

Supporting Information for:

## **DNAplotlib: programmable visualization of genetic designs and associated data**

*Bryan S. Der, Emerson Glassey, Bryan A. Bartley, Casper Enghuus, Daniel B. Goodman, D. Benjamin Gordon, Christopher A. Voigt and Thomas E. Gorochoowski*

### **Supplementary Text**

|   |   |
|---|---|
| 1. Rendering Pipeline                             | 2 |
| 1.1 Overview of DNARenderer                       | 2 |
| 1.2 Considerations when Developing Visualizations | 3 |
| 2. Part Renderers                                 | 3 |
| 2.1 SBOL Renderers                                | 4 |
| 2.2 Trace Renderers                               | 5 |
| 3. Regulation Renderers                           | 6 |
| 3.1 Standard Regulation Renderers                 | 7 |
| 4. Input-Output                                   | 7 |
| 4.1 Loading Design and Experimental Data          | 7 |
| 4.2 Loading Design Data from SBOL Files           | 8 |
| 4.3 Saving Visualizations to File                 | 8 |

## 1. Rendering Pipeline

### 1.1 Overview of DNARenderer

To visualize a genetic design, the DNARenderer object is used. This implements the main rendering pipeline (Figure 4a). To create this object a number of general parameters can be provided:

**DNARenderer** (scale=1.0, linewidth=1.0, linecolor=(0,0,0), backbone\_pad\_left=0.0, backbone\_pad\_right=0.0)

- *scale* : The scale that parts should be drawn at (default = 1).
- *linewidth* : Global line width to draw the design with (default = 1; can be overwritten by part or regulation customization).
- *linecolor* : Global line color to draw the design with (default is black; can be overwritten by part or regulation customization).
- *backbone\_pad\_left* : Padding to include at start of design (default = 0).
- *backbone\_pad\_right* : Padding to include at end of design (default = 0).

Once this object has been created it can be used to visualize as many designs as necessary. A number of methods are included that implement the main rendering pipeline, simplify the selection of part and regulation renderers, and allow annotation of existing designs.

#### **SBOL\_part\_renderers ()**

Returns a dictionary that maps each standard part type to the built-in SBOL part rendering functions (see Section 2 for a full list).

#### **trace\_part\_renderers ()**

Returns a dictionary that maps each standard part type to the built-in trace-based part rendering functions (see Section 2 for a full list).

#### **std\_reg\_renderers ()**

Returns a dictionary that maps each standard regulation type to the built-in regulation rendering functions (see Section 3 for a full list).

#### **renderDNA** (ax, parts, part\_renderers, regs=None, reg\_renderers=None, plot\_backbone=True)

This function implements the rendering pipeline. The user must provide the matplotlib axis to draw to (ax), the design of the construct as a list of parts (parts), and the part renderers that should be used for each part type encountered in the design (part\_renderers). Optionally, a set of regulatory links can be provided (regs), regulation rendering functions for displaying these (reg\_renderers), and a flag to state whether the backbone line should be plotted (plot\_backbone). Once called, the function cycles through each part in the design and calls the relevant part renderer. Then, if present, all regulatory links are processed in a similar way using the regulation renderers provided. Finally, the start and end position of the design are returned to simplify the setting of x-axis limits.

**annotate** (ax, part\_renderers, part, annotate\_zorder=1000)

Once a design has been rendered it may be necessary to add further annotations to highlight points of interest. To do this, the `annotate` function allows any part renderer to be called to draw any element at a specified location. It requires an axis to draw to (ax), a list of part renderers that are available (part\_renderers), and a part object (part). As annotations must sit on top of any existing design, by default they are rendered at a position above a normal design with a `zorder = 1000`. This can be varied to layer annotations on top of each other with parts having a higher `annotate_zorder` being placed above those with lower values.

### 1.2 Considerations when Developing Visualizations

All drawing by DNAplotlib is performed to a matplotlib axis. As DNAplotlib does not directly manipulate properties of this axis it is essential that the size and aspect ratio be defined correctly. If using SBOLv part renderers then this means using the `set_aspect('equal')` function such that parts are drawn and correctly sized. For trace-based renderers this is not necessary. However, the `scale` property should be used when calling the `DNARenderer.renderDNA(...)` method to account for differences in the length of a design. This parameter will stretch the symbols to ensure they are displayed correctly. We recommend consulting the examples gallery on the project website to understand the standard settings and stages of creating visualizations of differing types.

## 2. Part Renderers

Each individual part within a design is drawn by a part renderer. Functions for each type of part must be provided to the `DNARenderer.renderDNA(...)` method when it is invoked. It then coordinates the calling of these functions to draw the entire design. To enable this process to work, part renderers are simply functions that draw a specific type of part and have the general function signature:

**part\_renderer** (ax, type, num, start, end, prev\_end, scale, linewidth, opts)

- **ax** : Axis object to render the part to.
- **type** : Type of part to be drawn (this allows for a single function to potentially be used to draw multiple types of part).
- **num** : The part index in the design.
- **start** : Start position of the part.
- **end** : End position of the part.
- **prev\_end** : End position of the previous part drawn.
- **scale** : The scale that the part should be drawn at (default = 1).
- **linewidth** : Global line width to draw with, unless this is overridden in opts.
- **opts** : Dictionary of customization options (see Figure 5 for details).

As each part is processed in the design, the corresponding function for that type is called with all of the required parameters above. The appearance can be customized by including an `opts` element in the

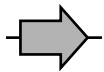
definition of the part (see Figure 5 for details). Built-in part rendering functions are described in the following sections.

## 2.1 SBOL Renderers

These capture the complete set of current SBOLv standardized parts. When drawing a design with these renderers the relative lengths of each part are ignored and only ordering and orientation information are maintained.



**sbol\_promoter (...)**  
Draws a promoter part.



**sbol\_cds (...)**  
Draws a coding region part.



**sbol\_terminator (...)**  
Draws a terminator.



**sbol\_rbs (...)**  
Draws a ribosome binding site (RBS).



**sbol\_ribozyme (...)**  
Draws a ribozyme site.



**sbol\_protein\_stability (...)**  
Draws a protein stability site.



**sbol\_protease (...)**  
Draws a protease site.



**sbol\_ribonuclease (...)**  
Draws a ribonuclease site.



**sbol\_scar (...)**  
Draws a scar. Normally used for scars introduced during cloning procedures.














**sbol\_empty\_space (...)**  
Draws an empty space. Useful for altering the spacing between groups of elements.



**sbol\_5\_overhang (...)**  
Draws a 5'-end overhang.







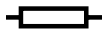
**sbol\_3\_overhang (...)**  
Draws a 3'-end overhang.

|   |   |
|---|---|
|    | <b>sbol_blunt_restriction_site (...)</b><br>Draws a blunt-end restriction enzyme cut site.        |
|    | <b>sbol_primer_binding_site (...)</b><br>Draws a primer binding site.                             |
|    | <b>sbol_5_sticky_restriction_site (...)</b><br>Draws a 5' sticky end restriction enzyme cut site. |
|    | <b>sbol_3_sticky_restriction_site (...)</b><br>Draws a 3' sticky end restriction enzyme cut site. |
|    | <b>sbol_user_defined (...)</b><br>Draws a user-defined component. Shown as a simple rectangle.    |
|    | <b>sbol_signature (...)</b><br>Draws a signature element.   |
|    | <b>sbol_restriction_site (...)</b><br>Draws a standard restriction enzyme cut site.               |
|   | <b>sbol_spacer (...)</b><br>Draws a spacer element.   |
|  | <b>sbol_origin (...)</b><br>Draws an origin of replication.                                       |
|  | <b>sbol_operator (...)</b><br>Draws an operator site.   |
|  | <b>sbol_insulator (...)</b><br>Draws an insulating element.                                       |

## 2.2 Trace Renderers

Because SBOLv parts do not provide information regarding the relative lengths of each element, they are unsuitable for plotting designs against data collected at a nucleotide-level resolution. To accommodate this use-case we provide a set of additional trace-based renderers where each element maintains the correct relative length within the design. This makes it possible to use standard plotting functions for bar or line graphs to display associated experimental information if the same x-axis limits are used.

|   |   |
|---|---|
|  | <b>trace_promoter (...)</b><br>Draws a promoter symbol with the actual relative extent of the part shown on the DNA backbone as a filled rectangle. |
|---|---|

- 
**trace\_rbs (...)**  
 Draws a ribosome binding site with the actual relative extent of the part shown on the DNA backbone as a filled rectangle.
- 
**trace\_cds (...)**  
 Draws a coding sequence with the length of the part relative to the actual size.
- 
**trace\_terminator (...)**  
 Draws a terminator symbol with the actual relative extent of the part shown on the DNA backbone as a filled rectangle.
- 
**trace\_user\_defined (...)**  
 Draws a user-defined part as a rectangle with a length relative to the actual size.

### 3. Regulation Renderers

It is often necessary to depict interactions between parts in a design. This is possible through the use of regulation arcs that are rendered after all parts have been drawn. Regulation arcs can potentially define any type of interaction. However, because genetic systems often include regulation that is linked to activation or repression, we include by default these two forms, in addition to a normal undirected connection type. Regulation renderers are simply functions that draw a connection between a source and target part, having the general function signature:

**regulation\_renderer** (ax, type, num, from\_part, to\_part, scale, linewidth, arc\_height\_index, opts)

- *ax* : Axis object to render the regulation to.
- *type* : Type of regulation arc to be drawn (this allows for a single function to potentially be used to draw multiple types of regulation).
- *num* : The regulation arc index in the design.
- *from\_part* : The source of the regulation arc.
- *to\_part* : The target of the regulation arc.
- *scale* : The scale that the regulation arc should be drawn at (default = 1).
- *linewidth* : Global line width to draw with, unless this is overridden in opts.
- *arc\_height\_index* : The height of each regulation arc is tailored to reduce overlapping links. The height index is the level that this arc should be drawn at.
- *opts* : Dictionary of customization options (see Figure 5 for details).

As with part renderers, regulation renderer functions are stored as a dictionary defining how different types of regulation should be drawn and provided to the `DNARenderer.renderDNA()` method when a visualization is rendered. As each regulation arc is processed in the design, the corresponding function for that type of regulation is called with all of the required parameters above. Similar to part renderers, aspects of the appearance can be customized by including an `opts` element in the definition of the regulation arc (see Figure 5 for details). Built-in regulation rendering functions are described in the following section.

### 3.1 Standard Regulation Renderers



#### **connect (...)**

Draws a non-directed connection between source and target parts as a line. Orientation of the target dictates if line is above or below the backbone.



#### **repress (...)**

Draws a repression arc (line with bar) between source and target parts. Orientation of the target dictates if arc is above or below the backbone.



#### **induce (...)**

Draws an activation arc (arrow) between source and target parts. Orientation of the target dictates if arc is above or below the backbone.

## 4. Input-Output

To simplify the creation of designs and provide access to numerous annotated sequences, we include several functions for loading design and experimental data, as well as access to functions for outputting completed visualization in a wide range of formats.

### 4.1 Loading Design and Experimental Data

#### **load\_design\_from\_gff** (filename, chrom, type\_map=dpl\_default\_type\_map, region=None)

Loads a DNAPlotlib compatible design from a standard GFF file. It assumes that this file contains 9 columns and at least a “Name” attribute is specified as a parameter for each annotation. Additional parameters are automatically loaded into the opts dictionary and can be used for styling an element in the design. As GFF files can contain multiple chromosomes, the chromosome of interest must be specified with the chrom parameter. Furthermore, because the type of annotation element within the GFF file might differ from what is understood by DNAPlotlib the type\_map parameter should be specified to define a dictionary mapping a type in the file to a type in the loaded design. By default, we include a standard mapping as follows:

```
dpl_default_type_map = { 'gene':      'CDS',
                        'promoter':  'Promoter',
                        'terminator': 'Terminator',
                        'rbs':       'RBS'}
```

The region loaded can be restricted using the region parameter which has the format [start\_bp, end\_bp]. If it is not provided (or set to ‘None’), all annotation elements for the chromosome are included.

#### **load\_profile\_from\_bed** (filename, chrom, region)

Loads nucleotide-level profile data generated by BEDTools. The user must provide a filename, chromosome for the profile (chrom) and a region in the format [start\_bp, end\_bp]. If the profile is contained within the file, the function returns the profile data as a list of floating point values. If the profile cannot be found in the file, then ‘None’ is returned.

#### *4.2 Loading Design Data from SBOL Files*

The Synthetic Biology Open Language<sup>1</sup> (SBOL) provides a standard for the exchange and serialization of genetic design data. DNAPlotlib is able to exploit this standard through a plugin to the pySBOL library (<https://github.com/SynBioDex/pySBOL>).

#### *4.3 Saving Visualizations to File*

Visualizations can be output to file by calling the `savefig()` function from matplotlib. The format of the output is automatically determined by the extension of the filename provided. Supported formats will depend on the users matplotlib installation, but generally includes: PNG, JPEG, PDF, PostScript (PS), Encapsulated PostScript (EPS) and Scalable Vector Graphics (SVG) formats. Other options can also be provided to alter the resolution in dots per inch (dpi) and whether the background should be transparent (if supported by the file format). See the matplotlib documentation for full details. For more complex visualizations that include animations, we recommend exporting each frame to a single image file and then combining these using an external tool such as Quicktime or MPlayer to encode the final video.