FIA Software 1RC
API Documentation

Orr Srour

July 3, 2010

Contents

1 Module fia_analysis

```
 FIA - Fluxomers Iterative Algorithm


SYNOPSIS:
    fia_analysis {-i[=--fia] PROJFILE | -t[=--ftbl] PROJFILE} [-o[=--output_file] OUTPUTFILE]

USAGE:
    fia_analysis performs the following two main steps:

        1) Metabolic pathway analysis.

            Here FIA transforms the text file representing the metabolic network into
            mathematical matrices required later for the MFA optimization process.

        2) Metabolic fluxes evaluation.

            Here FIA tries to find the fluxes that best suit the given metabolic system.
            The output log of this stage is a file called    <PROJFILE>.out,
            or else if specified by the "-o" flag.

    fia_analysis supports two types of input file formats: FIA and 13CFlux FTBL.
    The two are very much alike, except for the fact that the FIA format supports only declaration of
    uni-directional fluxes, and does not contain the FLUXES and INEQUALITIES sections.
    For bi-directional fluxes, FIA simply defines two seperated fluxes.
    When analyzing 13CFlux FTBL files, fia_analysis looks for C=0 constraints in the FLUXES XCH section
    in order to determine directionality of fluxes.

OPTIONS:
    -i --fia
        Specifies that the metabolic pathway input file <PROJFILE> is a FIA formated file.
        All fluxes are assumed unidirectional (for bi-directional specify 2 seperated fluxes).

    -t --ftbl
        Specifies that the metabolic pathway input file <PROJFILE> is a 13CFlux FTBL formated file.
        Flux is assumed unidirectional unless C=0 constraint is applied to it in the FLUXES XCH section
        The FIA formated file will be saved under <PROJFILE>.fia .

   -o --output_file
        optional.
        Specifies the name of the result log output file. The default file name is PROJFILE.out

    at least one of the above must be supplied.

USAGE EXAMPLE:

fia_analysis -t temp.ftbl
    Runs anaylsis & evaluation of the network defined in the FTBL formated input file temp.ftbl .
```

```
    same as:
        fia_analysis --ftbl temp.ftbl

fia_analysis --fia temp.fia -a   |
    Runs only network analysis on the given FIA formated input file.
    Same as:
        fia_analysis -i temp.fia -a
        fia_analysis -i temp.fia --analyze_only
        fia_analysis --fia temp.fia --analyze_only

fia_analysis temp.fia -i -e
    Runs only the fluxes evaluation on the FIA formated input file temp.fia .
    This assumes analysis has been done before on the temp.fia file.
    Same as:
        fia_analysis -i temp.fia --evaluate_only
        fia_analysis --fia temp.fia -e
        fia_analysis --fia temp.fia --evaluate_only
```

Version: 0.7.0

1.1 Functions

one_div_vec(*vec*)

This function returns a vector which is 1/vec.

Parameters

`vec:` input vector

(type=scipy.array)

Return Value

1/vec (element wise division).

(type=scipy.array)

sdiag(*a*)

Fast sparse diagonal matrix creator. Simply returns a sparse matrix with a as its diagonal.

Parameters

`a:` input array

(type=array)

Return Value

sparse matrix with a in its diagonal.

(type=scipy.sparse matrix)

expand_x_values(*cteq*)

This function transforms input with (\)'s into all the possible sets of inputs with 0's and 1's replacing the (\)s.

Parameters

`cteq:` Input string. (\)'s will be replaced with 1's and 0's.

(type=string)

Return Value

List of possible variations of cteq, with all the possible 0's and 1's replacements for \'s in the input string.

(type=list)

create_ms_isotopomers_dict(*no_of_atoms*, *indices_to_change*)

This function returns all the possible permutations of no_of_atoms atoms, seperated by the number of 1's in them. (indices_to_change) specifies which of the atoms can be changed (and the counting is then done only on these atoms). For example, if we call:

```
create_ms_isotopomers_dict(4,[0,1,2])
```

we get:

```
[['000x'], ['001x', '010x', '100x'], ['011x', '101x', '110x'], ['111x']]
```

The first element represents all 0's vec (for indices_to_change), the second a vector with only one 1 element, the third with two 1's elements, and the third with three 1's elements.

This function is used for mass-spectrometry measurements analysis.

Parameters

`no_of_atoms:` Number of atoms in the counted molecule.

(type=number)

`indices_to_change:` List of indicies on which we want to count.

(type=list)

Return Value

List of lists - all possible permutations for the molecule, sorted by number of ones.

(type=list)

num2bin(*num*, *length*)

Converts number to binary format with at (length) bits.

Parameters

`num:` The number to be converted

(type=number)

`length:` Length of the output binary number.

(type=number)

Return Value

Binary string representing the input number. Zeros are left appended if necessary in order to maintain the required length.

(type=string)

save_text_vec(*filename*, *vec*)

This fucntion saves the input vector in text TAB seperated format. Used for MATLAB debugging purposes.

matlab_save_sparsed_matrix(*filename*, *mat*)

This fucntion saves the input matrix in text TAB seperated format loadable by MATLAB. Used for MATLAB debugging purposes.

sort_dict_by_values(*d*)

This function sorts the input dictionary by its values.

null(*A*, *eps*=`1e-15`)

This function finds the null space of the incoming matrix A.

find_init_point(*self*)

This function generates the initial point given for the optimization process. It is based upon the standard init point finding methods of iterior point algorithms: In order to find a valid solution for *S*u = 0*, *U* u <= s* we solve:

```
min(s)

s.t.:

 [I,-I]*u + [0*i,10*i] >= s
 Su = S_b
```

We then use this initial point with the propogation equation, resulting with the first point for the algorithm.

get_normG(*u*, *self*)

Main MFA optimization objective function value. This function calculates || self.tG * x - self.tB || for a given u vector by finding the valid x vector and then substituting it in the objective function.

get_normG_grad(*u*, *self*)

Main MFA optimization objective function gradient calculation. This function calculates d (|| self.tG * x - self.tB ||) / du for a given u vector.

load_data(*fname*)

tic()

toc()

main(*argv*)

rand(*d0*, *d1*, *dn*, *...*)

```
Random values in a given shape.

Create an array of the given shape and propagate it with
random samples from a uniform distribution
over ``[0, 1)``.

Parameters
----------
d0, d1, ..., dn : int
    Shape of the output.

Returns
-------
out : ndarray, shape ``(d0, d1, ..., dn)``
    Random values.

See Also
--------
random

Notes
-----
This is a convenience function. If you want an interface that
takes a shape-tuple as the first argument, refer to
`random`.

Examples
--------
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

randn(*d0*, *d1*, *dn*, *...*)

```
Returns zero-mean, unit-variance Gaussian random numbers in an
array of shape (d0, d1, ..., dn).

Note:  This is a convenience function. If you want an
            interface that takes a tuple as the first argument
            use numpy.random.standard_normal(shape_tuple).
```

1.2 Variables

Name	Description
umfpack	**Value:** `um.UmfpackContext()`
prog_width	**Value:** `40`

continued on next page

Name	Description
display_prog	**Value:** `False`
ALLOW_THREADS	**Value:** `1`
BUFSIZE	**Value:** `10000`
CLIP	**Value:** `0`
ERR_CALL	**Value:** `3`
ERR_DEFAULT	**Value:** `0`
ERR_DEFAULT2	**Value:** `2084`
ERR_IGNORE	**Value:** `0`
ERR_LOG	**Value:** `5`
ERR_PRINT	**Value:** `4`
ERR_RAISE	**Value:** `2`
ERR_WARN	**Value:** `1`
FLOATING_POINT_SUPPORT	**Value:** `1`
FPE_DIVIDEBYZERO	**Value:** `1`
FPE_INVALID	**Value:** `8`
FPE_OVERFLOW	**Value:** `2`
FPE_UNDERFLOW	**Value:** `4`
False_	**Value:** `False`
Inf	**Value:** `inf`
Infinity	**Value:** `inf`
MAXDIMS	**Value:** `32`
NAN	**Value:** `nan`
NINF	**Value:** `-inf`
NZERO	**Value:** `-0.0`
NaN	**Value:** `nan`
PINF	**Value:** `inf`
PZERO	**Value:** `0.0`
RAISE	**Value:** `2`
SHIFT_DIVIDEBYZERO	**Value:** `0`
SHIFT_INVALID	**Value:** `9`
SHIFT_OVERFLOW	**Value:** `3`
SHIFT_UNDERFLOW	**Value:** `6`
ScalarType	**Value:** `(<type 'int'>, <type 'float'>, <type 'complex'>, <type 'l...`
True_	**Value:** `True`
UFUNC_BUFSIZE_DEFAULT	**Value:** `10000`
UFUNC_PYVALS_NAME	**Value:** `'UFUNC_PYVALS'`
WRAP	**Value:** `1`
__package__	**Value:** `None`
absolute	**Value:** `<ufunc 'absolute'>`
add	**Value:** `<ufunc 'add'>`
arccosh	**Value:** `<ufunc 'arccosh'>`
arcsinh	**Value:** `<ufunc 'arcsinh'>`
arctan	**Value:** `<ufunc 'arctan'>`
arctan2	**Value:** `<ufunc 'arctan2'>`
bitwise_and	**Value:** `<ufunc 'bitwise_and'>`
bitwise_not	**Value:** `<ufunc 'invert'>`
bitwise_or	**Value:** `<ufunc 'bitwise_or'>`
bitwise_xor	**Value:** `<ufunc 'bitwise_xor'>`

continued on next page

Name	Description
c_	**Value:** <numpy.lib.index_tricks.CClass object at 0x2bec6d0>
cast	**Value:** {<type 'numpy.int64'>: <function <lambda> at 0x2c68c80>, ...
ceil	**Value:** <ufunc 'ceil'>
conj	**Value:** <ufunc 'conjugate'>
conjugate	**Value:** <ufunc 'conjugate'>
cos	**Value:** <ufunc 'cos'>
cosh	**Value:** <ufunc 'cosh'>
deg2rad	**Value:** <ufunc 'deg2rad'>
degrees	**Value:** <ufunc 'degrees'>
divide	**Value:** <ufunc 'divide'>
e	**Value:** 2.71828182846
equal	**Value:** <ufunc 'equal'>
exp	**Value:** <ufunc 'exp'>
exp2	**Value:** <ufunc 'exp2'>
expm1	**Value:** <ufunc 'expm1'>
fabs	**Value:** <ufunc 'fabs'>
floor	**Value:** <ufunc 'floor'>
floor_divide	**Value:** <ufunc 'floor_divide'>
fmax	**Value:** <ufunc 'fmax'>
fmin	**Value:** <ufunc 'fmin'>
fmod	**Value:** <ufunc 'fmod'>
frexp	**Value:** <ufunc 'frexp'>
greater	**Value:** <ufunc 'greater'>
greater_equal	**Value:** <ufunc 'greater_equal'>
hypot	**Value:** <ufunc 'hypot'>
index_exp	**Value:** <numpy.lib.index_tricks.IndexExpression object at 0x2bec790>
inf	**Value:** inf
infty	**Value:** inf
invert	**Value:** <ufunc 'invert'>
isfinite	**Value:** <ufunc 'isfinite'>
isinf	**Value:** <ufunc 'isinf'>
isnan	**Value:** <ufunc 'isnan'>
ldexp	**Value:** <ufunc 'ldexp'>
left_shift	**Value:** <ufunc 'left_shift'>
less	**Value:** <ufunc 'less'>
less_equal	**Value:** <ufunc 'less_equal'>
little_endian	**Value:** True
log1p	**Value:** <ufunc 'log1p'>
logaddexp	**Value:** <ufunc 'logaddexp'>
logaddexp2	**Value:** <ufunc 'logaddexp2'>
logical_and	**Value:** <ufunc 'logical_and'>
logical_not	**Value:** <ufunc 'logical_not'>
logical_or	**Value:** <ufunc 'logical_or'>
logical_xor	**Value:** <ufunc 'logical_xor'>
maximum	**Value:** <ufunc 'maximum'>
mgrid	**Value:** <numpy.lib.index_tricks.nd_grid object at 0x2bec510>

continued on next page

Name	Description
minimum	**Value:** <ufunc 'minimum'>
mod	**Value:** <ufunc 'remainder'>
modf	**Value:** <ufunc 'modf'>
multiply	**Value:** <ufunc 'multiply'>
nan	**Value:** nan
nbytes	**Value:** {<type 'numpy.int64'>: 8, <type 'numpy.int16'>: 2, <type ...
negative	**Value:** <ufunc 'negative'>
newaxis	**Value:** None
not_equal	**Value:** <ufunc 'not_equal'>
ogrid	**Value:** <numpy.lib.index_tricks.nd_grid object at 0x2bec550>
ones_like	**Value:** <ufunc 'ones_like'>
pi	**Value:** 3.14159265359
r_	**Value:** <numpy.lib.index_tricks.RClass object at 0x2bec610>
rad2deg	**Value:** <ufunc 'rad2deg'>
radians	**Value:** <ufunc 'radians'>
reciprocal	**Value:** <ufunc 'reciprocal'>
remainder	**Value:** <ufunc 'remainder'>
right_shift	**Value:** <ufunc 'right_shift'>
rint	**Value:** <ufunc 'rint'>
s_	**Value:** <numpy.lib.index_tricks.IndexExpression object at 0x2bec810>
sctypeDict	**Value:** {0: <type 'numpy.bool_'>, 1: <type 'numpy.int8'>, 2: <typ...
sctypeNA	**Value:** {'?': 'Bool', 'B': 'UInt8', 'Bool': <type 'numpy.bool_'>,...
sctypes	**Value:** {'complex': [<type 'numpy.complex64'>, <type 'numpy.compl...
sign	**Value:** <ufunc 'sign'>
signbit	**Value:** <ufunc 'signbit'>
sin	**Value:** <ufunc 'sin'>
sinh	**Value:** <ufunc 'sinh'>
square	**Value:** <ufunc 'square'>
subtract	**Value:** <ufunc 'subtract'>
tan	**Value:** <ufunc 'tan'>
tanh	**Value:** <ufunc 'tanh'>
true_divide	**Value:** <ufunc 'true_divide'>
trunc	**Value:** <ufunc 'trunc'>
typeDict	**Value:** {0: <type 'numpy.bool_'>, 1: <type 'numpy.int8'>, 2: <typ...
typeNA	**Value:** {'?': 'Bool', 'B': 'UInt8', 'Bool': <type 'numpy.bool_'>,...
typecodes	**Value:** {'All': '?bhilqpBHILQPfdgFDGSUVO', 'AllFloat': 'fdgFDG', ...

1.3 Class MyWriter

A file-handler replacement that duplicates the actions applied to the hanlder to a log file handler as well.

1.3.1 Methods

__init__(*self*, *stdout*, *filename*)

Initialization of the file-handler duplicator.

Parameters

`stdout:` Originial file handler to be forked
(type=file)

`filename:` Name of the log file for which actions should be replicated into
(type=string)

write(*self*, *text*)

Replacement for the file handler's write method.

flush(*self*)

Replacement for the file handler's flush method.

close(*self*)

Replacement for the file handler's close method.

1.4 Class FtblFile

This class is the main object of the FIA algorithm. Its main purposes are:

1. Conversion of FIA and FTBL files into python mathematical objects.
2. Solution of the MFA optimization problem

1.4.1 Methods

__init__(*self*, *projname*, *inpfiletype*)

Parameters

`projname:` Name of the project to be analyzed.
(type=string)

`inpfiletype:` Type of the input file to be analyzed. Can be either 'fia' (for fia input files) or 'ftbl' (for ftbl input files)
(type=string)

save_ftblfile(*self*, *filename*)

This function saves the entire FtblFile class to a file using python's pickle packge. This enables the user to use the data analyzed later without the need of recalculating the transition or LU matrices (which can take a bit of a time).

save_data(*self*)

MATLAB debugging function. This function saves the mathematical objects used for the optimizaiton process in files in order to be analyzed by MATLAB[a].

[a] http://www.mathworks.com

get_flux_name(*self*, *flux_num*)

returns the name of fluxer whose index is flux_num.

Parameters

`flux_num:` The number of the fluxomer we are looking for.

(type=number)

Return Value

The name of the fluxomer.

(type=string)

load_file(*self*, *filename*)

This function loads the input file into the FtblClass object by:

1. Dropping comments (lines starting with "//")
2. Replacing EOL "\n" or "\r" with ""
3. Erasing spaces

Eventually we are left with a file containing only TABs. This object is then transformed into an array of lists - every array element is a line of the parsed file, and splitted by TABs into a list object.

Parameters

`filename:` Name of the input file to be examined.

(type=string)

Return Value

List of lists containing the TAB splitted parsed version of the input file.

(type=list)

find_segment(*self*, *seg_name*)

This function finds the block "seg_name" block segment in the parsed input file `self.ftbl_file` object. New segments begin with every non-idented line in the input file. Their names are the first string in these lines.

Parameters

`seg_name:` Name of the segment to be analyzed

(type=string)

Return Value

A list containing the segment's rows, tab seperated.

(type=list)

create_pools_dict(*self*)

This function creates the initial metabolic pools dictionary object by analyzing the NETWORK section of the input file. The function creates the `self.pools` dictrionary object - Metabolic pools dictionary for which the keys are pools names, and the values are the number of carbon atoms for each pool.

create_flux_dict(*self*)

This function creates and updates the main building blocks objects used for the metabolic pathway analysis.

(section) Classification of fluxes

Classifiction of fluxes into: uni-directional fluxes, bi-directional fluxes (FTBL files only) and input/output fluxes. The uni and bi direictional analysis is only needed for FTBL files (FIA assumes all the fluxes are uni-directional). The decision is made by looking for 0 constrained exchange fluxes in the FLUXES section of the FTBL file.

(section) Re-generation of the NETWORK section - adding necessary virtual fluxes

When we are given with a chemical reaction that transforms two elements of the same metabolic pool into others, we translate it into a regular multi-input chmecial reaction by adding a virtual pool and flux to our system:

As an example:

```
f: A + A -> B + C     (flux "f" transforms A+A into B+C)
```

is transformed into:

```
f: A + A_s -> B + C   (flux "f" transofmrs A+A_s ito B+C)
f_A: A -> A_s         (flux "f_A" transforms A into A_s)
```

This transformation is being done on the input file object itself (hence by changing the parsed NETWORK segment of the input file). The same applies for reversed fluxes of bi-diredctional FTBL files fluxes.

(section) Analysis of extra-cellular pools

Finding pools which are global input or output to the system, and save them in the `self.excell_pools` object. These pools are recognized as pools with no input / output fluxes going into / exiting them.

(section) Re-generation of the NETWORK section - adding necessary reversed fluxes

The mathematical analysis of the problem consists of flux vector representing uni-directional fluxes. Due to this reason, we need to transform the NETWORK section to contain only uni-directional fluxes by adding to it necessary reversed fluxes. This only applies when FTBL input files are supplied (since again, FIA files are assumed to contain only uni-diretional fluxes in anyway).

(section) Generation of the global fluxes vector

Generation of the global fluxes (as opposed to fluxomers) vector dictionary object. The dictrionary `self.flux_nw_vec` keys are the names of the global fluxes in our system, and the values are the number of carbon atoms each of them transforms.

(section) Generation of the equality matrix

The equality matrix `self.S` and its associated equality vector `self.S_b` are the mathematical constraints of the global optimization problem That should apply on the global fluxes vector (hence self.S * x == self.S_b). The equality matrix is constructed out of 2 main elements:

1. The stoichiometric equations - as analyzed from the parsed NETWORK section of the input file

2. The eqaulities constraints from the EQUALITIES section of the input file.

(section) Generation of the equality measurements

Equalities arised from the constraints section of the FLUXES section (only applies to FTBL

get_metabolic_flux_index(*self*, *flux_name*)

This function returns the index of the global metabolic flux "flux_name".

Parameters

`flux_name:` The name of the flux we want to find.

(type=string)

Return Value

The index of the global metabolic flux "flux_name".

(type=number)

create_flux_meas_mat(*self*)

This function parses the FLUX_MEASUREMENTS segment of the ftbl file. It creates the following matrices:

1. `self.flux_meas_mat` - The matrix choosing the measured metabolic fluxes out of the fluxomer vector.
2. `self.flux_meas_values` - The measured values.
3. `self.flux_meas_variance` - The measurement variance vector.

Eventually we will search for:

```
M{min [ (self.flux_meas_mat * x - self.flux_meas_values_).T * diag(self.flux_meas_variance) * (sel
```

create_global_meas_mat(*self*)

This function creates the objective function for the MFA optmization problem (the global "G" measurement matrix).

The objective of the MFA problem is to minimize:

```
M{|| tG * x - tB ||^2}
```

The matrix `self.tG` is created by concatanation of the 3 possible measurments matrices:

1. Label measurements (`self.labelinp_num` and `self.labelinp_denom`).
2. Fluxes measurement (`self.labelmeas_num` and `self.labelmeas_denom`).
3. Mass-Spectrometry measurements (`self.ms_meas_num` and `self.ms_meas_denom`).

The vector `self.tB` is created by concatanation of appropriate values of the above.

trnasform_eq_into_matrix(*self*, *eq*)

This function transforms string equation of fluxes into a fluxomers stoichiometric matrix. For example:

```
"*+flux1/0*-flux2/1" -> [1 0 0 0; 0 0 0 -1])
```

Parameters

`eq`: An string equation. Every equation element is seperated by either "*+" or "*-" with the fluxomer name following it. No preciding numbers are allowed, only plain fluxomer names. For multiple instances of the same fluxomer, simply add/substract it more than once to the equation.

(type=string)

Return Value

The fluxomer stoichiometric matrix represented by eq.

(type=CSR matrix)

create_label_meas_eq(*self*)

This function parses the LABEL_MEASUREMENTS segment of the ftbl file. The output of the function is the matrices and vectors:

1. `self.labelmeas_num` - The label measurements numerator matrix
2. `self.labelmeas_denom` - The label measurements denominator matrix
3. `self.labelmeas_value` - The label measurements values vector
4. `self.labelmeas_var` - The label measurements values vector

The objective function will be to minimize:

```
M{ ||diag(self.labelmeas_var) * [self.labelmeas_num - diag(self.labelmeas_value) * self.labelmeas
```

create_ms_meas_eq(*self*)

This function parses the MASS_SPECTROMETRY segment of the ftbl file. The output of the function is the matrices and vectors:

1. `self.ms_meas_num` - The MS measurements numerator matrix
2. `self.ms_meas_denom` - The MS measurements denominator matrix
3. `self.ms_meas_value` - The MS measurements values vector
4. `self.ms_meas_var` - The MS measurements values vector

We always treat these measurement as a ratio measurement between the MS values, relative to the 0 MS measurement (hence we add to the objective the ratios: mass1/mass0, mass2/mass0 etc').

The objective function will be to minimize:

```
M{ ||diag(self.ms_meas_var) * [self.ms_meas_num - diag(self.ms_meas_value) * self.ms_meas_denom]*x
```

create_U_matrix(*self*)

This function constructs the U matrix, which transforms the fluxomers vector into the metaoblic fluxes vector. hence:

```
M{u = U*x}
```

where:

```
u - the metabolic flux vector
x - the fluxomers vector
```

create_inp_eq_matrix(*self*)

This function parses the LABEL_INPUT segment. The output of the function is the matrices and vectors:

1. `self.labelinp_num` - The flux value measurements numerator matrix
2. `self.labelinp_denom` - The flux value measurements denominator matrix
3. `self.labelinp_value` - The flux value measurements values vector

The objective function will be to minimize:

```
M{ ||diag(self.labelinp_var) * [self.labelinp_num - diag(self.labelinp_value) * self.labelinp_deno
```

Note: since not supplied by the FTBL files, the value of self.labelinp_value is a global assigned constant.

create_label_transition_mat(*self*)

This function creates the propogation transition matrix:

```
\M{x(t) = P x(t-1)} .
```

P is provided using the following sub matrices:

1. `self.trans_mat_a` - the *H_1* matrix (in CSR format).
2. `self.trans_mat_b` - the *H_2* matrix (in CSR format).
3. `self.trans_mat_a_gamma` - the *g_1* matrix (in CSR format).
4. `self.trans_mat_a_beta` - the *g_2* matrix (in CSR format).
5. `self.trans_mat_b_beta` - the *g_3* matrix (in CSR format)

In addition, this function creates the beta (ratios) vector:

1. `self.beta_vec` - the text names of the beta vector elements
2. `self.beta_vec_M` - the numerator for construction of the beta vec.
3. `self.beta_vec_N` - the denominator for constuction of the beta vec.

eventually, we have:

```
M{beta_vec = (self.beta_vec_M * u) / (self.beta_vec_N * u)}
```

create_lu_trans_matrix(*self*)

This function computes the LU decomposition of the propogation transition matrices. We find a factorization for:

```
L{self.trans_mat_a} and L{self.trans_mat_b}
```

by finding the LU factorizaiton of the matix constructed out of concatanation of the two above.

The result are the matrices:

1. `self.al1` - the left side of the pseudo LU of self.trans_mat_a
2. `self.al2` - the left side of the pseudo LU of self.trans_mat_b
3. `self.au` - the right side of the pseudo LU factorizaiton of both self.trans_mat_a and self.trans_mat_b .

evaluate(*self*)

This function runs the main optimization process. It uses the `scipy.optimize.fmin_slsqp` for the actual optimization process, and prints out the results in a TABed formatted table.

1.4.2 Class Variables

Name	Description
filename	**Value:** ''

1.4.3 Instance Variables

Name	Description
ProjName	Project's name
labelinp_var	Label input variance (it is not supplied with FTBL files).
InputFileType	input file type (fia or ftbl)
excell_pools	Dictionary of the external (global input or output) metabolic pools
ftbl_file	Parsed input file object (TAB spliited lists for the various lines of the input file) **Value:** ''
pools	Metabolic pools dictionary (Keys are pools names, values are the number of carbon atoms for each pool)

Index