projplot

Release 1.0.1

Kanika Chopra, Martin Lysy

Nov 15, 2022

CONTENTS

Kanika Chopra, Martin Lysy – University of Waterloo

November 14, 2022

projplot provides a set of tools to assess whether or not an optimization algorithm has converged to a local optimum. Its main function does this by visualizing the "projection plots" of the objective function $f(\boldsymbol{x})$ – that is, by plotting f against each coordinate of $\boldsymbol{x}$ with the other coordinates fixed at the corresponding elements of the candidate optimal solution $\boldsymbol{x}_{opt}$.

This package has a similar goal to the R package **optimCheck**.

CHAPTER

ONE

INSTALLATION

This package is available on PyPi and can be installed as follows:

```
pip install projplot
```

1.1 Example

An overview of the package functionality is illustrated with the following example. Let

$$f(\boldsymbol{x}) = \boldsymbol{x}'\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{b}'\boldsymbol{x}$$

denote a quadratic objective function in $\boldsymbol{x}$, which is in the d-dimensional real space. If $\boldsymbol{A}$ is a positive-definite $d \times d$ matrix, then the unique minimum of $f(\boldsymbol{x})$ is $\boldsymbol{x}_{opt} = \boldsymbol{A}^{-1}\boldsymbol{b}$.

For example, suppose we have

```
import numpy as np

A = np.array([[3., 2.],
              [2., 7.]])
b = np.array([1., 10.])
```

Then we have that the optimal solution is $\boldsymbol{x}_{opt} = (-0.765, 1.647)$. Now, **projplot** allows us to complete a visual check. The following information will need to be provided:

- The objective function (`obj_fun`): This can be either a vectorized or non-vectorized function.
- Optimal values (`x_opt`): This will be the optimal solution for your function.
- Upper and lower bounds for each parameter (`x_lims`): This will provide an initial range of values to observe.
- Parameter names (`x_names`): These are the names of your parameters in the plots.
- The number of points to plot for each parameter (`n_pts`): This is the number of points that each parameter will be evaluated at for their respective plot.

1.1.1 Setup

```
# Optimal values
x_opt = np.array([-0.765,  1.647])

# Upper and lower bounds for each component of x
x_lims = np.array([[-3., 1], [0, 4]])

# Parameter names
x_names = ["x1", "x2"]

# Number of evaluation points per coordinate
n_pts = 10
```

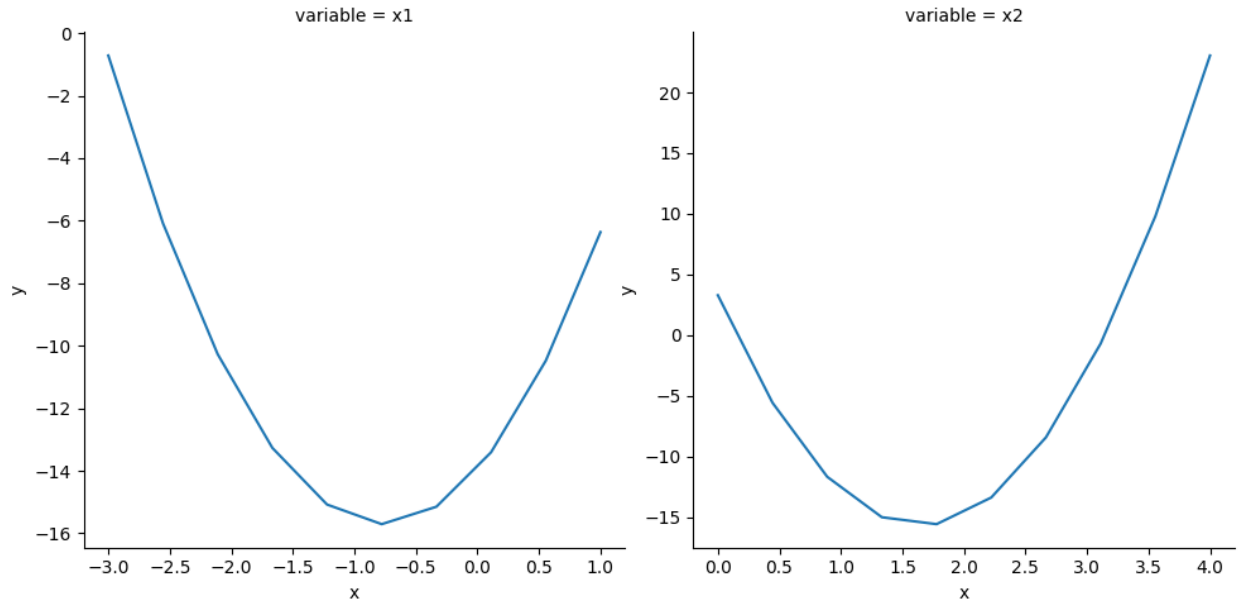
This package can be used with one function or with intermediary functions for more advanced users.

1.1.2 Basic Use Case

This example will walk through how to use the main function `projplot.proj_plot()`.

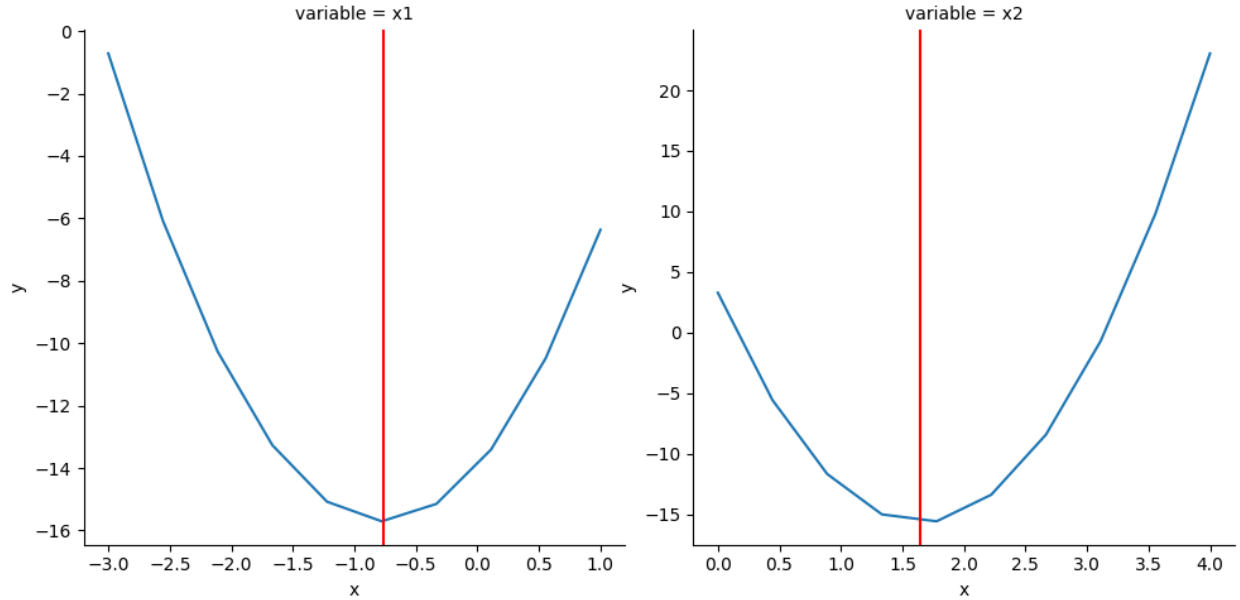
```
import projplot as pjp


def obj_fun(x):
    '''Compute x'Ax - 2b'x.'''
    y = np.dot(np.dot(x.T, A), x) - 2 * np.dot(b, x)
    return y


# Obtain plots without vertical x lines
pjp.proj_plot(obj_fun, x_opt=x_opt, x_lims=x_lims,
              x_names=x_names, n_pts=n_pts)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4fd4b71710>
```




```
# Obtain plots with vertical x lines
pjp.proj_plot(obj_fun, x_opt=x_opt, x_lims=x_lims,
              x_names=x_names, n_pts=n_pts,
              opt_vlines=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4fd21bad10>
```



1.1.3 Advanced Use Cases

In these cases, the calculation of the x-value matrix, projection DataFrame and plotting are done separately. Another added feature is that the user is able to plot vertical lines on the projection plots by providing an array whereas with `projplot.proj_plot()` this can only be done at the optimal values.

```
# Generate first round of x_values
x_vals = pjp.proj_xvals(x_opt, x_lims, n_pts)
x_vals
```

```
array([[-3.        ,  1.647     ],
       [-2.55555556,  1.647     ],
       [-2.11111111,  1.647     ],
       [-1.66666667,  1.647     ],
       [-1.22222222,  1.647     ],
       [-0.77777778,  1.647     ],
       [-0.33333333,  1.647     ],
       [ 0.11111111,  1.647     ],
       [ 0.55555556,  1.647     ],
       [ 1.        ,  1.647     ],
       [-0.765     ,  0.        ],
       [-0.765     ,  0.44444444],
       [-0.765     ,  0.88888889],
       [-0.765     ,  1.33333333],
       [-0.765     ,  1.77777778],
       [-0.765     ,  2.22222222],
       [-0.765     ,  2.66666667],
       [-0.765     ,  3.11111111],
       [-0.765     ,  3.55555556],
       [-0.765     ,  4.        ]])
```

```
# Obtain a DataFrame for plotting
plot_data = pjp.proj_data(obj_fun, x_vals, x_names)
plot_data
```

```
            y         x variable
0   -0.715737 -3.000000       x1
1   -6.084033 -2.555556       x1
2  -10.267144 -2.111111       x1
3  -13.265070 -1.666667       x1
4  -15.077811 -1.222222       x1
5  -15.705367 -0.777778       x1
6  -15.147737 -0.333333       x1
7  -13.404922  0.111111       x1
8  -10.476922  0.555556       x1
9   -6.363737  1.000000       x1
10   3.285675  0.000000       x2
11  -5.580498  0.444444       x2
12 -11.681239  0.888889       x2
13 -15.016547  1.333333       x2
14 -15.586424  1.777778       x2
15 -13.390868  2.222222       x2
16  -8.429881  2.666667       x2
```

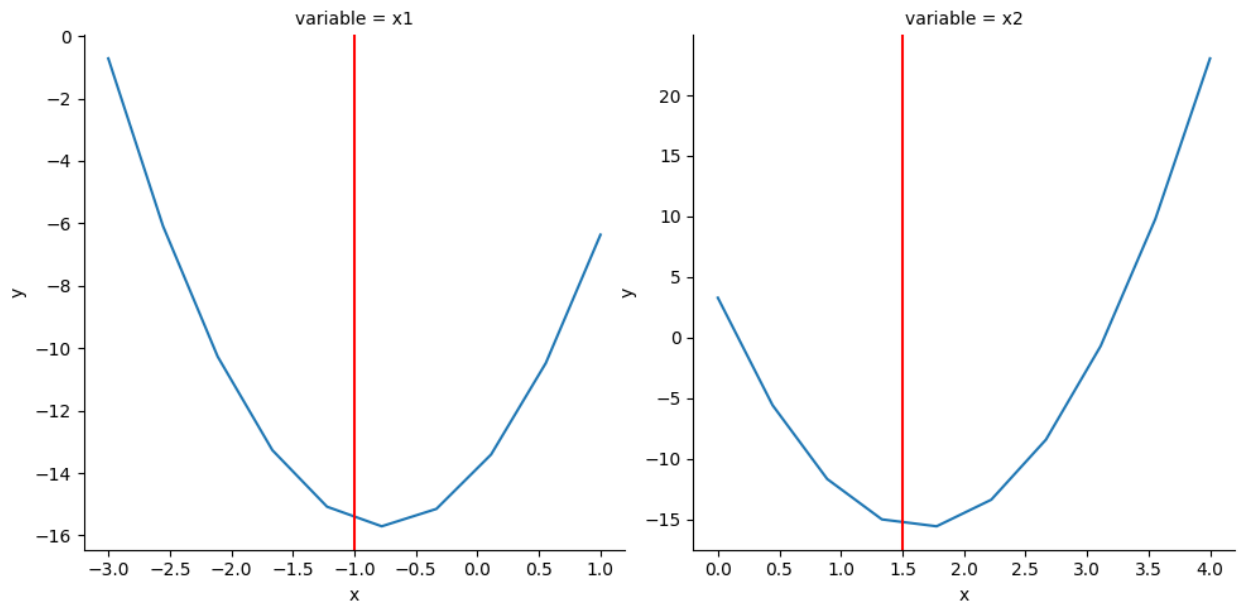
(continues on next page)

(continued from previous page)

```
17  -0.703461  3.111111       x2
18   9.788391  3.555556       x2
19  23.045675  4.000000       x2
```

```
# Plot vertical line at value specified by vlines
vlines = np.array([-1., 1.5]) # different from x_opt
pjp.proj_plot_show(plot_data, vlines=vlines)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4fd48bc690>
```



Vectorized Function

In the above, `obj_fun()` can only take a single vector `x` at a time. Inside `projplot.proj_plot()` (or `projplot.proj_data()`) the function is run through a for-loop on each value of `x`. Alternatively, `obj_fun()` can be vectorized over each row of `x` by providing the **projplot** functions with the argument `vectorized=True`.

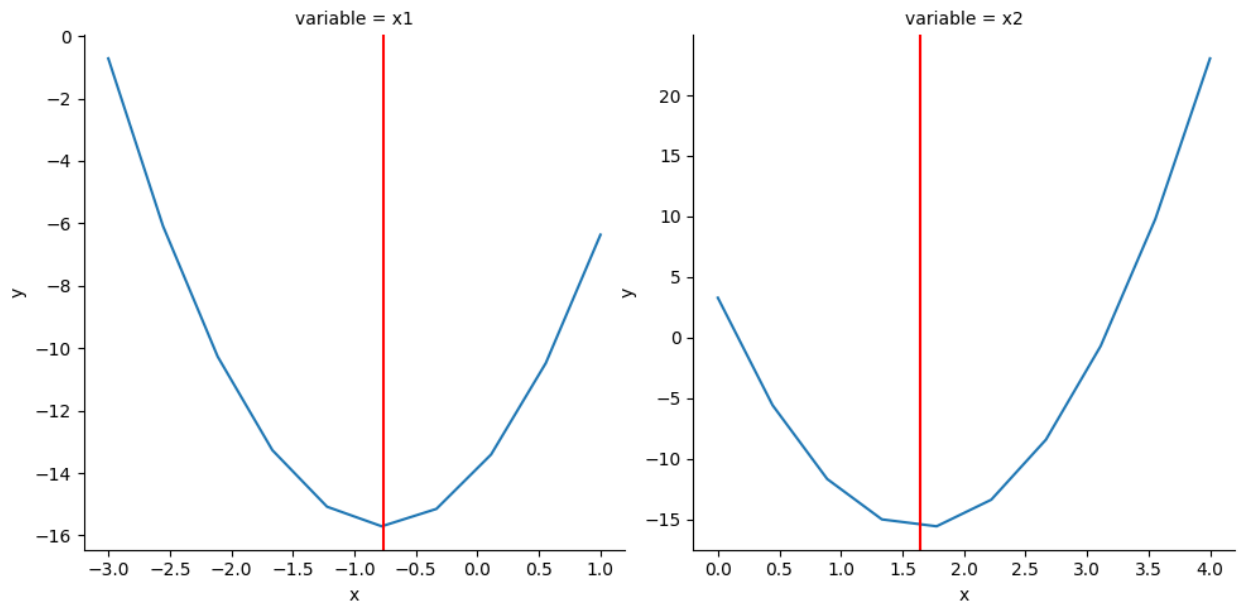
```
def obj_fun_vec(x):
    '''
    Vectorized computation of x'Ax - 2b'x.

    Params:
        x: A nx2 vector.

    Returns:
        The output of x'Ax - 2b'x applied to each row of x.
    '''
    x = x.T
    y = np.diag(x.T.dot(A).dot(x)) - 2 * b.dot(x)
    return y
```

(continues on next page)

(continued from previous page)

```
pjp.proj_plot(obj_fun_vec, x_opt=x_opt, x_lims=x_lims,
              x_names=x_names, n_pts=n_pts,
              vectorized=True,
              opt_vlines=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4fd1df1350>
```



We can see that the produced plots for the vectorized and non-vectorized function are identical. Vectorized functions have the advantage of running more efficiently; however, they are not necessary to utilize **projplot**.

1.2 FAQ

Does my function need to be vectorized?

No, it does not need to be vectorized in order for you to use this tool. There is a `vectorized` parameter that allows for both vectorized functions and non-vectorized functions. If your function is not vectorized, we will iterate through the x-values to generate the x-value matrix that will be used for the projection plots. If your function is vectorized, this will run more efficiently with generating the projection plots.

What is the point of generating the x-value matrix separately?

The x-value matrix generates the combinations with the varying parameters that we will be inputting into the objective function to visualize the resulting changes in the output. By having this outputted separately, the user is able to view the values that will be inputted prior to plotting and alter it. In the future, an `equalize()` function will be added to fine-tune the scale to be more accurate. An example of what the x-value matrix looks like is given below (based on the documentation *example*):

```
import numpy as np
import projplot as pjp

# setup
```

(continues on next page)

(continued from previous page)

```
x_opt = np.array([-0.765,  1.647]) # optimal values
x_lims = np.array([[-3., 1], [0, 4]]) # plot limits
n_pts = 10 # number of plotting points

x_vals = pjp.proj_xvals(x_opt, x_lims, n_pts)
x_vals
```

```
array([[-3.        ,  1.647     ],
       [-2.55555556,  1.647     ],
       [-2.11111111,  1.647     ],
       [-1.66666667,  1.647     ],
       [-1.22222222,  1.647     ],
       [-0.77777778,  1.647     ],
       [-0.33333333,  1.647     ],
       [ 0.11111111,  1.647     ],
       [ 0.55555556,  1.647     ],
       [ 1.        ,  1.647     ],
       [-0.765     ,  0.        ],
       [-0.765     ,  0.44444444],
       [-0.765     ,  0.88888889],
       [-0.765     ,  1.33333333],
       [-0.765     ,  1.77777778],
       [-0.765     ,  2.22222222],
       [-0.765     ,  2.66666667],
       [-0.765     ,  3.11111111],
       [-0.765     ,  3.55555556],
       [-0.765     ,  4.        ]])
```

Can I see the data that is plotted as a DataFrame?

Yes, if you want to see the data that is being plotted as a `pandas.DataFrame`, you can set `plot=False` in `projplot.proj_plot()` and it will return the DataFrame of values that would have been plotted. If we were to assign this to a variable `plot_data` and call it, we would have the following DataFrame outputted (based on the documentation *example*):

```
x_names = ["x1", "x2"] # variable names

# objective function
def obj_fun(x):
    A = np.array([[3., 2.],
                  [2., 7.]])
    b = np.array([1., 10.])
    y = np.dot(np.dot(x.T, A), x) - 2 * np.dot(b, x)
    return y

plot_data = pjp.proj_data(obj_fun, x_vals, x_names)
plot_data
```

```
            y         x variable
0   -0.715737 -3.000000       x1
1   -6.084033 -2.555556       x1
2  -10.267144 -2.111111       x1
```

(continues on next page)

(continued from previous page)

```
3  -13.265070 -1.666667       x1
4  -15.077811 -1.222222       x1
5  -15.705367 -0.777778       x1
6  -15.147737 -0.333333       x1
7  -13.404922  0.111111       x1
8  -10.476922  0.555556       x1
9   -6.363737  1.000000       x1
10   3.285675  0.000000       x2
11  -5.580498  0.444444       x2
12 -11.681239  0.888889       x2
13 -15.016547  1.333333       x2
14 -15.586424  1.777778       x2
15 -13.390868  2.222222       x2
16  -8.429881  2.666667       x2
17  -0.703461  3.111111       x2
18   9.788391  3.555556       x2
19  23.045675  4.000000       x2
```

Do I have to include names for each parameter?

No, as a default if the list of names is empty, the function will label them `x1`, `x2`, ..., `xd` based on d parameters.

What is the point of the `opt_vlines` and `vlines` parameters?

This allows the user to see where the solution for each parameter lies on the plot. For exxample, if the projection plot is given for values between -2 and 2 and was minimized at 0, if we believed the minimum was at -1, we would be able to visually tell that our optimization didn't work since the vertical line would not be at 0.

With `projplot.proj_plot()` you are only able to plot vertical lines at the optimal values using `opt_vlines`. However, for the more advanced users, vertical lines (`vlines`) can be plotted at any values as long as an array is provided that is the length of the parameters for `projplot.proj_plot_show()`.

1.3 API Reference

This page contains auto-generated API reference documentation[1].

1.3.1 `projplot`

Submodules

`projplot.projplot`

Module Contents

[1] Created with sphinx-autoapi

Functions

proj_xvals(x_opt, x_lims, n_pts)	Generate a matrix of projection plot x-values.
proj_plot_show(plot_data[, vlines])	Create a projection plot based on the output of `proj_data()`.
proj_data(fun, x_vals[, x_names, vectorized])	Generate projection plot data from the objective function and an x-value matrix returned by `proj_xvals()`.
proj_plot(fun, x_opt, x_lims[, x_names, n_pts, ...])	Generate projection plots.

`projplot.projplot.`**`proj_xvals`**(*x_opt*, *x_lims*, *n_pts*)

Generate a matrix of projection plot x-values.

Parameters

- **x_opt** (*NumPy array*) – An array of parameter values.
- **x_lims** (*NumPy array*) – An array of limits or a 2 x x_opt.shape[0] matrix of lower and upper limits for each parameter.
- **n_pts** (*int*) – The number of points to plot.

Returns

An array of all possible combinations of the x-values based on the limits (x_lims) and optimal values (x_opt).

Return type

(NumPy array)

Example

```
>>> proj_xvals(np.array([1,15]), np.array([[0,2], [10, 20]]), 3)
array([[ 0., 15.],
       [ 1., 15.],
       [ 2., 15.],
       [ 1., 10.],
       [ 1., 15.],
       [ 1., 20.]])
```

`projplot.projplot.`**`proj_plot_show`**(*plot_data*, *vlines=None*)

Create a projection plot based on the output of `proj_data()`.

Parameters

- **plot_data** (*DataFrame*) – A DataFrame that contains columns for the calculated y-value, varying x value and the respective *x_opt* name associated with the varying x.
- **vlines** (*optional Array*) – An array of x-values to plot a vertical line at for each projection plot. The length of this array should equal the number of parameters being optimized.

Returns

A plot for each unique *x_opt* using the x and y values in *plot_data* with optional vertical lines.

`projplot.projplot.`**`proj_data`**(*fun*, *x_vals*, *x_names=[]*, *vectorized=False*)

Generate projection plot data from the objective function and an x-value matrix returned by `proj_xvals()`.

Parameters

- **fun** (*Python function*) – The objective function that is being optimized.
- **x_vals** (*NumPy array*) – A matrix of the x_vals, this should be outputted from proj_xvals().
- **x_names** (*optional List*) – A list of the names respective to varying x-values for plotting.
- **vectorized** (*Bool*) – True if the objective function is vectorized, else False.

Returns
DataFrame with columns *y*, *x*, and *variable* containing: the y-value in each projection plot; the corresponding x-values; and the name of the variables in *x_names*.

Return type
(pandas.DataFrame)

projplot.projplot.**proj_plot**(*fun*, *x_opt*, *x_lims*, *x_names=None*, *n_pts=100*, *vectorized=False*, *plot=True*, *opt_vlines=False*)

Generate projection plots.

Parameters

- **fun** (*Python function*) – The objective function that is being optimized
- **x_opt** (*NumPy array*) – An array of parameter values
- **x_lims** (*NumPy array*) – An array of limits or a 2 x x_opt.shape[0] matrix of lower and upper limits for each parameter
- **x_names** (*anyof List None*) – A list of the names respective to varying x-values for plotting
- **n_pts** (*int*) – The number of points to plot
- **vectorized** (*Bool*) – TRUE if the objective function is vectorized, else FALSE
- **plot** (*Bool*) – TRUE if the user wants a plot outputted, else FALSE
- **opt_vlines** (*Bool*) – If this parameter is set to True, then a vertical line is plotted at each parameter's optimal value. The default is set to False.

Returns
If `plot=False`, the y-value in each projection plot appended to the x-values in a DataFrame format that's amenable to plotting is returned. If `plot=True`, a plot handle of the projection plots, which is a plot for each varying *x_opt* is returned and the plot is displayed.

Package Contents

Functions

proj_xvals(x_opt, x_lims, n_pts)	Generate a matrix of projection plot x-values.
proj_plot_show(plot_data[, vlines])	Create a projection plot based on the output of `proj_data()`.
proj_data(fun, x_vals[, x_names, vectorized])	Generate projection plot data from the objective function and an x-value matrix returned by `proj_xvals()`.
proj_plot(fun, x_opt, x_lims[, x_names, n_pts, ...])	Generate projection plots.

projplot.**proj_xvals**(*x_opt*, *x_lims*, *n_pts*)

Generate a matrix of projection plot x-values.

Parameters

- **x_opt** (*NumPy array*) – An array of parameter values.
- **x_lims** (*NumPy array*) – An array of limits or a 2 x x_opt.shape[0] matrix of lower and upper limits for each parameter.
- **n_pts** (*int*) – The number of points to plot.

Returns
An array of all possible combinations of the x-values based on the limits (x_lims) and optimal values (x_opt).

Return type
(NumPy array)

Example

```
>>> proj_xvals(np.array([1,15]), np.array([[0,2], [10, 20]]), 3)
array([[ 0., 15.],
       [ 1., 15.],
       [ 2., 15.],
       [ 1., 10.],
       [ 1., 15.],
       [ 1., 20.]])
```

projplot.**proj_plot_show**(*plot_data*, *vlines=None*)

Create a projection plot based on the output of `proj_data()`.

Parameters

- **plot_data** (*DataFrame*) – A DataFrame that contains columns for the calculated y-value, varying x value and the respective *x_opt* name associated with the varying x.
- **vlines** (*optional Array*) – An array of x-values to plot a vertical line at for each projection plot. The length of this array should equal the number of parameters being optimized.

Returns
A plot for each unique *x_opt* using the x and y values in *plot_data* with optional vertical lines.

projplot.**proj_data**(*fun*, *x_vals*, *x_names=[]*, *vectorized=False*)

Generate projection plot data from the objective function and an x-value matrix returned by `proj_xvals()`.

Parameters

- **fun** (*Python function*) – The objective function that is being optimized.
- **x_vals** (*NumPy array*) – A matrix of the x_vals, this should be outputted from proj_xvals().
- **x_names** (*optional List*) – A list of the names respective to varying x-values for plotting.
- **vectorized** (*Bool*) – True if the objective function is vectorized, else False.

Returns
DataFrame with columns *y*, *x*, and *variable* containing: the y-value in each projection plot; the corresponding x-values; and the name of the variables in *x_names*.

Return type
(pandas.DataFrame)

projplot.**proj_plot**(*fun*, *x_opt*, *x_lims*, *x_names=None*, *n_pts=100*, *vectorized=False*, *plot=True*, *opt_vlines=False*)

Generate projection plots.

Parameters

- **fun** (*Python function*) – The objective function that is being optimized
- **x_opt** (*NumPy array*) – An array of parameter values
- **x_lims** (*NumPy array*) – An array of limits or a 2 x x_opt.shape[0] matrix of lower and upper limits for each parameter
- **x_names** (*anyof List None*) – A list of the names respective to varying x-values for plotting
- **n_pts** (*int*) – The number of points to plot
- **vectorized** (*Bool*) – TRUE if the objective function is vectorized, else FALSE
- **plot** (*Bool*) – TRUE if the user wants a plot outputted, else FALSE
- **opt_vlines** (*Bool*) – If this parameter is set to True, then a vertical line is plotted at each parameter's optimal value. The default is set to False.

Returns

If `plot=False`, the y-value in each projection plot appended to the x-values in a DataFrame format that's amenable to plotting is returned. If `plot=True`, a plot handle of the projection plots, which is a plot for each varying *x_opt* is returned and the plot is displayed.

PYTHON MODULE INDEX

p

INDEX

M

P