
BESPA

---

a Backplane for Electron microscopy Single Particle Analysis

USER MANUAL

ver 0.9 release 352

2009 March 16, Yutaka Ueno, AIST

ABSTRACT

BESPA is a software package for image and volume data processing in single particle analysis.

OVERVIEW

Single particle analysis is a method for structural studies of protein and biological macromolecules. From observed images of the isolated molecules, the three dimensional volume can be reconstructed. The method was well demonstrated by pioneering works with numbers of software systems available in both free distribution and commercial packages. BESPA is yet another software tool for the single particle analysis to develop new algorithms and methods. Two major methods are studied: the reference free classification of images and the angular reconstitution technique. Currently, BESPA is not a complete program suite, but provides some original programs for image processing and three dimensional reconstruction. Our recent development and improvement will be released on the main web site.

http://staff.aist.go.jp/yutaka.ueno/bespa/

1.BASICS

Current Linux version of BESPA is started from a shell command line by a startup script "bespa", which is in most case installed in /usr/local/bin, a directory for a shared program. The body of BESPA is installed in its program directory. A basically it runs in a terminal window taking a script file to do a job. For a command prompt of the terminal, BESPA runs the script file and give a messages.

```
% bespa a-script-file
```

There are also useful built-in commands to start pre-defined script files. For example,

```
% bespa late
```

This commnad 'late' does a translational alignment of images. It will ask parameters from a conosole. Actually it runs a built-in script "late.msy". These scripts are installed in the BESPA program directory. The image data file is usually stored in an original file format called CDY. Usually a file extension ".cdy" is used for the file. To browse these image data or volume data, simply give the file name to the bespa command line, and a new window comes to display. In addition to a X-window display. A version for Microsoft Windows is also available.

```
% bespa image-file.cdy
```

---

2. Using Built-in Commands

Several commands are started asking filenames and parameters from the console. The console user-interface is not very friendly, so only backspace will work to edit the input line. It has history file in the current directory, named ".bespahistory" saving previous parameters. By typing "*", the parameter input will stop. "$" is to call a shell command just like "$ls" for directory listing. "?" should give a help message to be annotated more.

--- some implementation note

On unix, control-Z usually works to suspend the program, that will be revived by a shell command fg. Control-C will stop a program, but once job started an external program, it only stops the external program only. (bespa does not stop the process group like other unix shells)
Following message may appears when a history file is not found. please ignore them just not to use history.

```
XXX dofile () file not exist    .bespahistory
xxx file not found     .bespahistory   nil
```

2.1 late ---- translational alignment of images

Before classifying images of single particle, images are aligned to have a particle in the center. Also it is always nice to select good particles excluding pictures with neighbor objects. At first, a total sum of all image is calculated. This blob sum image is made by adding images only with centering adjustment. Only subset of the images, ~100, and four rotated images in 90 degrees step are averaged. Then images are aligned to the blob sum at the best match in terms of cross correlation function. These scores are sorted and requested numbers of upper images are selected.

```
% bespa late
 --- moonscript path    /home/uenoyt/bistro/ddlager/makeup/bespa.msy
moonscript bespa
cdybase --- cdy module prototype
  ---------------------------------------------
  late.run --- select particle images good for the first classification
    by lateral alignment (v08.10.27)

XXX dofile () file not exist    .bespahistory
 xxx file not found     .bespahistory   nil

. input image [] :
. output selected images [] :
. a blob particle image [] (optional) :
. the same selection as a previous file [] (optional) :
. number of selection [] : 1
. gausian smooothing [2] (optional) : 1.5
```

```
. coarsen images for quick run [2] (optional) :
. subset for making blob [100] :
. max shift [4] :
```

For filenames, please add extension ".cdy", By default, bespa create an image or volume data file with its own data format CDY. For Gaussian smoothing filters, use pixel unit for the radius parameter. Radius 0.5 yield images mostly unchanged. This smoothing proceeds before coarsening, i.e. radius 1.5 to 2.5 is ideal for the case with coarsening 2 in pixels. This radius parameter is the same as one used in Adobe Photoshop.
"The same selection as a previous file" is an option to make another class average image with different smoothing and coarsening conditions.

2.1 scla ---- a spectral clustering of image (eigenvector based)

As a reference free classification of single particle images at random orientation of molecule, images are grouped into clusters by all pair-wise similarity of images. Since the similarity values are evaluated with translational and rotational alignment, obtained groups are independent to in-plane orientation of the molecule. After clustering data, an iterative alignment of member images will save image file contains averaged images for the group, the class sum images.

```
% bespa scla
 --- moonscript path    /home/uenoyt/bistro/ddlager/makeup/bespa.msy
moonscript bespa
cdybase --- cdy module prototype
  ------------------------------------------------
  scla.run --- a reference free classification of single particle images
    by spectral clustering algorithm (v08.10.27)

. input cdy image [] : a
. use similarity matrix input [] (optional) :
. output group average [] : g.cdy
. output group data [] :


. gausian smooothing [1.5] (optional) :
. coarsen images for quick run [2] (optional) :
. number of groups [32] :

. sigma, local scaling gaussian function [0.3] :
. use mpi node (0/5/10/17) [0] (optional) :
. factor space dimension [80] :
. save intermediate factor coordinate [tmpfrc.lua] :
. mov/rot parameter file [] (optional) :
```

IMAGIC file is also supported if input image file with extension ".img" or ".hdr" are used,. The header file should exist in the same directory. For output, use filename with extension ".img", so that the program first create the group average (class sum) images into "tmpimv.cdy", then convert it to the IMAGIC-V format.

"sigma", the local scaling Gaussian function , is a technical term in spectral clustering. It controls how far relationship of similarity will be suppressed. The default value 0.3 usually

works in most case. However, in case the result is not satisfactory, other values between 0.2 to 0.8 should be tested. check eigenvalues appeard in the console log. If the eigenvalue fall off to zero too rapidly, decrease to 0.25. Typical eigenvalues should be 1.0 for the first, the second should be around 0.5, and the last is less 0.1. Please note this parameter was not well documented in literature of this algorithm.

```
--eigen status 0
  eigenval  vectors
 e  1.0000:  0.1191  0.1014  0.1156  0.1014  0.1071
 e  0.3290:  0.0579  0.0407  0.0304  0.0222  0.0187
 e  0.2798:  0.0683  0.1217 -0.1239 -0.0548  0.1168
 e  0.2121: -0.1440  0.1916 -0.1377  0.0291 -0.0249
 e  0.1907: -0.0838  0.1564  0.0952  0.0393 -0.1033
```

The class sum images are made without coarsening and gaussian filters. These parameters are the same as command "late" in previous chapter.

This script crates some temporary files:
tmpimg.cdy --- images filtered and coarsened
XXXXsmx.cdy --- the similarity matrix (reused in a next run)
tmpgrp.lua ---- k-means clustering result file
tmpfrc.lua ---- the factor coordinates of each data

The similarity matrix is useful to change number of groups with the other parameter unchanged. Other files are mostly checking purpose or they could be used for another custom script.

NOTE: Number of group is less than specified in rare occasion if k-means algorithm generate an empty group.

2.2 east ---- Euler angle search by triplet

From some of the characteristic views of single particle images, the projection euler angle to each images are estimated from mutual relationship between projection images of the same object. For this angular reconstitution method, the script tries to find many candidate set starting form some of good combination of triplet images. First, the sinogram of images are calculated, and all cross sinogram correlation functions are saved to a file. Then many triplet sets which satisfy mutual angular relationship are generated and sorted in terms of better values of the cross sinogram correlation. Since the result always depends on quality of images and noises at this moment, obtained Euler angles set in a single search are not always correct and subject to further tests.

```
% bespa east
 --- moonscript path    /home/uenoyt/bistro/ddlager/makeup/bespa.msy
moonscript bespa
cdybase --- cdy module prototype
  ------------------------------------------------
```

```
east.run --- euler angle search using triplet match seeds
  based on common line matching (v08.10.25)

. input cdy image [] :
. use cross sinogram file [] (optional) :
. output euler angle [tmp.tdr] :
. output 3D reconstruction [tmp.3d] :

. shift to fit center of gravity [] :
. gausian smooothing [1.5] (optional) :
. circke mask radius [0] (optional) :
. number of candidate [4] :
. maximum angular weight [10] :
. angle step [4] :
```

The output is the list of candidate Euler angles for each images and a sample 3D reconstruction of the 1st candidate. The list is sorted in terms of residual of common line matching. The file format is just like followings

At first, images are centered with their center of gravity up to the provided range of shifts in x-y direction. The circular mask is also applied and saved as a file, "tmpimg.cdy". Then sinogram is calculated and saved in a file "tmp.csg". The cross sinogram correlation function to all pairs of images are saved in a file "tmp.csx". This file becomes quite large in size, but can be reused in next search.
The main search is testing all possible triplet of images with good common line fit, and sort them. Up to the provided number of the candidates are saved. Then, next step is assign the best fit Euler angles of all images to every candidate models. The search is a iterative alignment of common line profile to minimize the total residual. It was an implementation of "simultaneous minimization algorithm" by Penczek (1996), they introduced the angular weight. This program asks for the minimum angular weight because if two images assigns the same or very close euler angles, the weight becomes too small to disturb convergence of the loop. The default value ~10 is fine, or 90 degree will impose totally even angular weight.

Please note the Euler angle definition is different from IMAGIC. They can converted (-gamma,-beta,-alpha) to be used in IMAGIC, where the scan line of images is unchanged. While IMAGIC takes the first scan line of image as Y coordinates, while BESPA takes it as X coordinates. File format conversion programs in BESPA keeps the scan line data intact without swapping xy axis.

2.3 cr3d --- Create 3D volume (not installed)

Once Euler angles to the characteristic views are assigned, 3D reconstruction of the volume is performed by the weighted back projection method. One of the candidates in calculated Euler angle sets can be selected in this script.

NOTE: This command is not correctly installed in version v08.10.27. Please use a script file

"cr3d.msy".

```
unix % bespa cr3d.msy
 opening script cr3d.msy
 --- moonscript path    /home/ueno/bistro/ddlager/makeup/bespa.msy
moonscript bespa
cdybase --- cdy module prototype
  ------------------------------------------
  cr3d.run --- create 3d reconstruciton
    weighted back projection (v08.10.25)

. input cdy image [../c3dwork4/mix10.cdy] :
. euler angle file [tmp.tdr] :
. output 3D reconstruction [tmp1.3d] :
. model number among candidates in a file [1] (optional) :
. make it a mirror [] (optional) :
. padding in real space for FFT  [16] (optional) :
. only with xcc value more than threshold [] (optional) :
 --- VMS version saved  .bespahistory_76
bespa.cr3d.run { ---
  [1]="cr3d.run",
  ["pad"]=16,
  ["modelno"]=1,
  ["imgfile"]="../c3dwork4/mix10.cdy",
  ["tdrfile"]="tmp.tdr",
  ["volout"]="tmp1.3d",
}
 --- VMS version saved  tmp1.3d_2
/home/ueno/bistro/ddlager/makeup/em/crtdrb -proj ../c3dwork4/mix10.cdy -m 1 -pad 16
tmp.tdr tmp1.3d

model section : 1
file=../c3dwork4/mix10.cdy sid=7  52.0 126.0   0.0 up=0 dx=0 dy=0 -8.54 ~ 46.62
 mask 15.5 filter 12.025/37 file=../c3dwork4/mix10.cdy sid=90 102.0 108.0   0.0 up=0
dx=0 dy=0 -8.45 ~ 49.11
 mask 15.5 filter 12.025/37 file=../c3dwork4/mix10.cdy sid=35 120.0  54.0   0.0 up=0
dx=0 dy=0 -9.57 ~ 52.00
 mask 15.5 filter 12.025/37 file=../c3dwork4/mix10.cdy sid=10  94.0 144.0   0.0 up=0
dx=0 dy=0 ..............................................................-- ...
(74)
 mem.allocation 53 save file tmp1.3d
 data range -0.5326 4.339 scale 26.0000
 interpreter finished.  cr3d.msy
```

2.4 toimv --- export to IMAGIC-5

Image or volume data are exported to IMAGIC-5 format. The command requires a file name to the argument.

```
unix % bespa toimv tmp12.cdy
 --- moonscript path    /home/uenoyt/bistro/ddlager/makeup/bespa.msy
moonscript bespa
cdybase --- cdy module prototype
using external program
 aux program :  /home/uenoyt/bistro/ddlager/makeup/em/cdy2imv tmp12.cdy tmp12.imv
 file name tmp12.cdy   comment :
```

```
 making 2 files, header and raw data : tmp12.hed tmp12.img
 using scale 1 base 0
  new files created
 size  41x41     coord 5.00(2.00)  5.00(2.00)   size 67KB  range -9.1 ~ 159.3   note :
euler={107.25,51.52,287.44}}
nz 1  size  41x41     coord 5.00(2.00)  5.00(2.00)   size 67KB  range -30.7 ~ 169.7
note : euler={129.64,81.19,171.86}}
nz 1  size  41x41     coord 5.00(2.00)  5.00(2.00)   size 67KB  range -2.7 ~ 166.4
note : euler={136.38,85.90,342.80}}
nz 1  size  41x41     coord 5.00(2.00)  5.00(2.00)   size 67KB  range -0.0 ~ 175.3
note : euler={71.94,63.60,50.98}0}}
nz 1  ---more images nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz
1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1 nz 1
nz 1 nz 1  adjust multi image header 100
 ... 3D volume
 program stop   3840
unix %
```

--- note

There might be a message of "3D Volume" for converting images, but never mind. The same information is saved to the IMAGIC header file for a 3D volume or multiple 2D images.

---

3. Using Script

A script is a program for BESPA written in the programming language Lua. Usually a script is to describe flow of jobs and automate some tasks, while a program is to describe processing details of each jobs. Many program defines functions for a processing task with necessary input and output. Technically, there are no deference between a program and a script. The Lua interpreter that execute them is a main infrastructure of this software system. The language itself has an ordinary procedural syntax so that it is feasible for most scientist with casual programming knowledge. This section introduces some of basic methods and necessary programming for the image processing. It helps understanding some of image processing jobs in a script and writing a new script or modifying existing scripts for a specific purpose. In current version, processing for the volume data are very limited.

3.1 Using a variable for an image data

A script file usually state a command of data processing by calling pre-defined functions. One data set of image pixels is handled as a variable.

```
one=cindy:read("sample.cdy")
```

This function cindy:read() load image data file on memory and name it a variable "one". In fact, this image data also contain many information accompanied with this data set, so they are stored in a data container for the image data. This data container is called a "table " data in Lua language. Therefore, it is good to consider the image is loaded into a container on memory, which is referred as a variable "one". For example, we can see the size of image :

```
print(one.nx, one.ny)
```

The image data size nx and ny is stored in this container, a "table" as a property of image, that can be specified by a period format. These property values are called "field" of this "table" in Lua language. There is also a useful function to access pixel value of image data directory.

```
print(one.pixel[1][1])
```

This is a two dimensional array pixel, where the bracket is used to access array data in Lua language. For this type of data container, some operations are defined.

```
two=cindy:zoom(one,2.0)
three=cindy:add(one,two)
four=cindy:affine(one,45,1,1)     --- rotate 45 degree, then move (1,1) in (x,y)
cindy:display(four)
```

After two minus, "--", the rest of line become a comment in the program. There is a program module "cindy" for image processing tasks. Functions defined in this module can be called as :

```
cindy:functionname(parameters)
```

Functions to modify image data usually return a new image data, a Lua-table, so they can be referred as a new variable. If you do not need previous image data, we can use the same variable as previous one to refer the new image data, then previous data will be released off memory.

3.2 Loading Multiple Images from a File

Usually multiple images are saved in a single file because number of images usually becomes thousands. Image loader function only load single images using index to the data starting from 1.

```
one=cindy:read("simple.cdy",3)
```

For programming efficiency, the file pointer is also used to load files and read sucessive data in a file as a stream.

```
fp=openfile("simple.cdy","rb")

list={}
for idx=1,10 do
        list[idx]=cindy:read(fp)
end
```

3.3 Managing List Data

One useful feature of Lua programming language is the list data. It is called as a "table" data in Lua. The simple list of numbers could also be used as an array just like previous example. The table data are created when it needed at any place in the script.

```
list={1,2,5,6,7}
list={ name="test", size=10 }
```

The list can be nested to describe hierarchical data. For example, the Euler angle data in BESPA is described in a hierarchical table.

3.4 Operations to Volume Data

Currently, not many functions are available for volume data, even though they can be loaded as the same way as the image data. If it was used by the image processing functions, the first slice of the volume is used and resulted new 2d images.
Available functions are follows:

```
cindy:calcstat3d()
cindy:display()
```

In this release, the 3D reconstruction is calculated by an auxiliary program "em/crtrb". There are experimental functions just like cdy_project3d(), cdy_rotate3d(), which is used in some scripts are described in "C-language functions" later in this document.

3.4 Sample Script

There are sample script files in bistro/ddlager/samples to demonstrate basic image manipulations.

3.5 Calling an External Program

There is a function to launch an external program in BESPA, which is actually in the standard programming library in Lua using the function provided by the operating system: execute(). For other programming library implemented in Lua, please refer to the Reference Manual of Lua (ver 4.0) available on the web : http://www.lua.org/manual/4.0

Additional programs used in BESPA are in a subdirectory named "em" of the BESPA program directory, the place of installation. For example, a user "guest" installed source code of BESPA in a directory "ueno/" , then it builds the BESPA program directory at :

```
/home/guest/ueno/bistro/ddlager/makeup/
```

3.6 Installing a built-in command

Several built-in commands are configured by the file "config.msy" in the BESPA directory. This configuration file is evaluated everytime the bespa command runs. The built in command is managed in the talbe "msycmd.command". For example, a new command "myf" to call a Lua funciton named "myfunction()" will be set in folloing.

```
msycmd.command.myf={ lua=myfunction}
```

There is a sample script file "mylib.msy" to demonstrate adding this new command. If an original script works fine, the function may be installed as a command in this way. During a

development, it is useful to use a lua program library for the script file to be edited extensively. The lua program library is loaded as follows:

```
bespa -l mylib.msy   myf
```

The file mylib.msy in the current directory will be loaded at first, then the command line argument "myf" is evaluated to call the function just installed in the library. Using script files will be much useful just included in a sample file "uselib.msy". If this script works, then it may be installed to "em/" directory in BESPA and loaded with "config.msy". To change the first menu messages, append text into msycmd.intro_message.

---

4. Using Graphical Environment

There is also a graphical user-interface and simple graphics facility for writing simple graphs or displaying image data. Although hey are usually used in script files to add user interfaces using mouse and keyboard, implemented graphical environment in BESPA is experimental. Please note these specification will change in later releases.
Here is a brief introduction what kind and how they are used in a script. This simple script will give a window with two buttons and small boxes. They it will ask your mouse selection on the buttons. After clicking mouse on a button, the script end and print which button you selected.

```
---- a simple program for gui and graphics
---- with lager interpreter (2008 Dec 8)

moonscript("bespa")

  YoyDon()

  messagebox=lag.dialog{
        a=lag.button{"test-A"; y=10},
        b=lag.button{"test-B"; y=40},
  }

  box=lag.dialogbox.new(messagebox)
  box:open()

  YovMove(100,100)
  YovBox(10,10)
  YovMove(120,100)
  YovBox(10,10)

  ret=box:modal()    --- wait a mouse selection input from user
  box:close()

  print("-----",ret)
```

The dialog box is a window with user-interface items: button, checkbox, keyboard entry box, or slider box. There are simple graphics library for rendering 2-dimensional primitives. In addtion, 3-dimensional graphics rendering is also supported with double buffering animation support. They are not full-featured graphics library, but provides a subset of very ordinal one. Please refer to the Programmers Guide for LAGER (not available yet)

---

5. The C-languageMmodules

This summarize the internal functions in BESPA written in C-language

_cdy_add_lua
_cdy_affine_lua
_cdy_axis_lua
_cdy_blt2d_lua
_cdy_calcstat_lua
cdy_circav_lua
cdy_circlemask_lua
cdy_covariance_lua
cdy_cshift_lua
cdy_cspan_cov_lua
cdy_cspan_lua
cdy_cspan_stat_lua
cdy_fft2d_lua
cdy_filter_lua
cdy_flip_lua
cdy_friedel2d_lua
cdy_phase2d_lua
cdy_superpose_lua
calcproject3d
cdy_cubic3d_lua
cdy_fitsinogram_lua
cdy_noise_lua
cdy_polar2d_lua
cdy_project3d_lua
cdy_rotate3d_lua
cdy_sinogram_lua

ACKNOWLEDGMENT

A software development of BESPA was first started in 1998 in AIST (Electro Technical Laboratory) and then a current prototype was developed through a collaboration works between CNRS and AIST. In particular, we wish to thank to Dr.Patrick Schultz, Dr. Bruno Klahols, and members of their laboratory at CNRS/ IGBMC for testing the code. Current development of this work was supported by the Strategic International Cooperative Program, Japan Science and Technology Agency (JST). Previous supports were from the Grant-in-Aid(08283101:"Genome Science") for Scientific Research on Priority Areas from the Ministry of Education, Science, Sports and Culture of Japan, and from Real World Computing Project of the Ministry of International Trade and Industry.

REFERENCES

1. a first report

Yutaka Ueno, Katsutoshi Takahashi, Kiyoshi Asai & Chikara Sato. BESPA: Software Tools for Three-dimensional Structure Reconstruction From Single Particle Images of Proteins. Genome Informatics (1999) 10:241–242.
http://staff.aist.go.jp/yutaka.ueno/bespa/bespa-GIW99P11.pdf

2. reference free classification of images

Yutaka Ueno, Masaaki Kawata & Shinji Umeyama (2005) "Intrinsic Classification of Single Particle Images by Spectral Clustering"
Proc. Biosignal Processing and Classification. (INSTICC Press, Portugal) 60-67.

3. a web site

BESPA: a Backplane for Electron microscopy Single Particle Analysis
2008 Oct 31. Yutaka Ueno, AIST Tsukuba Japan.
http://staff.aist.go.jp/yutaka.ueno/bespa/
