leveldiagram

Release 0.1.0

David Meyer

Dec 05, 2022

DOCUMENTATION

A python library for generating AMO physics level diagrams with matplotlib.

CHAPTER

ONE

INTRODUCTION

```
%matplotlib inline
```

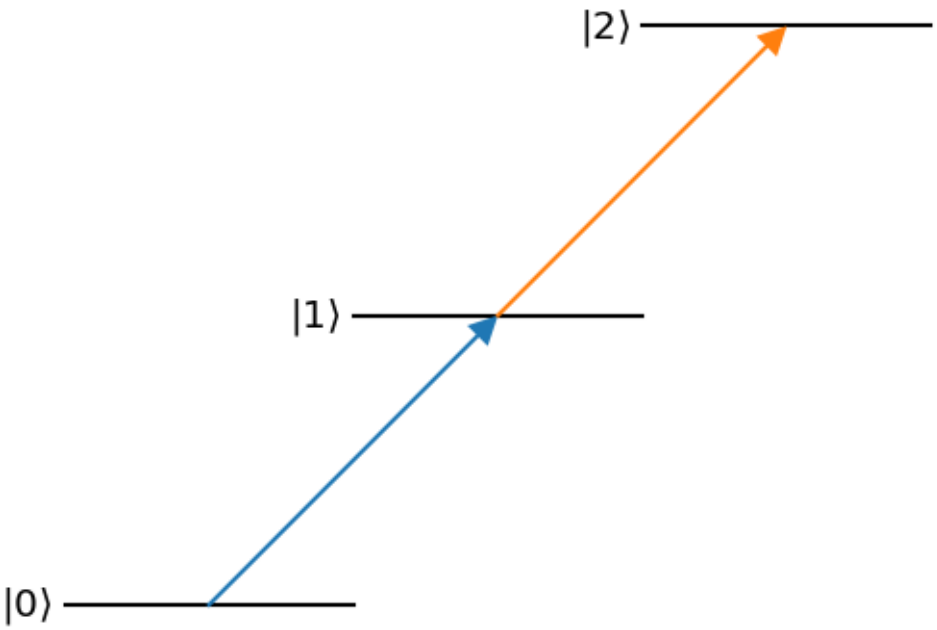
```
import networkx as nx
import leveldiagram as ld
```

To begin, the system is defined using a direction graph, provided by the networkx.DiGraph class. The nodes of this graph represent the levels, the edges represent the desired couplings.

Passing a simple graph to the base level diagram constructor LD will produce a passable output for simple visualization.

```
nodes = (0,1,2)
edges = ((0,1),(1,2))
graph = nx.DiGraph()
graph.add_nodes_from(nodes)
graph.add_edges_from(edges)
```

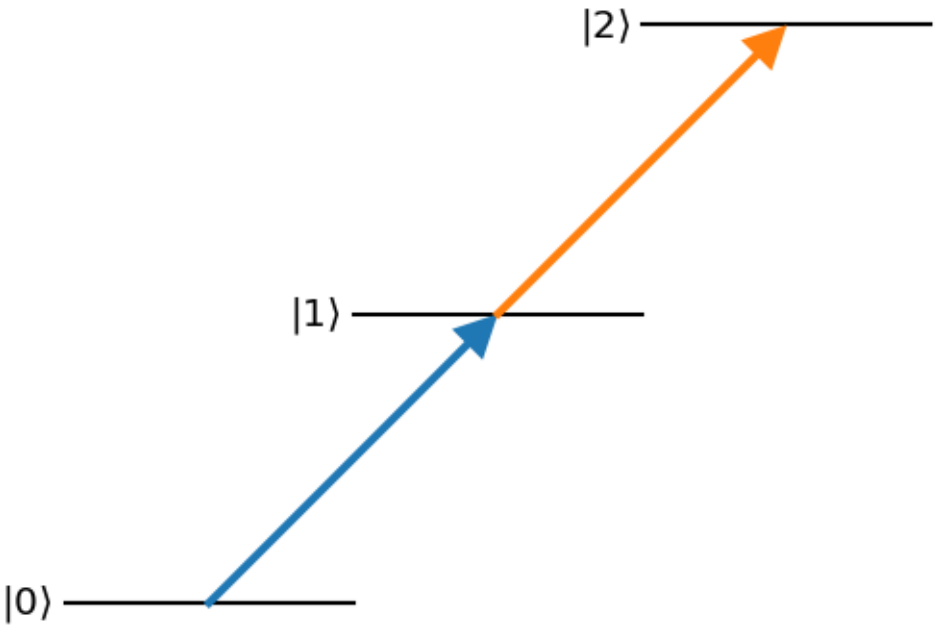
```
d = ld.LD(graph)
d.draw()
```



In keeping with peak matplotlib form, getting something that looks nicer requires applying custom configuration settings that control many of the aspects of the diagram.

Gloabl settings can be controlled by passing in keyword argument dictionaries to the constructor.

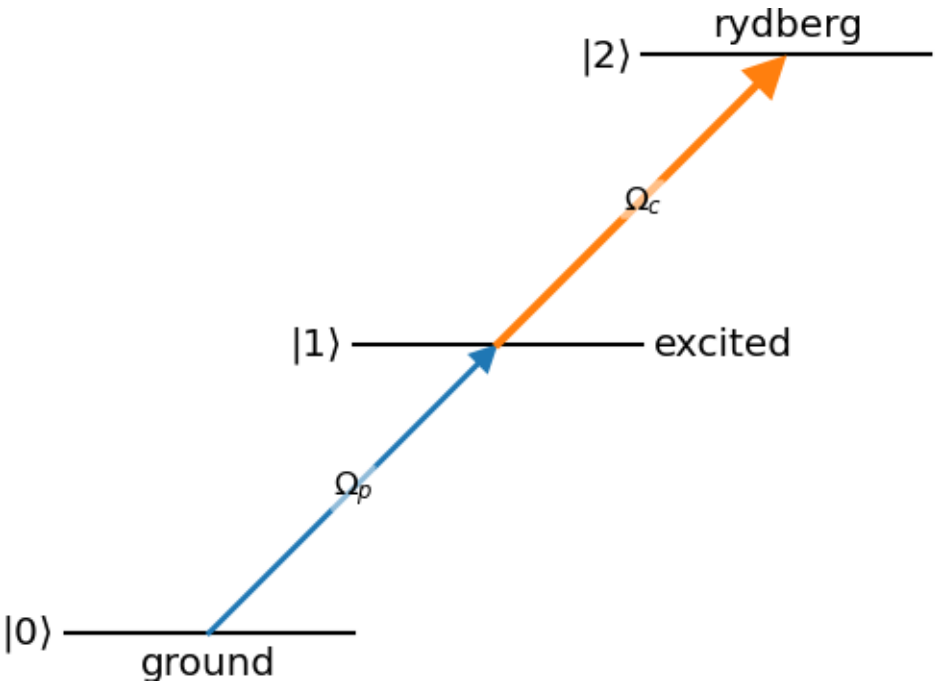
```
d = ld.LD(graph,
          coupling_defaults = {'arrowsize':0.15,'lw':3})
d.draw()
```



NetworkX graphs take an internal structure of nested dictionaries. Leveldiagram utilizes this to provide keyword argument control over each element in the graph.

```
nodes = ((0,{'bottom_text':'ground'}),
         (1,{'right_text':'excited'}),
         (2,{'top_text':'rydberg'}))
edges = ((0,1, {'label':'$\\Omega_p$', 'lw':2}),
         (1,2, {'label':'$\\Omega_c$','lw':3, 'arrowsize':0.15}))
graph = nx.DiGraph()
graph.add_nodes_from(nodes)
graph.add_edges_from(edges)
```

```
d = ld.LD(graph)
d.draw()
```


rydberg
|2⟩
Ω_c
|1⟩
excited
Ω_p
|0⟩
ground

CHAPTER

TWO

DETAILED API DOCUMENTATION

Documention of the classes and methods provided by **leveldiagram**

2.1 Level Diagram Constructors

class leveldiagram.**LD**(*graph*, *ax=None*, *default_label='left_text'*, *level_defaults=None*, *coupling_defaults=None*, *wavycoupling_defaults=None*)

Basic Level Diagram drawing class.

This class is used to draw a level diagram based on a provided Directional Graph. The nodes of this graph define the energy levels, the edges define the couplings.

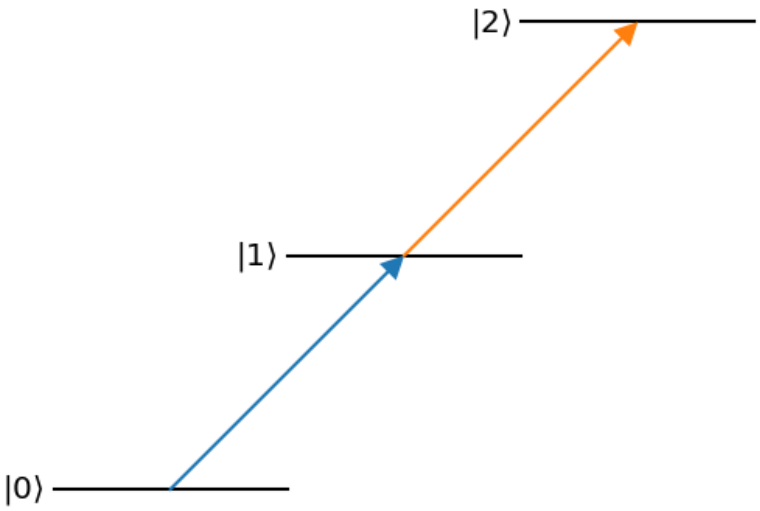
Parameters

- **graph** (*networkx.DiGraph*) – Graph object that defines the system to diagram
- **ax** (*matplotlib.axes.Axes, optional*) – Axes to add the diagram to. If None, creates a new figure and axes. Default is None.
- **default_label** (*str, optional*) – Sets which text label direction to use for default labelling, which is the node index inside a key. Valid options are `'left_text'`, `'right_text'`, `'top_text'`, `'bottom_text'`. If 'none', no default labels are not generated.
- **level_defaults** (*dict, optional*) – *EnergyLevel* default values for whole diagram. Provided values override class defaults. If None, use class defaults.
- **coupling_defaults** (*dict, optional*) – *Coupling* default values for whole diagram. Provided values override class defaults. If None, use class defaults.
- **wavycoupling_defaults** (*dict, optional*) – *WavyCoupling* default values for whole diagram. Provided values override class defaults. If None, use class defaults.

In keeping with the finest matplotlib traditions, default options and behavior will produce a *reasonable* output from a graph.

Examples

```
>>> nodes = (0,1,2)
>>> edges = ((0,1), (1,2))
>>> graph = nx.DiGraph()
>>> graph.add_nodes_from(nodes)
>>> graph.add_edges_from(edges)
>>> d = ld.LD(graph)
>>> d.draw()
```



To get more refined diagrams, global options can be set by passing keyword argument dictionaries to the constructor. Options per level or coupling can be set by setting keyword arguments in the dictionaries of the nodes and edges of the graph.

_coupling_defaults = {'arrowsize': 0.1, 'label_kw': {'fontsize': 'large'}}

Coupling default parameters dictionary

_level_defaults = {'color': 'k', 'text_kw': {'fontsize': 'x-large'}, 'width': 1}

EnergyLevel default parameters dictionary

_wavycoupling_defaults = {'halfperiod': 0.1, 'waveamp': 0.1}

WavyCoupling default parameters dictionary

couplings: Dict[Tuple[int, int], *Coupling*]

Stores couplings to be drawn

draw()

Add artists to the figure.

This calls `matplotlib.axes.Axes.autoscale_view()` to ensure plot ranges are increased to account for objects.

It may be necessary to increase plot margins to handle labels near edges of the plot.

generate_couplings()

Creates the Coupling and WavyCoupling artisits from the graph edges.

They are saved to the *couplings* dictionary.

generate_levels()

Creates the EnergyLevel artists from the graph nodes.

They are saved to the *levels* dictionary.

levels: Dict[int, *EnergyLevel*]

Stores levels to be drawn

2.2 Artist Primitives

Customized matplotlib artist primitives

class leveldiagram.artists.**Coupling**(*start*, *stop*, *arrowsize*, *arrowratio=1*, *tail=False*, *arrow_kw=None*, *label=''*, *label_offset='center'*, *label_rot=False*, *label_flip=False*, *label_kw=None*, ***kwargs*)

Bases: Line2D

Coupling artist for showing couplings between levels.

This artist is a conglomeration of artists.

- Line2D for the actual coupling path
- Polygon for the arrow heads
- Text for the label

Sufficient methods are overridden from the base Line2D class to ensure the other artists are rendered whenever the main artist is rendered.

Parameters

- **start** (*2-element collection*) – Coupling start location in data coordinates
- **stop** (*2-element collection*) – Coupling end location in data coordinates
- **arrowsize** (*float*) – Size of arrowheads in x-data coordinates
- **arrowratio** (*float, optional*) – Aspect ratio of the arrowhead. Default is 1 for equal aspect ratio.
- **tail** (*bool, optional*) – Whether to draw an identical arrowhead at the coupling base. Default is False.
- **arrow_kw** (*dict, optional*) – Dictionary of keyword arguments to pass to matplotlib.patches.Polygon constructor. Note that keyword arguments provided to this function will clobber identical keys provided here.
- **label** (*str, optional*) – Label string to apply to the coupling. Default is no label.
- **label_offset** (*str, optional*) – Offset direction for the label. Options are `'center'`, `'left'`, and `'right'`. Default is center of the coupling line.
- **label_rot** (*bool, optional*) – Label will be justified along the coupling arrow axis if True. Default is False, so label is oriented along x-axis always.
- **label_flip** (*bool, optional*) – Apply a 180 degree rotation to the label. Default is False.
- **label_kw** (*dict, optional*) – Dictionary of keyword arguments to pass to the matplotlib.text.Text constructor.
- **kwargs** – Optional keyword arguments passed to the matplotlib.lines.Line2D constructor and the matplotlib.patches.Polygon constructor for the arrowhead. Note that `'color'` will be automatically changed to `'facecolor'` for the arrowhead to avoid extra lines.

draw(*renderer*)

Draw the Artist (and its children) using the given renderer.

This has no effect if the artist is not visible (`.Artist.get_visible` returns False).

Parameters

renderer (`.RendererBase` subclass.) –

Notes

This method is overridden in the Artist subclasses.

init_arrowheads(***kwargs*)

Creates the arrowhead(s) for the coupling as matplotlib polygon objects.

Parameters
kwargs – Optional keyword arguments to pass to the `matplotlib.patches.Polygon` constructor.

init_label(*label*, *label_offset*, *label_rot*, *label_flip*, ***label_kw*)

Creates the coupling label text object.

Parameters

- **label** (*str*) – Label string to apply to the coupling.
- **label_offset** (*str*) – Offset direction for the label. Options are `'center'`, `'left'`, and `'right'`.
- **label_rot** (*bool*) – Label will be justified along the coupling arrow axis if True.
- **label_flip** (*bool*) – Apply a 180 degree rotation to the label.
- **label_kw** – Keyword arguments to pass to the `matplotlib.text.Text` constructor.

init_path()

Calculates the desired path for the line of the coupling.

The returned path is a line along the x-axis or the correct length. Transforms are used to move and rotate this path to the end location. This method of making the couplings is a little convoluted, but it allows for simple definition of very general paths (line sine waves) without distortions.

Returns

- **x** (*numpy.ndarray*) – x-coordinates of the data points for the un-rotated, un-translated path
- **y** (*numpy.ndarray*) – y-coordinates of the data points for the un-rotated, un-translated path (ie all zeros)

Return type
Tuple[*ndarray*, *ndarray*]

set_axes(*axes*)

set_figure(*figure*)

Set the `.Figure` instance the artist belongs to.

Parameters
fig (`.Figure`) –

set_transform(*transform*)

Set the artist transform.

Parameters
t (`.Transform`) –

class leveldiagram.artists.**EnergyLevel**(*energy*, *xpos*, *width*, *right_text=''*, *left_text=''*, *top_text=''*, *bottom_text=''*, *text_kw=None*, ***kwargs*)

Bases: `Line2D`

Energy level artist.

This object also implements a number of potential Text artists for labelling. It also includes helper methods for getting the exact coordinates of anchor points for connected coupling arrows and the like.

Parameters

- **energy** (*float*) – y-axis position of the level
- **xpos** (*float*) – x-axis position of the level
- **width** (*float*) – Width of the level line, in units of the x-axis
- **right_text** (*str, optional*) – Text to put to the right of the level
- **left_text** (*str, optional*) – Text to put to the left of the level
- **top_text** (*str, optional*) – Text to put above the level
- **bottom_text** (*str, optional*) – Text to put below the level
- **text_kw** (*dict, optional*) – Dictionary of keyword-arguments passed to `matplotlib.text.Text`
- **kwargs** – Passed to the `matplotlib.lines.Line2D` constructor

draw(*renderer*)

Draw the Artist (and its children) using the given renderer.

This has no effect if the artist is not visible (`.Artist.get_visible` returns False).

Parameters
renderer (`.RendererBase` subclass.) –

Notes

This method is overridden in the Artist subclasses.

get_anchor(*loc='center'*)

Returns an anchor on the level in plot coordinates.

Parameters
loc (*str or collection of 2 elements*) – What reference point to return. `'center'`, `'left'`, `'right'` gives those points of the level. A 2-element iterable is interpreted as offsets from the center location.

Raises
TypeError – If `loc` is not accepted string or a 2-element iterable.

Return type
ndarray

get_center()

Returns coordinates of the center of the level line.

Returns
x,y coordinates

Return type
numpy.ndarray

get_left()

Returns coordinates of the left of the level line.

Returns
x,y coordinates

Return type
numpy.ndarray

get_right()

Returns coordinates of the right of the level line.

Returns
x,y coordinates

Return type
numpy.ndarray

set_axes(*axes*)

Parameters
axes (*Axes*) –

set_data(*x*, *y*)

Set the x and y data.

Parameters
***args** (*(2, N) array or two 1D arrays*) –

set_figure(*figure*)

Set the .Figure instance the artist belongs to.

Parameters
fig (.Figure) –

set_transform(*transform*)

Overridden to add padding offsets to labels.

text_labels: Dict[str, Text]

Text label objects to add to the level

class leveldiagram.artists.**WavyCoupling**(*start*, *stop*, *waveamp*, *halfperiod*, *arrowsize*, *arrowratio=1*, *tail=False*, *arrow_kw=None*, *label=''*, *label_offset='center'*, *label_rot=False*, *label_flip=False*, *label_kw=None*, ***kwargs*)

Bases: *Coupling*

Coupling that uses a sine wave for the line.

This artists only differs from *Coupling* in that the path uses a sine wave.

Parameters

- **start** (*2-element collection*) – Coupling start location in data coordinates
- **stop** (*2-element collection*) – Coupling end location in data coordinates
- **waveamp** (*float*) – Amplitude of the sine wave in y-coordiantes
- **halfperiod** (*float*) – Length of a half-period of the sinewave in x-coordinates.
- **arrowsize** (*float*) – Size of arrowheads in x-data coordinates
- **arrowratio** (*float, optional*) – Aspect ratio of the arrowhead. Default is 1 for equal aspect ratio.
- **tail** (*bool, optional*) – Whether to draw an identical arrowhead at the coupling base. Default is False.
- **arrow_kw** (*dict, optional*) – Dictionary of keyword arguments to pass to matplotlib.patches.Polygon constructor. Note that keyword arguments provided to this function will clobber identical keys provided here.
- **label** (*str, optional*) – Label string to apply to the coupling. Default is no label.
- **label_offset** (*str, optional*) – Offset direction for the label. Options are 'center', 'left', and 'right'. Default is center of the coupling line.

- **label_rot** (*bool, optional*) – Label will be justified along the coupling arrow axis if True. Default is False, so label is oriented along x-axis always.
- **label_flip** (*bool, optional*) – Apply a 180 degree rotation to the label. Default is False.
- **label_kw** (*dict, optional*) – Dictionary of keyword arguments to pass to the `matplotlib.text.Text` constructor.
- **kwargs** – Optional keyword arguments passed to the `matplotlib.lines.Line2D` constructor and the `matplotlib.patches.Polygon` constructor for the arrowhead. Note that `'color'` will be automatically changed to `'facecolor'` for the arrowhead to avoid extra lines.

Warns

UserWarning – If wave amplitude is larger and arrowhead.

`init_path()`

Calculates the desired path for the line of the coupling.

The returned path is a sine wave along the x-axis or the correct length. Transforms are used to move and rotate this path to the end location.

Returns

- **x** (*numpy.ndarray*) – x-coordinates of the data points for the un-rotated, un-translated path
- **y** (*numpy.ndarray*) – y-coordinates of the data points for the un-rotated, un-translated path

Return type

Tuple[*ndarray*, *ndarray*]

2.3 Utilities

Miscellaneous utility functions

`leveldiagram.utils.bra_str(s)`

Put a bra around the string in matplotlib.

Parameters

s (*Any*) – Object to be converted to a string and placed inside a bra.

Returns

A string that will render as ⟨s|

Return type

str

`leveldiagram.utils.deep_update(mapping, *updating_mappings)`

Helper function to update nested dictionaries.

Lifted from pydantic

Returns

Deep-updated copy of `mapping`

Return type

dict

Parameters

- **mapping** (*dict*) –
- **updating_mappings** (*dict*) –

leveldiagram.utils.**ket_str**(*s*)

Put a ket around the string in matplotlib.

Parameters
s (*Any*) – Object to be converted to string and placed inside a ket.

Returns
A string that will render as $|s\rangle$

Return type
str

CHAPTER

THREE

CHANGELOG

3.1 Latest

3.1.1 Improvements

- stuff

3.1.2 Bug Fixes

- other stuff

3.1.3 Deprecations

- other other stuff

3.2 v0.1.0

Initial release.

Includes the artist primitives `EnergyLevel`, `Coupling`, and `WavyCoupling`. Also includes the base leveldiagram creation class `LD`.

CHAPTER

FOUR

ARTIST TESTS

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from leveldiagram.artists import Coupling, WavyCoupling, EnergyLevel
```

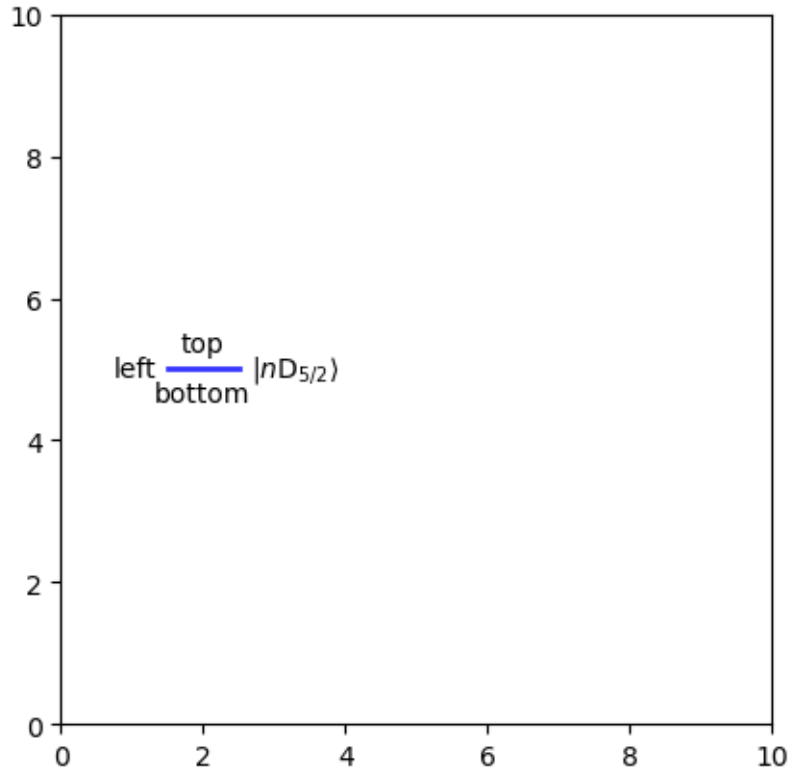
4.1 Level Diagram Tests

```
plt.close('all')
fig, ax = plt.subplots(1)

ax.set_xlim((0,10))
ax.set_ylim((0,10))

x = np.linspace(0,25,151)
y = np.sin(x)
lev = EnergyLevel(5, 2, 1,
                  r'$\left|n\mathrm{D}_{5/2}\right\rangle$', 'left', 'top', 'bottom',
                  color='b',alpha=0.8,linestyle='-', lw=2)
ax.add_line(lev)

ax.set_aspect('equal')
```



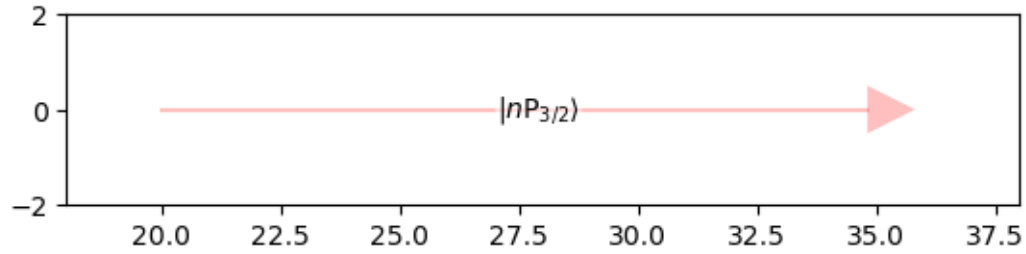
4.2 Coupling Tests

```
plt.close('all')
fig, ax = plt.subplots(1)

ax.set_xlim((18,38))
ax.set_ylim((-2,2))

ax.add_line(Coupling((20,0),(20+15.81,0),1,1,color='r',alpha=0.25,linestyle='-',␣
↪tail=False, arrow_kw={'ec':'none'},
                label=r'$|n\mathrm{P}_{3/2}\rangle$', label_offset='center', label_kw=
↪{'rotation_mode':'default'}))

ax.set_aspect('equal')
```




```
plt.close('all')
fig, ax = plt.subplots(1)

ax.set_xlim((5,40))
```

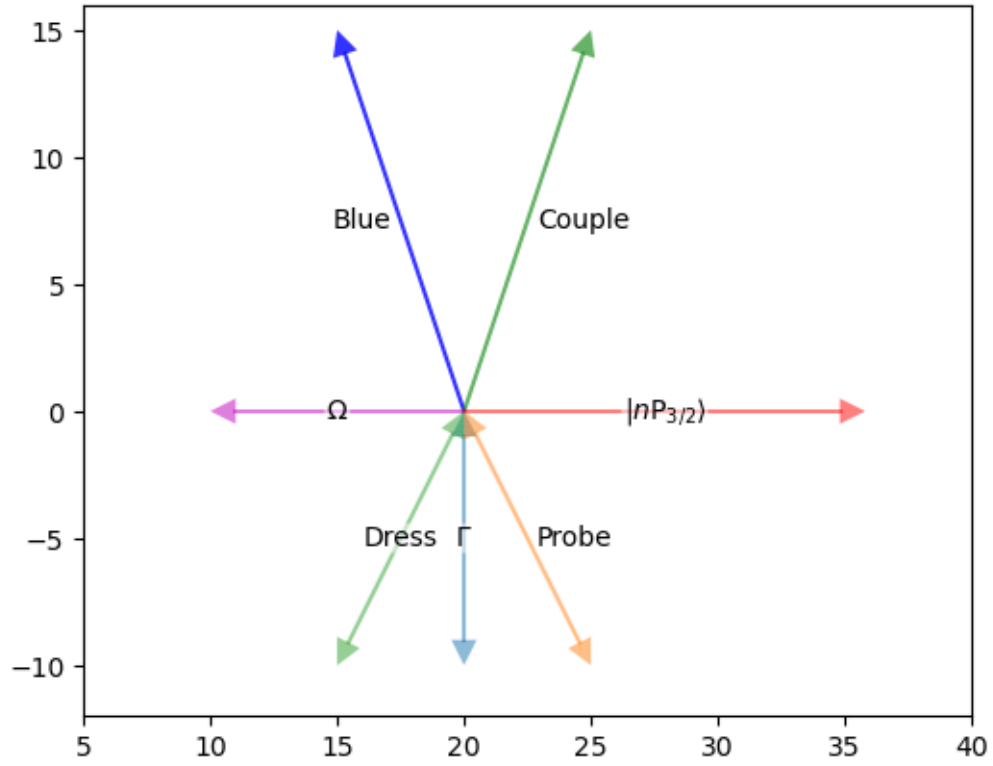
(continues on next page)

(continued from previous page)

```
ax.set_ylim((-12,16))

ax.add_line(Coupling((20,0),(15,15),1,1,color='b',alpha=0.8,linestyle='-', arrow_kw={
↪'ec':'none'},
                label='Blue', label_offset='left', label_kw={'rotation_mode':'default
↪'}))
ax.add_line(Coupling((20,0),(25,15),1,1,color='g',alpha=0.6,linestyle='-',
            label='Couple', label_offset='right', label_kw={'rotation_mode':'default'}
↪))
ax.add_line(Coupling((20,0),(20+15.81,0),1,1,color='r',alpha=0.5,linestyle='-', arrow_
↪kw={'ec':'none'},
                label=r'$|n\mathrm{P}_{3/2}\rangle$', label_offset='center', label_kw=
↪{'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (10,0), 1,1, color='m', alpha=0.5, linestyle='-',
                    label=r'$\Omega$', label_offset='center', label_kw={'rotation_mode
↪':'default'}))
ax.add_line(Coupling((20,0), (20,-10), 1,1, tail=True, color='C0', alpha=0.5,␣
↪linestyle='-', arrow_kw={'ec':'none'},
                    label=r'$\Gamma$', label_offset='center', label_kw={'rotation_mode
↪':'default'}))
ax.add_line(Coupling((20,0), (25,-10), 1,1, tail=True, color='C1', alpha=0.5,␣
↪linestyle='-',
           label='Probe', label_offset='right', label_kw={'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (15,-10), 1,1, tail=True, color='C2', alpha=0.5,␣
↪linestyle='-', arrow_kw={'ec':'none'},
            label='Dress', label_offset='center', label_kw={'rotation_mode':'default'}
↪))

ax.set_aspect('equal')
```

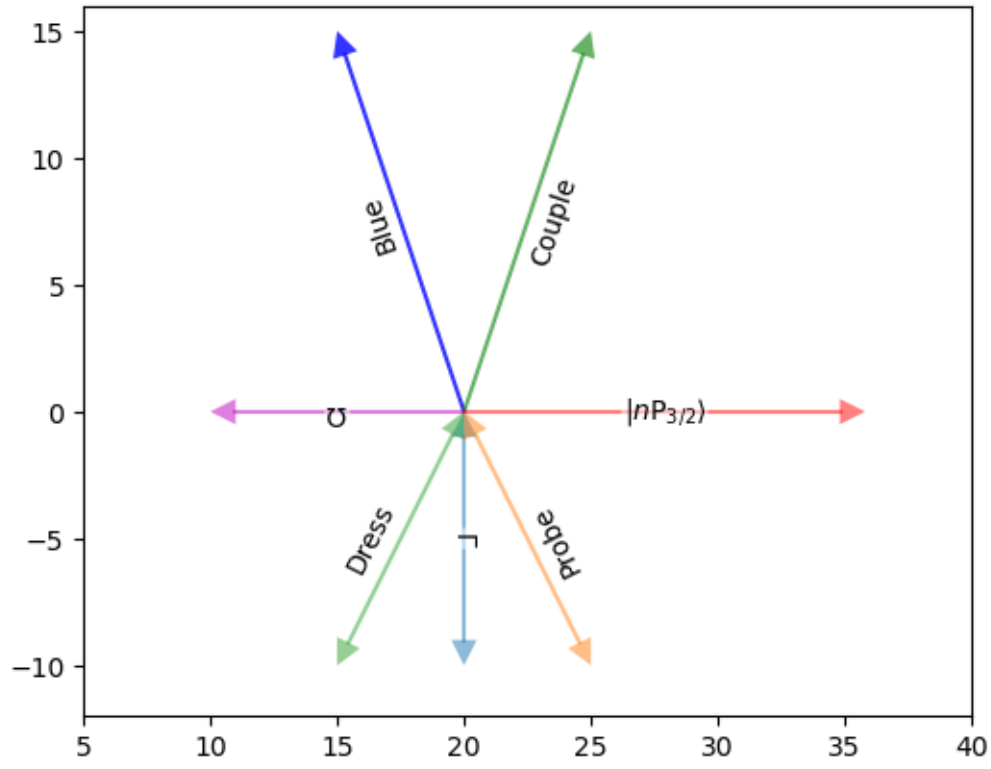


```
plt.close('all')
fig, ax = plt.subplots(1)

ax.set_xlim((5,40))
ax.set_ylim((-12,16))

ax.add_line(Coupling((20,0),(15,15),1,1,color='b',alpha=0.8,linestyle='-',
                label='Blue', label_offset='left', label_rot=True, label_kw={
↪'rotation_mode':'default'}))
ax.add_line(Coupling((20,0),(25,15),1,1,color='g',alpha=0.6,linestyle='-',
            label='Couple', label_offset='right', label_rot=True, label_kw={'rotation_
↪mode':'default'}))
ax.add_line(Coupling((20,0),(20+15.81,0),1,1,color='r',alpha=0.5,linestyle='-',
                label=r'$|n\mathrm{P}_{3/2}\rangle$', label_offset='center', label_
↪rot=True, label_kw={'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (10,0), 1,1, color='m', alpha=0.5, linestyle='-',
                    label=r'$\Omega$', label_offset='center', label_rot=True, label_
↪kw={'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (20,-10), 1,1, tail=True, color='C0', alpha=0.5,␣
↪linestyle='-',
                    label=r'$\Gamma$', label_offset='center', label_rot=True, label_
↪kw={'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (25,-10), 1,1, tail=True, color='C1', alpha=0.5,␣
↪linestyle='-',
           label='Probe', label_offset='right', label_rot=True, label_flip=True,␣
↪label_kw={'rotation_mode':'default'}))
ax.add_line(Coupling((20,0), (15,-10), 1,1, tail=True, color='C2', alpha=0.5,␣
↪linestyle='-',
            label='Dress', label_offset='left', label_rot=True, label_flip=True,␣
↪label_kw={'rotation_mode':'default'}))

ax.set_aspect('equal')
```



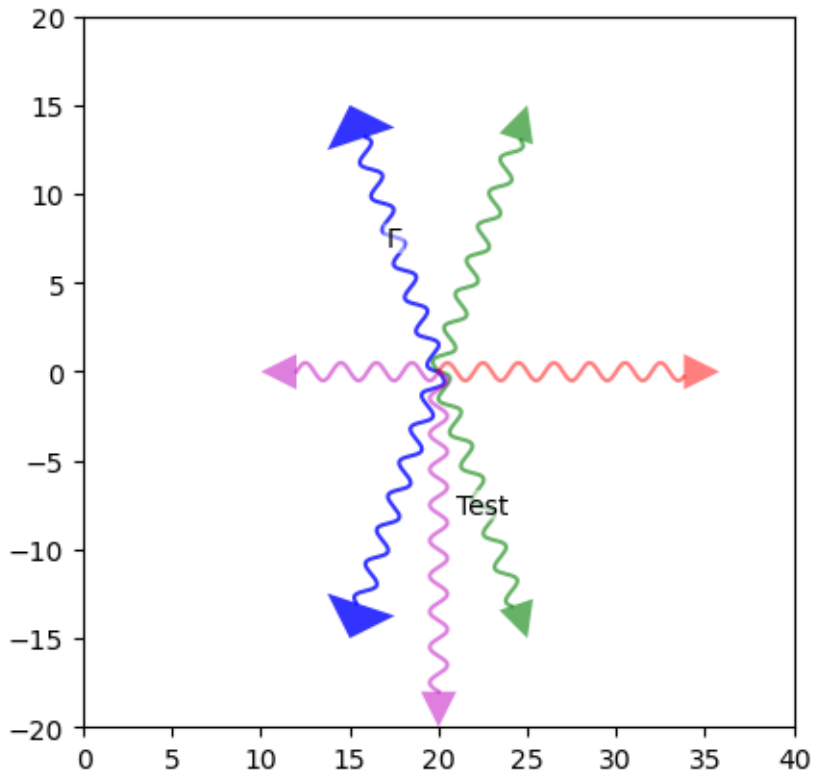
4.3 Wavy Coupling Tests

```
plt.close('all')
fig, ax = plt.subplots(1)

ax.set_xlim((0,40))
ax.set_ylim((-20,20))

x = np.linspace(0,25,151)
y = np.sin(x)
ax.add_line(WavyCoupling((20,0),(15,15),1,1,2,2,color='b',alpha=0.8,linestyle='-',␣
↪label=r'$\Gamma$'))
ax.add_line(WavyCoupling((20,0),(25,15),1,1,2,color='g',alpha=0.6,linestyle='-'))
ax.add_line(WavyCoupling((20,0),(15,-15),1,1,2,2,color='b',alpha=0.8,linestyle='-'))
ax.add_line(WavyCoupling((20,0),(25,-15),1,1,2,color='g',alpha=0.6,linestyle='-',␣
↪label='Test'))
ax.add_line(WavyCoupling((20,0),(20+15.81,0),1,1,2,color='r',alpha=0.5,linestyle='-',␣
↪arrow_kw={'ec':'none'}))
ax.add_line(WavyCoupling((20,0), (10,0), 1,1,2, color='m', alpha=0.5, linestyle='-'))
ax.add_line(WavyCoupling((20,0), (20,-20), 1,1,2, color='m', alpha=0.5, linestyle='-
↪'))


ax.set_aspect('equal')
```



CHAPTER

FIVE

LD TESTS

```
%matplotlib inline
```

```
%load_ext autoreload
%autoreload 2
```

```
import networkx as nx
```

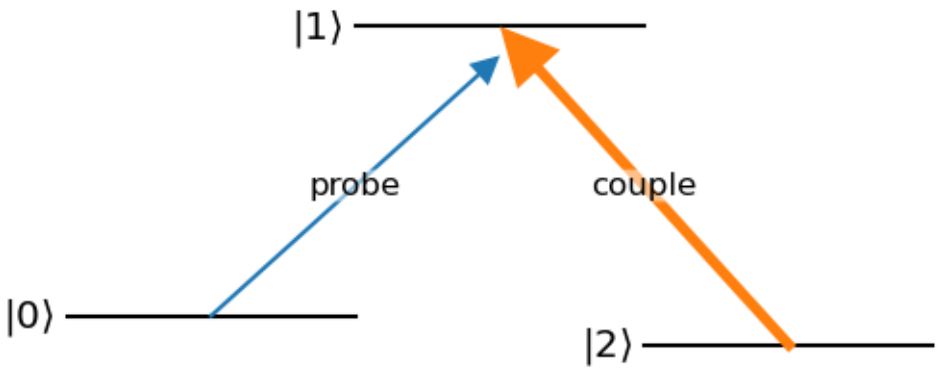
```
import leveldiagram as ld
```

5.1 Basic 3-level diagrams

5.1.1 Lambda

```
lambda_nodes = ((0),
                (1),
                (2, {'energy':-0.1}))
lambda_edges = ((0,1,{'detuning':0.1, 'label':'probe'}),
                (2,1,{'label':'couple', 'lw':4, 'arrowsize':0.2}))
lambda_graph = nx.DiGraph()
lambda_graph.add_nodes_from(lambda_nodes)
lambda_graph.add_edges_from(lambda_edges)
```

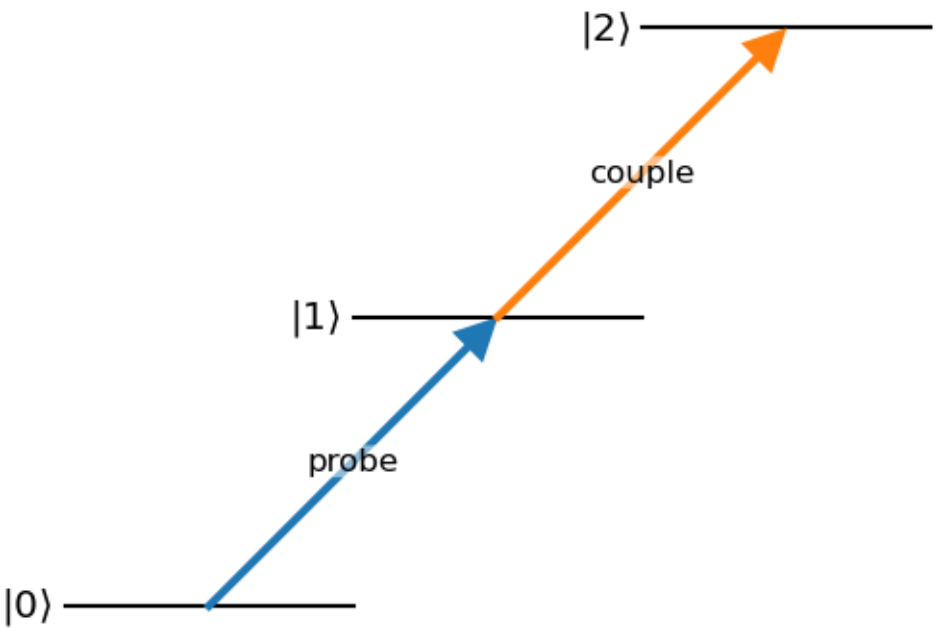
```
d = ld.LD(lambda_graph)
d.draw()
```



5.1.2 Ladder

```
ladder_nodes = (0,1,2)
ladder_edges = ((0,1,{'label':'probe'}),
                (1,2,{'label':'couple'}))
ladder_graph = nx.DiGraph()
ladder_graph.add_nodes_from(ladder_nodes)
ladder_graph.add_edges_from(ladder_edges)
```

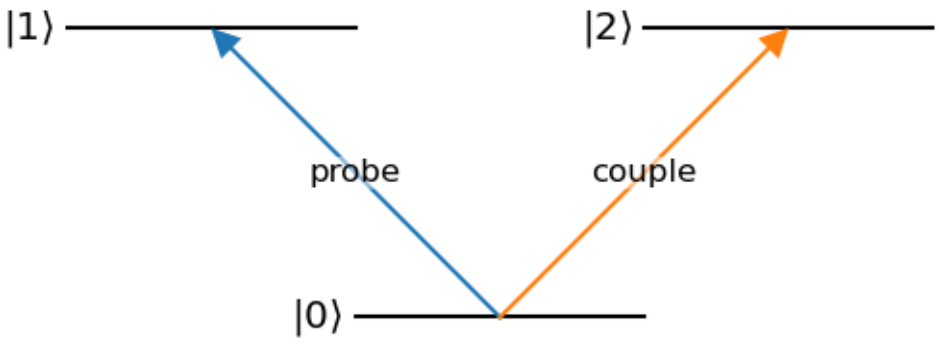
```
d = ld.LD(ladder_graph,
          coupling_defaults = {'arrowsize':0.15,'lw':3})
d.draw()
```



5.1.3 Vee

```
v_nodes = ((0),
           (1,{'energy':1,'xpos':-1}),
           (2,{'energy':1, 'xpos':1}))
v_edges = ((0,1,{'label':'probe'}),
           (0,2,{'label':'couple'}))
v_graph = nx.DiGraph()
v_graph.add_nodes_from(v_nodes)
v_graph.add_edges_from(v_edges)
```

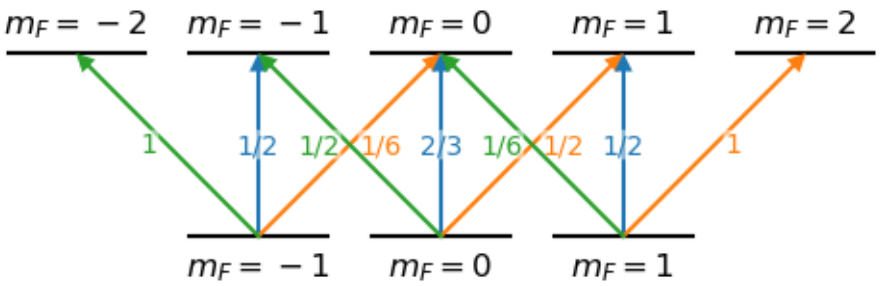
```
d = ld.LD(v_graph)
d.draw()
```



5.2 Hyperfine Diagram

```
hf_nodes =  [((f,i), {('top' if f==2 else 'bottom') + '_text':'$m_F='+f'{i:d}'+'$',
                      'energy':f-1,
                      'xpos':i,
                      'width':0.75,
                      'text_kw':{'fontsize':'large'}})
             for f in [1,2]
             for i in range(-f,f+1)]
lin_couples = [((1,i),(2,i),{'label':l,'color':'C0',
                             'label_kw':{'fontsize':'medium','color':'C0'}})
               for i,l in zip(range(-1,2), ['1/2','2/3','1/2'])]
sp_couples = [((1,i),(2,i+1),{'label':l,'color':'C1',
                              'label_offset':'right',
                             'label_kw':{'fontsize':'medium','color':'C1'}})
              for i,l in zip(range(-1,2), ['1/6','1/2','1'])]
sm_couples = [((1,i),(2,i-1),{'label':l, 'color':'C2',
                              'label_offset':'left',
                             'label_kw':{'fontsize':'medium','color':'C2'}})
              for i,l in zip(range(-1,2), ['1','1/2','1/6'])]
hf_edges = lin_couples + sp_couples + sm_couples
hf_graph = nx.DiGraph()
hf_graph.add_nodes_from(hf_nodes)
hf_graph.add_edges_from(hf_edges)
```

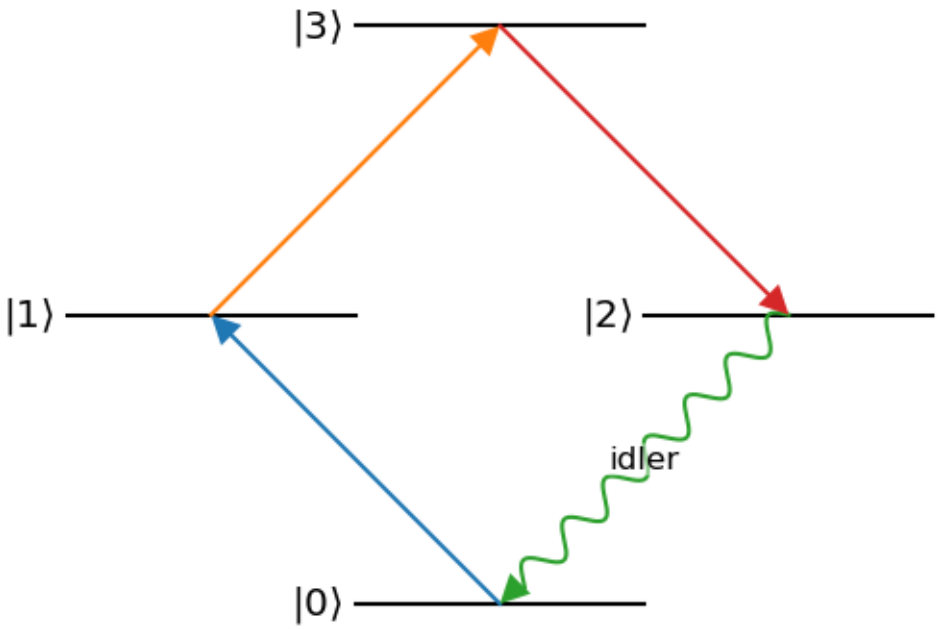
```
d = ld.LD(hf_graph, default_label = 'none')
d.ax.margins(y=0.2)
d.draw()
```



5.3 4-wave Mixing Diagram

```
fwm_nodes = ((0),
             (1,{'xpos':-1}),
             (2,{'xpos':1,'energy':1}),
             (3,{'energy':2,'xpos':0}))
fwm_edges = ((0,1),
             (1,3),
             (3,2),
             (2,0,{'label':'idler', 'wavy':True}))
fwm_graph = nx.DiGraph()
fwm_graph.add_nodes_from(fwm_nodes)
fwm_graph.add_edges_from(fwm_edges)
```

```
d = ld.LD(fwm_graph)
d.draw()
```



PYTHON MODULE INDEX

INDEX

Symbols

B

C

D

E

G

I

K

L

M

S

T

W