Cookbook

pydna

Björn Johansson
CBMA

University of Minho

Braga

Portugal

What is pydna?

Pydna is a python package that provides functions and data types to deal with double stranded DNA. It depends on Biopython (a python bioinformatics package), networkx (a graph theory package) and numpy (a mathematics package).

What does Python dna provide?

Python dna provide classes and functions for molecular biology using python. Notably, PCR, cut and paste cloning (sub-cloning) and homologous recombination between linear DNA fragments are supported. Most functionality is implemented as methods for the double stranded DNA sequence record classes “Dseq” and "Dseqrecord", which are a subclasses of the Biopython Seq and SeqRecord classes, respectively.

Pydna was designed to semantically imitate how sub-cloning experiments are typically documented in scientific literature. One use case for pydna is to create executable documentation for a sub-cloning experiment. The pydna code unambiguously describe the experiment, and can be executed to yield the sequence of the of the resulting DNA molecule(s) and all intermediary steps. Pydna code describing a sub cloning is reasonably compact and also meant to be easily readable.

Typical usage at the command line could look like this:

```
>>> import pydna
>>> seq = pydna.Dseq("GGATCCAAA","TTTGGATCC", ovhg=0)
>>> seq
Dseq(-9)
GGATCCAAA
CCTAGGTTT
```

The example above shows an example usage of the Dseq class which is a double stranded version of the Biopython seq class. This is the main pydna data type along with the Dseqrecord class which is a double stranded version of the Biopython SeqRecord class.

The Dseq object was initialized using two strings and a value for the stagger (ovhg) between the DNA strands in the 5' (left) extremity. This is of course not a practical way of creating a Dseq object in most cases, but there are other more practical methods as we will see further on.

The Dseq object comes with a cut method that takes one or more restriction

enzymes as arguments. A list is returned with the fragments produced in the digestion:

```
>>> from Bio.Restriction import BamHI
>>> a,b = seq.cut(BamHI)
>>> a
Dseq(-5)
G
CCTAG
>>> b
Dseq(-8)
GATCCAAA
    GTTT
```

The fragments a and b formed in the example above can be religated together by the addition operator:

```
>>> a+b
Dseq(-9)
GGATCCAAA
CCTAGGTTT
>>> b+a
Dseq(-13)
GATCCAAAG
    GTTTCCTAG
>>> b+a+b
Dseq(-17)
GATCCAAAGGATCCAAA
    GTTTCCTAGGTTT
```

The Dseq objects keep track of the structure of the DNA ends and only allow ligation of compatible fragments:

```
>>> b+a+a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
   File "/usr/local/lib/python2.7/dist-packages/pydna/dsdna.py", line
217, in __add__
    raise TypeError("sticky ends not compatible!")
TypeError: sticky ends not compatible!
>>>
```

Two examples are given in this tutorial (Example 1 and 2). The data files that are referred to in this document can be found in the folder “cookbook_files” that was downloaded together with this file. Alternatively, the examples can be solved on-line using pydna live.

pydna live

Python 2.7.3 with pydna and Biopython are avaliable for testing interactively online at http://pydna-shell.appspot.com/ (Fig1).

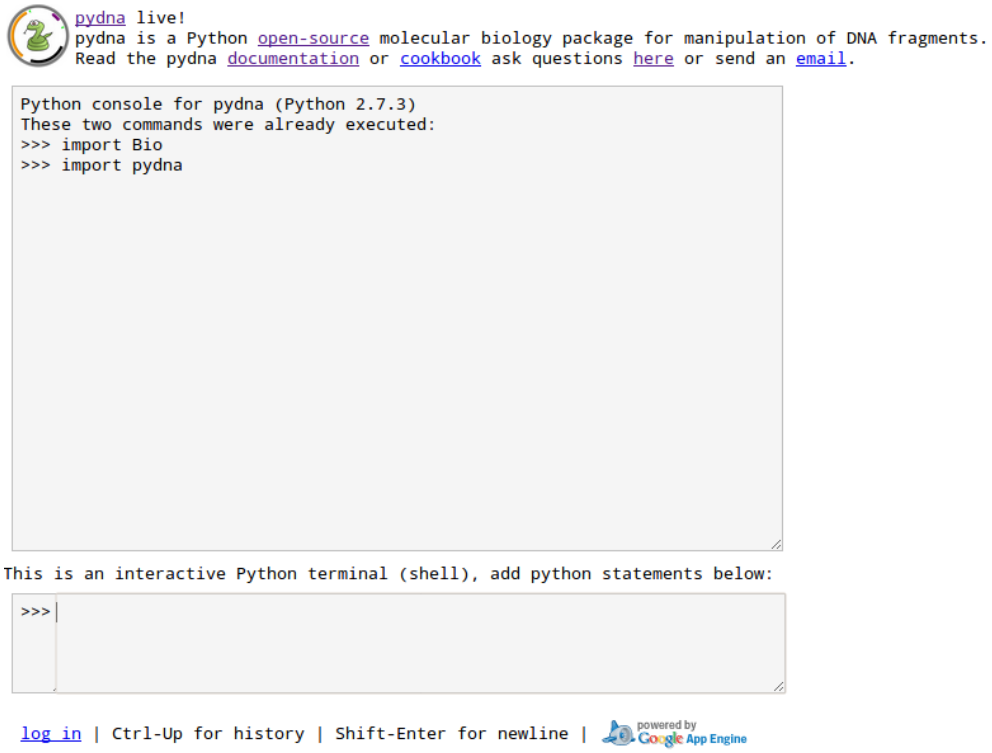

Fig. 1

The Biopython package is not completely supported by pydna live since pydna live runs on the google app engine, which currently does not permit C-extensions. However, all functionality needed for pydna is provided.

All files referred to in this cookbook are provided in the sub directory “cookbook_files”. This means that you can execute the statements given here directly as they are written by copy and paste (leaving out the prompt “>>>”). If you perform these examples on your own system, you have to adjust file paths when reading and writing files.pydna_cookbook_0-5-0 (copy)

Example 1: Sub cloning by restriction digestion and ligation

The construction of the vector YEp24PGK_XK is described on page 4250 in the publication below:

Johansson et al., “Xylulokinase Overexpression in Two Strains of Saccharomyces cerevisiae Also Expressing Xylose Reductase and Xylitol Dehydrogenase and Its Effect on Fermentation of Xylose and Lignocellulosic Hydrolysate” Applied and Environmental Microb

Briefly, the XKS1 gene from *Saccharomyces cerevisiae* is amplified by PCR using two primers called primer1 and primer3. The primers add restriction sites for *Bam*HI to the ends of the XKS1 gene. The gene is digested with *Bam*HI and ligated to the YEp24PGK plasmid that has previously been digested with *Bgl*II which cut the plasmid in one location. The two enzymes are compatible so fragments cut with either enzyme can be ligated together. Fig 1 shows an image outlining the strategy.

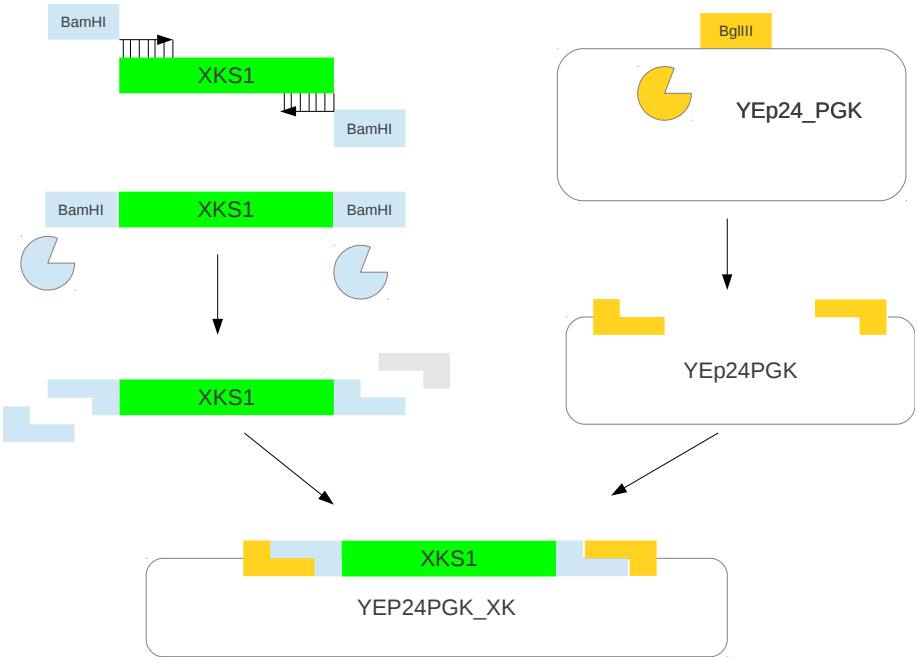

Fig. 2: Construction of the YEp24PGK_XK vector

We will replicate this cloning strategy using the tools provided by pydna.

Start your interactive python session and import the pydna module.

```
Python 2.7.3 (default, Aug  1 2012, 05:14:39)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import pydna
>>> pydna.__version__
'0.6.0'
```

Use the pydna read function to assign the primers and template sequence to SeqRecord objects for the primers (they are single stranded) and a Dseqrecord object for the template.

```
>>> p1 = pydna.read("primer1.txt", ds=False)
>>> p3 = pydna.read("primer3.txt", ds=False)
>>> XKS1 = pydna.read("XKS1_orf.txt")
```

We use the pydna PCR function to make the PCR product from the primers and the template sequence. The result should be a linear Dseqrecord of 1829 bp.

```
>>> PCR_prod = pydna.pcr(p1 ,p3 ,XKS1)
>>> PCR_prod
Amplicon(1829)
```

We then cut the PCR product with BamHI.

```
>>> from Bio.Restriction import BamHI
>>> stuffer1, insert, stuffer2 = PCR_prod.cut(BamHI)
```

The stuffer1 and stuffer2 sequences are the small DNA pieces at each end. The seq property shows the Dseq object that is held by the of the Dseqrecord object.

```
>>> stuffer1.seq
Dseq(-7)
GCG
CGCCTAG
>>> stuffer2.seq
Dseq(-11)
GATCCAGATCT
    GTCTAGA
>>> insert.seq
Dseq(-1819)
GATCCTCTAGAATGGTTTGT...GGAAAAGACTCTCATCTAAG
    GAGATCTTACCAAACA...CCTTTTCTGAGAGTAGATTCCTAG
```

We then cut the YEp24PGK plasmid with the BglII enzyme.

```
>>> YEp24PGK = pydna.read("YEp24PGK.txt")
```

```
>>> from Bio.Restriction import BglII
>>> YEp24PGK_BglII = YEp24PGK.cut(BglII).pop()
```

We now have two linear DNA molecules insert and YEp24PGK_BglII. We then add them together to form a larger linear molecule:

```
>>> YEp24PGK_XK = YEp24PGK_BglII + insert
>>> YEp24PGK_XK
Dseqrecord(-11456)
```

The plasmid is still linear, but we can change this property with the looped method.

```
>>> YEp24PGK_XK = YEp24PGK_XK.looped()
>>> YEp24PGK_XK
Dseqrecord(o11452)
```

The number indicate the size and the “o” that the DNA is circular. The molecule appears smaller, but this is since the sticky ends annealed together. The sync method will rotate the new plasmid, so that it starts at the same position as the old plasmid. This makes the final sequence easier to read.

```
>>> YEp24PGK_XK = YEp24PGK_XK.synced(YEp24PGK)
```

The seguid method give the seguid of the sequence.

```
>>> YEp24PGK_XK.seguid()
'HRVpCEKWcFsKhw/W+25ednUfldI'
```

We then write the plasmid to a file:

```
>>> YEp24PGK_XK.write("YEp24PGK_XK_vector.gb")
```

You can now open the saved sequence file with your favorite sequence editor.

If you are doing these examples on pydna live, the google app engine does not permit saving files. You can print the content of the file instead:

```
>>> print YEp24PGK_XK.format("gb")
```

Beware! This is a long output. The output window can be made wider if necessary for a correct print out.

Example 2: Sub cloning by homologous recombination

The construction of the vector pGUP1 is described in the publication:

Régine Bosson, Malika Jaquenoud, and Andreas Conzelmann, “GUP1 of Saccharomyces Cerevisiae Encodes an O-acyltransferase Involved in Remodeling of the GPI Anchor,” Molecular Biology ofthe Cell 17, no. 6 (June 2006): 2636–2645.

Our objective is to replicate the cloning steps using pydna so that we can have the final sequence of the plasmid.

The cloning is described in the paper on page 2637 on the upper left side of the publication:

"*The expression vectors harboring GUP1 or GUP1H447A were obtained as follows: the open reading frame of GUP1 was amplified by PCR using plasmid pBH2178 (kind gift from Morten Kielland-Brandt) as a template and using primers and , underlined sequences being homologous to the target vector pGREG505 (Jansen et al., 2005). The PCR fragment was purified by a PCR purification kit (QIAGEN, Chatsworth, CA) and introduced into pGREG505 by co transfection into yeast cells thus generating pGUP1 (Jansen et al., 2005).*"

Briefly, two primers (GUP1rec1sens and GUP1rec2AS) were used to amplify the GUP1 gene from *Saccharomyces cerevisiae* chromosomal DNA using the two primers

```
>GUP1rec1sens
gaattcgatatcaagcttatcgataccgatgtcgctgatcagcatcctgtc
```

```
>GUP1rec2AS
gacataactaattacatgactcgaggtcgactcagcattttaggtaaattccg
```

Then the vector pGREG505 was digested with the restriction enzyme *Sal*I. This is not mentioned in Bosson et. al, but they make a reference to Jansen 2005:

Jansen G, Wu C, Schade B, Thomas DY, Whiteway M. 2005. Drag&Drop cloning in yeast. Gene, 344: 43–51.

Jansen et al describe the pGREG505 vector and that it is digested with *Sal*I before cloning. The *Sal*I digests the vector in two places, so a fragment containing the HIS3 gene is removed.

The SalI sites are visible in the plasmid drawing in Fig. 3.

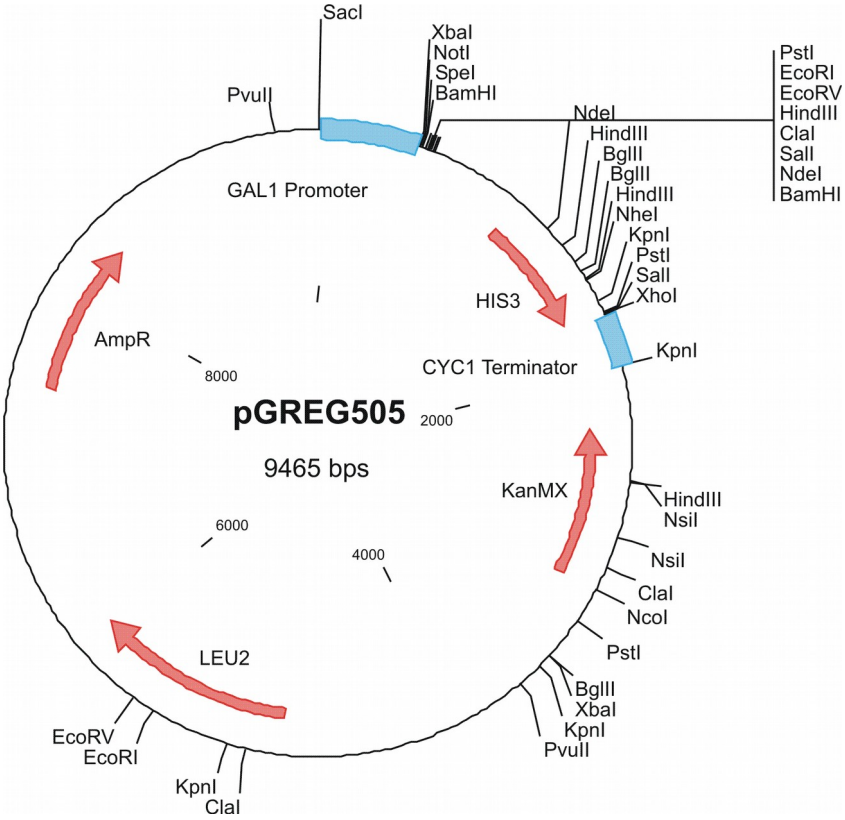

Fig. 3: pGREG505 vector

In the final step, the digested vector is mixed with the PCR product and transformed into yeast.

This is a cloning in three steps:

A. PCR of the GUP1 locus using GUP1rec1sens GUP1rec2AS, resulting in a linear insert.

B. Digestion of the plasmid pGREG505 with SalI, This step is not mentioned above, but evident from (2). This digestion removes a DNA fragment containing the HIS3 marker gene from the final construct.

C. Recombination between the linear insert and the linear vector.

See Fig. 4 for a graphical representation.

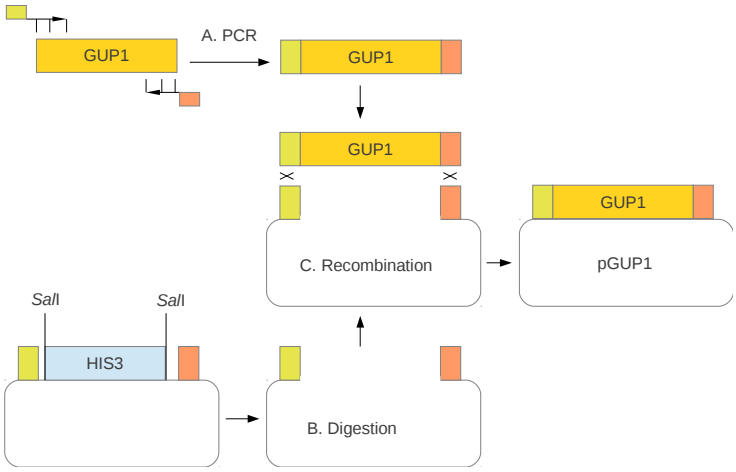

Fig. 4: Construction of pGUP1 by homologous recombination

We will now replicate the cloning procedure using pydna. For this we use Python interactively:

First we import pydna and verify the version. The current directory should be where you have the data files mentioned before.

```
Python 2.7.3 (default, Aug  1 2012, 05:14:39)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pydna
>>> pydna.__version__
'0.6.0'
```

The primer sequences, template and vector is read into SeqRecord objects (primers) and Dseqrecord objects (gene and plasmid).

```
>>> GUP1rec1sens = pydna.read("GUP1rec1sens.txt", ds=False)
>>> GUP1rec2AS = pydna.read("GUP1rec2AS.txt", ds=False)
>>> GUP1_locus = pydna.read("GUP1_locus.gb")
>>> pGREG505 = pydna.read("pGREG505.gb")
```

The PCR product is made from the primers and the template sequence.

```
>>> insert = pydna.pcr(GUP1rec1sens, GUP1rec2AS, GUP1_locus)
```

We need to import the *Sal*I restriction enzyme from Biopython.

```
>>> from Bio.Restriction import SalI
```

We cut the vector with *Sal*I

```
>>> lin_vect, his3 = pGREG505.cut(SalI)
>>> lin_vect
Dseqrecord(-8301)
```

We use the pydna circular assembly function to assemble the two linear DNA sequences into a circular DNA molecule. The variable cp will have a list of Dseqrecords.

```
>>> asm = pydna.Assembly((lin_vect, insert))
```

We check the list. It should have only one recombination product. The number indicate the size and the “o” that the DNA is circular.

We analyze the sequences for overlaps longer than or equal to 25 bp:

```
>>> asm.analyze_overlaps(limit = 25)
'4 sequences analyzed of which 2 have shared homologies with totally
2 overlaps'
```

```
>>> asm.create_graph()
"A graph with 5', 3' and 2 internal nodes was created"
```

```
>>> asm.assemble_circular_from_graph()
'1 circular products were formed'
```

```
>>> asm.circular_products
[Dseqrecord(o9981)]
```

```
>>> pGUP1 = asm.circular_products[0]
```

The pydna sync method makes sure that our new vector starts from the same position as the pGREG vector. This makes our recombinant plasmid easier to read.

```
>>> pGUP1 = pGUP1.synced(pGREG505)
```

Finally, we calculate the seguid for the sequence.

```
>>> pGUP1.seguid()
'42wIByERn2kSe/Exn405RYwhffU'
```

Finally we write the sequence to a file

```
>>> pGUP1.write("pGUP1_vector.gb")
```

If you are doing these examples on pydna live, the google app engine does not permit saving files. You can print the content of the file instead:

```
>>> print pGUP1.format("gb")
```

Beware! This is a long output. The output window can be made wider if necessary for a correct print out.