Python-dna

TP

Version 0.08

Björn Johansson

Braga

Jan 17

2012

What is Python-dna?

Python dna is a python package that provides functions and data types to deal with double stranded DNA. It depends on Biopython (a 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 and homologous recombination between linear DNA fragments are supported. Most functionality is implemented as methods for the double stranded DNA sequence record class "drecord", which is a subclass of the Biopython SeqRecord class.

Python-DNA might be useful to automate the simulation of sub cloning experiments using python. This could be helpful to generate examples for teaching purposes.

Python-dna was designed to semantically imitate how sub cloning experiments are typically documented in Scientific literature. Python-dna code describing a sub cloning is reasonably compact and meant to be easily readable.

One use case for Python-dna is to create a sort of executable documentation describing a subcloning experiment. The Python-dna code unambiguously describe a sub cloning experiment, and can be executed to yield the sequence of the of the resulting DNA molecule.

Typical usage at the command line could look like this:

```
>>> import pydna
>>> seq = pydna.dseq("GGATCCAAA","TTTGGATCC",ovhg=0)
>>> seq
dseq(-9)
GGATCCAAA
CCTAGGTTT
>>> from Bio.Restriction import BamHI
>>> a,b = seq.cut(BamHI)
>>> a
dseq(-5)
G
CCTAG
>>> b
```

```
dseq(-8)
GATCCAAA
    GTTT
>>> a+b
dseq(-9)
GGATCCAAA
CCTAGGTTT
>>> b+a
dseq(-13)
GATCCAAAG
    GTTTCCTAG
>>> b+a+b
dseq(-17)
GATCCAAAGGATCCAAA
    GTTTCCTAGGTTT
>>> 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!
>>>
```

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 Python-dna data type along with the drecord class which is a double stranded version of the Biopython SeqRecord class.

Two examples are given in this tutorial (which is under heavy development). The data files that are referred to in this document can be found in the sub folder “cookbook_files”.

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 Microbiology 67, no. 9 (September 2001): 4249–4255. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC93154/

Briefly, the XKS1 gene from *Saccharomyces cerevisiae* is amplified by PCR using two primers called primer1 and primer 3. 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.

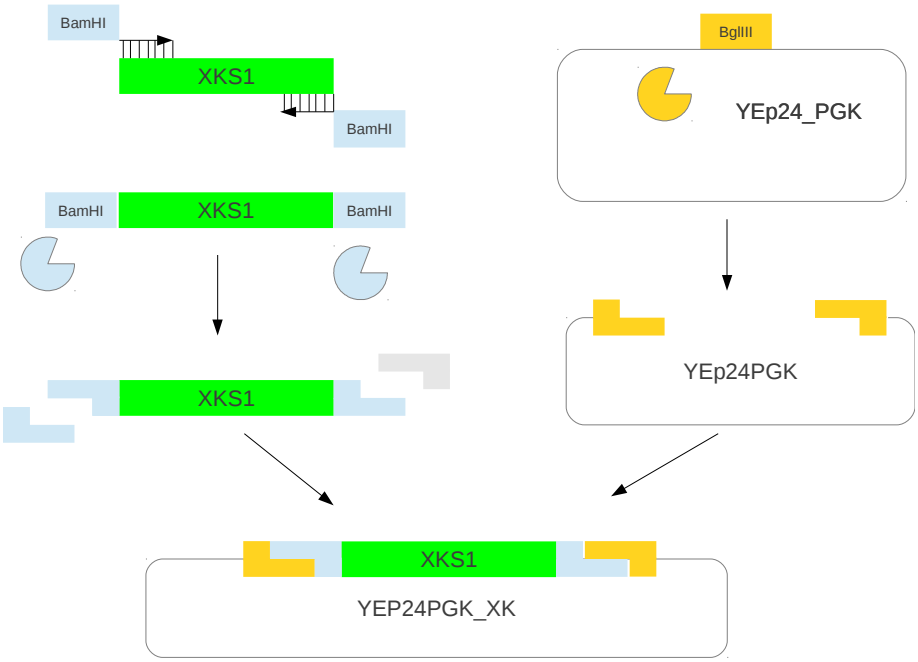

Fig. 1: Construction of the YEp24PGK_XK vector

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__
'0.0.8'
```

Use the pydna read function to assign the primers and template sequence to one drecord object each.

```
>>> pydna.read("primer1.txt")
drecord(-35)
>>> p1 = pydna.read("primer1.txt")
>>> p3 = pydna.read("primer3.txt")
>>> 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 drecord of 1829 bp.

```
>>> PCR_prod = pydna.pcr(p1,p3,XKS1)
>>> PCR_prod
Drecord(-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 property of the drecord 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 then add the insert to the linear plasmid

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

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

```
>>> YEp24PGK_XK.loop()
>>> YEp24PGK_XK
drecord(o11452)
```

The sync function will rotate the new plasmid, so that it starts at the same position as the old plasmid.

```
>>> YEp24PGK_XK = YEp24PGK_XK.sync(YEp24PGK)
>>> YEp24PGK_XK = pydna.sync(YEp24PGK_XK, YEp24PGK)
```

We then write the plasmid to a file:

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

The seguid command give the seguid of the sequence. Only the first four characters are shown.

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

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

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 of the Cell 17, no. 6 (June 2006): 2636–2645. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1474799/

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 were used to amplify the GUP1 gene from *Saccharomyces cerevisiae* chromosomal DNA using the two primers

```
>GUP1rec1sens
gaattcgatatcaagcttatcgataccgatgtcgctgatcagcatcctgtc
```

and

```
>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. http://www.ncbi.nlm.nih.gov/pubmed/15656971

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 small fragment containing the HIS3 gene is removed.

Make sure you can find the SalI sites in Fig. 1:

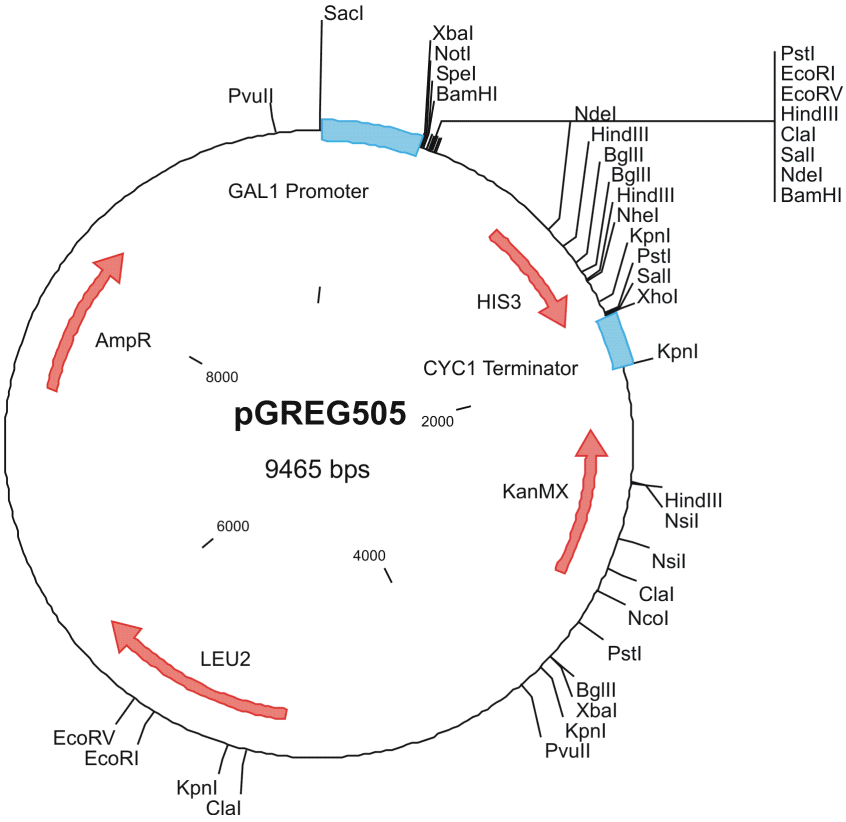

Fig. 2: pGREG505 vector

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. 3 for a graphical representation.

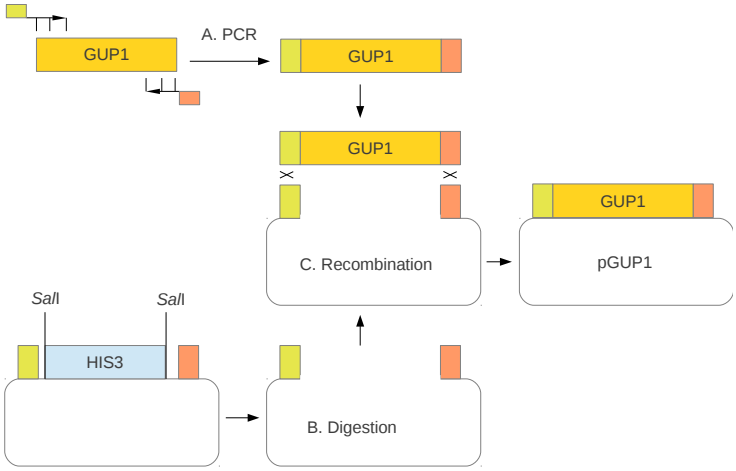

Fig. 3: Construction of pGUP1 by homologous recombination

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__
'0.0.8'
```

The primer sequences, template and vector is read into one drecord object each.

```
>>> GUP1rec1sens = pydna.read("GUP1rec1sens.txt")
>>> GUP1rec2AS = pydna.read("GUP1rec2AS.txt")
>>> 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 SalI restriction enzyme from Biopython

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

We cut the vector:

```
>>> his3, linear_vector = pGREG505.cut(SalI)
```

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 drecords.

```
>>> fs, cp = pydna.circular_assembly((insert, linear_vector),
limit=28)
```

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

```
>>> cp
[drecord(o9981)]
```

We pop the list:

```
>>> pGUP1 = circular_recombination_products.pop()
```

The pydna sync function makes sure that our new vector starts from the same position as the pGREG vector. This is mainly cosmetics.

```
>>> pGUP1 = pydna.sync(pGUP1, pGREG505)
```

Finally we write the sequence to a file

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

Finally, we calculate the seguid for the sequence.

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