Mont Hubbard**

**May 21, 2018**

# Contents:

This is the documentation for "skijumpdesign: A Ski Jump Design Tool for Equivalent Fall Height" based on the work presented in[1]. The software includes a library for two dimensional skiing simulations and a graphical web application for designing basic ski jumps.




[1] Levy, Dean, Mont Hubbard, James A. McNeil, and Andrew Swedberg. "A Design Rationale for Safer Terrain Park Jumps That Limit Equivalent Fall Height." Sports Engineering 18, no. 4 (December 2015): 227–39. https://doi.org/10.1007/s12283-015-0182-6.

CHAPTER 1

# Installation

## 1.1 conda

The library and web application can be installed into the root conda environment from the Conda Forge channel at Anaconda.org:

```
$ conda install -c conda-forge skijumpdesign
```

## 1.2 pip

The library and web application can be installed from PyPi using pip[1]:

```
$ pip install skijumpdesign
```

If you want to run the unit tests and/or build the documentation use:

```
$ pip install skijumpdesign[dev]
```

## 1.3 setuptools

Download and unpack the source code to a local directory, e.g. `/path/to/skijumpdesign`.

Open a terminal. Navigate to the `skijumpdesign` directory:

```
$ cd /path/to/skijumpdesign
```

Install with[1]:

[1] Note that you likely want to install into a user directory with pip/setuptools. See the pip and setuptools documentation on how to do this.

```
$ python setup.py install
```

## 1.4 Optional dependencies

If pycvodes is installed it will be used to speed up the flight simulation and the landing surface calculation significantly. This library is not trivial to install on all operating systems, so you will need to refer its documentation for installation instructions. If you are using conda and 64 bit Linux, this package can be installed using:

```
$ conda install -c conda-forge -c bjodah pycvodes
```

## 1.5 Development Installation

Clone the repository with git:

```
git clone https://gitlab.com/moorepants/skijumpdesign
```

Navigate to the cloned `skijumpdesign` repository:

```
$ cd skijumpdesign/
```

Setup the custom development conda environment named `skijumpdesign` to ensure it has all of the correct software dependencies. To create the environment type:

```
$ conda env create -f environment-dev.yml
```

To activate the environment type[2]:

```
$ conda activate skijumpdesign-dev
(skijumpdesign-dev)$
```

## 1.6 Heroku Installation

When installing into a Heroku instance, the application will make use of the `requirements.txt` file included in the source code which installs all of the dependencies needed to run the software on a live Heroku instance.

[2] This environment will also show up in the Anaconda Navigator program.

CHAPTER 2

# Running the Web Application

## 2.1 User

After *installing* skijumpdesign type:

```
$ skijumpdesign
```

to run the web application. This should launch the application on your computer's port 8050. You can view the application by visiting `localhost:8050` in your preferred web browser. `<CTRL + C>` will stop the server.

## 2.2 Developer

The `bin/skijumpdesign` entry point is not available unless you install the software. The following shows how to launch the app from the Python file.

### 2.2.1 In a terminal

Navigate to the `skijumpdesign` directory on your computer:

```
$ cd /path/to/skijumpdesign
```

Activate the custom Conda environment with:

```
$ conda activate skijumpdesign-dev
```

Now run the application with:

```
(skijumpdesign-dev)$ python skijumpdesignapp.py
```

You should see something like:

```
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Open your web browser and enter the displayed URL to interact with the web app. Type `<CTRL>+C` in the terminal to shutdown the web server.

### 2.2.2 In Spyder

Open Anaconda Navigator, switch to the `skijumpdesign-dev` environment, and then launch Spyder. Set the working directory to the `/path/to/skijumpdesign` directory. In the Spyder IPython console execute:

```
In [1]: run skijumpdesignapp.py
```

If successful, you will see something like:

```
* Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Open your web browser and enter the displayed URL to interact with the web app.

To shutdown the web app, close the tab in your web browser. Go back to Spyder and execute `<CTRL>+C` to shutdown the web server.

# CHAPTER 3

# Example EFH Jump Design

The following page describes how to construct a typical equivalent fall height ski jump landing surface using the `skijumpdesign` API. Make sure to *install* the library first.

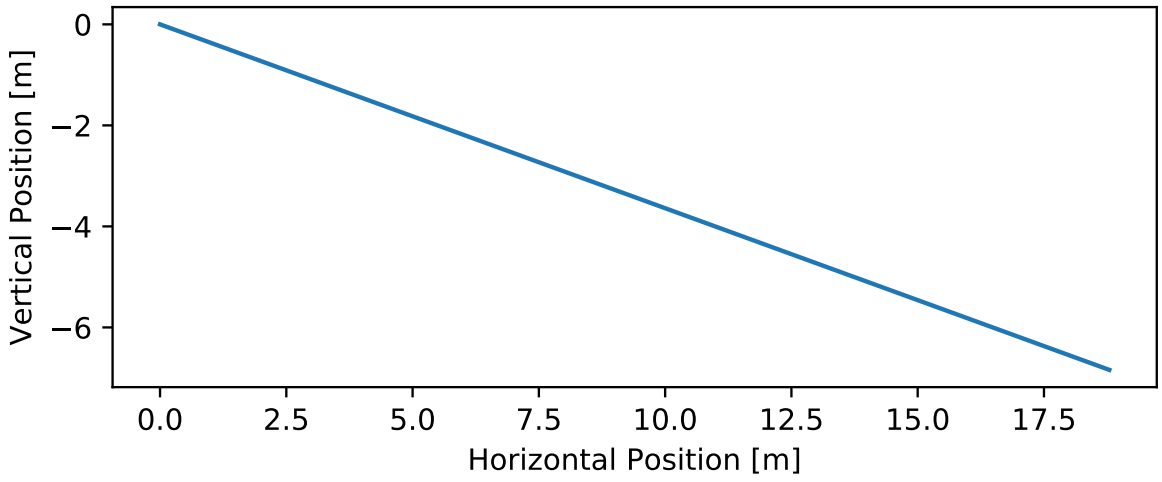
## 3.1 Approach

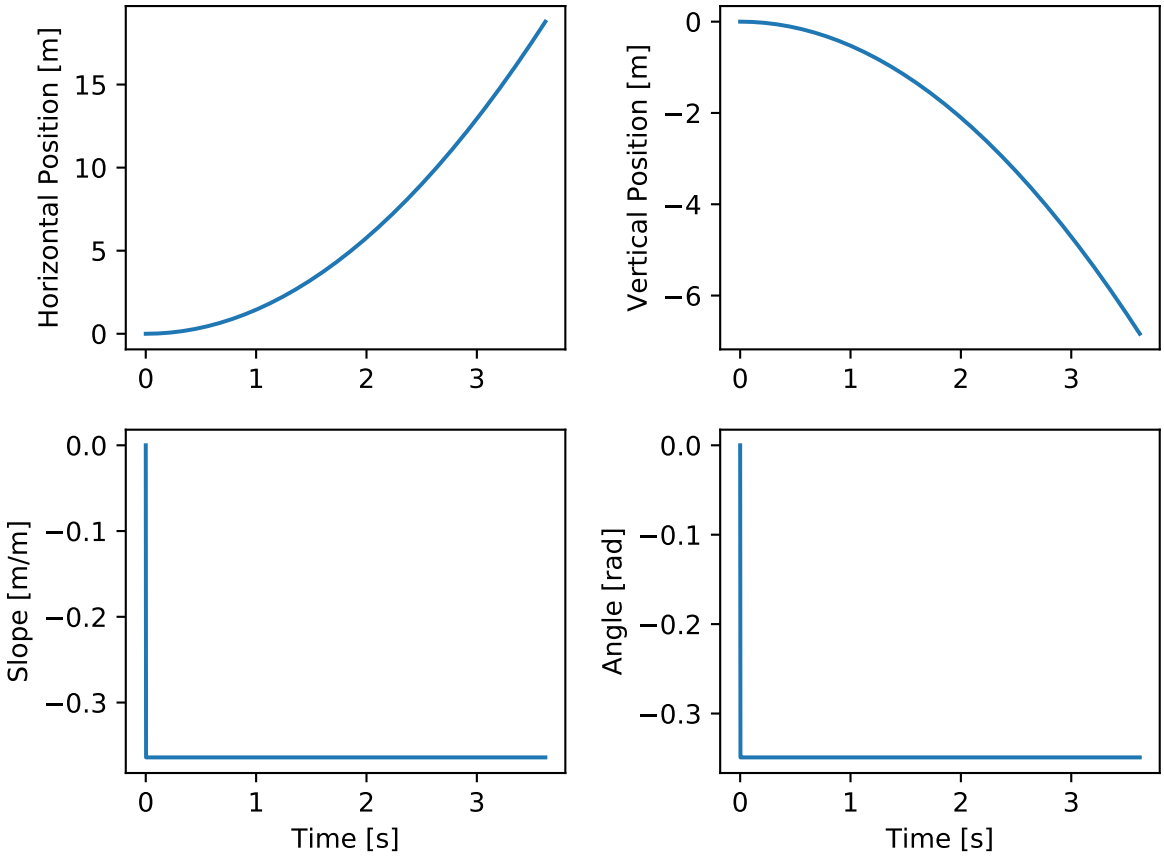
Start by creating a 20 meter length of an approach surface (also called the in-run) which is flat and has a downward slope angle of 20 degrees. The resulting surface can be visualized with the `FlatSurface.plot()` method.

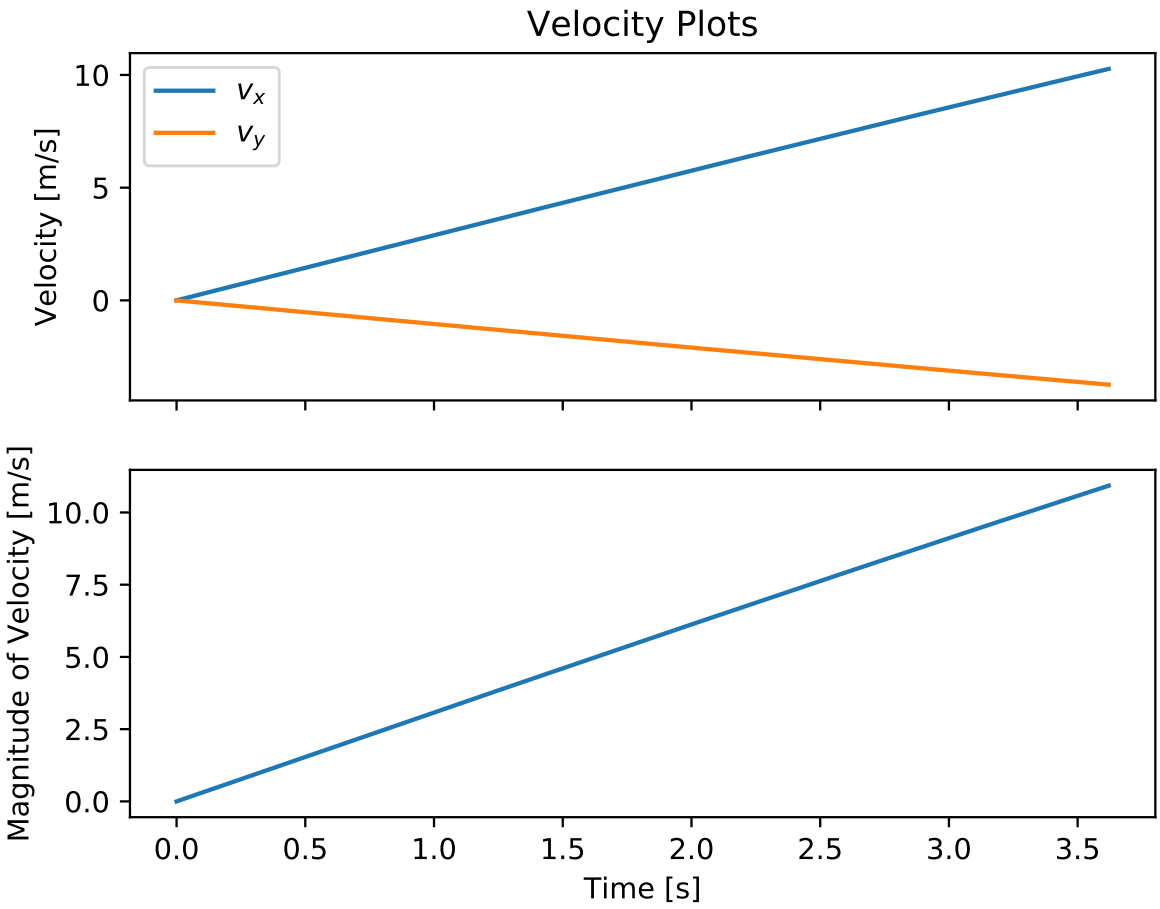
```
from skijumpdesign import FlatSurface

approach_ang = -np.deg2rad(20)  # radians
approach_len = 20.0  # meters

approach = FlatSurface(approach_ang, approach_len)

approach.plot()
```

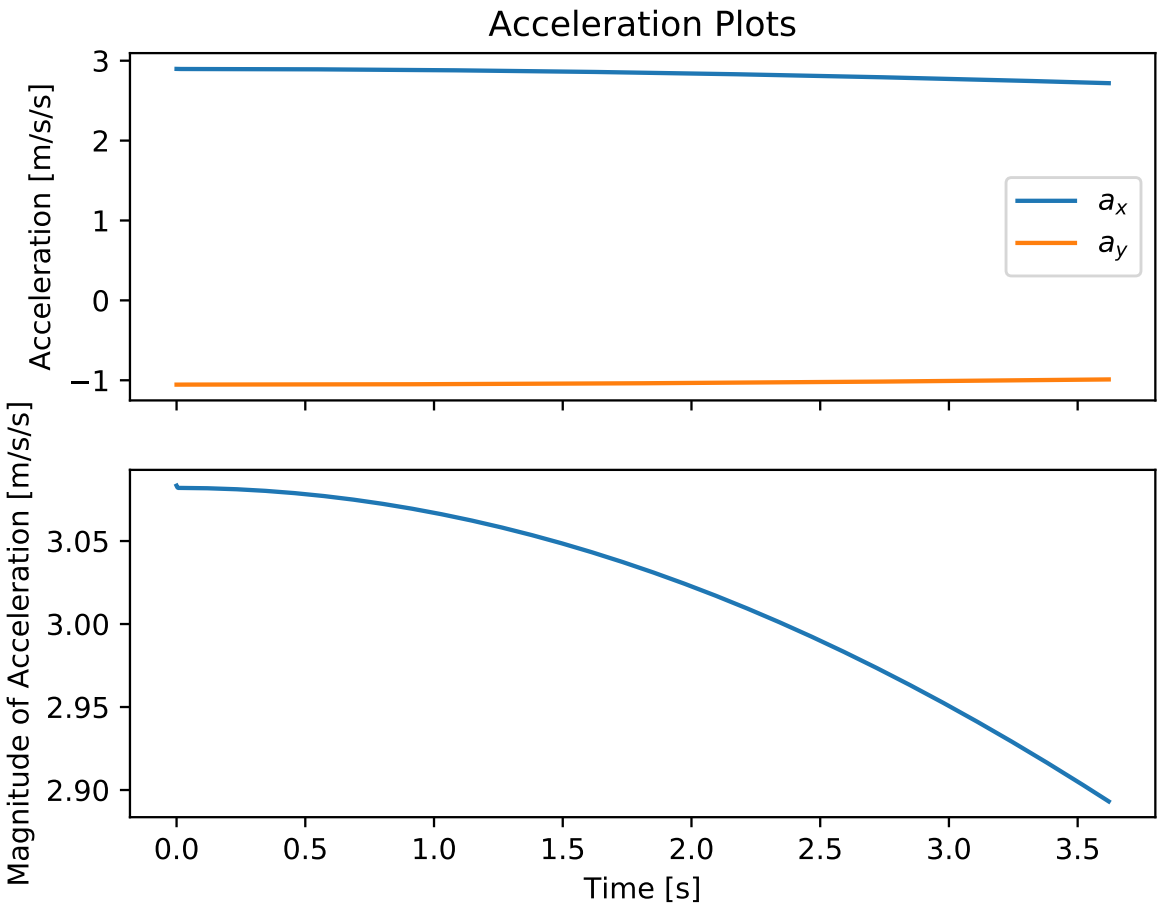
Now that a surface has been created, a skier can be created. The skier can "ski" along the approach surface using the `slide_on()` method which generates a skiing simulation trajectory.

```
from skijumpdesign import Skier

skier = Skier()

approach_traj = skier.slide_on(approach)

approach_traj.plot_time_series()
```


Vertical Position [m]
0
−2
−4
−6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Horizontal Position [m]


Horizontal Position [m]
15
10
5
0
0
1
2
3
Vertical Position [m]
0
−2
−4
−6
0
1
2
3
Slope [m/m]
0.0
−0.1
−0.2
−0.3
0
1
2
3
Time [s]
Angle [rad]
0.0
−0.1
−0.2
−0.3
0
1
2
3
Time [s]

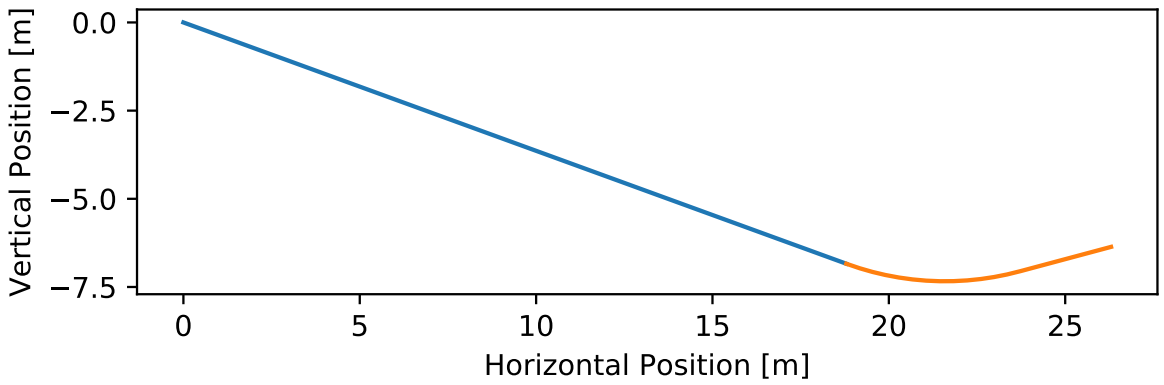

Velocity Plots
Velocity [m/s]
10
5
0
$v_x$
$v_y$
Magnitude of Velocity [m/s]
10.0
7.5
5.0
2.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Time [s]


Acceleration Plots
Acceleration [m/s/s]
3
2
1
0
−1
$a_x$
$a_y$
Magnitude of Acceleration [m/s/s]
3.05
3.00
2.95
2.90
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Time [s]

## 3.2 Takeoff

The takeoff ramp is constructed with a clothoid-circle-clothoid-flat surface to transition from the approach parent slope angle to the desired takeoff angle, in this case 15 degrees.

```
from skijumpdesign import TakeoffSurface

takeoff_entry_speed = skier.end_speed_on(approach)

takeoff_ang = np.deg2rad(15)

takeoff = TakeoffSurface(skier, approach_ang, takeoff_ang,
                         takeoff_entry_speed, init_pos=approach.end)

ax = approach.plot()
takeoff.plot(ax=ax)
```




The trajectory of the skier on the takeoff can be examined also.

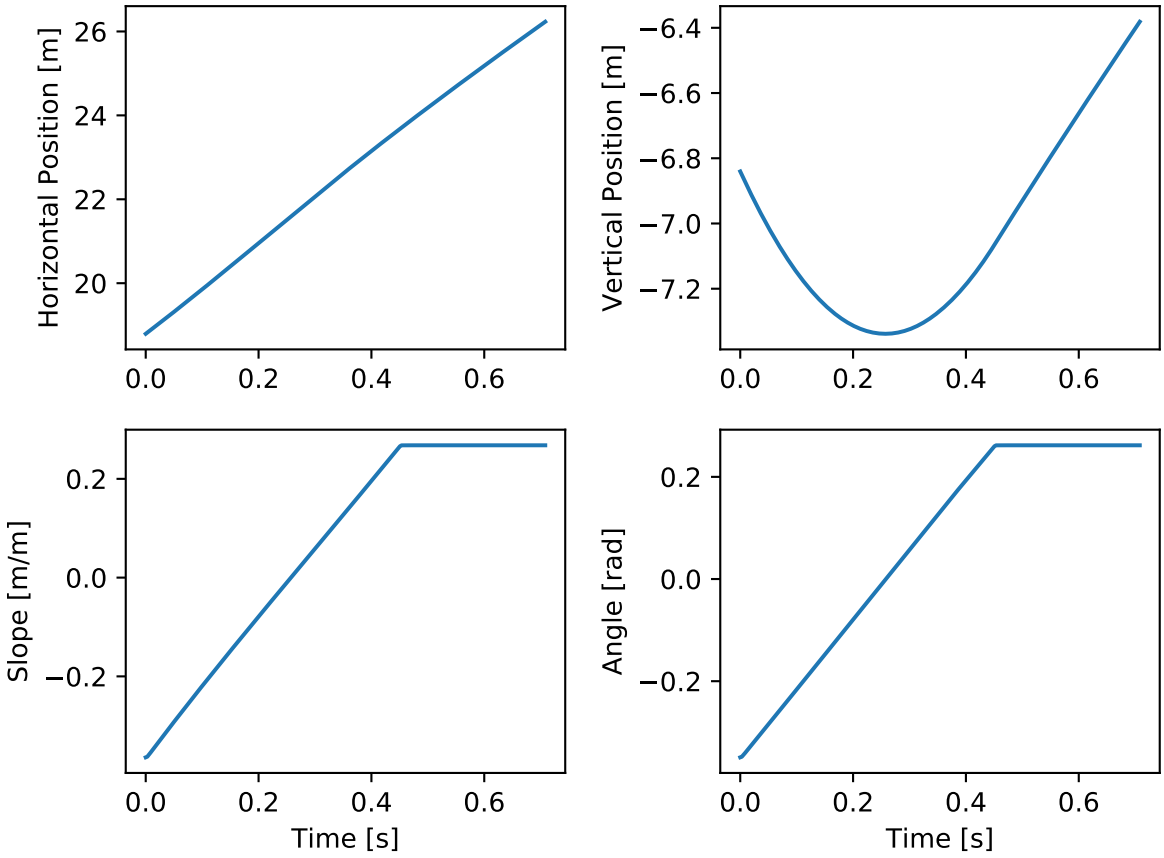
```
takeoff_traj = skier.slide_on(takeoff, takeoff_entry_speed)

takeoff_traj.plot_time_series()
```

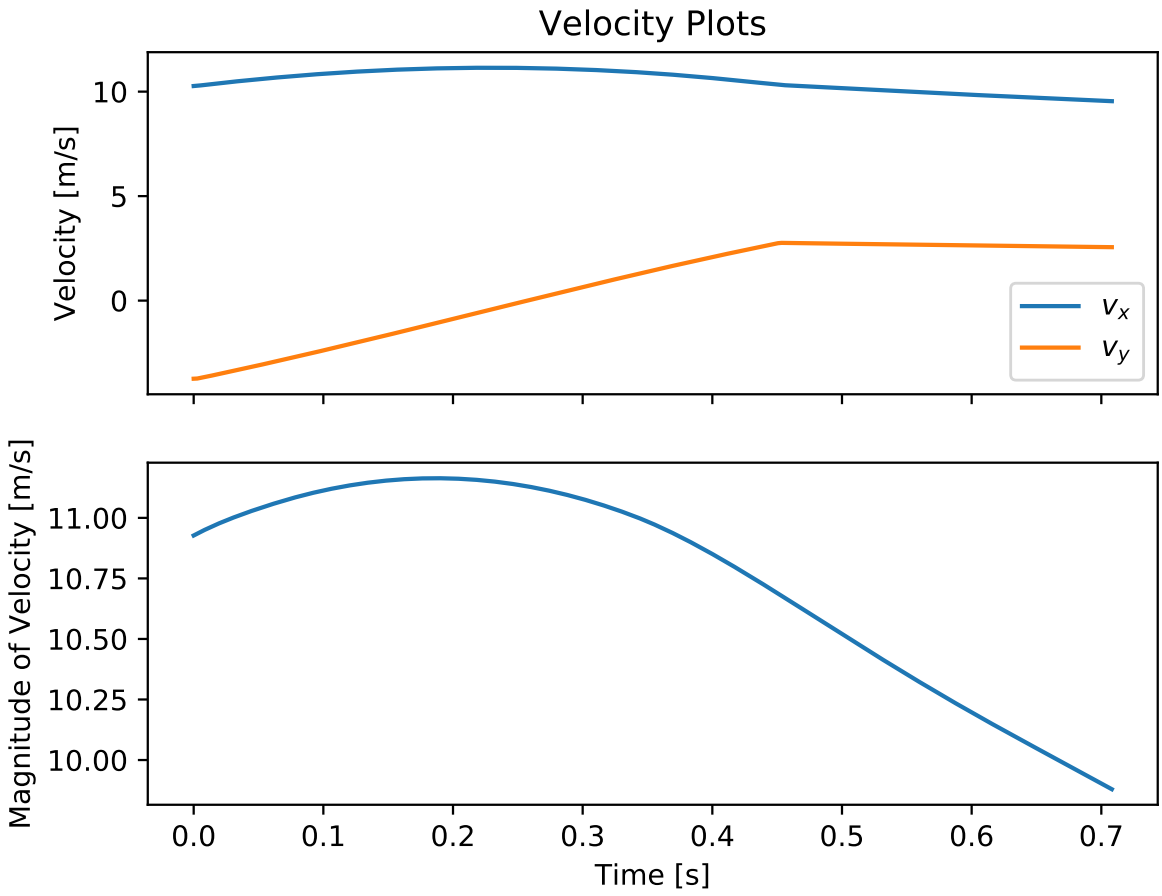
## 3.3 Flight

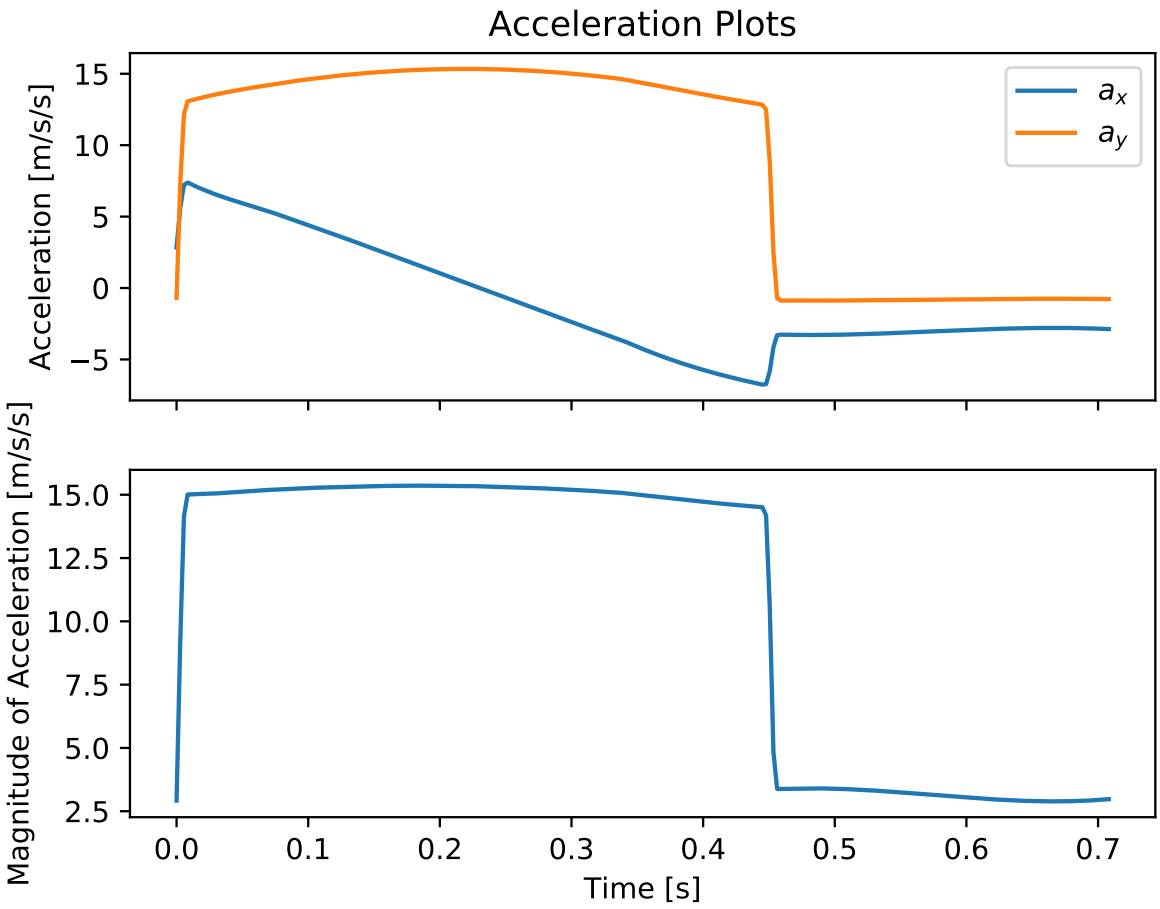
Once the skier leaves the takeoff ramp at the maximum (design) speed they will be in flight. The `Skier.fly_to()` method can be used to simulate this longest flight trajectory.

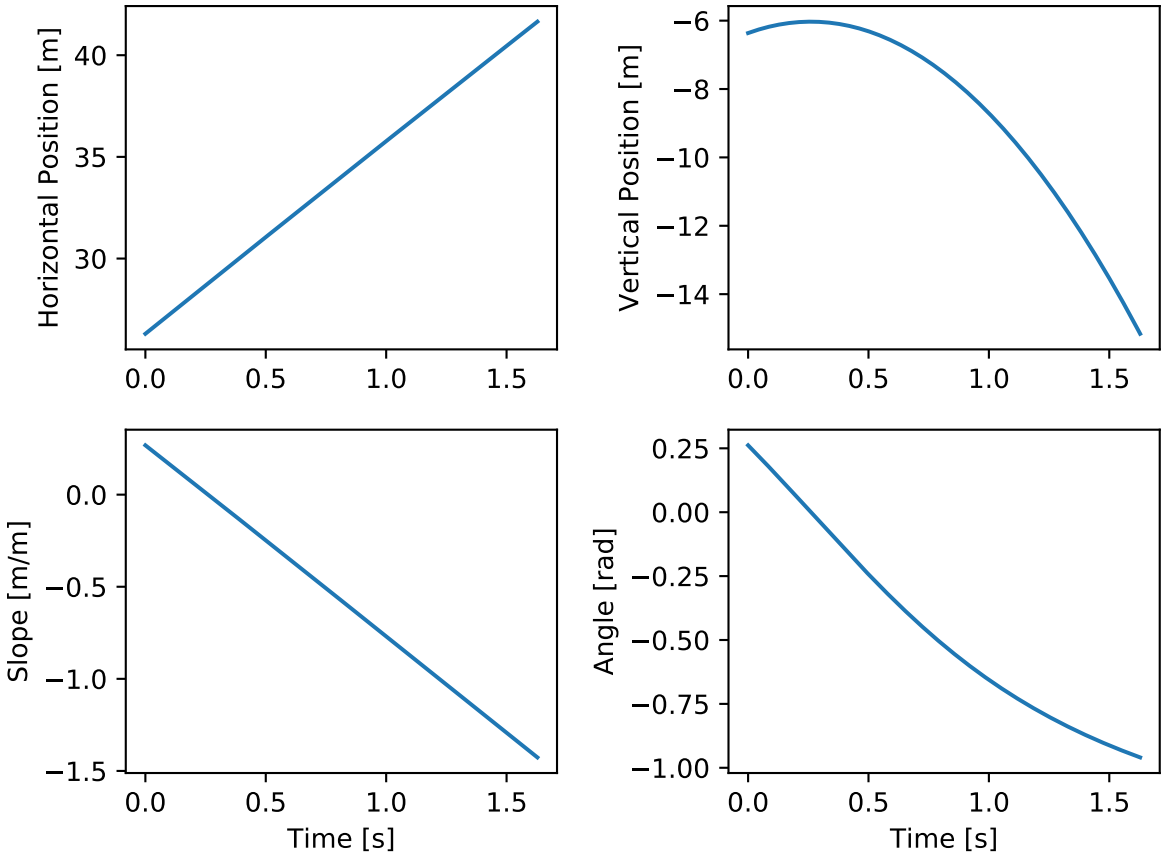
```
takeoff_vel = skier.end_vel_on(takeoff, init_speed=takeoff_entry_speed)

flight = skier.fly_to(approach, init_pos=takeoff.end,
                      init_vel=takeoff_vel)

flight.plot_time_series()
```

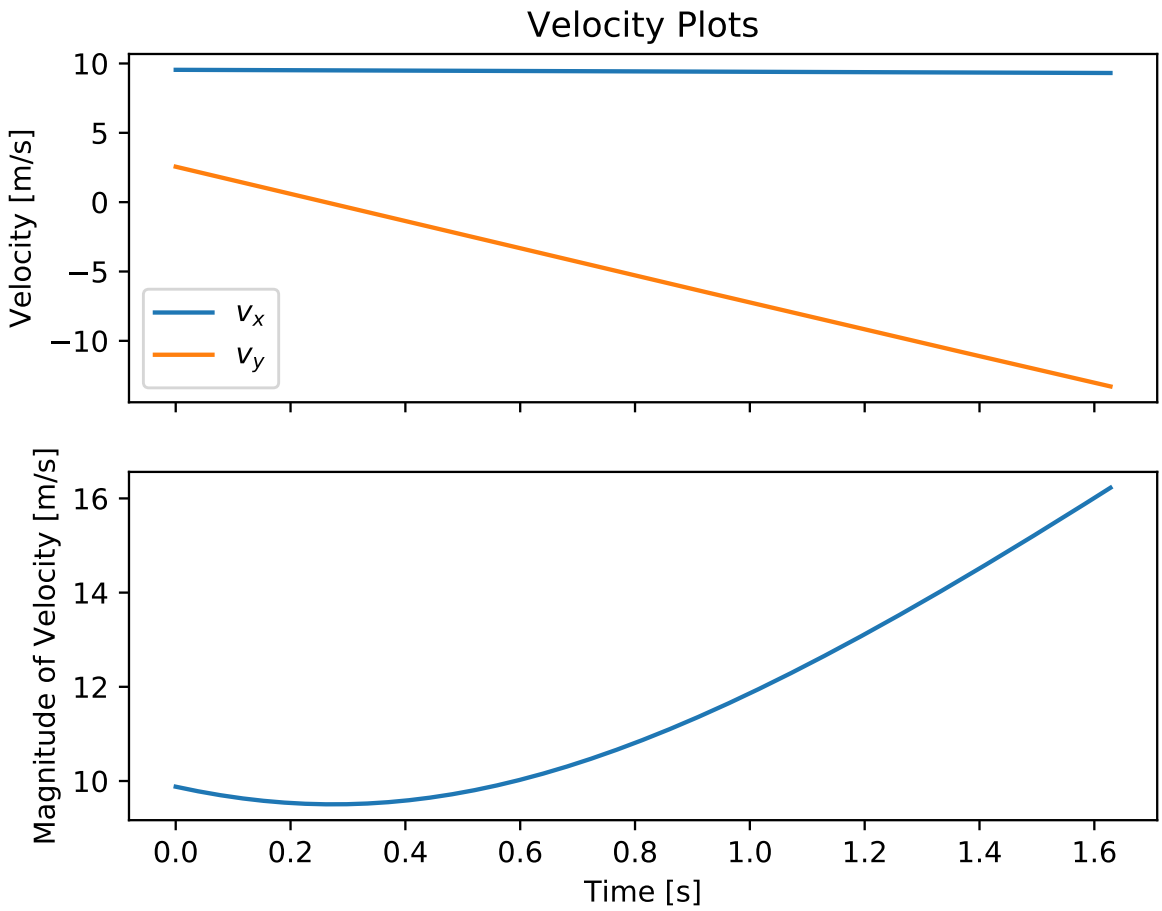
The design speed flight trajectory can be plotted alongside the surfaces.

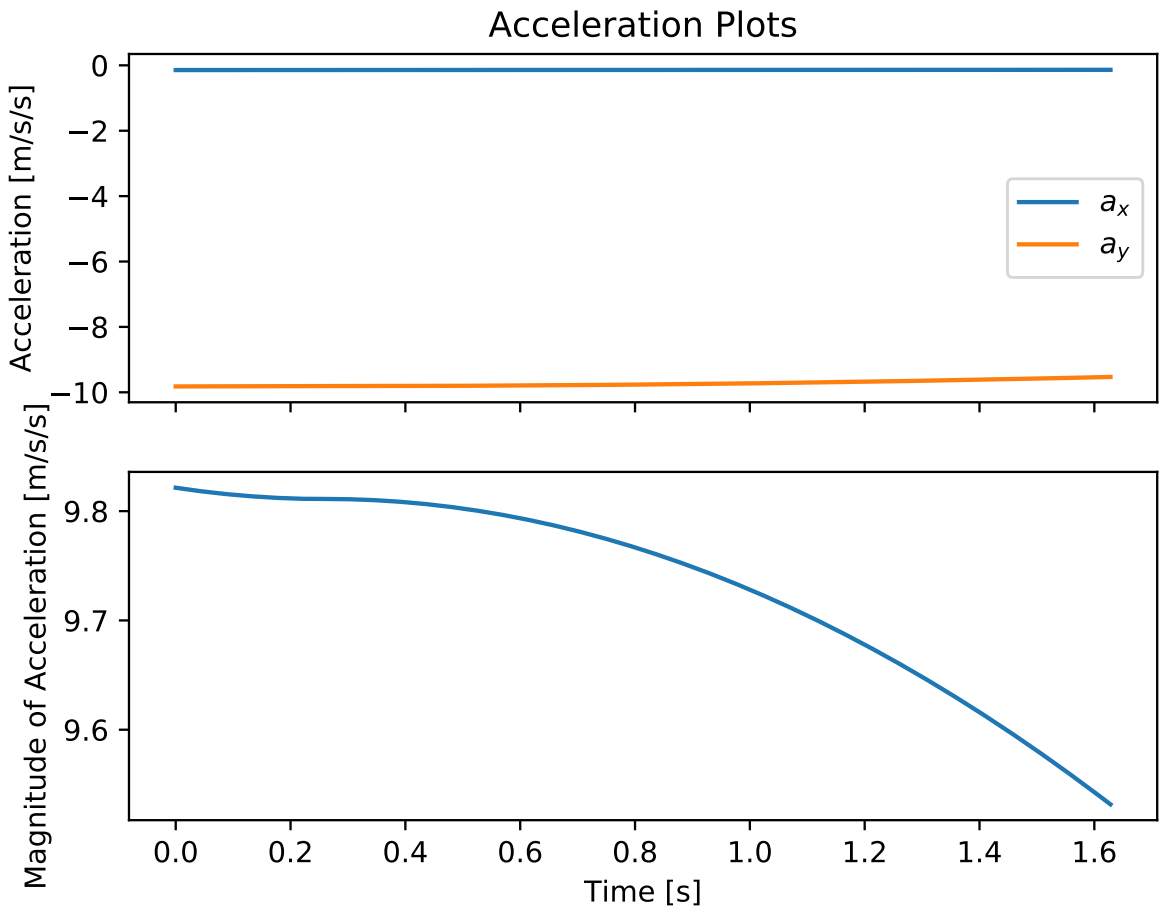

Horizontal Position [m]
26
24
22
20
0.0
0.2
0.4
0.6
Vertical Position [m]
−6.4
−6.6
−6.8
−7.0
−7.2
Slope [m/m]
0.2
0.0
−0.2
Time [s]
Angle [rad]


Velocity Plots
Velocity [m/s]
10
5
0
$v_x$
$v_y$
Magnitude of Velocity [m/s]
11.00
10.75
10.50
10.25
10.00
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time [s]

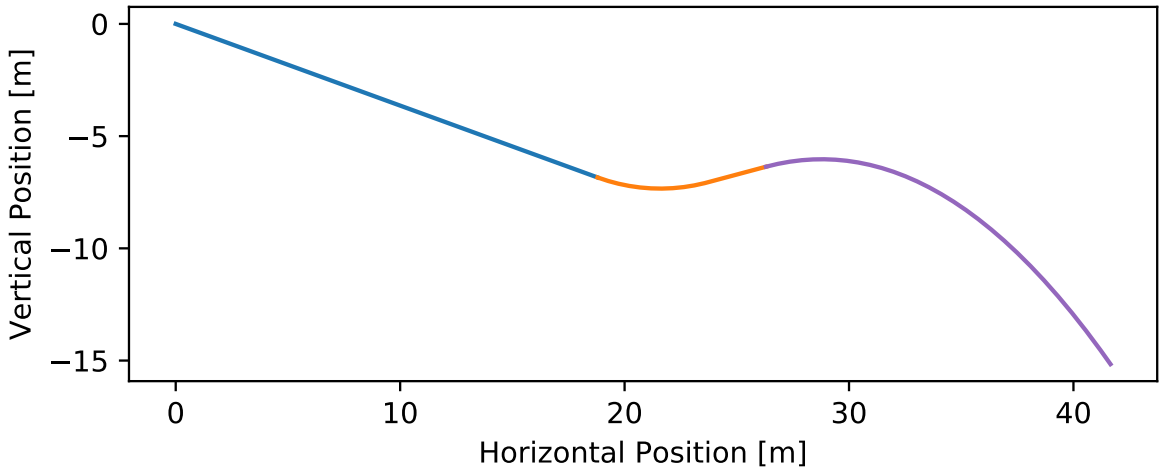

Acceleration Plots
Acceleration [m/s/s]
15
10
5
0
−5
$a_x$
$a_y$
Magnitude of Acceleration [m/s/s]
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time [s]


Horizontal Position [m]
40
35
30
0.0
0.5
1.0
1.5
Vertical Position [m]
−6
−8
−10
−12
−14
Slope [m/m]
0.0
−0.5
−1.0
−1.5
Time [s]
Angle [rad]
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Time [s]


Velocity Plots
Velocity [m/s]
10
5
0
−5
−10
$v_x$
$v_y$
Magnitude of Velocity [m/s]
16
14
12
10
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time [s]


Acceleration Plots
Acceleration [m/s/s]
0
−2
−4
−6
−8
−10
$a_x$
$a_y$
Magnitude of Acceleration [m/s/s]
9.8
9.7
9.6
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time [s]

```
ax = approach.plot()
ax = takeoff.plot(ax=ax)
flight.plot(ax=ax, color='#9467bd')
```




## 3.4 Landing Transition

There is a single infinity of landing surfaces that satisfy the efh differential equation and provide the desired equivalent fall height. The algorithm selects the one of these that is closest to the parent slope, and hence is least expensive to build, but which still is able to transition back to the parent slope with slope continuity and simultaneously is constrained to experience limited normal acceleration. The final part of this step is to determine the landing transition curve which connects the optimum (cheapest) constant efh landing surface to the parent slope.

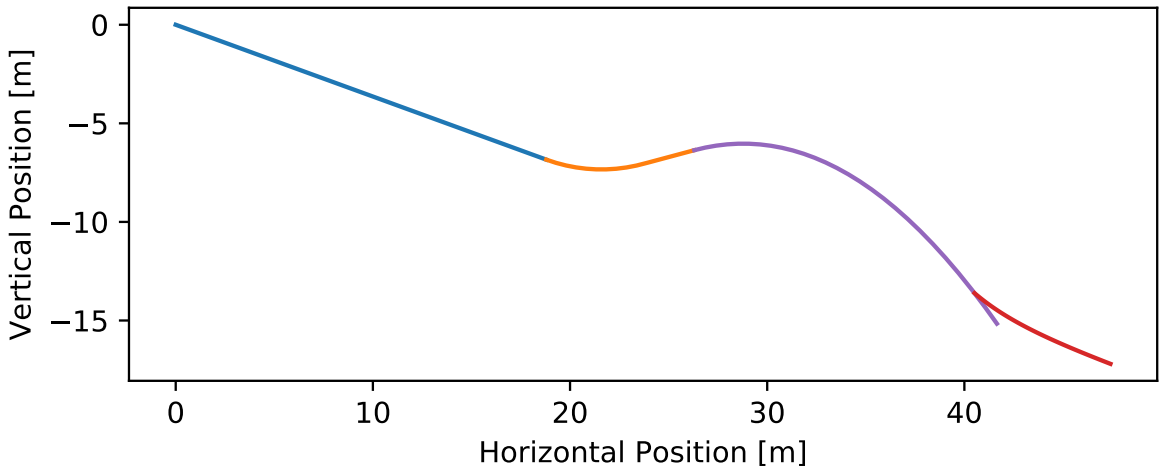
```
from skijumpdesign import LandingTransitionSurface

fall_height = 0.5

landing_trans = LandingTransitionSurface(approach,
    flight, fall_height, skier.tolerable_landing_acc)

ax = approach.plot()
ax = takeoff.plot(ax=ax)
ax = flight.plot(ax=ax, color='#9467bd')
landing_trans.plot(ax=ax, color='#d62728')
```

## 3.5 Landing

Finally, the equivalent fall height landing surface can be generated which accommodates all takeoff speeds below the maximum takeoff (design) speed above.

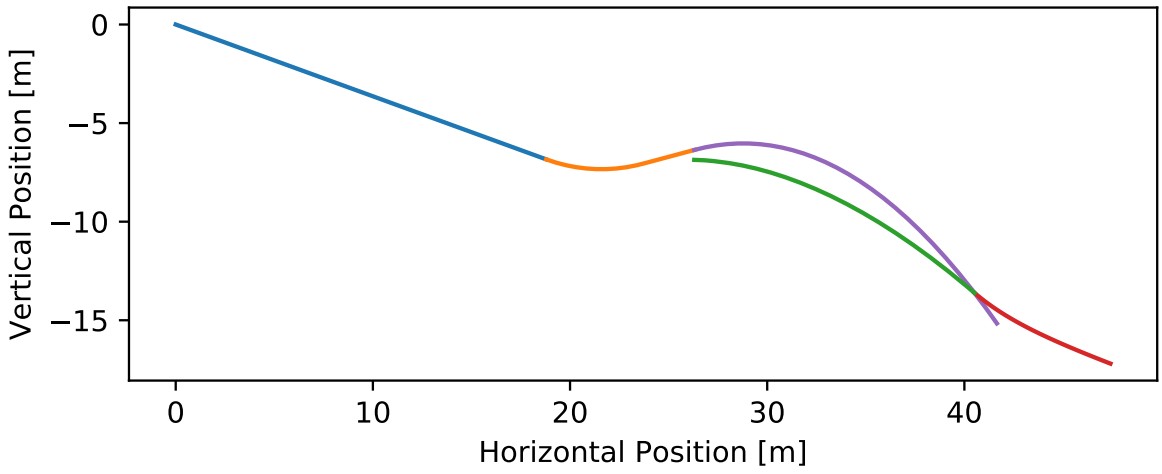
```
from skijumpdesign import LandingSurface

slope = FlatSurface(approach_ang, np.sqrt(landing_trans.end[0]**2 +
                                          landing_trans.end[1]**2) + 1.0)
```

(continues on next page)




(continued from previous page)

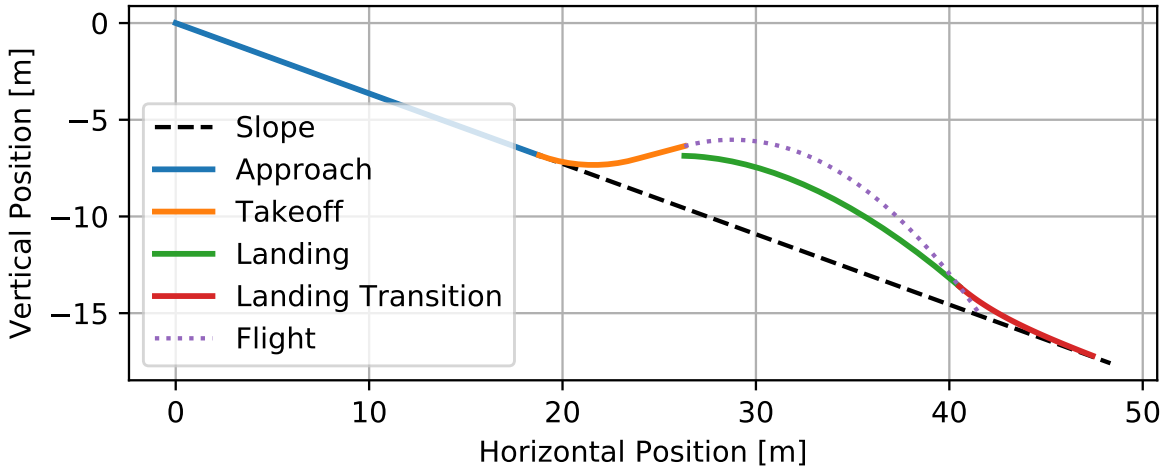
```
landing = LandingSurface(skier, takeoff.end, takeoff_ang,
                         landing_trans.start, fall_height,
                         surf=slope)

ax = approach.plot()
ax = takeoff.plot(ax=ax)
ax = flight.plot(ax=ax, color='#9467bd')
ax = landing_trans.plot(ax=ax, color='#d62728')
landing.plot(ax=ax, color='#2ca02c')
```




## 3.6 Entire Jump

There is a convenience function for plotting the jump:

```
from skijumpdesign import plot_jump

plot_jump(slope, approach, takeoff, landing, landing_trans, flight)
```



CHAPTER 4

# Application Programming Interface (API)

## 4.1 skijumpdesign/functions.py

skijumpdesign.functions.**make_jump**

Returns a set of surfaces and output values that define the equivalent fall height jump design and the skier's flight trajectory.

**Parameters**

- **slope_angle** (*float*) – The parent slope angle in degrees. Counter clockwise is positive and clockwise is negative.
- **start_pos** (*float*) – The distance in meters along the parent slope from the top (x=0, y=0) to where the skier starts skiing.
- **approach_len** (*float*) – The distance in meters along the parent slope the skier travels before entering the takeoff.
- **takeoff_angle** (*float*) – The angle in degrees at end of the takeoff ramp. Counter clockwise is positive and clockwise is negative.
- **fall_height** (*float*) – The desired equivalent fall height of the landing surface in meters.
- **plot** (*boolean*) – If True a matplotlib figure showing the jump will appear.

**Returns**

- **slope** (*FlatSurface*) – The parent slope starting at (x=0, y=0) until a meter after the jump.
- **approach** (*FlatSurface*) – The slope the skier travels on before entering the takeoff.
- **takeoff** (*TakeoffSurface*) – The circle-clothoid-circle-flat takeoff ramp.
- **landing** (*LandingSurface*) – The equivalent fall height landing surface.
- **landing_trans** (*LandingTransitionSurface*) – The minimum exponential landing transition.
- **flight** (*Trajectory*) – The maximum velocity flight trajectory.

- **outputs** (*dictionary*) – A dictionary of output values.

skijumpdesign.functions.**plot_jump**(*slope*, *approach*, *takeoff*, *landing*, *landing_trans*, *flight*)
Returns a matplotlib axes with the jump and flight plotted given the surfaces created by `make_jump()`.

skijumpdesign.functions.**snow_budget**(*parent_slope*, *takeoff*, *landing*, *landing_trans*)
Returns the jump's cross sectional snow budget area of the EFH jump.

**Parameters**

- **parent_slope** (`FlatSurface`) – A FlatSurface that spans before and after the jump.
- **takeoff** (`TakeoffSurface`) – The clothiod-circle-clothiod-flat takeoff surface.
- **landing** (`LandingSurface`) – The EFH landing surface.
- **landing_trans** (`LandingTransitionSurface`) – The EFH landing transition surface.

**Returns** The cross sectional snow budget (area between the parent slope and jump curve) in meters squared.

**Return type** float

## 4.2 skijumpdesign/skiers.py

**class** skijumpdesign.skiers.**Skier**(*mass=75.0*, *area=0.34*, *drag_coeff=0.821*, *friction_coeff=0.03*, *tolerable_sliding_acc=1.5*, *tolerable_landing_acc=3.0*)
Bases: `object`

Class that represents a skier which can slide on surfaces and fly in the air.

Instantiates a skier with default properties.

**Parameters**

- **mass** (*float, optional*) – The mass of the skier.
- **area** (*float, optional*) – The frontal area of the skier.
- **drag_coeff** (*float, optional*) – The air drag coefficient of the skier.
- **friction_coeff** (*float, optional*) – The sliding friction coefficient between the skis and the slope.
- **tolerable_sliding_acc** (*float, optional*) – The maximum normal acceleration in G's that a skier can withstand while sliding.
- **tolerable_landing_acc** (*float, optional*) – The maximum normal acceleration in G's that a skier can withstand when landing.

**drag_force**(*speed*)
Returns the drag force in Newtons opposing the speed of the skier.

**end_speed_on**(*surface*, ***kwargs*)
Returns the ending speed after sliding on the provided surface. Keyword args are passed to Skier.slide_on().

**end_vel_on**(*surface*, ***kwargs*)
Returns the ending velocity (vx, vy) after sliding on the provided surface. Keyword args are passed to Skier.slide_on().

**fly_to**(*surface*, *init_pos*, *init_vel*, *fine=True*, *compute_acc=True*, *logging_type='info'*)
Returns the flight trajectory of the skier given the initial conditions and a surface which the skier contacts at the end of the flight trajectory.

**Parameters**

- **surface** (Surface) – A landing surface. This surface must intersect the flight path.
- **init_pos** (*2-tuple of floats*) – The x and y coordinates of the starting point of the flight.
- **init_vel** (*2-tuple of floats*) – The x and y components of the skier's velocity at the start of the flight.
- **fine** (*boolean*) – If True two integrations occur. The first finds the landing time with coarse time steps and the second integrates over a finer equally spaced time steps. False will skip the second integration.
- **compute_acc** (*boolean, optional*) – If true acceleration will be calculated. If false acceleration is set to zero.
- **logging_type** (*string*) – The logging level desired for the non-debug logging calls in this function. Useful for suppressing too much information since this runs a lot.

**Returns** **trajectory** – A trajectory instance that contains the time, position, velocity, acceleration, speed, and slope of the flight.

**Return type** *Trajectory*

**Raises**

- InvalidJumpError if the skier does not contact a surface within
- `Skier.max_flight_time`.

**friction_force**(*speed*, *slope=0.0*, *curvature=0.0*)
Returns the friction force in Newtons opposing the speed of the skier.

**Parameters**

- **speed** (*float*) – The tangential speed of the skier in meters per second.
- **slope** (*float, optional*) – The slope of the surface at the point of contact.
- **curvature** (*float, optional*) – The curvature of the surface at the point of contact.

**max_flight_time = 30.0**

**samples_per_sec = 360**

**slide_on**(*surface*, *init_speed=0.0*, *fine=True*)
Returns the trajectory of the skier sliding over a surface.

**Parameters**

- **surface** (Surface) – A surface that the skier will slide on.
- **init_speed** (*float, optional*) – The magnitude of the velocity of the skier at the start of the surface which is directed tangent to the surface.
- **fine** (*boolean*) – If True two integrations occur. The first finds the exit time with coarse time steps and the second integrates over a finer equally spaced time steps. False will skip the second integration.

**Returns** **trajectory** – A trajectory instance that contains the time, position, velocity, acceleration, speed, and slope of the slide,

**Return type** *Trajectory*

**Raises**

- InvalidJumpError if skier can't reach the end of the surface within
- 1000 seconds.

**speed_to_land_at**(*landing_point*, *takeoff_point*, *takeoff_angle*, *surf*)

Returns the magnitude of the velocity required to land at a specific point given launch position and angle.

**Parameters**

- **landing_point** (*2-tuple of floats*) – The (x, y) coordinates of the desired landing point in meters.
- **takeoff_point** (*2-tuple of floats*) – The (x, y) coordinates of the takeoff point in meters.
- **takeoff_angle** (*float*) – The takeoff angle in radians.
- **surf** (Surface) – This should most likely be the parent slope but needs to be something that ensures the skier flies past the landing point.

**Returns** **takeoff_speed** – The magnitude of the takeoff velocity.

**Return type** float

## 4.3 skijumpdesign/surfaces.py

**class** skijumpdesign.surfaces.**ClothoidCircleSurface**(*entry_angle*, *exit_angle*, *entry_speed*, *tolerable_acc*, *init_pos=(0.0, 0.0)*, *gamma=0.99*, *num_points=200*)

Bases: *skijumpdesign.surfaces.Surface*

Class that represents a surface made up of a circle bounded by two clothoids.

Instantiates a clothoid-circle-clothoid curve.

**Parameters**

- **entry_angle** (*float*) – The entry angle tangent to the start of the left clothoid in radians.
- **exit_angle** (*float*) – The exit angle tangent to the end of the right clothoid in radians.
- **entry_speed** (*float*) – The magnitude of the skier's velocity in meters per second as they enter the left clothiod.
- **tolerable_acc** (*float*) – The tolerable normal acceleration of the skier in G's.
- **init_pos** (*2-tuple of floats*) – The x and y coordinates of the start of the left clothoid.
- **gamma** (*float*) – Fraction of circular section.
- **num_points** (*integer, optional*) – The number of points in each of the three sections of the curve.

**class** skijumpdesign.surfaces.**FlatSurface**(*angle*, *length*, *init_pos=(0.0, 0.0)*, *num_points=100*)

Bases: *skijumpdesign.surfaces.Surface*

Class that represents a flat surface angled relative to the horizontal.

Instantiates a flat surface that is oriented at a counterclockwise angle from the horizontal.

**Parameters**

- **angle** (*float*) – The angle of the surface in radians. Counterclockwise (about z) is positive, clockwise is negative.
- **length** (*float*) – The distance in meters along the surface from the initial position.
- **init_pos** (*2-tuple of floats, optional*) – The x and y coordinates in meters that locate the start of the surface.
- **num_points** (*integer, optional*) – The number of points used to define the surface coordinates.

**angle**

Returns the angle wrt to horizontal in radians of the surface.

**distance_from**(*xp*, *yp*)

Returns the shortest distance from point (xp, yp) to the surface.

**Parameters**

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

**Returns** **distance** – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

**Return type** float

**class** skijumpdesign.surfaces.**HorizontalSurface**(*height*, *length*, *start=0.0*, *num_points=100*)

Bases: *skijumpdesign.surfaces.Surface*

Instantiates a class that represents a horizontal surface at a height above the x axis.abs

**Parameters**

- **height** (*float*) – The height of the surface above the horizontal x axis in meters.
- **length** (*float*) – The length of the surface in meters.
- **start** (*float, optional*) – The x location of the start of the left most point of the surface.
- **num_points** (*integer, optional*) – The number of (x,y) coordinates.

**distance_from**(*xp*, *yp*)

Returns the shortest distance from point (xp, yp) to the surface.

**Parameters**

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

**Returns** **distance** – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

**Return type** float

**class** skijumpdesign.surfaces.**LandingSurface**(*skier*, *takeoff_point*, *takeoff_angle*, *max_landing_point*, *fall_height*, *surf*)

Bases: *skijumpdesign.surfaces.Surface*

Class that defines an equivalent fall height landing surface.

Instantiates a surface that ensures impact velocity is equivalent to that from a vertical fall.

**Parameters**

- **skier** (*Skier*) – A skier instance.
- **takeoff_point** (*2-tuple of floats*) – The point at which the skier leaves the takeoff ramp.
- **takeoff_angle** (*float*) – The takeoff angle in radians.
- **max_landing_point** (*2-tuple of floats*) – The maximum x position that the landing surface will attain in meters. In the standard design, this is the start of the landing transition point.
- **fall_height** (*float*) – The desired equivalent fall height in meters. This should always be greater than zero.
- **surf** (*Surface*) – A surface below the full flight trajectory, the parent slope is a good choice. It is useful if the distance_from() method runs very fast, as it is called a lot internally.

**allowable_impact_speed**
Returns the perpendicular speed one would reach if dropped from the provided fall height.

**class** skijumpdesign.surfaces.**LandingTransitionSurface**(*parent_surface*, *flight_traj*, *fall_height*, *tolerable_acc*, *num_points=100*)

Bases: *skijumpdesign.surfaces.Surface*

Class representing a acceleration limited exponential curve that transitions the skier from the landing surface to the parent slope.

Instantiates an exponentially decaying surface that connects the landing surface to the parent slope.

**Parameters**

- **parent_surface** (*FlatSurface*) – The parent slope in which the landing transition should be tangent to on exit.
- **flight_traj** (*Trajectory*) – The flight trajectory from the takeoff point to the parent slope.
- **fall_height** (*float*) – The desired equivalent fall height for the jump design in meters.
- **tolerable_acc** (*float*) – The maximum normal acceleration the skier should experience in the landing.
- **num_points** (*integer*) – The number of points in the surface.

**acc_error_tolerance = 0.001**

**allowable_impact_speed**
Returns the perpendicular speed one would reach if dropped from the provided fall height.

**calc_trans_acc**(*x*)
Returns the acceleration in G's the skier feels at the exit transition occurring if the transition starts at the provided horizontal location, x.

**delta = 0.01**

**find_parallel_traj_point**()
Returns the position of a point on the flight trajectory where its tangent is parallel to the parent slope. This is used as a starting guess for the start of the landing transition point.

**find_transition_point**()
Returns the horizontal position indicating the intersection of the flight path with the beginning of the landing transition. This is the last possible transition point, that by definition minimizes the transition snow budget, that satisfies the allowable transition acceleration.

### Notes

This uses Newton's method to find an adequate point but may fail to do so with some combinations of flight trajectories, parent slope geometry, and allowable acceleration. A warning will be emitted if the maximum number of iterations is reached in this search and the curve is likely invalid.

**max_iterations = 1000**

**class** skijumpdesign.surfaces.**Surface**(*x*, *y*)
Bases: object

Base class for a 2D curve that represents the cross section of a surface expressed in a standard Cartesian coordinate system.

Instantiates an arbitrary 2D surface.

**Parameters**

- **x** (*array_like, shape(n,)*) – The horizontal, x, coordinates of the slope. x[0] should be the left most horizontal position and corresponds to the start of the surface. This should be monotonically increasing.
- **y** (*array_like, shape(n,)*) – The vertical, y, coordinates of the slope. y[0] corresponds to the start of the surface.

**area_under**(*x_start=None*, *x_end=None*, *interval=0.05*)
Returns the area under the curve integrating wrt to the x axis at 0.05 m intervals using the trapezoidal rule.

**distance_from**(*xp*, *yp*)
Returns the shortest distance from point (xp, yp) to the surface.

**Parameters**

- **xp** (*float*) – The horizontal, x, coordinate of the point.
- **yp** (*float*) – The vertical, y, coordinate of the point.

**Returns** **distance** – The shortest distance from the point to the surface. If the point is above the surface a positive distance is returned, else a negative distance.

**Return type** float

---

**Note:** This general implementation can be slow, so implement overloaded distance_from() methods in subclasses when you can.

---

**end**
Returns the x and y coordinates at the end point of the surface.

**height_above**(*surface*)
Returns an array of values giving the height each point in this surface is above the provided surface.

**length**()
Returns the length of the surface in meters via a numerical line integral.

**plot**(*ax=None*, ***plot_kwargs*)
Returns a matplotlib axes containing a plot of the surface.

**Parameters**

- **ax** (*Axes*) – An existing matplotlib axes to plot to.
- **plot_kwargs** (*dict*) – Arguments to be passed to Axes.plot().

**shift_coordinates**(*delx*, *dely*)
Shifts the x and y coordinates by delx and dely respectively. This modifies the surface in place.

**start**
Returns the x and y coordinates at the start point of the surface.

**class** skijumpdesign.surfaces.**TakeoffSurface**(*skier*, *entry_angle*, *exit_angle*, *entry_speed*, *time_on_ramp=0.25*, *gamma=0.99*, *init_pos=(0.0, 0.0)*, *num_points=200*)
Bases: *skijumpdesign.surfaces.Surface*

Class that represents a surface made up of a circle bounded by two clothoids with a flat exit surface.

Instantiates the takeoff curve with the flat takeoff ramp added to the terminus of the clothoid-circle-clothoid curve.

**Parameters**

- **skier** (Skier) – A skier instance.
- **entry_angle** (*float*) – The entry angle tangent to the start of the left clothoid in radians.
- **exit_angle** (*float*) – The exit angle tangent to the end of the right clothoid in radians.
- **entry_speed** (*float*) – The magnitude of the skier's velocity in meters per second as they enter the left clothiod.
- **time_on_ramp** (*float, optional*) – The time in seconds that the skier should be on the takeoff ramp before launch.
- **gamma** (*float, optional*) – Fraction of circular section.
- **init_pos** (*2-tuple of floats, optional*) – The x and y coordinates of the start of the left clothoid.
- **num_points** (*integer, optional*) – The number of points in each of the three sections of the curve.

## 4.4 skijumpdesign/trajectories.py

**class** skijumpdesign.trajectories.**Trajectory**(*t*, *pos*, *vel=None*, *acc=None*, *speed=None*)
Bases: object

Class that describes a 2D trajectory.

Instantiates a trajectory.

**Parameters**

- **t** (*array_like, shape(n,)*) – The time values of the trajectory.

- **pos** (*array_like, shape(n, 2)*) – The x and y coordinates of the position.
- **vel** (*array_like, shape(n, 2), optional*) – The x and y components of velocity. If not provided numerical differentiation of position will be used.
- **acc** (*array_like, shape(n, 2), optional*) – The x and y components of acceleration. If not provided numerical differentiation of velocity will be used.
- **speed** (*array_like, shape(n, 2), optional*) – The magnitude of the velocity. If not provided it will be calculated from the velocity components.

**duration**
Returns the duration of the trajectory in seconds.

**plot**(*ax=None*, ***plot_kwargs*)
Returns a matplotlib axes containing a plot of the trajectory position.

**Parameters**

- **ax** (*Axes*) – An existing matplotlib axes to plot to.
- **plot_kwargs** (*dict*) – Arguments to be passed to Axes.plot().

**plot_time_series**()
Plots all of the time series stored in the trajectory.

**shift_coordinates**(*delx*, *dely*)
Shifts the x and y coordinates by delx and dely respectively. This modifies the surface in place.

## 4.5 skijumpdesign/utils.py

**exception** skijumpdesign.utils.**InvalidJumpError**
Bases: Exception

Custom class to signal that a poor combination of parameters have been supplied to the surface building functions.

skijumpdesign.utils.**speed2vel**(*speed*, *angle*)
Returns the x and y components of velocity given the magnitude and angle of the velocity vector.

**Parameters**

- **speed** (*float*) – Magnitude of the velocity vector in meters per second.
- **angle** (*float*) – Angle of velocity vector in radians. Clockwise is negative and counter clockwise is positive.

**Returns**

- **vel_x** (*float*) – X component of velocity in meters per second.
- **vel_y** (*float*) – Y component of velocity in meters per second.

skijumpdesign.utils.**vel2speed**(*hor_vel*, *ver_vel*)
Returns the magnitude and angle of the velocity vector given the horizontal and vertical components.

**Parameters**

- **hor_vel** (*float*) – X component of velocity in meters per second.
- **ver_vel** (*float*) – Y component of velocity in meters per second.

**Returns**

- **speed** (*float*) – Magnitude of the velocity vector in meters per second.
- **angle** (*float*) – Angle of velocity vector in radians. Clockwise is negative and counter clockwise is positive.

CHAPTER 5

# References

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## S

# Index

## S



## T



## V