oapackage Documentation

Pieter Eendebak, Alan Vazquez

Dec 23, 2019
1 Introduction 3
2 Example notebooks 7
3 Data representation 31
4 Generation of designs 37
5 Normal form of arrays 41
6 Properties of designs 45
7 C++ library 53
8 References 127
9 Indices and tables 129
10 License 131

Bibliography 133

Index 135
The Orthogonal Array package contains functionality to generate and analyse orthogonal arrays, optimal designs and conference designs.
Orthogonal arrays, optimal designs and conference designs are important tools for the design of experiments [EN95] [HSS99] [WH09]. The Orthogonal Array package contains functionality to generate and analyse these types of designs. To generate the arrays and designs, the package uses the exhaustive enumeration algorithm of [SEN10] and the optimization algorithm of [ES17]. To analyze the arrays and designs, the package includes a wide variety of relevant statistical and combinatorial criteria. A large collection of orthogonal arrays, optimal designs and conference designs generated with the package are available in the Orthogonal Array package website [Een18].

1.1 Example usage

The Orthogonal Array package can be used to generate and manipulate arrays and designs. Additionally, it can calculate some of their statistical properties. The following example shows how to generate an orthogonal array with 8 runs and 2 factors, and calculate three relevant statistical properties:

Calculate D-efficiency

```
>>> import oapackage
>>> array=oapackage.exampleArray(0) # define an orthogonal array
>>> array.showarray()
array:
  0 0
  0 0
  0 1
  0 1
  1 0
  1 0
  1 1
  1 1
>>> D = array.Defficiency() # calculate the D-efficiency for estimating the interaction effects model
>>> array_rank = array.rank() # calculate the rank of the design
>>> print('D-efficiency %f, rank %d' % (D, array_rank) )
D-efficiency 1.000000, rank 2
>>> gwlp = array.GWLP() # calculate the generalized word length pattern
>>> print('Generalized wordlength pattern: %s' % gwlp) )
Generalized wordlength pattern: (1.0, 0.0, 0.0)
```

The statistical properties of the arrays and designs are introduced in Properties of designs.
1.2 Interfaces

The Orthogonal Array package has interfaces in C++ and Python for generating, manipulating and analyzing all the types of arrays and designs. In this documentation, you will find references to both the Python and the C++ interface. The package also includes several command line tools.

For the generation of optimal designs [ES17], the Orthogonal Array package has also a Matlab interface; see the documentation README.Matlab.md.

1.3 License

The code is available under a BSD style license; see the file LICENSE for details. If you use this code or any of the results, please cite this program as follows:


1.4 Acknowledgements

The code and ideas for this package have been contributed by Eric Schoen, Ruben Snepvangers, Vincent Brouerius van Nidek, Alan Roberto Vazquez and Pieter Thijs Eendebak.

1.5 Installation

The package is continously tested on Linux and Windows. The Python interface is available from the Python Package Index. The package can be installed from the command line using pip:

```
$ pip install OApackage
```

The source code for the package is available on https://github.com/eendebakpt/oapackage. The command line tools use a cmake build system. From the command line, type the following:

```
$ mkdir -p build
$ cd build
$ cmake ..
$ make
$ make install
```

This creates the command line utilities and a C++ library.

To compile the Python interface use:

```
$ python setup.py build
$ python setup.py install --user
```

The Python interface requires Numpy [TheScipycommunity12], Matplotlib [Hun07] and Swig. The code has been tested with Python 2.7, 3.5, 3.6 and 3.7.
The R interface to the optimal design functionality of the package is available from CRAN. For the Matlab and Octave interface to the optimal design functionality see the file README.Matlab.md.

1.6 Related sites of orthogonal arrays

There are several related sites available online which include collections of orthogonal arrays. For instance, the website of Neil Sloane [Slo14], the website of Hongquan Xu [Xu18], the SAS website managed by Warren Kuhfeld [Kuh18], and the R package _DoE.base_ [GAX18] include lists and surveys of attractive orthogonal arrays gathered from different sources.
This section contains several examples for generating, manipulating and analyzing arrays and designs using the Orthogonal Arrays package. The examples are shown using Jupyter notebooks, which can be found on github https://github.com/eendebakpt/oapackage/tree/master/docs/examples.

## 2.1 Example script for Python interface to Orthogonal Array package

Pieter Eendebak pieter.eendebak@gmail.com

```python
import numpy as np
import oapackage
print('oapackage version: %s' % oapackage.version() )
oapackage version: 2.6.0

Load an example array.

```python
array=oapackage.exampleArray(0)
array.showarray()
```

array:
0 0
0 0
0 1
0 1
1 0
1 0
1 1
1 1

Calculate properties of the array such as the D-efficiency for the main effects model, the generalized word length pattern and the rank.

```python
print('D-efficiency %f, rank %d' % (array.Defficiency(), array.rank()) )
print('Generalized wordlength pattern: %s' % str(array.GWLP()))
```

D-efficiency 1.000000, rank 2
Generalized wordlength pattern: (1.0, 0.0, 0.0)

Calculate the generalized word length pattern for another example array.

```python
array=oapackage.exampleArray(11)
print('Generalized wordlength pattern: %s' % oapackage.oahelper.gwlp2str(array.GWLP()))
```

7
Generalized wordlength pattern: 1.00, 0.02273, 0.03926, 0.5434, 0.8244, 2.217, 1.043, 0.126, 0.002066

2.1.1 Indexing

The array_link object can be indexed as a normal array.

```
[12]: array[0:5, 2:3]
array[[1, 1], [0, 1], [1, 0], [0, 0], [1, 1]]

[12]: array_link: 5, 1

[14]: array[0:5, 2:4].showarray()
array:
1 1
0 1
1 0
0 0
1 1

[15]: print(array[0,2])
1
```

2.1.2 Numpy

We can convert between Numpy arrays and array_link objects. Note that an array_link is always integer valued.

```
[19]: X=(4*np.random.rand(20, 10)).astype(int)
array=oapackage.array_link(X)
array.showarraycompact()

2003200102
0322120001
3030110033
0311131120
2200100200
211233121
3110300211
3302313313
2310311302
1212232322
3003300322
0312311331
3222200313
2001123302
2222111302
0020322112
0101223211
0223300100
1232030100
0320032113

Convert from array_link back to Numpy array.
```

2.2 Example arrays

The package contains several example arrays and designs. They can be retrieved using the method `exampleArray`.

```python
import oapackage
al = oapackage.exampleArray(-1, 1)
```

<table>
<thead>
<tr>
<th>Array Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exampleArray 0</td>
<td>array in OA(8, 2, 2^2)</td>
</tr>
<tr>
<td>exampleArray 1</td>
<td>array 3 in OA(16, 2, 2^5)</td>
</tr>
<tr>
<td>exampleArray 2</td>
<td>array 6 in OA(16, 2, 2^6)</td>
</tr>
<tr>
<td>exampleArray 3</td>
<td>array ? in OA(32, 3, 2^7)</td>
</tr>
<tr>
<td>exampleArray 4</td>
<td>array 4 in OA(16, 2, 2^7)</td>
</tr>
<tr>
<td>exampleArray 5</td>
<td>array 0 in OA(24, 2, 4 3 2^a)</td>
</tr>
<tr>
<td>exampleArray 6</td>
<td>array in OA(4, 2, 2^a)</td>
</tr>
<tr>
<td>exampleArray 7</td>
<td>array 0 in OA(4, 2, 2^a)?</td>
</tr>
<tr>
<td>exampleArray 8</td>
<td>array in OA(40, 3, 2^7)</td>
</tr>
<tr>
<td>exampleArray 9</td>
<td>array in A(40, 2^7), D-optimal</td>
</tr>
<tr>
<td>exampleArray 10</td>
<td>array in OA(9, 3^3)</td>
</tr>
<tr>
<td>exampleArray 11</td>
<td>D-optimal array in OA(44, 2^8)</td>
</tr>
<tr>
<td>exampleArray 12</td>
<td>even-odd array OA(64, 2^13)</td>
</tr>
<tr>
<td>exampleArray 13</td>
<td>array in OA(25, 2^5)</td>
</tr>
<tr>
<td>exampleArray 14</td>
<td>design in D(28, 2^5), D-efficiency is low</td>
</tr>
<tr>
<td>exampleArray 15</td>
<td>design in D(56, 2^10), D-efficiency is low</td>
</tr>
<tr>
<td>exampleArray 16</td>
<td>array in OA(32, 2, 2^5)</td>
</tr>
<tr>
<td>exampleArray 17</td>
<td>unique array in OA(64, 4, 2^7)</td>
</tr>
<tr>
<td>exampleArray 18</td>
<td>conference matrix of size 16, 7</td>
</tr>
<tr>
<td>exampleArray 19</td>
<td>conference matrix of size 4, 3</td>
</tr>
<tr>
<td>exampleArray 20</td>
<td>first LMC-0 double conference matrix in DC(24,3)</td>
</tr>
<tr>
<td>exampleArray 21</td>
<td>second LMC-0 double conference matrix in DC(16,4)</td>
</tr>
<tr>
<td>exampleArray 22</td>
<td>LMC-0 double conference matrix in DC(32,4)</td>
</tr>
<tr>
<td>exampleArray 23</td>
<td>LMC-0 double conference matrix in DC(32,6)</td>
</tr>
<tr>
<td>exampleArray 24</td>
<td>design in OA(64, 3, 2^16) (even-odd)</td>
</tr>
<tr>
<td>exampleArray 25</td>
<td>design in OA(64, 3, 2^16) (even-odd)</td>
</tr>
<tr>
<td>exampleArray 26</td>
<td>design in OA(64, 3, 2^16) (even-odd)</td>
</tr>
<tr>
<td>exampleArray 27</td>
<td>design in OA(64, 3, 2^16) (even-odd)</td>
</tr>
<tr>
<td>exampleArray 28</td>
<td>conference design in C(4, 3) in LMC0 form</td>
</tr>
<tr>
<td>exampleArray 29</td>
<td>conference design in C(4, 3)</td>
</tr>
<tr>
<td>exampleArray 30</td>
<td>conference design in C(8,4)</td>
</tr>
<tr>
<td>exampleArray 31</td>
<td>conference design in C(8,4)</td>
</tr>
<tr>
<td>exampleArray 32</td>
<td>first double conference design in DC(18,4)</td>
</tr>
<tr>
<td>exampleArray 33</td>
<td>second double conference design in DC(18,4)</td>
</tr>
<tr>
<td>exampleArray 34</td>
<td>third double conference design in DC(18,4)</td>
</tr>
<tr>
<td>exampleArray 35</td>
<td>first double conference design in DC(20,4)</td>
</tr>
<tr>
<td>exampleArray 36</td>
<td>second double conference design in DC(20,4)</td>
</tr>
<tr>
<td>exampleArray 37</td>
<td>third double conference design in DC(20,4)</td>
</tr>
<tr>
<td>exampleArray 38</td>
<td>LMC0 conference design in C(30,3)</td>
</tr>
<tr>
<td>exampleArray 39</td>
<td>first LMC0 conference design in C(8,6)</td>
</tr>
<tr>
<td>exampleArray 40</td>
<td>first conference design in C(14, 5)</td>
</tr>
<tr>
<td>exampleArray 41</td>
<td>second conference design in C(14, 5)</td>
</tr>
<tr>
<td>exampleArray 42</td>
<td>third conference design in C(14, 5)</td>
</tr>
<tr>
<td>exampleArray 43</td>
<td>2x2 array with zeros and a single value -1</td>
</tr>
<tr>
<td>exampleArray 44</td>
<td>D-optimal strength 3 orthogonal array in OA(40,3, 2^7)</td>
</tr>
<tr>
<td>exampleArray 45</td>
<td>first conference design in C(20,8)</td>
</tr>
</tbody>
</table>

(continues on next page)
exampleArray 46: second conference design in $C(20,8)$
exampleArray 47: third conference design in $C(20,8)$
exampleArray 48: last conference design in $C(20,8)$
exampleArray 49: array 4347 $C(20,8)$
exampleArray 50: array 4506 $C(20,8)$
exampleArray 51: first array in $C(12,4)$
exampleArray 52: second array in $C(12,4)$
exampleArray 53: third array in $C(12,4)$
exampleArray 54: root array in $OA(6,2,3^1 2^1)$
exampleArray: no example array with index 55 exists

Select an example array and show it.

```
[3]: al = oapackage.exampleArray(2, 1)
    al.showarray()
```

exampleArray 2: array 6 in $OA(16, 2, 2^6)$
array:
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 0 1 1 1 1 1
0 1 1 0 0 1 0 1
0 1 1 1 0 1 0 1
0 1 1 1 1 0 1 0
0 1 1 1 1 1 0 1
1 0 1 0 0 1 0 1
1 0 1 0 1 1 0 1
1 0 1 1 0 0 1 1
1 0 1 1 1 0 0 1
1 1 0 0 1 1 0 0
1 1 0 1 0 0 1 1
1 1 1 0 0 1 0 1
1 1 1 0 1 0 0 1
1 1 1 1 0 0 1 1
1 1 1 1 1 0 0 1
1 1 1 1 1 1 0 1
```

### 2.3 Example to write and read files with arrays from disk

The package can write arrays and designs to a custom file format. There is a text based format (specified by `oapackage.ATEXT`) and a binary format (specified by `oapackage.ABINARY`). See the online documentation for details on the file formats.

```
[1]: import oapackage
    import tempfile
```

Create a list of two example arrays.

```
[2]: lst = [oapackage.exampleArray(32), oapackage.exampleArray(33)]
    print(lst)
[array_link: 18, 4, array_link: 18, 4]
```

Write the two arrays to a file on disk.

```
[3]: filename = tempfile.mktemp('.oa')
    _=oapackage.writearrayfile(filename, lst, oapackage.ATEXT)
```
Display information about the file written.

```python
# [4]: oapackage.oainfo(filename)
file C:\Users\EENDEB-1\AppData\Local\Temp\tmpxc4dns16.oa: 18 rows, 4 columns, 2 arrays, mode text, nbits 0
```

The arrays can be read from disk again using the command `readarrayfile`.

```python
# [6]: lst2=oapackage.readarrayfile(filename)
print(lst2)
(array_link: 18, 4, array_link: 18, 4)
```

The first array in the list that was read from disk is:

```python
# [7]: lst2[0].showarray()
array:
 0 0 0 0
 0 0 0 0
 1 1 1 1
 1 1 1 1
 1 1 -1 -1
 1 1 -1 -1
 1 -1 1 -1
 1 -1 1 -1
 1 -1 -1 1
 1 -1 -1 -1
-1 1 1 -1
-1 1 1 -1
-1 1 -1 1
-1 1 -1 1
-1 -1 1 1
-1 -1 1 1
-1 -1 -1 -1
-1 -1 -1 -1
```

### 2.4 Enumerate orthogonal arrays

The Orthogonal Array package can completely enumerate all orthogonal arrays of a specified class. In this notebook, we enumerate specific classes of three-level orthogonal arrays and mixel-level orthogonal arrays.

First, we specify the class of three-level orthogonal arrays to enumerate. For example, we consider three-level orthogonal arrays of strength 2 with 27 runs and 8 factors.

```python
# [1]: import oapackage
run_size = 27
strength=2
number_of_factors=8
factor_levels = 3
arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print(arrayclass)
arrayclass: N 27, k 8, strength 2, s {3,3,3,3,3,3,3,3}, order 0
```

Second, we create the root array as the starting point of our enumeration.

---

**2.4. Enumerate orthogonal arrays** 11
Third, extend the root array. It is also possible to extend a list of arrays.

```python
list3columns = oapackage.extend_arraylist(ll2, arrayclass)
print('extended to %d arrays with 3 columns' % len(list3columns))
list4columns = oapackage.extend_arraylist(list3columns, arrayclass)
print('extended to %d arrays with 4 columns' % len(list4columns))
```

extended to 9 arrays with 3 columns
extended to 711 arrays with 4 columns

It is also possible to extend selected arrays from a list.

```python
ll = oapackage.extend_arraylist(list4columns[0:8], arrayclass)
print('extended first 2 arrays to %d arrays with 5 columns' % len(ll))
```

extended first 2 arrays to 189 arrays with 5 columns

By adding one column at a time we can enumerate all three-level orthogonal arrays of strength 2 with 27 runs and 8 factors. The total computation time for this would be a couple of hours.
2.4.1 Mixed-level orthogonal arrays

The package can also enumerate mixed-level orthogonal arrays. For instance, consider enumerating all 16-run strength-2 orthogonal arrays with one four-level factor and nine two-level factors.

```
run_size = 16
strength=2
number_of_factors = 10
factor_levels=[4,2,2,2,2,2,2,2,2,2]
arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print(arrayclass)
arrayclass: N 16, k 10, strength 2, s {4,2,2,2,2,2,2,2,2,2}, order 0
```

Create the root array as the starting point of our enumeration.

```
al=arrayclass.create_root()
al.showarraycompact()
```

```
00
00
01
01
10
10
11
11
20
20
21
21
30
30
31
31
```

For these arrays, we can extend a single array or lists of arrays.

```
array_list=[arrayclass.create_root()]
array_list_3columns=oapackage.extend_arraylist(array_list, arrayclass)
array_list_4columns=oapackage.extend_arraylist(array_list_3columns, arrayclass)
print('extended to %d arrays with 3 columns' % len(array_list_3columns))
print('extended to %d arrays with 4 columns' % len(array_list_4columns))
```

```
extended to 3 arrays with 3 columns
extended to 10 arrays with 4 columns
```

Finally, enumerate all 16-run strength-2 orthogonal arrays with one four-level factor and nine two-level factors.

```
arrays = array_list_4columns
for extension_column in range(5, number_of_factors+1):
    extensions=oapackage.extend_arraylist(arrays, arrayclass)
    print('extended to %d arrays with %d columns' % (len(extensions), extension_column))
    arrays=extensions
```

```
extended to 28 arrays with 5 columns
extended to 65 arrays with 6 columns
extended to 110 arrays with 7 columns
```

(continues on next page)
extended to 123 arrays with 8 columns
extended to 110 arrays with 9 columns
extended to 72 arrays with 10 columns

2.4.2 Notes

- The numbers of isomorphism classes for various types of classes can be found at the webpage series of orthogonal arrays.
- For larger number of arrays the command line tools are more convenient and more memory efficient.

2.5 Generate orthogonal arrays with high D-efficiency

This notebook contains example code from the article Two-level designs to estimate all main effects and two-factor interactions by Eendebak, P. T. and Schoen, E. D. This example shows how to generate orthogonal arrays with a high $D$-efficiency in a reasonable amount of time (< 1 minute). For more results and details, see the paper.

Generate a D-optimal orthogonal array of strength 2 with 32 runs and 7 factors.

```python
import numpy as np
import matplotlib.pyplot as plt
import oapackage
%matplotlib inline

run_size = 32
number_of_factors = 7
factor_levels = 2
strength = 2
nkeep = 24  # Number of designs to keep at each stage

arrayclass = oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print('In this example we generate orthogonal arrays in the class: {}'.format(arrayclass))

# First, generate orthogonal arrays with the function extend_arraylist. Next, keep the arrays with the best D-efficiency.
arraylist = [arrayclass.create_root()]

options = oapackage.OAextend()
options.setAlgorithmAuto(arrayclass)

for extension_column in range(strength + 1, number_of_factors + 1):
    print('extend arrays with {} columns with a single column'.format(arraylist[0].n_columns))
    arraylist_extensions = oapackage.extend_arraylist(arraylist, arrayclass, options)

    dd = np.array([a.Defficiency() for a in arraylist_extensions])
    ind = np.argsort(dd)[::-1][0:nkeep]
```

(continues on next page)
selection = [ arraylist_extensions[ii] for ii in ind]
dd=dd[ind]
    print(' generated %d arrays, selected %d arrays with D-efficiency %.4f to %.4f'
    % (len(arraylist_extensions), len(ind), dd.min(), dd.max() ) )
arraylist = selection

extend 1 arrays with 2 columns with a single column
    generated 5 arrays, selected 5 arrays with D-efficiency 0.0000 to 1.0000
extend 5 arrays with 3 columns with a single column
    generated 19 arrays, selected 19 arrays with D-efficiency 0.0000 to 1.0000
extend 19 arrays with 4 columns with a single column
    generated 491 arrays, selected 24 arrays with D-efficiency 0.9183 to 1.0000
extend 24 arrays with 5 columns with a single column
    generated 2475 arrays, selected 24 arrays with D-efficiency 0.9196 to 1.0000
extend 24 arrays with 6 columns with a single column
    generated 94 arrays, selected 24 arrays with D-efficiency 0.7844 to 0.8360

Show the best array from the list of D-optimal orthogonal arrays.

[20]: selected_array = selection[0]
print('Generated a design in OA(%d, %d, 2^%d) with D-efficiency %.4f
    % (selected_array.n_rows, arrayclass.strength, selected_array.n_columns, dd[0] )
print('The array is (in transposed form):

    selected_array.transposed().showarraycompact()

Generated a design in OA(32, 2, 2^7) with D-efficiency 0.8360
The array is (in transposed form):

00000000000000001111111111111111
00000000111111110000000111111111
000011100011111001111110000000111
000110010110111011101101100111001
001010101011011011011011011011010
010010101101101101101101101101101
011100100110110110110110110110110

We calculate the $D_\infty$, $D_\sigma$- and $D_1$-efficiencies.

[21]: efficiencies = np.array([array.Defficiencies() for array in arraylist])
print(efficiencies)

[[0.8360354 0.72763273 1. ]
 [0.83481532 0.78995185 1. ]
 [0.82829522 0.70849985 1. ]
 [0.82829522 0.65037401 1. ]
 [0.82545286 0.7514444 1. ]
 [0.8185915  0.65157148 1. ]
 [0.81676643 0.68965719 1. ]
 [0.81559279 0.72218391 1. ]
 [0.81557999 0.66966068 1. ]
 [0.81302642 0.74299714 1. ]
 [0.80973102 0.73354611 1. ]
 [0.80286424 0.76455544 1. ]
 [0.79907785 0.72697978 1. ]
 [0.79762174 0.58515515 1. ]
 [0.79514152 0.65199837 1. ]
 [0.79348419 0.74167976 1. ]]
Visualize the $D$-efficiencies using a scatter plot.

```python
[19]: plt.plot(efficiencies[:,0], efficiencies[:,1], '.b')
plt.title('Scatterplot of efficiencies of designs with %d columns' % arraylist[0].n_cols)
plt.xlabel('D-efficiency')
plt.ylabel('Ds-efficiency')
```

![Scatterplot of efficiencies of designs with 7 columns](scatterplot.png)

2.6 Generate D-efficient designs

This notebook contains example code from the article Two-level designs to estimate all main effects and two-factor interactions by Eendebak, P. T. and Schoen, E. D. This example shows how to generate D-efficient designs with a user-specified optimization function.

```python
[1]: import numpy as np
    import oapackage
    import oapackage.Doptim

Define the class of designs to generate.

```python
[4]: run_size = 40
    number_of_factors=7
    factor_levels=2
    strength=0
```
arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print('We generate D-efficient designs with %d rows and %d columns
˓→number_of_factors) )

We generate D-efficient designs with 40 rows and 7 columns

Generate a single D-efficient design using $\alpha = (1, 2, 0)$ as the parameters for the optimization function. For details on this parameter and its corresponding optimization function, see Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions.

[5]: alpha=[1,2,0]
scores, design_efficiencies, designs, ngenerated = oapackage.Doptimize(arrayclass,˓→nrestarts=30, optimfunc=alpha, selectpareto=True)

Doptim: optimization class 40.2-2-2-2-2-2-2
Doptimize: iteration 0/30
Doptimize: iteration 29/30
Doptim: done (12 arrays, 1.5 [s])

[6]: print('nGenerated %d designs, the efficiencies for these designs are:' % ˓→len(designs))
for ii, d in enumerate(designs):
    dd = d.Defficiencies()
    print('array %d: D-efficiency %.4f, Ds-efficiency %.4f' % (ii, dd[0], dd[1]) )

D=[d.Defficiency() for d in designs]
best=np.argmax(D)
print('nThe design with the highest D-efficiency (%.4f) is:
˓→D[best] )

designs[best].transposed().showarraycompact()

Generated 12 designs, the efficiencies for these designs are:
array 0: D-efficiency 0.8775, Ds-efficiency 0.9783
array 1: D-efficiency 0.9040, Ds-efficiency 0.9627
array 2: D-efficiency 0.8632, Ds-efficiency 0.9827
array 3: D-efficiency 0.9047, Ds-efficiency 0.9596
array 4: D-efficiency 0.8498, Ds-efficiency 0.9854
array 5: D-efficiency 0.8908, Ds-efficiency 0.9627
array 6: D-efficiency 0.8829, Ds-efficiency 0.9631
array 7: D-efficiency 0.8857, Ds-efficiency 0.9600
array 8: D-efficiency 0.8859, Ds-efficiency 0.9552
array 9: D-efficiency 0.8972, Ds-efficiency 0.9323
array 10: D-efficiency 0.8905, Ds-efficiency 0.9345
array 11: D-efficiency 0.8745, Ds-efficiency 0.9394

The design with the highest D-efficiency (0.9047) is:

11000110000101011101110001110101
11110001111011010101100111001001
10110111101100110000100100111001
1101110110000101100110011001001
01010110111010001110011101010101
01000110111111011110100011000011
011100100100000110111111111111001

Optimizing with a different optimization target leads to different D-efficient designs. Below we compare the sets of

2.6. Generate D-efficient designs

17
designs generated with optimization target \([1,0,0]\) and \([1,2,0]\).

```python
[7]: scores0, design_efficiencies0, designs0, _ = oapackage.Doptimize(arrayclass,
→nrestarts=30, optimfunc=[1,0,0], selectpareto=True)

Doptim: optimization class 40.2-2-2-2
Doptimize: iteration 0/30
Doptimize: iteration 29/30
Dopt: done (8 arrays, 1.4 [s])
```

```python
[9]: def combineEfficiencyData(lst):
    data = np.zeros((0, 4))
    for jj, dds in enumerate(lst):
        dds_index=np.hstack((dds, jj * np.ones((len(dds),1)) ) )
        data=np.vstack( (data, dds_index) )
    return data

design_efficiencies_combined=combineEfficiencyData([design_efficiencies0, design_ ˓→efficiencies])
plot_handles=oapackage.generateDscatter(design_efficiencies_combined, ndata=3, lbla=[ ˓→'Optimize $D$', 'Optimize $D+2D_s$'], verbose=0)

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

Pareto: 20 optimal values, 20 objects
```

![Graph showing D-efficiency vs D_s-efficiency](image-url)

<Figure size 432x288 with 0 Axes>
2.7 Example script to use Nauty from Python

Nauty can be used to reduce any graph into a normal form. In this notebook, we show how to use the Nauty functionality from Python.

```python
import numpy as np
import oapackage

# Define a function to invert a permutation.

def inverse_permutation(perm):
    inverse = [0] * len(perm)
    for i, p in enumerate(perm):
        inverse[p] = i
    return inverse

# We define a graph with 5 nodes. The graph is defined by the incidence matrix of size 5 x 5 and a coloring with two colors.

graph = np.zeros((5,5), dtype=int); graph[0,1]=graph[0,2]=graph[0,3]=graph[1,3]=1;
graph = np.maximum(graph, graph.T)  # make array symmetric
colors = [0,0,0,1,1]

# Reduce the graph to normal form using Nauty.

t = oapackage.reduceGraphNauty(graph, colors=colors, verbose=0)
tri = inverse_permutation(t)
graph_reduced = oapackage.transformGraphMatrix(graph, tri)

print('normal form reduction:
%s' % (t,))
print('input graph: 

%r' % (graph,))
print('reduced graph: 

%r' % (graph_reduced,))

colors_r = [colors[idx] for idx in tri]
print('colors reduced: %s' % (colors_r,))
```

2.7. Example script to use Nauty from Python 19
normal form reduction: (1, 2, 0, 4, 3)
input graph:
[[0 1 1 1 0]
 [1 0 0 1 0]
 [1 0 0 0 0]
 [1 1 0 0 0]
 [0 0 0 0 0]]
reduced graph:
[[0 0 1 0 1]
 [0 0 1 0 0]
 [1 1 0 0 1]
 [0 0 0 0 0]
 [1 0 1 0 0]]
colors reduced: [0, 0, 1, 1, 1]

Apply a random permutation to the graph and reduce the graph again.

```python
perm = np.random.permutation(5); iperm = inverse_permutation(perm)
print('permutation: %s' % (perm,))
graph2 = graph[perm, :][:, perm]
print('permutation: %s' % (perm,))
```

Show the transformed matrix and color vector.

```python
print(graph2)
print(colors2)
```

```python
[0 0 0 0 0]
[0 0 0 1 1]
[0 0 0 0 1]
[0 1 0 0 1]
[0 1 1 1 0]
```

```python
[1, 1, 0, 0, 0]
```

```python
tr2 = oapackage.reduceGraphNauty(graph2, colors=colors2, verbose=0)
tr2i = inverse_permutation(tr2)
print('tr2: %s' % (tr2,))
print('input graph: ')
print(graph2)
print('reduced graph: ')
print(graph2reduced)
```

```python
print('colors2r: %s' % (colors2r,))
```

```python
[3, 2, 4, 0, 1]
input graph:
[[0 0 0 0 0]
 [0 0 0 1 1]
 [0 0 0 1 1]
 [0 1 0 0 1]
 [0 1 0 0 1]]
```

(continues on next page)
Check that the two reduced graphs are equal.

```python
[10]: if np.all(graph_reduced==graph2reduced):
    print('reduced arrays are equal!')
reduced arrays are equal!
```

## 2.8 Analyse isomorphisms of a set of orthogonal arrays with N=56

In this example, we show how to identify isomorphic arrays from a list of two-level strength-2 orthogonal arrays with 56 runs and 28 factors.

```python
import numpy as np
import oapackage
import oapackage.graphtools
from oapackage.graphtools import selectIsomorphismClasses

sols=oapackage.readarrayfile('OAN56K28.oa')
arrayclass=oapackage.arraylink2arraydata(sols[0])
print(arrayclass)
print('loaded %d arrays' % len(sols))
arrayclass: N 56, k 28, strength 2, s {2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,-2,2,2,2,2}, order 0
loaded 9 arrays

b,mm=selectIsomorphismClasses(sols, verbose=0)
print('from %d arrays selected %d unique isomorphism classes' % (len(mm),np.unique(b).size))
print('indices: %s' % str(b))
from 9 arrays selected 6 unique isomorphism classes
indices: [0 4 5 3 1 5 3 2 4]
```

Verify that the first two arrays are indeed non-isomorphic:

```python
jj=np.abs(sols[0].Jcharacteristics(4))
n0, _=np.histogram(jj, [0,8,16,24])
print(n0)
jj=np.abs(sols[1].Jcharacteristics(4))
n1, _=np.histogram(jj, [0,8,16,24])
print(n1)
```
Since the first two arrays have different $J$-characteristics (see the section Statistical properties of orthogonal arrays for details), they are non-isomorphic. For the isomorphic arrays, it is possible to obtain the array transformation to make the arrays identical using the function `reduceConferenceTransformation`.

### 2.9 Generation and analysis of conference designs

In this notebook, we show how to generate conference designs and calculate properties of these designs. For details on conference designs and their properties, see Properties of conference designs and A Classification Criterion for Definitive Screening Designs, Schoen et al., 2018 and [Enumeration and Classification of Definitive Screening Designs] (in preparation).

Load required libraries and define the class of conference designs to enumerate.

```python
import oapackage
classification_class=oapackage.conference_t(12, 6, 0)
print(classification_class)
conference class: number of rows 12, number of columns 6
```

Define the root array and extend the lists of conference designs.

```python
conference_designs=[[classification_class.create_root_three_columns()]]
for ii, ncols in enumerate(range(4, 8)):
    arrays = oapackage.extend_conference(conference_designs[ii], classification_class, _
                                           verbose=0)
    conference_designs.append(arrays)
    print('extension resulted in %d designs with %d columns' % (len(arrays), ncols))
extension resulted in 9 designs with 4 columns
extension resulted in 42 designs with 5 columns
extension resulted in 123 designs with 6 columns
extension resulted in 184 designs with 7 columns
```

### 2.9.1 Calculate properties of conference designs

Here, we show how to calculate relevant properties of conference designs. Select a 12-run 7-factor conference design generated previously.

```python
design = conference_designs[4][0]
design.showarray()
array:
  0 1 1 1 1 1 1
  1 0 -1 -1 -1 -1 -1
  1 1 0 -1 -1 1 1
  1 1 1 0 1 -1 -1
  1 1 1 -1 0 -1 1
  1 1 -1 1 1 0 -1
  1 1 -1 1 -1 1 0
  1 -1 1 1 -1 1 -1
(continues on next page)
A sensible criterion to evaluate conference designs is the so-called $F_4$-vector (Schoen et al., 2019). We can compute the $F_4$-vector of a conference design as follows.

Schoen et al., 2019 showed that conference designs are good building blocks for definitive screening designs (Xiao et al. 2012). The Orthogonal Array package can construct a definitive screening design from a conference design.

For the resulting definitive screening design, the Orthogonal Array package can compute some statistical properties based on projections into a smaller number of factors.
2.10 Example code for delete-one-factor projections

Any orthogonal array can be reduced to delete-one-factor projection form using `reduceDOPform`. The method is described in the article *A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections* by Eendebak, P. T. In this notebook, we reduce an example array to its delete-one-factor projection form.

Load required libraries and select an example orthogonal array with 16 runs and 7 factors.

```python
import oapackage
al = oapackage.exampleArray(4)
al = oapackage.reduceDOPform(al)
al.showarray()
```

![Array](array-image)

A key property of the delete-of-factor projection form is that the generalized word length patterns (GWLPs) of the projections are ordered. For details on the GWLP, see *Statistical properties of orthogonal arrays*.

```python
print('GWLP %s' % str(al.GWLP()))
for ii in range(0, al.n_columns):
    bl = al.deleteColumn(ii)
    print('Delete column %d: GWLP %s' % (ii, str(bl.GWLP())))

GWLP (1.0, 0.0, 0.0, 3.5, 2.5, 0.5, 0.5, 0.0)
Delete column 0: GWLP (1.0, 0.0, 0.0, 1.5, 1.0, 0.5, 0.0)
Delete column 1: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
Delete column 2: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
Delete column 3: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
Delete column 4: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
Delete column 5: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
Delete column 6: GWLP (1.0, 0.0, 0.0, 3.0, 2.0, 0.0, 0.0)
```

The symmetry group of the projection GWLPs can be calculated. This symmetry group determines how fast an array can be reduced to normal form.

```python
dof_values = oapackage.projectionDOFvalues(al)
sg = oapackage.symmetry_group(dof_values, False)
sg.show(1)
symmetry group: 7 elements, 4 subgroups: 1 2 3 1
```
It is also possible to reduce mixed-level arrays to their delete-of-factor projection forms. We now show an example
involving an orthogonal array with 24 runs, one four-level factor, one three-level factor and three two-level factors.

```
[8]: al = oapackage.exampleArray(5)
adshowarray()
dopgwlp = oapackage.projectionGWLPs(al)
pdopgwlp = [x.raw_values() for x in dopgwlp]
pdopgwlp

array:
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 1 1 1 1
0 2 1 1 1
0 2 1 1 1
1 0 0 1 1
1 0 0 1 1
1 1 0 0 1
1 1 1 1 0
1 2 1 0 0
1 2 1 0 0
2 0 1 0 1
2 0 1 0 1
2 1 0 1 0
2 1 1 0 1
2 2 0 1 0
2 2 0 1 0
3 0 1 1 0
3 0 1 1 0
3 1 0 1 1
3 1 1 0 0
3 2 0 0 1
3 2 0 0 1

delete-one-factor GWLP values [(1.0, 0.0, 0.0, 0.0, 0.6666666666666666), (1.0, 0.0, 0.0
→ 2.111111111111111, 0.0), (1.0, 0.0, 0.0, 1.8888888888888888, 0.4444444444444444),
→ (1.0, 0.0, 0.0, 2.3333333333333335, 0.0), (1.0, 0.0, 0.0, 1.8888888888888888, 0.
→ 4444444444444444)]
```

The delete-of-factor values for mixel-level arrays consists of the factor level of the deleted column and the GWLP of
the projection.

```
[9]: oapackage.projectionGWLPs
dopgwlp = oapackage.projectionGWLPs(al)
dofclass = oapackage.arraylink2arraydata(al)
dofvalues = oapackage.projectionDOFvalues(al)
factor_levels = arrayclass.factor_levels()

for column, dof_value in enumerate(dofvalues):
    print('column %d: factor level %s' % (column, factor_levels[column]))
    print(' delete-of-factor value: %s' % (list(dof_value.raw_values()),))
```

2.10. Example code for delete-one-factor projections

(continues on next page)
Show the reduced array.

```python
[10]: reduced_array=al.reduceDOP()
   al.showarray()
```

array:

```
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 1 1 1 1
0 2 1 1 1
0 2 1 1 1
1 0 0 1 1
1 0 0 1 1
1 1 0 0 1
1 1 1 1 0
1 2 1 0 0
1 2 1 0 0
2 0 1 0 1
2 0 1 0 1
2 1 0 1 0
2 1 1 0 1
2 2 0 1 0
2 2 0 1 0
3 0 1 1 0
3 0 1 1 0
3 1 0 1 1
3 1 1 0 0
3 2 0 0 1
3 2 0 0 1
```

### 2.11 Calculate a Pareto optimal set

Pareto optimality (or multi-objective optimization) allows one to search for optimal solutions for an optimization problem with multiple objectives. The Pareto class in the Orthogonal Array package allows one to calculate the Pareto optimal elements (called the Pareto frontier).

```python
[1]: import numpy as np
   import matplotlib.pyplot as plt
   %matplotlib inline
   import oapackage
```

First, define a dataset of 50 random vectors. The vectors have length 2, so there are 2 objectives to be optimized.

```python
[2]: datapoints=np.random.rand(2, 50)
   for ii in range(0, datapoints.shape[1]):
       w=datapoints[:,ii]
```
Calculate a Pareto optimal set

```python
define fac = 0.6 + 0.4 * np.linalg.norm(w)
data points[:, ii] = (1 / fac) * w

h = plt.plot(data points[0, :], data points[1, :], '.b', markersize=16, label='Non Pareto-optimal')
plt.title('The input data', fontsize=15)
plt.xlabel('Objective 1', fontsize=16)
plt.ylabel('Objective 2', fontsize=16)

Create a structure (ParetoDoubleLong) to keep track of the data.

```python
pareto = oapackage.ParetoDoubleLong()
for ii in range(0, data points.shape[1]):
    w = oapackage.doubleVector((data points[0, ii], data points[1, ii]))
    pareto.addvalue(w, ii)
pareto.show( verbose=1)

Pareto: 12 optimal values, 12 objects

Plot the results.

```python
lst = pareto.allindices()  # the indices of the Pareto optimal designs

optimal_data points = data points[:, lst]

h = plt.plot(data points[0, :], data points[1, :], '.b', markersize=16, label='Non Pareto-optimal')
hp = plt.plot(optimal_data points[0, :], optimal_data points[1, :], '.r', markersize=16, label='Pareto optimal')
plt.xlabel('Objective 1', fontsize=16)
plt.ylabel('Objective 2', fontsize=16)
plt.xticks([])
plt.yticks([])
plt.legend(loc=3, numpoints=1)
```

2.11. Calculate a Pareto optimal set
2.12 Isomorphism reduction for conference designs

In this example, we show how to test if two conference designs are isomorphic. We consider conference designs with 10 runs and 3 factors, and calculate a reduction to their normal form. Using the reduction, we determine if the two designs are isomorphic.

[1]:
```python
import oapackage
import numpy as np
A=np.array([0,1,1, 1,0,1, 1,1,1, 1,1,-1, 1,1,-1, 1,-1,1, 1,-1,1, 1,-1,-1, 1,-1,]).reshape(10,3)
B=np.array([0,1,1, 1,0,-1, 1,1,1, 1,1,1, 1,1,-1, 1,1,-1, 1,-1,0, 1,-1,1, 1,-1,1, 1,-1,]).reshape(10,3)
array1=oapackage.makearraylink( A )
array2=oapackage.makearraylink( B )
array1.showarray()
array2.showarray()
```

(array:
0 1 1
1 0 1
1 1 0
1 1 1
1 1 -1
1 1 -1
1 -1 1
1 -1 1
1 -1 -1
1 -1 -1

(array:
0 1 1
1 0 -1
1 1 1
1 1 1
1 1 -1
1 1 -1

(continues on next page)
We calculate the normal forms of the conference designs using the function `reduceConference` or `reduceConferenceTransformation`. The result of the former is the reduced design, while the result of the latter is an object describing the transformation to normal form. The normal form is calculated using Nauty.

```
[2]: help(oapackage.reduceConference)
Help on function reduceConference in module oalib:
reduceConference(arg1, verbose=0)
    reduceConference(array_link arg1, int verbose=0) -> array_link
    reduceConference(array_link arg1) -> array_link

    Reduce conference matrix to normal form using Nauty

    See also: reduceConferenceTransformation

[3]: T1=oapackage.reduceConferenceTransformation(array1, verbose=1)
T2=oapackage.reduceConferenceTransformation(array2, verbose=1)
T1.show()
reduceConferenceTransformation: reduce design with 10 rows, 3 columns
    row permutation: {2,0,1,9,7,8,5,6,3,4}
    row flips: {1,1,-1,1,1,1,1,1,1,1}
    column permutation: {0,1,2}
    col flips: {1,-1,-1}

We can check whether the designs are isomorphic by comparing the normal forms.

```

```
[4]: design_equal = T1.apply(array1)==T2.apply(array2)
print('designs isomorphic? %s' % design_equal)
designs isomorphic? 1

The designs are isomorphic. So, it is possible to calculate a reduction of the second design into the first design.

```

```
[5]: TT=T1.inverse() * T2
TT.show()
    row permutation: {0,1,8,9,6,7,2,4,5,3}
    row flips: {1,1,-1,1,1,1,1,1,1,1}
    column permutation: {0,1,2}
    col flips: {1,-1,-1}

[6]: r1=T1.apply(array1)
r1.showarray()
r2=T2.apply(array2)
r2.showarray()

```
Calculate some properties of a conference design; see the section Properties of conference designs for details.

```
[7]: print('array 1: is_conference() %d' % array1.is_conference() )
print('array 1: J2-characteristics %s' % (oapackage.Jcharacteristics_
conference(array1, number_of_columns=2), ) )
array 1: is_conference() 1
array 1: J2-characteristics (0, 0, 0)
```

```
[8]: help(oapackage.Jcharacteristics_conference)
Help on function Jcharacteristics_conference in module oalib:

Jcharacteristics_conference(array, number_of_columns, verbose=0)
    Jcharacteristics_conference(array_link array, int number_of_columns, int_
    →verbose=0) -> intVector
Jcharacteristics_conference(array_link array, int number_of_columns) -> intVector
```
All designs handled by the OApacakge are integer valued. The designs (whether these are orthogonal arrays, optimal designs or conferences designs) are stored in an \texttt{array\_link()} object. The definitions of orthogonal arrays, optimal designs and conference designs are included in the section \textit{Definitions of arrays and designs}.

### 3.1 Data structures

The package contains several data structures. Here, we describe the main structures and their use.

- \texttt{array\_link()} The structure containing an orthogonal array is called the \texttt{array\_link()} structure. Lists of arrays are stored in the \texttt{arraylist\_t()} object, which is implemented as a std::deque container.

- \texttt{arrayfile\_t()} This object allows for reading and writing of arrays to disk.

- \texttt{arraydata\_t()} The structure describing a certain class of orthogonal arrays or optimal designs.

- \texttt{conference\_t()} The structure describing a certain class of conference designs.

- \texttt{array\_transformation\_t()} The structure describing a transformation of an orthogonal array, which includes row and column permutations, as well as permutations to the symbols in one or more columns.

- \texttt{conference\_transformation\_t()} The structure describing a transformation of conference design or double conference design, which includes row and column permutations, as well as sign switches to the elements in one or more rows and columns.

### 3.2 Representing arrays

The structure containing an orthogonal array is called the \texttt{array\_link} structure. It consists of a specified number of rows and columns, the data (integer valued) and an index. In the Python interface, the \texttt{array\_link()} object can be indexed just as normal arrays.

It is also possible to convert to a Numpy array. The \texttt{array\_link} object implements the Python array interface, so most operations from packages such as Numpy work on the \texttt{array\_link()} object.

#### Array representation and indexing in Python

```python
>>> import oapackage; import numpy as np
>>> al=oapackage.exampleArray(0)
>>> al.showarray()
...`
The C++ class is `array_link`.

### 3.3 Classes of arrays

The `arraydata_t` object represents data about a class of orthogonal arrays, e.g. the class OA($N; t; s^k$); see Definitions of arrays and designs. The `conference_t` object represents data about a class of conference designs.

### 3.4 Array transformations

Transformations of (orthogonal) arrays consist of row, column and level permutations. A transformation is represented by the `array_transformation_t` object.

For a given transformation, the column permutations are applied first, then the level permutations and finally the row permutations. The level and column permutations are not commutative.

The conference design transformations also allow for row sign switches and are described by the class `conference_transformation_t`.
3.5 Reading and writing arrays

Reading and writing arrays to disk can be done with the arrayfile_t class.

Write an array or a list of arrays to disk

```python
>>> import oapackage
>>> list_of_arrays = [oapackage.exampleArray(24), oapackage.exampleArray(25)]
>>> _ = oapackage.writearrayfile('test.oa', list_of_arrays)
>>> oapackage.oainfo('test.oa')
file test.oa: 64 rows, 16 columns, 2 arrays, mode text, nbits 0
>>> al=oapackage.exampleArray()
>>> af=oapackage.arrayfile_t('test.oa', al.n_rows, al.n_columns)
>>> af.append_array(al)
>>> print(af)
file test.oa: 8 rows, 2 columns, 1 arrays, mode text, nbits 8
>>> af.closefile()
```

The arrays can be written in text or binary format. For more details on the file format, see the section File formats.

The Python interface is oalib.arrayfile_t() and the C++ interface is arrayfile_t.

3.6 File formats

The Orthogonal Array package stores arrays in a custom file format. There is a text format which is easily readable by humans and a binary format which is faster to process and memory efficient.

3.6.1 Plain text array files

Arrays are stored in plain text files with extension .oa. The first line contains the number of columns, the number of rows and the number of arrays (or -1 if the number of arrays is not specified). Then, for each array, a single line with the index of the array, followed by \(N\) lines containing the array.

A typical example of a text file is the following:

```
5 8 1
1
0 0 0 0 0 0 1 1
0 1 1 0 0
0 1 1 1 1 1 0 1
1 0 1 0 1
1 0 1 1 0
1 1 0 0 1
1 1 0 1 0
-1
```

This file contains exactly 1 array with 8 rows and 5 columns.
3.6.2 Binary array files

Every binary file starts with a header, which has the following format:

```
[INT32] 65 (magic identifier)
[INT32] b: Format: number of bits per number. Currently supported are 1 and 8
[INT32] N: number of rows
[INT32] k: number of columns
[INT32] Number of arrays (can be -1 if unknown)
[INT32] Reserved integer
[INT32] Reserved integer
```

The format of the remainder of the binary file depends on the binary format specified. For the normal binary format, the format is as follows. For each array, the number is specified in the header:

```
[INT32] Index
[ Nxk elements] The elements contain b bits
```

If the number of bits per number is 1 (e.g. a 2-level array), then the data is padded with zeros to a multiple of 64 bits. The data of the array is stored in column-major order. The binary file format allows for random access reading and writing. The binary diff and binary diff zero formats are special formats.

A binary array file can be compressed using gzip. Most tools in the Orthogonal Array package can read these compressed files transparently. Writing to compressed array files is not supported at the moment.

3.6.3 Data files

The analysis tool (oaanalyse) writes data to disk in binary format. The format consists of a binary header:

```
[FLOAT64] Magic number 30397995;
[FLOAT64] Magic number 12224883;
[FLOAT64] nc: Number of rows
[FLOAT64] nr: Number of columns
```

After the header there follow nc×nr [FLOAT64] values.

3.6.4 MD5 sums

To check data integrity on disk, the packages includes functions to generate MD5 sums of designs.

```
Calculate md5 sum of a design

>>> import oapackage; al=oapackage.exampleArray(0)
>>> al.md5()
'6454c492239a8e01e3c01a864583abf2'
```

The C++ functions are array_link::md5() and md5().
3.7 Command line interface

Several command line tools are included in the Orthogonal Array package. For each tool, help can be obtained from the command line by using the switch \( -h \). The tools include the following:

- **oainfo** This program reads Orthogonal Array package data files and reports the contents of the files. For example:

  ```bash
  $ oainfo result-8.2-2-2-2.oa
  Orthogonal Array package 1.8.7
  oainfo: reading 1 file(s)
  file result-8.2-2-2.oa: 8 rows, 3 columns, 2 arrays, mode text, nbits 0
  $```

- **oacat** Shows the contents of a file with orthogonal arrays for a data file.
- **oacheck** Checks or reduces an array to canonical form.
- **oaextendingle** Extends a set of arrays in LMC form with one or more columns.
- **oacat** Shows the contents of an array file or data file.
  
  **Usage:**
  oacat [OPTIONS] [FILES]

- **oajoin** Reads one or more files from disk and join all the array files into a single list.
- **oasplit** Takes a single array file as input and splits the arrays into a specified number of output files.
- **oapareto** Calculates the set of Pareto optimal arrays in a file with arrays.
- **oanlyse** Calculates various statistical properties of arrays in a file. The properties are described in section *Properties of designs*.

Fig. 1: Orthogonal array in OA(18, 23^a, 2).
The Orthogonal Array package can be used to generate several classes of arrays and designs. A collection of arrays and designs generated by the package is available on the website http://www.pietereendebak.nl/oapackage/index.html.

4.1 Generation of orthogonal arrays

A list of arrays in LMC form (i.e., lexicographically minimum in columns) can be extended to a list of arrays in LMC form with one additional column. Details about the algorithm are described in [SEN10].

The main function for array extension is the function `extend_arraylist()`. The arguments for this function are the list of arrays to extend, a specification of the class of arrays in `arraydata_t` and the options `OAextend` for the algorithm.

An example of a session that extends an array is:

```python
>>> import oapackage
>>> nrows=8; ncols=3;
>>> arrayclass=oapackage.arraydata_t(2, nrows, 2, ncols)
>>> root_array=arrayclass.create_root()
>>> root_array.showarraycompact()
00
00
01
01
10
10
11
11
>>> array_list=oapackage.extend_array(root_array, arrayclass)
>>> print('found %d extensions of the root array' % len(array_list))
found 2 extensions of the root array
```

A more detailed example is included in *Enumerate orthogonal arrays*. 
4.2 Conference designs

A conference design is an $N \times k$ matrix with entries 0, -1, +1 such that i) in each column the symbol 0 occurs exactly one time and ii) all columns are orthogonal to each other. For details on conference designs, see the section Properties of conference designs and [SEG19]. An example of a session to generate conference designs is the following:

Generate conference designs with 8 rows

```python
>>> import oapackage
>>> conference_class=oapackage.conference_t(8, 6, 0)
>>> array = conference_class.create_root_three_columns()
>>> array.showarray()
array:
    0 1 1
    1 0 -1
    1 1 0
    1 1 1
    1 1 -1
    1 -1 1
    1 -1 1
    1 -1 -1

>>> l4=oapackage.extend_conference ([array], conference_class, verbose=0)
>>> l5=oapackage.extend_conference ( l4, conference_class, verbose=0)
>>> l6=oapackage.extend_conference ( l5, conference_class, verbose=0)
>>> print('number of non-isomorphic conference designs with 6 columns: %d ' % len(l6))
number of non-isomorphic conference designs with 6 columns: 11
```

An example notebook with more functionality is included in Generation and analysis of conference designs. The full interface for conference designs is available in the Interface for conference designs.

The main functions to extend conference and double conference designs are extend_conference() and extend_double_conference(), respectively. The low-level functions for generating candidate extension columns of conference and double conference designs are generateConferenceExtensions() and generateDoubleConferenceExtensions(), respectively.

The conference designs are generated in LMC0 form.

4.3 Calculation of D-efficient designs

D-efficient designs (sometimes called D-optimal designs) can be calculated with the function oapackage.Doptim.Doptimize(). This function uses a coordinate-exchange algorithm to generate designs with good properties for the $D$-efficiency. With the coordinate-exchange algorithm, the following target function $T$ is optimized:

$$T = \alpha_1 D_{eff} + \alpha_2 D_{s,eff} + \alpha_3 D_{1,eff}$$

Here, $\alpha$ is a weight vector specified by the user. Details on the $D_{eff}$, $D_{s,eff}$ and $D_{1,eff}$ can be found in the section Optimality criteria for D-efficient designs.

A Python script to generate D-efficient designs with 40 runs and 7 factors is shown below.

Example of Doptimize usage
We generate optimal designs with: arrayclass: N 40, k 7, strength 0, s {2,2,2,2,2,2,2}, order 0

Generated 10 designs, the best D-efficiency is 0.9198

The parameters of the Doptimize() function are documented in the code.

To calculate the $D_1$, $D_{s1}$- and $D_1$-efficiencies of the designs, we can use the method Defficiencies(). For details on these efficiencies, see the section Optimality criteria for D-efficient designs and [ES17].

In [ES17], it is shown that one can optimize a linear combination of the $D$-efficiency and $D_{s1}$-efficiency to generate a rich set of good compromise designs. From the generated designs, the optimal ones according to Pareto optimality can be selected.

Fig. 1: Scatterplot for the $D$-efficiency and $D_{s1}$-efficiency for generated designs in OA(40; 2; 2^7). The Pareto optimal designs are colored, while the non-Pareto optimal designs are grey. For reference the strength-3 orthogonal array with highest D-efficiency is also included in the plot.

4.3. Calculation of D-efficient designs
4.4 Even-odd arrays

The even-odd arrays are a special class of orthogonal arrays with at least one of the odd $J_k$-characteristics unequal to zero. More information on this class of designs will appear later.
NORMAL FORM OF ARRAYS

The Orthogonal Array package contains functions to reduce arrays and designs to canonical form with respect to some ordering. The default ordering for orthogonal arrays is the lexicographic ordering in columns [SEN10]. The default ordering for conference designs is the LMC0 ordering [SEG19]. Alternative orderings include the delete-one-factor projection ordering introduced in [Een13] or the even-odd ordering. For a given ordering of a set of arrays, the minimal element of all arrays in an isomorphism class defines a unique representative of that isomorphism class.

Specialized packages such as Nauty [McK81], [MP13] can also reduce arrays to their canonical form using state-of-the-art, graph-isomorphism methods. However, these methods do not take into account the special structure of the arrays and so, they cannot be tailored to create normal forms of a specific form.

5.1 Reduction to LMC normal form

The Orthogonal Array package implements theory and methods from the article Complete enumeration of pure-level and mixed-level orthogonal arrays, Schoen et al. to reduce orthogonal arrays to their LMC normal form. The C++ function to perform the reduction is `reduceLMCform()`. An example on how to use this function is shown below.

```python
>>> import oapackage
>>> oapackage.set_srand(1)
>>> array = oapackage.exampleArray(1, 0).selectFirstColumns(3)
>>> array = array.randomperm()
>>> print('input array:'); array.transposed().showarraycompact()
input array:
1100010111010001
0101100110100011
0000001111101101
>>> reduced_array = oapackage.reduceLMCform(array)
>>> print('reduced array:'); reduced_array.transposed().showarraycompact()
reduced array:
0000000011111111
0000011100001111
0000101101111001
```

It is also possible to check whether an array is in normal form with the `LMCcheck()` method:

```python
>>> import oapackage
>>> array = oapackage.exampleArray(1)
>>> lmc_type = oapackage.LMCcheck(array)
>>> if lmc_type == oapackage.LMC_MORE:
...     print('array is in minimal form')
... elif lmc_type == oapackage.LMC_LESS:
...     print('array is not in minimal form')
array is in minimal form
```
5.2 Reduction to delete-one-factor projection form

The article A canonical form for non-regular arrays based on generalized word length pattern values of delete-one-factor projections [Een13] describes a canonical form of an orthogonal array based on delete-one-factor projections. The C++ interface to delete-one-factor projection form is `reduceDOPform()`. The reduction method works well for large arrays with a large variation in the projection values.

An example on how to use this reduction is shown in Example code for delete-one-factor projections, which can be found in the example notebooks section.

5.3 Reduction using graph isomorphisms

The function `reduceOAnauty()` reduces an orthogonal array to Nauty canonical form. To reduce general graphs to Nauty canonical form, the Orthogonal Array package includes the function `reduceGraphNauty()`.

Reduce a design to normal form using Nauty

```python
>>> oapackage.set_srand(1)
>>> al = oapackage.exampleArray(0).randomperm()
>>> al.showarray()
aarray:
0 0
0 1
1 1
0 1
1 0
0 0
1 0
1 1
>>> transformation=oapackage.reduceOAnauty(al, 0)
>>> transformation.show()
aarray transformation: N 8
column permutation: {0,1}
level perms:
{0,1}
{0,1}
row permutation: {0,5,1,3,4,6,2,7}
>>> alr=transformation.apply(al)
>>> alr.showarray()
aarray:
0 0
0 0
0 1
0 1
1 0
1 0
1 1
1 1
```
5.4 Normal forms for conference designs

For conference designs, a convenient normal form is the LMC0 ordering (sometimes also called L0 ordering) [SEG19].

LMC0 ordering

The LMC0 ordering for conference designs is defined in three steps:

**Definition LMC0 i:** Order of elements  The LMC0 order of the factor levels -1, 0 and +1 is 0 < +1 < -1.

**Definition LMC0 ii:** Order of columns  A column a is smaller than a column b according to LMC0 ordering, notated as a < b, if either of the following conditions hold:

1. If we replace the values -1 by +1 in both columns, then the first element where the columns differ is smaller in a than in b according to Definition 1.
2. The zeros in column a are in the same position as the zeros in column b, and the first element where the columns differ is smaller in a than in b according to Definition i.

**Definition LMC0 iii:** Order of designs  Conference design A is smaller than conference design B according to LMC0 ordering, notated as A < B, if the first column where the designs differ is smaller in A than in B.

The definition implies that the ordering of designs is column-by-column and that the position of zeros in the columns is dominant over the values +1, -1. To check whether a design is in LMC0 form, we can use \texttt{LMC0check()}:

Conference design in normal form

```python
>>> array = oapackage.exampleArray(53,1)
documentation 53: third array in C(12,4)
>>> array.showarray()
array:
  0  1  1  1
  1  0  1  1
  1  1  0  1
  1  1  1  0
  1  1  1  1
  1  1  1  1
  1  1  1  1
  1  1  1  1
  1  1  1  1
  1  1  1  1
  1  1  1  1
  1  1  1  1
>>> oapackage.LMC0check(array) == oapackage.LMC_LESS
True
```
**CHAPTER SIX**

**PROPERTIES OF DESIGNS**

This section shows the structural and statistical properties of the orthogonal arrays, conference designs and D-efficient designs generated by the Orthogonal Array package. The properties of the arrays and designs are calculated using the `array_link` object or functions from the package.

### 6.1 Definitions of arrays and designs

Before introducing the structural and statistical properties, we define orthogonal arrays, conference designs and D-efficient designs:

An orthogonal array (OA) of strength $t$, $N$ runs and $n$ factors at $s$ levels is an $N \times n$ array of symbols $0, \ldots, (s - 1)$, such that for every subset of $t$ columns, every $t$-tuple occurs equally often [Rao47]. The set of all strength-$t$ OAs with $N$ runs and $n$ factors at $s$ levels is denoted by $\text{OA}(N; t; s^n)$. If $s = 2$, the OA is called a two-level OA and the set of all strength-$t$ two-level OAs with $N$ runs and $n$ factors is denoted as $\text{OA}(N; t; 2^n)$.

For $N$ even, a conference design [SEG19] $C$ is an $N \times n$ array which satisfies $C^T C = (n - 1)I_n$, with $C_{ii} = 0$ and $C_{ij} \in \{-1, 1\}$, for $i \neq j$ and $i, j = 1, \ldots, n$. A $N \times N$ conference design $E$ such that $E^T E = (n - 1)I_n$ is called a conference matrix; see [EN95], [CD06] and [XLB12].

A D-optimal design [DAT07] $(X)$ is an $N \times n$ array which maximizes the $D$-efficiency, defined as $(\det(X_M^T X_M)^{1/p})/N$, for a given $N \times p$ model matrix $X_M$ (for details see Model matrices). The Orthogonal Array package uses a coordinate-exchange algorithm to generate designs that optimize the $D$-efficiency. Since there is no guarantee that the resulting designs have the largest possible $D$-efficiency, we refer to them as D-efficient designs in this documentation. An orthogonal array is called D-optimal orthogonal array if it provides the largest $D$-efficiency among all comparable orthogonal arrays.

### 6.2 Structural properties of an array

The OApackage can calculate the rank of an array, defined as the maximum number of linearly independent column or row vectors in the array. The rank of an array is useful for several other functions in the package. For two-level arrays, the OApackage can also check if the arrays are foldover arrays. A two-level array is called a foldover array if half of its runs images of the other half, in the sense that the factor levels are changed from 0 to 1 and from 1 to 0.

For example, to calculate the rank of a two-level orthogonal array and determine whether the array is a foldover array, one can use the methods `array_link::rank()` and `array_link::foldover()`:

---

**Calculate rank of array and test for foldover**

---
>>> array = oapackage.exampleArray(1) # Select an example two-level orthogonal array
>>> array.showarray() # Show the two-level orthogonal array
array:
 0 0 0 0 0
 0 0 0 0 0
 0 0 1 1 1
 0 0 1 0 1
 0 1 0 1 0
 0 1 1 0 0
 0 1 1 1 1
 0 1 1 1 1
 1 0 0 1 1
 1 0 1 0 1
 1 0 1 1 0
 1 0 1 1 0
 1 1 0 0 1
 1 1 0 1 0
 1 1 1 0 0

>>> print(array.rank()) # Calculate the rank of the array
5
>>> print(array.foldover()) # Determine if the array is foldover
False

Other structural properties such as whether an array involves two levels or is symmetric can be found in the documentation of array_link, which shows the full set of methods available.

### 6.3 Model matrices

For orthogonal arrays and conference designs, the OApackage can calculate different model matrices. The model matrices available depend on the type of array and design.

**Model matrices for two-level orthogonal arrays**

For two-level orthogonal arrays, the levels of the array are first coded according to the map $0 \rightarrow -1$ and $1 \rightarrow +1$. The coded matrix is referred to as the design matrix. The main effect contrast vectors are given by the columns in the design matrix. The contrast vectors associated to the two-factor interactions are calculated by taking products between two different columns in the design matrix. The model matrix consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the two-factor interactions.

**Model matrices for conference designs**

The model matrix for a conference design consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the second-order effects (two-factor interactions and quadratic effects). The main effect contrast vectors are given by the columns in the conference design. The contrast vectors associated to the second-order effects are calculated by taking products between two columns in the conference design.
Model matrices for mixed-level orthogonal arrays

For mixed-level orthogonal arrays, the main effect contrast vectors are defined by the Helmert contrasts. The contrast vectors associated to the two-factor interactions are calculated by taking products between two different columns in the matrix containing the Helmert contrasts of the array; see Model matrices for mixed-level orthogonal arrays for details. The model matrix consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the two-factor interactions.

An example on how to generate an interaction model matrix for a two-level orthogonal array is shown below.

Calculate interaction effects model matrix

```matlab
>>> array=oapackage.exampleArray(0,1)
exampleArray 0: array in OA(8,2, 2^2)
```

```matlab
array.showarray()
array:
 0 0
 0 0
 0 1
 0 1
 1 0
 1 0
 1 1
 1 1
```

```matlab
>>> M=oapackage.array2modelmatrix(array, 'i')
>>> print(M)
[[ 1. -1. -1. 1.]
 [ 1. -1. -1. 1.]
 [ 1. -1. 1. -1.]
 [ 1. -1. 1. -1.]
 [ 1. 1. -1. -1.]
 [ 1. 1. -1. -1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]]
```

6.4 Statistical properties of orthogonal arrays

Orthogonal arrays are commonly evaluated in terms of their generalized wordlength pattern \([XW01]\) (GWLP). Two-level OAs are also commonly evaluated in terms of their \(J_k\)-characteristics and \(F\)-vectors \([DT99]\). The OApackage can calculate all these statistical criteria: \(array\_link::GWLP()\), \(array\_link::Fvalues()\), \(array\_link::Jcharacteristics()\).

The following example shows how to calculate the GWLP, \(F_k\)-values and \(J_k\)-characteristics from an \(array\_link\) object:

Calculate GWLP and F-values

```matlab
>>> al=oapackage.exampleArray(1,1) # Select an example array
exampleArray 1: array 3 in OA(16, 2, 2^5)
```

```matlab
>>> gwlp = al.GWLP() # Calculate its generalized word length pattern
```

```matlab
>>> print('GWLP: %s' % str(gwlp))
```

(continues on next page)
GWLP: (1.0, 0.0, 0.0, 1.0, 1.0, 0.0)

```python
>>> print('F3-value: ' + str(al.Fvalues(3)))  # Calculate the F_3 values
F3-value: (4, 6)
```

```python
>>> print('F4-value: ' + str(al.Fvalues(4)))  # Calculate the F_3 values
F4-value: (1, 4)
```

```python
>>> print('J3-characteristics: ' + str(al.Jcharacteristics(3)))  # Calculate the J_3-
˓→characteristics
J3-characteristics: (-8, -8, 0, 0, -8, 0, -8, 0, 0)
```

We now briefly mention some technical details of the $J_k$-characteristics, the $F_k$-values and the GWLP.

### $J_k$-characteristics

To calculate $J_k$-characteristics of a two-level OA, the OApackage codes the levels of the array as $-1$ and $+1$. To this end, the package uses the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$. Let $D$ be an $N \times n$ with coded levels $-1$ and $+1$. For $S = \{l_1, \ldots, l_k\}$, a subset of $k$ different factors of $D = (d_{il})$, define

$$j_k(S; D) = \sum_{i=1}^{N} d_{i1} \cdots d_{ik}.$$ 

The $|j_k(S; D)|$ values are called the $J_k$-characteristics, which necessarily equal $N - 4q$ [DT02], where $q \leq N/4$ is a non-negative integer.

### $F_k$-values

The $F_k$-vector collects the frequencies of all the $J_k$-characteristics. More specifically, the vector $F_k = (f_{k1}, \ldots, f_{kv})$, where $v = N/4$ and $f_{ku}$ denotes the frequency of the $J_k$-characteristics which are equal to $4(v+1-u)$. When calculating an $F_k$-vector, the OApackage shows only the vector $(f_{k1}, \ldots, f_{kv})$, whose elements are referred to as the $F_k$-values.

### Generalized word length pattern

Consider an OA, $D$, of strength $t$ with $N$ runs and $n$ factors at $s$ levels. Let $X_0$ be a column of ones, $X_1$ the matrix involving the contrast vectors associated with the main effects, and $X_j$ the matrix involving the contrast vectors associated with the $j$-factor interactions, $j \geq 2$. We assume that the column vectors in $X_1$ are normalized so that they have the same length $\sqrt{N}$. For $j = 0, \ldots, n$, let

$$A_j(D) = N^{-2}1_N^T X_j X_j^T 1_N,$$

where $1_N$ denotes the $N \times 1$ column of ones. The value of $A_j(D)$ is invariant to the choice of the orthonormal contrasts used; see [XW01] for details. The vector $(A_0(D), \ldots, A_n(D))$ is called the generalized word length pattern (GWLP). To increase the speed of the computations for the GWLP, the OApackage uses the distance distribution and the MacWilliams identities as in [XW01] and [Xu09].
6.5 Optimality criteria for D-efficient designs

In [ES17], D-efficient designs for the model including the intercept, all main effects and all two-factor interactions are generated. The OApackage provides functionality to compute the optimality criteria used to generate the D-efficient designs in [ES17]. Moreover, the package can calculate the well-known $A$- and $E$-optimality criteria from the literature on Optimal Experimental Design [DAT07]. The functions to perform the calculations are `Defficiency()`, `DsEfficiency()`, `Aefficiency()` and `Eefficiency()`.

The following example shows how to calculate the $D$-, $D_s$-, $A$- and $E$-efficiency for a design that permits the estimation of the interaction model.

```
# Select an array that can estimate the interaction model
>>> a1 = oapackage.exampleArray(11, 1)
exampleArray 11: D-optimal array in OA(44, 2^8)
>>> print('D-efficiency: %.4f' % a1.Defficiency())
D-efficiency: 0.8879
>>> print('Ds-efficiency (Eendebak and Schoen, 2017): %.4f' % a1.DsEfficiency())
Ds-efficiency (Eendebak and Schoen, 2017): 0.8059
>>> print('A-efficiency for the interaction model: %.4f' % a1.Aefficiency())
A-efficiency for the interaction model: 0.7906
>>> print('E-efficiency for the interaction model: %.4f' % a1.Eefficiency())
E-efficiency for the interaction model: 0.3602
```

### Calculation of $D_s$-, $A$- and $E$-efficiency

Let $X$ be again the $N \times p$ interaction model matrix (see section Model matrices) consisting of a column of ones and the contrast vectors associated to the main and two-factor interactions of $n$ factors, where $p = 1 + n + (n)(n - 1)/2$. The $D_s$-, $A$- and $E$-efficiency are calculated using the eigenvalues of the singular-value decomposition (SVD) of $X$. To calculate the rank of a matrix, the lower-upper (LU) decomposition, as implemented in the Eigen package [Gl+10], is used.

Let $\lambda_1, \ldots, \lambda_p$ be the eigenvalues of the SVD of $X$. The OApackage calculates the $D_s$-, $A$- and $E$-efficiency of a design $D$ as follows:

$$D_{\text{eff}}(D) = \left( \prod_j \lambda_j \right)^{1/p}/N,$$

$$A_{\text{eff}}(D) = N(\sum_j \lambda_j^{-1})/m,$$

$$E_{\text{eff}}(D) = \min_j \lambda_j.$$

### $D_s$-efficiency and $D_1$-efficiency

In [ES17], the $D_s$-efficiency is used to assess the joint precision of the main effects in the interaction model. Let the interaction model matrix $X$ be split into $X_1$, containing the contrast vectors associated with the main effects only, and $X_{02}$, containing the intercept column and the contrast vectors associated
to the two-factor interactions. The $D_s$-criterion of a design $D$ is defined as

$$D_{s,\text{crit}}(D) = \frac{\det(X^T X)}{\det(X_{02}^T X_{02})},$$

where $X_{02}$ is necessarily of full rank. Similar to the calculations of the $D$-efficiency, the OApackage calculates the $D_s$-criterion using the eigenvalues of the SVD of the matrices $X$ and $X_{01}$. Finally, the package calculates the $D_s$-efficiency of $D$ as $D_{s,\text{eff}}(A) = D_{s,\text{crit}}(A)^{1/m}$, where $m$ is the number of factors.

In a similar way the $D_1$-efficiency of a design $A$ with $n$ factors and model matrix of intercept and main effects $X_{01}$, is defined as

$$D_{1,\text{eff}}(A) = \left(\det((X_{01})^T (X_{01}))\right)^{1/(n+1)}.$$  

### 6.6 Projection capacities

Other relevant statistical criteria to evaluate a two-level design with $N$ runs and $k$ factors include the so-called projection estimation capacity (PEC) and projection information capacity (PIC) [LST07]. These criteria focus on the projections of the two-level design onto a smaller number of factors. More specifically, the PEC and PIC summarize the performance of all the $N$-run subdesigns with $l \leq k$ factors in terms of the capacity to estimate the interaction model and the $D$-efficiency for this model, respectively.

The PEC and PIC are based on the so-called PEC and PIC sequences, which are formally defined as follows. Let $PEC_l$ denote the proportion of $N$-run $l$-factor subdesigns that permit the estimation of the interaction model in $l$ factors, that is, the model including the intercept, all $l$ main effects and all $l(l-1)/2$ two-factor interactions. The PEC sequence is the vector $(PEC_1, PEC_2, \ldots, PEC_k)$. Now, let $PIC_l$ denote the average $D$-efficiency for the interaction model in $l$ factors across all $N$-run $l$-factor subdesigns. The PIC sequence is the vector $(PIC_1, PIC_2, \ldots, PIC_k)$. The OApackage can calculate the PEC and PIC sequences of two-level designs with `PECsequence()` and `PICsequence()`, respectively.

The following example shows how to compute the PEC and PIC sequences of a two-level orthogonal array using the OApackage.

#### Calculate the PEC and PIC sequences

```plaintext
>>> al=oapackage.exampleArray(1,1)
exmapleArray 1: array 3 in OA(16, 2, 2^5)
>>> PEC = al.PECsequence()
>>> print('PEC sequence: %s' % ', '.join(['%.2f' % x for x in PEC]))
PEC sequence: 1.00,1.00,1.00,0.80,0.00
>>> PIC = al.PICsequence()
>>> print('PIC sequence: %s' % ', '.join(['%.2f' % x for x in PIC]))
PIC sequence: 1.00,1.00,0.95,0.66,0.00
```
6.7 Properties of conference designs

In [SEG19], it is shown that the $F_4$ vector is useful for classifying definitive screening designs [XLB12] that are generated by folding over a conference design. To calculate the $F_4$ vector, we first need to compute the $J_4$-characteristics of the conference design. The calculations for the $J_k$-characteristics of conference designs are similar as for orthogonal arrays; see *Statistical properties of orthogonal arrays*. Consider a definitive screening design constructed from an $N$-run conference design with at least four factors. The $F_4$ vector of this design collects the frequencies of the $J_4$-characteristics of $2N\lambda$ for $\lambda = 1, \ldots, N/4$ when $N$ is a multiple of 4, or $\lambda = 1, \ldots, (N^2)/4$ when $N$ is an odd multiple of 2.

Note: the $J_4$-values of a definitive screening design generated by folding over a conference design are twice the value of the $J_4$-values of the conference design. The $F_4$ vector of a conference design and the corresponding definitive screening design are equal.

### Calculate the F4 vector for a conference design

```python
>>> import oapackage
>>> array = oapackage.exampleArray(47, 1)
exampleArray 47: third conference design in C(20,8)
>>> F4 = array.FvaluesConference(4)
>>> print(F4)
(0, 2, 4, 51, 13)
>>> definitive_screening_design = oapackage.conference2DSD(array)
```

The individual $J_k$-characteristics can be calculated with the method `Jcharacteristics_conference()`. For conference designs, we can calculate the projection statistics using `conferenceProjectionStatistics()`.

### Calculate projection statistics for conference designs

```python
>>> array = oapackage.exampleArray(46, 1)
exampleArray 46: second conference design in C(20,8)
>>> pec, pic, ppc = oapackage.conference.conferenceProjectionStatistics(array)
>>> print('Projection estimation capacity for 4 columns: %.3f' % pec)
Projection estimation capacity for 4 columns: 0.986
>>> J3 = oapackage.Jcharacteristics_conference(array, number_of_columns = 3)
```
The full documentation for the C++ library can be built from the source code using doxygen. For convenience we provide links to the main functions on this page.

### 7.1 Interface for D-optimal designs

Contains functions to generate optimal designs.

For more information see “Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions”, P.T. Eendebak and E.D. Schoen, 2017

**Enums**

```cpp
define coordinate_exchange_method_t
    Different methods for the optimization. The default method DOPTIM_SWAP is a coordinate-exchange algorithm.

    Values:
    - DOPTIM_UPDATE: replace a random element with a random value
    - DOPTIM_SWAP: swap two elements at random
    - DOPTIM_FLIP: randomly flip an element between 0 and 1
    - DOPTIM_AUTOMATIC: automatically select one of the methods
    - DOPTIM_NONE: perform no optimization
```
Functions

```cpp
double scoreD(const std::vector<double> &efficiencies, const std::vector<double> &weights)
```

Calculate score from a set of efficiencies

The score is the weighted sum of the efficiencies.

**Return** Weighted sum of the efficiencies

**Parameters**

- `efficiencies`: Vector with calculated efficiencies
- `weights`: Weights for the efficiencies

```cpp
DoptimReturn Doptimize(const arraydata_t &arrayclass, int nrestarts, const std::vector<double> &alpha, int verbose, coordinate_exchange_method_t method = DOP-TIM_AUTOMATIC, int niter = 300000, double maxtime = 100000, int nabort = 5000)
```

Generates optimal designs for the specified class of designs

The method uses a coordinate-exchange algorithm to optimize a target function defined by the optimization parameters. The optimization is performed multiple times to prevent finding a design in a local minumum of the target function.

The method is described in more detail in “Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions”, Eendebak et al., 2015, Technometrics, https://doi.org/10.1080/00401706.2016.1142903.

**Return** A structure with the generated optimal designs

**Parameters**

- `arrayclass`: Class of designs to optimize
- `nrestarts`: Number of restarts to perform
- `alpha`: Optimization parameters. The target function is $\alpha_1 D + \alpha_2 D_s + \alpha D_1$
- `verbose`: Verbosity level
- `method`: Method for optimization algorithm
- `niter`: Maximum number of iterations for each restart
- `maxtime`: Maximum calculation time. If this time is exceeded, the function is aborted
- `nabort`: Maximum number of iterations when no improvement is found

```cpp
DoptimReturn DoptimizeMixed(const arraylist_t &sols, const arraydata_t &arrayclass, const std::vector<double> &alpha, int verbose = 1, int nabort = -1)
```

Function to generate optimal designs with mixed optimization approach

This function is beta code. See Doptimize for details of the parameters.

```cpp
array_link optimDeff(const array_link &array, const arraydata_t &arrayclass, std::vector<double> &alpha, int verbose = 1, coordinate_exchange_method_t optimmethod = DOP-TIM_AUTOMATIC, int niter = 100000, int n_abort = 0)
```

Optimize a design according to the optimization function specified.

**Arguments:**

**Return** Optimized designs

**Parameters**
• **array**: Array to be optimized
• **arrayclass**: Structure describing the design class
• **alpha**: 3x1 array with optimization parameters
• **verbose**: Verbosity level
• **optimmethod**: Optimization method to use
• **niter**: Number of iterations
• **nabort**: Number of iterations after which to abort when no improvements are found

```c
struct DoptimReturn
    #include <Deff.h> Structure containing results of the Doptimize function

Public Members

std::vector<std::vector<double>> dds
    calculated efficiencies for the generated designs

arraylist_t designs
    designs generated

int nrestarts
    number of restarts performed

int _nimproved
```

### 7.2 Interface for array properties

Contains functions to calculate properties of arrays.

Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

**Defines**

GWLPvalue

**Typedefs**

```c
typedef mvalue_t<double> DOFvalue
    delete-one-factor projection value
```
Enums

enum model_matrix_t

Values:

MODEL_CONSTANT
  only the intercept

MODEL_MAIN
  intercept and main effects

MODEL_INTERACTION
  intercept, main effects and second order interactions

MODEL_SECONDORDER
  intercept, main effects and second order effects (interactions and quadratic effects)

MODEL_INVALID
  invalid model

enum paretomethod_t

Values:

PARETOFUNCTION_DEFAULT

PARETOFUNCTION_J5

Functions

void DAEefficiencyWithSVD (const Eigen::MatrixXd &secondorder_interaction_matrix, double &Deff, double &vif, double &Eff, int &rank, int verbose)

Calculate D-efficiency and VIF-efficiency and E-efficiency values using SVD.

int array2rank_Deff_Beff (const array_link &al, std::vector<double> *ret = 0, int verbose = 0)

Calculate the rank of the second order interaction matrix of an orthogonal array

The model is the intercept, main effects and interaction effects The rank, D-efficiency, VIF-efficiency and E-efficiency are appended to the second argument

The return vector is filled with the rank, Defficiency, VIF efficiency and Eefficiency

double Defficiency (const array_link &orthogonal_array, int verbose = 0)

Calculate D-efficiency for a 2-level array using symmetric eigenvalue decomposition.

std::vector<double> Defficiencies (const array_link &array, const arraydata_t &arrayclass, int verbose = 0, int addDs0 = 0)

Calculate efficiencies for an array

Return Vector with the calculate D-efficiency, the main effect robustness (or Ds-optimality) and D1-efficiency for an orthogonal array

Parameters

  • array: Array to use in calculation
  • arrayclass: Specification of the array class
  • verbose: Verbosity level
  • addDs0: If True, then add the Ds0-efficiency to the output

double VIFefficiency (const array_link &orthogonal_array, int verbose = 0)

Calculate VIF-efficiency of matrix.
double Aefficiency (const array_link &orthogonal_array, int verbose = 0)
Calculate A-efficiency of matrix.

double Eefficiency (const array_link &orthogonal_array, int verbose = 0)
Calculate E-efficiency of matrix (1 over the VIF-efficiency)

std::vector<double> Aefficiencies (const array_link &orthogonal_array, int verbose = 0)
calculate various A-efficiencies

std::vector<double> projDeff (const array_link &array, int number_of_factors, int verbose = 0)
Calculate D-efficiencies for all projection designs

Return Vector with calculated D-efficiencies

Parameters

• array: Design to calculate D-efficiencies for
• number_of_factors: Number of factors into which to project
• verbose: Verbosity level

std::vector<double> PECsequence (const array_link &array, int verbose = 0)
Calculate the projection estimation capacity sequence for a design

The PECk of a design is the fraction of estimable second-order models in k factors. The vector (PEC1, PEC2, …., ) is called the projection estimation capacity sequence. See “Ranking Non-regular Designs”, J.L. Loeppky, 2004.

Return Vector with the calculated PEC sequence

Parameters

• array: Input array
• verbose: Verbosity level

std::vector<double> PICsequence (const array_link &array, int verbose = 0)
Calculate the projection information capacity sequence for a design.

The PICk of a design is the average D-efficiency of estimable second-order models in k factors. The vector (PIC1, PIC2, …., ) is called the PIC sequence.

Return Vector with the calculated PIC sequence

Parameters

• array: Input array
• verbose: Verbosity level

std::vector<double> distance_distribution (const array_link &array)
Return the distance distribution of a design

The distance distribution is described in “Generalized minimum aberration for asymmetrical fractional factorial designs”, Wu and Xu, 2001

std::vector<int> Jcharacteristics (const array_link &array, int number_of_columns = 4, int verbose = 0)
Calculate Jk-characteristics of a matrix

The calculated Jk-values are signed.

Return Vector with calculated Jk-characteristics

Parameters

7.2. Interface for array properties
std::vector<double> GWLP (const array_link &array, int verbose = 0, int truncate = 1)
Calculate GWLP (generalized wordlength pattern)

The method used for calculation is from Xu and Wu (2001), “Generalized minimum aberration for asymmetrical fractional factorial desings”. For non-symmetric arrays see “Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run Sizes”, Xu, Technometrics, 2009.

A more detailed description of the generalized wordlength pattern can also be found in the documentation at https://oapackage.readthedocs.io/.

Return Vector with calculated generalized wordlength pattern

Parameters
• array: Array to calculate the GWLP value for
• verbose: Verbosity level
• truncate: If True then round values near zero to solve double precision errors

std::vector<double> GWLPmixed (const array_link &array, int verbose = 0, int truncate = 1)
Calculate GWLP (generalized wordlength pattern) for mixed-level arrays.

The method used for calculation is from “Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run Sizes”, Xu, Technometrics, 2009.

Return Vector with calculated generalized wordlength pattern

Parameters
• array: Array to calculate the GWLP value for
• verbose: Verbosity level
• truncate: If True then round values near zero to solve double precision errors

std::vector<GWLPvalue> projectionGWLPs (const array_link &al)
calculate delete-one-factor GWLP (generalized wordlength pattern) projections

std::vector<GWLPvalue> sortGWLP (std::vector<GWLPvalue>)
sort a list of GWLP values and return the sorted list

doCulum $C_{L_2}$discrepancy (const array_link &array)
Calculate centered L2-discrepancy of a design

The method is from “A connection between uniformity and aberration in regular fractions of two-level factorials”, Fang and Mukerjee, 2000

array_link array2secondorder (const array_link &array)
Calculate second order interaction model for 2-level array

Return Array interaction effects

Parameters
• **array**: Array to calculate second order interaction model from

```cpp
array_link array2xf(const array_link &array)
```
calculate second order interaction model for 2-level array

**Return** Array with intercept, main effects and interaction effects

**Parameters**

- **array**: Array to calculate second order interaction model from

```cpp
array_link conference_design2modelmatrix(const array_link &conference_design, const char *mode, int verbose = 0)
```
Calculate model matrix for a conference design

**Return** Calculated model matrix

**Parameters**

- **conference_design**: Conference design
- **mode**: Can be ‘m’ for main effects, ‘i’ for interaction effects or ‘q’ for quadratic effects
- **verbose**: Verbosity level

Eigen::MatrixXd ```cpp
array2modelmatrix(const array_link &array, const char *mode, int verbose = 0)
```
Convert orthogonal array or conference design to model matrix

The model matrix consists of the intercept, main effects and (optionally) the interaction effects and quadratic effects. The order in the interaction effects is (c1, c2)=(0,0), (1,0), (2,0), (2,1), ... with c2<c1 for columns c1, c2. The size of the model matrix calculated by this function is given by `array2modelmatrix_sizes`.

For conference designs the method `conference_design2modelmatrix` is used. For orthogonal array the calculated is performed with `array2eigenModelMatrixMixed`.

**Return** Calculated model matrix

**Parameters**

- **array**: Orthogonal array or conference design
- **mode**: Type of model matrix to calculate. Can be ‘m’ for main effects, ‘i’ for interaction effects or ‘q’ for quadratic effects
- **verbose**: Verbosity level

std::vector<int> ```cpp
array2modelmatrix_sizes(const array_link &array)
```
Return the sizes of the model matrices calculated

**Return** List with the sizes of the model matrix for: only intercept; intercept, main; intercept, main, and interaction terms; intercept, main and full second order

**Parameters**

- **array**: Orthogonal array or conference designs

Eigen::MatrixXd ```cpp
array2xfeigen(const array_link &array)
```
calculate second order interaction model for 2-level array

**Return** Array with intercept, main effects and interaction effects

**Parameters**
• array: Array to calculate second order interaction model from

int arrayrankFullPivQR(const array_link &al, double threshold = -1)
    return rank of an array based on Eigen::FullPivHouseholderQR

int arrayrankColPivQR(const array_link &al, double threshold = -1)
    return rank of an array based on Eigen::ColPivHouseholderQR

int arrayrankFullPivLU(const array_link &al, double threshold = -1)
    return rank of an array based on Eigen::FullPivLU

int arrayrankSVD(const array_link &al, double threshold = -1)
    return rank of an array based on Eigen::JacobiSVD

int arrayrank(const array_link &array)
    calculate the rank of an array

int arrayrankInfo (const Eigen::MatrixXd&, int verbose = 1)
    Return rank of an array. Information about the different methods for rank calculation is printed to stdout.

int arrayrankInfo (const array_link &array, int verbose = 1)
    Return rank of an array. Information about the different methods for rank calculation is printed to stdout.

Eigen::MatrixXd arraylink2eigen (const array_link &array)
    convert array_link to Eigen matrix

double conditionNumber (const array_link &matrix)
    Return the condition number of a matrix.

void calculateParetoEvenOdd (const std::vector<std::string> infiles, const char *outfile, int verbose = 1, arrayfilemode_t afmode = ABINARY, int nrows = -1, int ncols = -1, paretomethod_t paretomethod = PARETOFUNCTION_DEFAULT)
    Calculate the Pareto optimal arrays from a list of array files

    Pareto optimality is calculated according to (rank; A3,A4; F4)

Pareto<mvalue_t<long>, long> parsePareto (const arraylist_t &arraylist, int verbose, paretomethod_t paretomethod = PARETOFUNCTION_DEFAULT)

mvalue_t<long> A3A4 (const array_link &al)
    calculate A3 and A4 value for array

    Return Object with A3 and A4

Parameters

• al: Array for which to calculate A3 and A4

mvalue_t<long> F4 (const array_link &al, int verbose = 1)
    calculate F4 value for 2-level array

template<class IndexType>
Pareto<mvalue_t<long>, IndexType>::pValue calculateArrayParetoRankFA (const array_link &array, int verbose)
    Calculate properties of an array and create a Pareto element

    The values calculated are:
    1) Rank (higher is better) 2) A3, A4 (lower is better) 3) F4 (lower is better, sum of elements is constant)

    Valid for 2-level arrays of strength at least 3

template<class IndexType>
**void addJmax (const array_link &al, typename Pareto<mvalue_t<long>, IndexType>::pValue &p, int verbose = 1)**

add Jmax criterium to Pareto set

**template<class IndexType>**

Pareto<mvalue_t<long>, IndexType>::pValue calculateArrayParetoJ5 (const array_link &al, int verbose)

Calculate Pareto element with J5 criterium.

**template<class IndexType>**

void parseArrayPareto (const array_link &array, IndexType i, Pareto<mvalue_t<long>, IndexType> &pset, int verbose)

Add array to list of Pareto optimal arrays

The values to be optimized are:
1) Rank (higher is better) 2) A3, A4 (lower is better) 3) F4 (lower is better, sum of elements is constant)

**double Cvalue2Dvalue (double Cvalue, int number_of_columns)**

convert C value to D-efficiency value

**double Dvalue2Cvalue (double Dvalue, int number_of_columns)**

convert D-efficiency value to C value

---

**class rankStructure**

#include <arrayproperties.h> Structure to efficiently calculate the rank of the second order interaction matrix of many arrays

The efficiency is obtained if the arrays share a common subarray. The theory is described in “Efficient rank calculation for matrices with a common submatrix”, Eendebak, 2016

---

**Public Types**

typedef Eigen::FullPivHouseholderQR<Eigen::MatrixXd> EigenDecomp

---

**Public Functions**

**rankStructure (const array_link &al, int nsup = 3, int verbose = 0)**

constructor

**rankStructure (int nsup = 3, int id = -1)**

constructor

void info () const

print information about the rank structure

void updateStructure (const array_link &al)

update the structure cache with a new array

int rankdirect (const Eigen::MatrixXd &array) const

calculate the rank of an array directly, uses special threshold

int rankxfdirect (const array_link &array) const

calculate the rank of the second order interaction matrix of an array directly

int rankxf (const array_link &array)

calculate the rank of the second order interaction matrix of an array using the cache system

---

7.2. Interface for array properties
Public Members

array_link alsub
int r
int verbose
   verbosity level
int ks
   number of columns of subarray in cache
int nsub
   number of columns to subtract from array when updating cache
int id
   used for debugging

Private Members

EigenDecomp decomp
   decomposition of subarray
Eigen::MatrixXd Qi
long ncalc
   internal structure
long nupdate

7.3 Interface for array tools

Contains the array_link class and related classes.
This file contains method and classes to work with (orthogonal) arrays.
Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

Defines

MPI_ARRAY_T

Typedefs

typedef Eigen::MatrixXd MatrixFloat
typedef Eigen::ArrayXd ArrayFloat
typedef Eigen::VectorXd VectorFloat
typedef double eigenFloat
typedef short int array_t
   data type for elements of orthogonal arrays
typedef const short int carray_t
   constant version of array_t
typedef short int rowindex_t
typedef int colindex_t
    type used for row indexing
typedef const int const_colindex_t
    type used for column indexing
typedef array_t *array_p
    pointer to array
    constant version of type used for column indexing
typedef carray_t *carray_p
    pointer to constant array
typedef rowindex_t *rowperm_t
typedef colindex_t *colperm_t
    type of row permutation
typedef array_t *levelperm_t
    type of column permutation
typedef int vindex_t
    type of level permutation
typedef signed char conf_t
    data type for elements of conference designs
typedef std::vector<conf_t> conference_column
    data type for column of a conference design
typedef std::vector<conference_column> conference_column_list
    list of columns of conference designs
typedef std::deque<array_link> arraylist_t
    container with arrays

Enums

denum ordering_t
    Values:
    
    ORDER_LEX
        lexicograph minimal by columns ordering
    ORDER_J5
        J5 based ordering.

Functions

void throw_runtime_exception (const std::string exception_message)
void eigenInfo (const MatrixFloat m, const char *str = "eigen", int verbose = 1)
    Print information about an Eigen matrix

Parameters

    • m: Matrix about which to print information
    • str: String to prepend in output

7.3. Interface for array tools
• verbose: Verbosity level

```cpp
void print_eigen_matrix(const MatrixFloat matrix)
    { Print Eigen matrix to stdout
```}

```cpp
void eigen2numpyHelper(double *pymat1, int n, const MatrixFloat &m)
```

```cpp
int sizeof_array_t()
    { return size in bytes of array_t type
```

```cpp
int sizeof_double()
    { return size in bytes of double type
```

```cpp
std::vector<int> possible_F_values(int N, int strength)
    { possible values for J-values of 2-level design
```

```cpp
bool file_exists(const std::string filename)
    { return true if the specified file exists
```

```cpp
bool file_exists(const char *filename)
    { return true if the specified file exists
```

```cpp
bool oa_file_exists(const char *filename)
    { return true if the specified oa file exists
```

```cpp
bool oa_file_exists(const std::string filename)
    { return true if the specified oa file exists
```

```cpp
arraydata_t* readConfigFile(const char *file)
    { Read array configuration from file.
```

```cpp
std::string printfstring(const char *message, ...)
    Function similar to printf returning C++ style string.
```

```cpp
 Return
 Parameters
 • message:
```

```cpp
void copy_array(const array_t* src, array_t* dst, const int nrows, const int ncols)
    Make a copy of an array.
```

```cpp
int destroy_array(array_t* array)
    Delete an array.
```

```cpp
 Return
 Parameters
 • array:
```

```cpp
static array_t* create_array(const int nrows, const int ncols)
    Create an array.
```

```cpp
 Return
 Parameters
 • nrows: Number of rows
 • ncols: Number of columns
```
array_t *create_array (const arraydata_t *ad)
Create an array from an arraydata_t structure.

array_t *clone_array (const array_t *const array, const rowindex_t nrows, const colindex_t ncols)
Clone an array.

int compareLMC (const array_link &lhs, const array_link &rhs)
Return -1 if the first array is smaller in LMC ordering than the second array, 0 if equal and 1 otherwise

array_link exampleArray (int idx = 0, int verbose = 0)
Return example array

Parameters
• idx: Index of example array to return
• verbose: If True, then print information about the array to stdout

std::vector<int> Jcharacteristics_conference (const array_link &array, int number_of_columns, int verbose = 0)
Calculate Jk-characteristics for a conference design

Return A vector of calculated inner products between all combinations of k columns.

Parameters
• array: Conference design
• number_of_columns: Specifies the number of columns to use
• verbose: Verbosity level

array_link hstack (const array_link &array1, const array_link &array2)
concatenate 2 arrays in vertical direction

concatenate 2 arrays in horizontal direction

array_link hstack (const array_link &array, const conference_column &column)
concatenate array and conference_column

array_link hstacklastcol (const array_link &A, const array_link &B)
concatenate the last column of array B to array A

conference_column vstack (const conference_column &column_top, const conference_column &column_bottom)
concatenate two columns

void perform_column_permutation (const array_link source, array_link &target, const std::vector<int> perm)
perform column permutation for an array

void perform_row_permutation (const array_link source, array_link &target, const std::vector<int> perm)
perform row permutation for an array

arraydata_t arraylink2arraydata (const array_link &array, int extracols = 0, int strength = 2)
create arraydata_t structure from array

Parameters
• array: Array to use as input specification for array class
• extracols: Number of extra columns to add to the number of columns of the array
• strength: Strength to set in the array class. If -1, then use the strength of the array

\texttt{arraylist\_t addConstant (const arraylist\_t &lst, int value)}

add a constant value to all arrays in a list

\texttt{std::vector<int> getJcounts (arraylist\_t *arraylist, int N, int k, int verbose = 1)}

Return number of arrays with j_{2n+1}=0 for number_of_arrays<m

void \texttt{create\_root (array\_t*array, const arraydata\_t*arrayclass)}

set first columns of an array to root form

void \texttt{create\_root (const arraydata\_t*arrayclass, arraylist\_t &solutions)}

Creates the root of an orthogonal array. The root is appended to the list of arrays.

int \texttt{array\_diff (carray\_p A, carray\_p B, const rowindex\_t r, const colindex\_t c, rowindex\_t &rpos, colindex\_t &cpos)}

Compare 2 arrays and return position of first difference.

void \texttt{fastJupdate (const array\_t*array, rowindex\_t N, const int J, const colindex\_t*column\_indices, array\_t*mnp)}

helper function to calculate J-values

int \texttt{jvalue (const array\_link &array, const int J, const int *column\_indices)}

Calculate J-value for a 2-level array

int \texttt{jvaluefast (const array\_t*array, rowindex\_t N, const int J, const colindex\_t *column\_indices)}

Calculate J-value for a column combination of a 2-level array

We assume the array has values 0 and 1. No boundary checks are performed.

\texttt{std::vector<jstruct\_t> analyseArrays (const arraylist\_t &arraylist, const int verbose, const int jj = 4)}

Analyze a list of arrays.

void \texttt{showArrayList (const arraylist\_t &lst)}

print a list of arrays to stdout

long \texttt{nArrays (const char *fname)}

return number of arrays in an array file

void \texttt{arrayfileinfo (const char *filename, int &number_of_arrays, int &number_of_rows, int &number_of_columns)}

return information about file with arrays

\textbf{Parameters}

• filename: Filename of array file

• number_of_arrays: Variable is set with number of arrays

• number_of_rows: Variable is set with number of rows

• number_of_columns: Variable is set with number of columns

\texttt{arraylist\_t readarrayfile (const char *fname, int verbose = 1, int *setcols = 0)}

Read all arrays in a file

\textbf{Return} List of arrays

\textbf{Parameters}

• fname: Filename to read from

• verbose: Verbosity level
• setcols: Pointer to return number of columns from array file

```c
int readarrayfile(const char *filename, arraylist_t *arraylist, int verbose = 1, int *setcols = 0, int *setrows = 0, int *setbits = 0)
```
Read all arrays in a file and append then to an array list

**Return**

**Parameters**

- **filename**: Filename to read from
- **arraylist**: Pointer to list of arrays
- **verbose**: Verbosity level
- **setcols**: Reference that is set with the number of columns from the file
- **setrows**: Reference that is set with the number of rows from the file
- **setbits**: Reference that is set with the number of bits from the file

```c
int writearrayfile(const char *filename, const arraylist_t &arraylist, arrayfile::arrayfilemode_t mode = arrayfile::ATEXT, int nrows = NRAUTO, int ncols = NRAUTO)
```
Write a list of arrays to file on disk

**Return** Value zero if succesfull

**Parameters**

- **filename**: Filename to use
- **arraylist**: List of arrays to write
- **mode**: Mode for the file with designs
- **nrows**: If the list of arrays is empty, use this number of rows for the design file
- **ncols**: If the list of arrays is empty, use this number of rows for the design file

```c
int writearrayfile(const char *filename, const array_link &array, arrayfile::arrayfilemode_t mode = arrayfile::ATEXT)
```
Write a single array to file.

```c
int append_arrayfile(const char *filename, const array_link array)
```
Append a single array to an array file. creates a new file if no file exists.

```c
void selectArrays(const std::string filename, std::vector<int> &idx, arraylist_t &fl, int verbose = 0)
```
Make a selection of arrays from binary array file, append to list.

```c
array_link selectArrays(const std::string filename, int index)
```
Select a single array from a file.

```c
arraylist_t selectArrays(const arraylist_t &input_list, std::vector<int> &idx)
```
Make a selection of arrays.

```c
arraylist_t selectArrays(const arraylist_t &input_list, std::vector<long> &idx)
```
Make a selection of arrays.

```c
void selectArrays(const arraylist_t &input_list, std::vector<int> &idx, arraylist_t &output_list)
```
Make a selection of arrays, append to list.

```c
void selectArrays(const arraylist_t &input_list, std::vector<long> &idx, arraylist_t &output_list)
```
Make a selection of arrays, append to list.

7.3. Interface for array tools
template<class Container, class IntType>
void \texttt{keepElements} (\texttt{Container} \&\texttt{al}, std::vector<\texttt{IntType}> \&\texttt{idx})
\hfill From a container keep all elements with specified indices.

template<class Container, class IntType>
void \texttt{removeElements} (\texttt{Container} \&\texttt{al}, std::vector<\texttt{IntType}> \&\texttt{idx})
\hfill From a container remove all elements with specified indices.

\begin{verbatim}
void \texttt{selectArraysMask} (const \texttt{arraylist} \&\texttt{al}, std::vector<\texttt{MType}> \&\texttt{mask}, \texttt{arraylist} \&\texttt{rl})
\end{verbatim}
\hfill Make a selection of arrays from a list, append to list.

\begin{verbatim}
void \texttt{appendArrays} (const \texttt{arraylist} \&\texttt{al}, \texttt{arraylist} \&\texttt{rl})
\end{verbatim}
\hfill Append selection of arrays to existing list.

\begin{verbatim}
void \texttt{write_array_format} (const \texttt{atype} \*\texttt{array}, \texttt{int} \texttt{nrows}, \texttt{int} \texttt{ncols}, \texttt{int} \texttt{width} = 3)
\end{verbatim}
\hfill Write a formatted array

\begin{verbatim}
void \texttt{write_array_format} (FILE \*\texttt{fid}, const \texttt{atype} \*\texttt{array}, \texttt{int} \texttt{nrows}, \texttt{int} \texttt{ncols})
\end{verbatim}
\hfill Write an array to a file pointer.

\begin{verbatim}
void \texttt{write_array_latex} (std::ostream \&\texttt{ss}, const \texttt{atype} \*\texttt{array}, \texttt{int} \texttt{nrows}, \texttt{int} \texttt{ncols})
\end{verbatim}
\hfill write an array in latex style

\begin{verbatim}
void \texttt{convert_array_file} (\texttt{std::string} \texttt{input_filename}, \texttt{std::string} \texttt{output_filename}, \texttt{arrayfile::arrayfilemode_t} \texttt{output_format}, \texttt{int} \texttt{verbose} = 0)
\end{verbatim}
\hfill Convert a file with arrays to a different format

\begin{verbatim}
bool \texttt{readbinheader} (FILE \*\texttt{fid}, \texttt{int} \&\texttt{nr}, \texttt{int} \&\texttt{nc})
\end{verbatim}
\hfill Read header for binary data file. Return true if valid header file

\hfill The header consists of 4 integers: 2 magic numbers, then the number of rows and columns

\begin{verbatim}
void \texttt{writebinheader} (FILE \*\texttt{fid}, \texttt{int} \texttt{number_rows}, \texttt{int} \texttt{number_columns})
\end{verbatim}
\hfill Write header for binary data file.

\begin{verbatim}
void \texttt{vector2doublebinfile} (const \texttt{std::string} \texttt{fname}, \texttt{std::vector<\texttt{Type}>} \texttt{vals}, \texttt{int} \texttt{writeheader} = 1)
\end{verbatim}
\hfill Write a vector of numeric elements to binary file as double values.

\begin{verbatim}
void \texttt{vectorvector2binfile} (const \texttt{std::string} \texttt{fname}, \texttt{const \texttt{std::vector<\texttt{vector<\texttt{double>}>}}} \texttt{vals}, \texttt{int} \texttt{writeheader}, \texttt{int} \texttt{na})
\end{verbatim}
\hfill Write a vector of vector elements to binary file.

\begin{verbatim}
\textbf{MatrixFloat} \texttt{array2eigenX1} (\texttt{const array_link} \&\texttt{array}, \texttt{int} \texttt{intercept} = 1)
\end{verbatim}
\hfill Convert 2-level array to main effects in Eigen format

\textbf{Return}  The main effects model

\textbf{Parameters}

- array: Array to convert
- intercept: If True, then include the intercept

68 Chapter 7. C++ library
MatrixFloat array2eigenX2 (const array_link &array)
Convert 2-level array to second order interaction matrix in Eigen format
The intercept and main effects are not included.

Return The second order interaction model
Parameters
  • array: Array to convert

MatrixFloat array2eigenModelMatrix (const array_link &array)
Convert 2-level array to second order interaction model matrix (intercept, main effects, interaction effects)

Return Eigen matrix with the model matrix
Parameters
  • array: Design of which to calculate the model matrix

std::pair<MatrixFloat, MatrixFloat> array2eigenModelMatrixMixed (const array_link &array, int verbose = 1)
Create first and second order model matrix for mixed-level orthogonal array
For 2-level arrays a direct calculation is used. For mixel-level arrays Helmert contrasts are used.

Return Pair with main effects and two-factor interaction model
Parameters
  • array: Input array
  • verbose: Verbosity level

std::vector<int> numberModelParams (const array_link &array, int order = -1)
Calculate number of parameters in the model matrix
A list of integers is returned, with the number of columns in:
  • The intercept (always 1)
  • The main effects
  • The interaction effects (second order interaction terms without quadratics)
  • The quadratic effects

Return List of sizes
Parameters
  • array: Orthogonal array or conference design
  • order: Not used any more

int arrayInFile (const array_link &array, const char *array_file, int verbose = 1)
return index of specified array in a file. returns -1 if array is not found

Return Position of array in list
Parameters
  • array: Array to find
  • array_file: Location if file with arrays

7.3. Interface for array tools
arrayInList(const array_link &array, const arraylist_t &arrays, int verbose = 1)

int arrayInList (const array_link &array, const arraylist_t &arrays, int verbose = 1)
return index of specified array in a list. returns -1 if array is not found

Return Position of array in list

Parameters

• array: Array to find
• arrays: List of arrays
• verbose: Verbosity level

Variables

const int NRAUTO = 0

struct arraydata_t
#include <arraytools.h> Specifies a class of arrays.
The specification includes the number of rows, number of columns, factor levels and strength.

Public Functions

arraydata_t ()
Specifies a class of orthogonal arrays
The specification includes the number of rows, number of columns, factor levels and strength.
An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array
with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs
equally often.

arraydata_t (array_t s, rowindex_t N, colindex_t strength, colindex_t ncols)
Specifies a class of orthogonal arrays
The specification includes the number of rows, number of columns, factor levels and strength.
An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array
with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs
equally often.

Parameters

• s: Factor levels
• N: Number of rows
• strength: Strength for class
• ncols: Number of columns for the class

arraydata_t (const std::vector<int> s, rowindex_t N, colindex_t strength, colindex_t ncols)
Specifies a class of orthogonal arrays
The specification includes the number of rows, number of columns, factor levels and strength.
An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

**Parameters**
- s: Factor levels
- N: Number of rows
- strength: Strength for class
- ncols: Number of columns for the class

```cpp
class arraydata_t(const array_t *s, rowindex_t N, colindex_t strength, colindex_t ncols)
```

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

```cpp
class arraydata_t(const arraydata_t &adp)
```

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

```cpp
class arraydata_t(const arraydata_t *adp, colindex_t newncols)
```

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, ..., s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

```cpp
arraydata_t()
```

```cpp
arraydata_t &operator= (const arraydata_t &ad2)
```

```cpp
int operator== (const arraydata_t &ad2)
```

```cpp
bool ismixed() const
```

return true if the class represents mixed-level arrays

```cpp
bool is2level() const
```

return true if the class represents a 2-level array

```cpp
array_link randomarray (int strength = 0, int ncols = -1) const
```

return random array from the class. this operation is only valid for strength 0 or 1

```cpp
void writeConfigFile (const char *filename) const
```

Write file with specification of orthogonal array class.

**Parameters**
- filename: Filename to write to
std::string idstr () const
std::string idstrseriesfull () const
std::string fullidstr (int series = 0) const
std::string latexstr (int cmd = 0, int series = 0) const
    return latex string describing the class
arraydata_t reduceColumns (int k)
std::string showstr () const
void show (int verbose = 1) const
void complete_arraydata ()
    Calculate derived data such as the index and column groups from a design.
void lmc_overflow_check () const
    check whether the LMC calculation will overflow
void complete_arraydata_fixlast ()
void complete_arraydata_splitn (int ns)
void set_colgroups (const std::vector<int> splits)
void set_colgroups (const symmetry_group &sg)
    set column group equal to that of a symmetry group
void show_colgroups () const
    show column groups in the array class
void calculate_oa_index (colindex_t strength)
    calculate the index of the orthogonal arrays in this class
array_link create_root (int n_columns = -1, int fill_value = 0) const
    return the root array for the class
int getfactorlevel (int idx) const
    return the factor level for the specified column return -1 if the column index is invalid
std::vector<int> getS () const
    return factor levels
std::vector<int> factor_levels () const
    return factor levels
void reset_strength (colindex_t strength)
    Reset strength of arraydata.

Parameters

    • strength: The strength to reset the structure to

colindex_t get_col_group (const colindex_t col) const
    Return index of the column group for a column.
bool is_factor_levels_sorted () const
    Return True if the factor levels are sorted from large to small.
**Public Members**

- `rowindex_t N`  
  number of runs

- `colindex_t ncols`  
  total number of columns (factors) in the design

- `colindex_t strength`  
  strength of the design

- `array_t *s`  
  pointer to factor levels of the array

- `ordering_t order`  
  Ordering used for arrays.

- `colindex_t ncolgroups`  
  number of groups of columns with the same number of levels

- `colindex_t *colgroupindex`  
  specifies for each column the index of the column group

- `colindex_t *colgroupsize`  
  specifies for each column the size of the column group

- `int oaindex`  
  index of the array

```c
struct array_link
#include <arraytools.h>
```

**Public Functions**

- `array_link ()`  
  A class representing an integer valued array

- `array_link (rowindex_t nrows, colindex_t ncols, int index)`  
  A class representing an integer valued array
  The array is initialized with zeros.

  **Parameters**

  - `nrows`: Number of rows
  - `ncols`: Number of columns
  - `index`: Number to keep track of lists of designs

- `array_link (rowindex_t nrows, colindex_t ncols, int index, carray_t *data)`  
  A class representing an integer valued array
  Initialize with data from a pointer.

- `array_link (const array_link&)`  
  A class representing an integer valued array
  Initialize with data from another `array_link` object.
array_link (Eigen::MatrixXd &eigen_matrix)
A class representing an integer valued array
Initialize with data from an Eigen matrix.

array_link (const array_link &array, const std::vector<int> &column_permutation)
A class representing an integer valued array
The array is initialized by permuting the columns of another array

Parameters
• array: Source to copy from
• column_permutation: The permutation to apply

array_link (const array_t *array, rowindex_t nrows, colindex_t ncols, int index = 0)
A class representing an integer valued array

array_link (const array_t *array, rowindex_t nrows, colindex_t ncolsorig, colindex_t ncols, int index)
A class representing an integer valued array
The array is initialized by copying the values from a vector.

~array_link ()

array_link clone () const

void showarray () const
print array to stdout

std::string showarrayString () const
print array to string

void showarraycompact () const
print array to stdout in compact format (no whitespace between elements)

void showproperties () const
print array properties to stdout

bool is2level () const
return true if the array is a 2-level array (e.g. only contains values 0 and 1)

bool is_mixed_level () const
return true is the array is a mixel-level array

bool is_orthogonal_array () const
return true is the array is array with values in 0, 1, ..., for each column

bool is_conference () const
return true if the array is a +1, 0, -1 valued array

bool is_conference (int number_of_zeros) const
return true if the array is a +1, 0, -1 valued array, with specified number of zeros in each column

bool isSymmetric () const
return true if the array is symmetric
void makeSymmetric()
make the array symmetric by copying the upper-right to the lower-left

array_link deleteColumn (int index) const
return array with selected column removed

array_link selectFirstRows (int nrows) const
return array with first number_of_arrays rows

array_link selectFirstColumns (int ncolumns) const
return array with first number_of_arrays columns selected

array_link selectLastColumns (int ncolumns) const
return array with last number_of_arrays columns selected

array_link selectColumns (const std::vector<int> c) const
select columns from an array

array_link selectColumns (int c) const
select single column from an array

void setColumn (int c, const std::vector<int> v)
set a column of the array to the given vector

void setColumn (int c, const std::vector<signed char> v)
set a column of the array to the given vector

array_link transposed () const
return transposed array

double Defficiency () const
calculate D-efficiency

double DsEfficiency (int verbose = 0) const
calculate main effect robustness (or Ds-optimality)

std::vector<double> Defficiencies (int verbose = 0, int addDs0 = 0) const
calculate D-efficiency, calculate main effect robustness (or Ds-optimality) and D1-efficiency for an orthogonal array

double VIFefficiency () const

double Aefficiency () const
calculate A-efficiency

double Eefficiency () const
calculate E-efficiency

std::vector<int> Fvalues (int number_of_columns) const
Calculate F-values of a 2-level matrix.
This assumes the strength is at least 3. Otherwise use the jstruct_t object

std::vector<int> FvaluesConference (int number_of_columns) const
Calculate F-values of a conference design

Return The Fk vector with k the number of columns specified

Parameters
- `number_of_columns`: Number of columns to use

```cpp
std::vector<int> Jcharacteristics (int jj = 4) const
    Calculate the Jk-characteristics of the matrix (the values are signed)
```

Return: Vector with calculated Jk values

Parameters

- `jj`: Number of columns to use

```cpp
std::vector<double> PECsequence (int verbose = 0) const
    Calculate the projective estimation capacity sequence.
```

```cpp
std::vector<double> PICsequence (int verbose = 0) const
    Calculate the projective information capacity sequence.
```

```cpp
int rank () const
    calculate rank of array
```

```cpp
std::vector<double> GWLP (int truncate = 1, int verbose = 0) const
    Calculate generalized wordlength pattern
```

See: `GWLP`

```cpp
int strength () const
    calculate strength of an array
```

```cpp
bool foldover () const
    return true if the array is a foldover array
```

```cpp
array_t min () const
array_t max () const
```

```cpp
double CL2discrepancy () const
    Calculate centered L2 discrepancy
    The method is from “A connection between uniformity and aberration in regular fractions of two-level factorials”, Fang and Mukerjee, 2000
```

```cpp
array_link randomperm () const
    apply a random permutation of rows, columns and levels of an orthogonal array
```

```cpp
array_link randomcolperm () const
    apply a random permutation of columns of an orthogonal array
```

```cpp
array_link randomrowperm () const
    apply a random permutation of rows of an orthogonal array
```

```cpp
MatrixFloat getModelMatrix (int order, int intercept = 1, int verbose = 0) const
    Calculate model matrix of an orthogonal array
    This function uses `array2eigenModelMatrixMixed` for the calculation.
```

Return: Calculated model matrix

Parameters
• order: For 0 return only the intercept; for 1 return intercept and main effects; for 2 return intercept, main effects and interaction effects.

• intercept: If 1, then include the intercept in the output.

• verbose: Verbosity level

array_link &operator=(const array_link &rhs)
array_link &deepcopy (const array_link &rhs)
array_link &shallowcopy (const array_link &rhs)

int operator==(const array_link &rhs) const
Return True if both arrays are equal.

Parameters
• rhs: Array to compare to

int operator!=(const array_link &rhs) const
int operator<(const array_link &rhs) const
int operator>(const array_link &rhs) const
int equalsize (const array_link &rhs) const
return true of two array have the same dimensions

array_link operator+(const array_link&) const
elementwise addition

array_link operator+(array_t value) const
elementwise addition

array_link operator-(const array_link&) const
array_link operator-(array_t value) const
array_link operator*(const array_link &rhs) const
elementwise multiplication

array_link operator*(array_t value) const
array_link operator*=(array_t value)
array_link operator+= (array_t value)
array_link operator-=(array_t value)

const array_t &atfast (const rowindex_t r, const colindex_t c) const
get element from array, no error checking, inline version

array_t &atfast (const rowindex_t r, const colindex_t c)
get element from array, no error checking, inline version

array_t _at (const rowindex_t, const colindex_t) const
get element at specified position, no bounds checking
array_t _at (const int index) const
get element at specified position, no bounds checking

array_t _at (const rowindex_t, const colindex_t) const
get element at specified position

array_t _at (const int index) const
get element at specified position

array_t &_at (const rowindex_t, const colindex_t)
get element at specified position

void setconstant (array_t value)
set all elements in the array to a value

void _setvalue (int row, int col, int value)
set value of an array

void _setvalue (int row, int col, double value)
set value of an array

void _setvalue (int row, int col, int value)
set value of an array, no bounds checking!

void negateRow (rowindex_t row)
multiply a row by -1

void show () const
print information about array

std::string showstr () const
return string describing the array

std::string md5 () const
return md5 sum of array representation (as represented with 32bit int datatype in memory)

bool columnEqual (int column_index, const array_link &rhs, int column_index_rhs) const
return true if two columns are equal

int firstColumnDifference (const array_link &A) const
return index of first different column

bool firstDiff (const array_link &A, int &r, int &c, int verbose = 1) const
Calculate row and column index of first difference between two arrays

The difference is according to the column-major ordering.

void create_root (const arraydata_t &arrayclass, int fill_value = 0)
create root in arraylink

double nonzero_fraction () const
return fraction of nonzero elements in array

void clear ()
fill array with zeros

void getarraydata (int *pymat1, int n)
template<class numtype>
void `setarraydata` (const `numtype` *tmp, int n)
  internal function

void `setarraydata` (std::vector<int> tmp, int n)
  special method for SWIG interface

template<class `numtype`>
void `setarraydata` (std::vector<`numtype`> tmp, int n)
  internal function

void `setcolumn` (int `target_column`, const `array_link` &`source_array`, int `source_column` = 0) const
  set column to values

void `init` (rowindex_t `r`, colindex_t `c`)

symmetry_group `row_symmetry_group` () const
  return the row_symmetry group of an array

`array_link reduceLMC` () const
  return the LMC form of the array

`array_link reduceDOP` () const
  return the delete-one-factor-projection form of the array

`MatrixFloat getEigenMatrix` () const
  return the array as an Eigen matrix

int `columnGreater` (int `c1`, const `array_link` &rhs, int `rhs_column`) const
  return true of specified column is smaller than column in another array

void `debug` () const

**Public Members**

`rowindex_t n_rows`
  Number of rows in array.

`colindex_t n_columns`
  Number of columns in array.

int `index`
  Index number.

`array_t *array`
  Pointer to array data.

**Public Static Attributes**

const int `INDEX_NONE` = 0
const int `INDEX_ERROR` = -1
const int `INDEX_DEFAULT` = 0
Private Functions

bool equal_size (const array_link &array) const
    return true if both arrays have the same size

bool _valid_index (const rowindex_t r, const colindex_t c) const

bool _valid_index (int index) const

Friends

std::ostream &operator<< (std::ostream&, const array_link &A)
    print an array to output stream

class jstructbase_t
    #include <arraytools.h> struct to hold data of an array, e.g. J-characteristic. Abstract base class
    Subclassed by jstructconference_t

Public Functions

int maxJ () const
    calculate maximum J value

std::vector<int> Jvalues () const
    calculate possible values in F vector

std::vector<int> calculateF () const
    Calculate histogram of J values

    \return Histogram of J values
    \hrule
    The histogram bins are given by the values of @ref Jvalues.

virtual void calc (const array_link &array) = 0
    Calculate the J-values for a given array.

void show ()
    Show contents of structure.

void showdata (int verbose = 1)

std::string showstr ()

int allzero ()
    return 1 if all vals are zero
**Public Members**

std::vector<int> **values**
- calculated J-characteristics

std::vector<int> **jvalues**

std::map<int, int> **jvalue2index**
- map from Jk-value to index in the jvalues variable

int **jj**
- number of columns

**struct symmdata**
- #include <arraytools.h> structure containing data related to symmetries of arrays

**Public Functions**

**symmdata** (const **array_link** & **al**, int **minlen** = 1)

void **show** (int **verbose** = 1) **const**

std::vector<int> **checkIdx** (int **col** = -1) **const**
- list with indices set to check for symmetry reductions

**Public Members**

**array_link** **rowvalue**

**array_link** **orig**

**array_link** **ft**

**class jstruct_t**
- #include <arraytools.h> struct to hold data of an array, e.g. J-characteristic, rank


**Public Functions**

**jstruct_t** ()
- Create an object to calculate J-characteristics.

**jstruct_t** (const **array_link** & **al**, int **jj** = 4)
- Create an object to calculate J-characteristics.

**jstruct_t** (const int **N**, const int **K**, const int **jj** = 4)
- Create an object to calculate J-characteristics.

**jstruct_t** (const **jstruct_t** & **js**)
- Create an object to calculate J-characteristics.

**~jstruct_t** ()

**jstruct_t &operator=(**const **jstruct_t** & **rhs**)
```cpp
int maxJ() const
    calculate maximum J value

int number_J_values(int strength) const
    Calculate the number of possible J values that can occur for the given strength.

std::vector<int> Fval(int strength = 3) const
    Calculate possible values in F vector

        Return  Vector with possible Jk values (ordered from high to low)

        Parameters
            • strength: Strength to use

std::vector<int> calculateF(int strength = 3) const
    calculate histogram of J values for a 2-level array

void calculateAberration()
    Calculate aberration value
    This is equal to the sum of the squares of all Jk values, divided by the number of rows squared.
    The calculated aberration is stored in the variable aberration.

void show() const
    Show contents of structure.

void showdata()

std::string showstr()

int allzero() const
    return 1 if all J values are zero, otherwise return 0

Public Members

int N
    number of rows in array

int k
    number of columns in array

int jj
    J-characteristic that is calculated.

int nc
    number of column combinations possible

std::vector<int> values
    contains calculated J-values

double aberration
    calculated aberration
```
Private Functions

void init (int N, int k, int jj)
init data structures

void calc (const array_link &al)
calculate J-characteristics of a 2-level array

void calcj4 (const array_link &al)
calculate J-characteristics of a 2-level array, special function for jj=4

void calcj5 (const array_link &al)
calculate J-characteristics of a 2-level array, special function for jj=5

class jstructconference_t: public jstructbase_t
#include <arraytools.h> Calculate J-characteristics of conference designs

Public Functions

jstructconference_t (int N, int jj = 4)
Create structure to calculate J-characteristics of conference designs

Parameters
• N: Number of rows
• jj: Number of columns to use for the Jk-characteristics

jstructconference_t (const array_link &array, int jj = 4)
Calculate J-characteristics of a conference design

Parameters
• array: Array to calculate the J-characteristics for
• jj: Number of columns to use for the Jk-characteristics

Private Functions

void calcJvalues (int N, int jj)

void calc (const array_link &array)
Calculate the J-values for a given array.

class array_transformation_t
#include <arraytools.h> Contains a transformation of an array.

Contains an array transformation. The transformation consists of column, row and level permutations. The level and column permutations are not commutative (since the level permutations are tied to a particular column). We apply the column permutations first.
Public Functions

array_transformation_t (const arraydata_t *arrayclass)
array_transformation_t (const arraydata_t &arrayclass)
array_transformation_t ()
array_transformation_t (const array_transformation_t &transformation)

// copy constructor

array_transformation_t &operator=(const array_transformation_t &at)

// assignment operator

~array_transformation_t ()

void show() const
show the array transformation

bool isIdentity() const
return true if the transformation is equal to the identity

array_transformation_t inverse() const
return the inverse transformation

void reset()
return the transformation to the identity transformation

void randomize()
initialize to a random transformation

void randomizecolperm()
initialize with a random column permutation

void randomizerowperm()
initialize with a random row permutation

array_link apply (const array_link &array) const
apply transformation to an array_link object

int operator== (const array_transformation_t &t2) const
Comparison operator.

array_transformation_t operator* (const array_transformation_t b) const
composition operator. the transformations are applied from the left

void apply (array_t *sourcetarget) const
apply transformation to an array (inplace)

void apply (const array_t *source, array_t *target) const
apply transformation to an array

void print_transformed (carray_t *source) const
apply transformation and show resulting array

void show (std::ostream &out) const

std::vector<int> rowperm() const
return the row permutation of the transformation
std::vector<int> \textbf{colperm}() \texttt{const}
return the column permutation of the transformation

std::vector<int> \textbf{lvlperm}(\texttt{int} \texttt{c}) \texttt{const}
return the level permutations of the transformation

\texttt{void \textbf{setrowperm}}(std::vector<int> \texttt{row\_permutation})
set the row permutation of the transformation

\texttt{void \textbf{setcolperm}}(std::vector<int> \texttt{column\_permutation})
set the column permutation of the transformation

\texttt{void \textbf{setlevelperm}}(\texttt{int} \texttt{column\_index}, std::vector<int> \texttt{lvl\_permutation})
set the level permutation of the transformation

\textbf{Public Members}

\texttt{rowperm\_t rperm}
row permutation

\texttt{colperm\_t cperm}
column permutation

\texttt{levelperm\_t *lperms}
level permutations

\texttt{const arraydata\_t *ad}
type of array

\textbf{Private Functions}

\texttt{void \textbf{allocate\_data\_structures}()}
initialize permutation structures

\texttt{void \textbf{free\_data\_structures}()}
free permutation structures and \texttt{arraydata\_t} structure

\texttt{class \textbf{conference\_transformation\_t}}
\#include <arraytools.h> Contains a transformation of a conference matrix.

Contains an array transformation. The transformation consists of column permutations, row permutations and
sign switches for both the rows and columns.

The sign switches and the permutations are not commutative. We apply the permutations first and then the sign
flips.

\textbf{Public Functions}

\texttt{conference\_transformation\_t()} 

\texttt{conference\_transformation\_t(\texttt{int} nrows, \texttt{int} ncols)}
default constructor

\texttt{conference\_transformation\_t(const array\_link &al)}

\texttt{conference\_transformation\_t(const conference\_transformation\_t &T)}
void show (int verbose = 1) const
  show the array transformation

bool isIdentity () const
  return true if the transformation is equal to the identity

conference_transformation_t inverse () const
  return the inverse transformation

void reset ()
  return the transformation to the identity transformation

void randomize ()
  initialize to a random transformation

void randomizecolperm ()
  initialize with a random column permutation

void randomizerowperm ()
  initialize with a random row permutation

void randomizecolflips ()
  initialize with random col switches

void randomizerowflips ()
  initialize with random row switches

array_link apply (const array_link &al) const
  apply transformation to an array_link object

int operator== (const conference_transformation_t &rhs) const

conference_transformation_t operator* (const conference_transformation_t &rhs) const
  composition operator. the transformations are applied from the left
  E.g. (T1*T2)(x) = T1(T2(x))

void setrowperm (std::vector<int> rp)

void setcolperm (std::vector<int> cp)

Public Members

std::vector<int> rperm
  row permutation of the transformation

std::vector<int> cperm
  column permutation of the transformation

std::vector<int> cswitch
  sign flips for the columns

std::vector<int> rswitch
  sign flips for the rows

int nrows
  number of rows

int ncols
  number of columns
Private Functions

void init (int nr, int nc)

struct arraywriter_t
#include <arraytools.h> structure to write arrays to disk, thread safe

Public Functions

arraywriter_t ()
~arraywriter_t ()
void flush ()
flush all output files
void writeArray (const array_link &A)
write a single array to disk
void writeArray (const arraylist_t &lst)
write a list of arrays to disk
void initArrayFiles (const arraydata_t &ad, int kstart, const std::string prefix, arrayfilemode_t mode = ABINARY_DIFF)
initialize the result files
int nArraysWritten () const
return the total number arrays written to disk
void closeafiles ()

Public Members

std::vector<arrayfile_t *> afiles
Pointers to different data files.

bool writearrays
only write arrays if this variable is true

int nwritten
number of arrays written to disk

int verbose
verbosity level

namespace arrayfile
 Enums

`enum arrayfilemode_t`
file format mode

*Values:*

- **ATEXT**: text based format
- **ALATEX**: write arrays to a text file in a format that can be parsed by LaTeX
- **ABINARY**: binary format
- **ABINARY_DIFF**: binary format storing differences of arrays
- **ABINARY_DIFFZERO**: binary format storing differences of arrays and zero offsets
- **AERROR**
- **A_AUTOMATIC**: automatically determine the format
- **A_AUTOMATIC_BINARY**: automatically determine the format (but binary)

`enum afilerw_t`
file mode for array file

*Values:*

- **READ**
- **WRITE**
- **READWRITE**

`struct arrayfile_t`
#include <arraytools.h> Structure for reading or writing a file with arrays.

The format of the file is determined by the `arrayfilemode_t`. The format described in detail in the documentation of the OApakage https://oapackage.readthedocs.io/en/latest/.

 Public Functions

`arrayfile_t()`
Structure for reading or writing a file with arrays

`arrayfile_t(const std::string filename, int verbose = 1)`
Structure for reading or writing a file with arrays

*Parameters*

- **filename**: File to open for reading
- **verbose**: Verbosity level
```cpp
arrayfile_t (const std::string filename, int nrows, int ncols, int narrays = -1, arrayfilemode_t mode = ATEXT, int number_of_bits = 8)
Structure for reading or writing a file with arrays

Parameters
• filename: File to open
• nrows: Number of rows
• ncols: Number of columns
• narrays: Specify a number of arrays, or -1 to add dynamically
• mode: File mode
• number_of_bits: Number of bits to use for storage. For 2-level arrays only 1 bit is needed

~arrayfile_t()
    destructor function, closes all filehandles

void createfile (const std::string filename, int nrows, int ncols, int narrays = -1, arrayfilemode_t mode = ATEXT, int number_of_bits = 8)
    Open a new file for writing and (if opened) close the current file.

void closefile()
    close the array file

int isopen () const
    return true if file is open

int seek (int pos)
    seek to specified array position

int read_array (array_link &a)
    read array and return index

array_link readnext ()
    read next array from the file

arraylist_t readarrays (int nmax = NARRAYS_MAX, int verbose = 1)
    read set of array from the file

void flush()
    flush any open file pointer

bool isbinary () const
    return true if the file has binary format

int append_arrays (const arraylist_t &arrays, int startidx = -1)
    append list of arrays to the file

void append_array (const array_link &a, int specialindex = -1)
    append a single array to the file

int swigcheck () const
    return True if code is wrapper by SWIG

std::string showstr () const
    return string describing the object

size_t pos () const
    return current position in file
```

7.3. Interface for array tools 89
bool hasrandomaccess () const
    return true of the file format has random access mode

void updatenumbers ()

int read_array (array_t *array, const int nrows, const int ncols)
    read array and return index

void finisharrayfile ()

void setVerbose (int v)
    set verbosity level

int getnbits ()

**Public Members**

std::string filename
    location of file on disk

int iscompressed
    True of the file is compressed with gzip.

int nrows
    number of rows of the arrays

int ncols
    number of columns of the arrays

int nbits
    number of bits used when storing an array

arrayfilemode_t mode
    file mode, can be ATEXT or ABINARY, ABINARY_DIFF, ABINARY_DIFFZERO

afilerw_t rwmode
    file opened for reading or writing

int narrays
    number of arrays in the file

int narraycounter

FILE *nfid

int gzfid
    pointer to compressed file

int verbose
    verbosity level
Public Static Functions

```c
static arrayfile::arrayfilemode_t parseModeString(const std::string format)
    parse string to determine the file mode
```

```c
static int arrayNbits(const arraydata_t &ad)
    return number of bits necessary to store an array
```

```c
static int arrayNbits(const array_link &A)
    return number of bits necessary to store an array
```

Public Static Attributes

```c
const int NARRAYS_MAX = 2 * 1000 * 1000 * 1000
    maximum number of arrays in structure
```

Protected Functions

```c
void writeheader()
```

```c
void read_array_binary(array_t *array, const int nrows, const int ncols)
    Read a binary array from a file.
```

Private Functions

```c
int headersize() const
    return header size for binary format array
```

```c
int barraysize() const
    return size of bit array
```

```c
size_t afwrite(void *ptr, size_t t, size_t n)
    wrapper function for fwrite or gzwrite
```

```c
size_t afread(void *ptr, size_t sz, size_t cnt)
    wrapper function for fread or gzread
```

```c
int read_array_binary_zero(array_link &a)
```

```c
void write_array_binary(carray_t *array, const int nrows, const int ncols)
```

```c
void write_array_binary(const array_link &A)
```

```c
void write_array_binary_diff(const array_link &A)
    Write an array in binary diff mode to a file
        We only write the section of columns of the array that differs from the previous array.
```

```c
void write_array_binary_diffzero(const array_link &A)
    Write an array in binary diffzero mode
```
Private Members

array_link diffarray

7.4 Interface for conference designs

Contains functionality to generate and analyse conference designs. For more information see:

- "A Classification Criterion for Definitive Screening Designs", Schoen et al., The Annals of Statistics, 2019

Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

Typedefs

typedef CandidateGeneratorConference CandidateGenerator

Functions

void print_column (const conference_column &column, const char *msg = 0)
print a candidate extension

void showCandidates (const std::vector<conference_column> &column_candidates)
Show a list of candidate extensions

Parameters

- column_candidates: List of candidates to show

array_link conference2DSD (const array_link &conference_design, bool add_zeros = 1)
Convert conference design to definitive screening design

The DSD is created by appending the negated design to the conference design and then appending a row of zeros.

Return The DSD generated from the conference design

Parameters

- conference_design: Array with the conference design
- add_zeros: If True, then append a row of zeros

array_link reduceConference (const array_link &, int verbose = 0)
Reduce conference matrix to normal form using Nauty

See reduceConferenceTransformation

candidate_transformation_f reduceConferenceTransformation (const array_link &conference_design, int verbose)
Reduce conference matrix to normal form using Nauty

The design is converted to a graph representation. The graph is then reduced using Nauty to normal form and the resulting graph translated back to a conference design.
Return  A transformation that converts the input design to normal form

Parameters

- conference_design: Design to be reduced to normal form
- verbose: Verbosity level

`arraylist_t extend_conference (const arraylist_t &lst, const conference_t conference_type, int verbose, int select_isomorphism_classes = 0)`

Extend a list of conference designs with a single column.

The list of conference designs is extended by adding to each design the candidate extensions generated by CandidateGenerator.

The extension algorithm tried to generate designs in LMC0 normal form and prune any designs that are not in LMC0 form.

Return  List of generated conference designs

Parameters

- lst: List of conference designs
- conference_type: Type specification for the conference designs
- verbose: Verbosity level
- select_isomorphism_classes: If True then select only a single design for each isomorphism class specified by the conference type.

`arraylist_t extend_conference_plain (const arraylist_t &lst, const conference_t conference_type, int verbose, int select_isomorphism_classes = 0)`

Extend a list of conference designs with a single column, plain version without caching.

Research function.

`arraylist_t extend_conference_restricted (const arraylist_t &lst, const conference_t conference_type, int verbose)`

Extend a list of conference designs with a single column

Research function.

`arraylist_t extend_double_conference (const arraylist_t &lst, const conference_t conference_type, int verbose)`

Extend a list of double conference matrices with an additional column

The list of designs is extended by adding each design with the candidate extensions generated by CandidateGeneratorDouble.

Return  List of generated double conference designs

Parameters

- lst: List of double conference designs
- conference_type: Type specification for the double conference designs
- verbose: Verbosity level

`arraylist_t selectConferenceIsomorpismClasses (const arraylist_t &list, int verbose, matrix_isomorphism_t itype = CONFERENCE_ISOMORPHISM)`

select representatives for the isomorphism classes of a list of conference arrays
std::vector<int> selectConferenceIsomorphismIndices(const arraylist_t &lst, int verbose, matrix_isomorphism_t itype = CONFERENCE_ISOMORPHISM)

select representatives for the isomorphism classes of a list of conference arrays, return indices of classes

arraylist_t selectLMC0doubleconference(const arraylist_t &list, int verbose, const conference_t &ctype)

Select double conference designs in LMC0 form

**Return** List with only the designs in the input list that are in LMC0 normal form.

**Parameters**
- **list**: List of double conference designs
- **verbose**: Verbosity level
- **ctype**: Specification of the class of designs

arraylist_t selectLMC0(const arraylist_t &list, int verbose, const conference_t &ctype)

Select conference designs in LMC0 form

**Return** List with only the designs in the input list that are in LMC0 normal form.

**Parameters**
- **list**: List of conference designs
- **verbose**: Verbosity level
- **ctype**: Specification of the class of designs

std::vector<conference_column> generateConferenceExtensions(const array_link &array, const conference_t &conference_type, int zero_index, int verbose = 1, int filter_symmetry = 1, int filterj2 = 1)

Generate candidate extensions for a conference design

**Return** List of generated extensions

**Parameters**
- **array**: Design to be extended
- **conference_type**: Class of conference designs
- **zero_index**: Index of zero in candidate column
- **verbose**: Verbosity level
- **filter_symmetry**: If True, filter based on symmetry
- **filterj2**: If True, filter based on J2 values

std::vector<conference_column> generateConferenceRestrictedExtensions(const array_link &array, const conference_t &conference_type, int zero_index, int verbose = 1, int filter_symmetry = 1, int filterip = 1)
Generate candidate extensions for restricted isomorphism classes

```cpp
std::vector<conference_column> generateDoubleConferenceExtensions(const array_link &array, const conference_t &conference_type, int verbose = 1, int filter_symmetry = 1, int filterip = 1, int filterJ3 = 0, int filter_symmetry_inline = 1)
```

generated extensions for double conference matrices in LMC0 form

```cpp
std::vector<conference_column> generateSingleConferenceExtensions(const array_link &array, const conference_t &conference_type, int zero_index, int verbose, int filter_symmetry, int filterj2, int filterj3, int filter_symmetry_inline = 0)
```

generated extensions for conference matrices in LMC0 form

```cpp
int maxz (const array_link &al, int column_index = -1)
   return max position of zero in array, returns -1 if no zero is found
```

The parameter k specifies the column to search in. For k=-1 all columns are searched.

```cpp
bool compareLMC0 (const array_link &array_first, const array_link &array_second)
   Return true if the first array is smaller in LMC-0 ordering than the second array
```

```cpp
arraylist_t sortLMC0 (const arraylist_t &arrays)
   sort list of conference designs according to LMC0 ordering
```

```cpp
lmc_t LMC0checkDC (const array_link &al, int verbose = 0)
```

```cpp
lmc_t LMC0check (const array_link &array, int verbose = 0)
```

```cpp
bool isConferenceFoldover (const array_link &array, int verbose = 0)
   return true if the design is a foldover array
```

```cpp
std::vector<int> double_conference_foldover_permutation (const array_link &double_conference)
   For a double conference design return a row permutation to a single conference design

   If the design is not a foldover design then the first element of the returned permutation is -1.

   **Return** Permutation such that the top block of the resulting design forms a single conference design

   **Parameters**

   • double_conference: A double conference design
```

```cpp
int minz (const array_link &al, int column_index)
   return minimal position of zero in specified column of a design
```

```cpp
class conference_t
   #include <conference.h> Structure representing the type of conference designs.
```
Public Types

enum conference_type
    Type of conference design.
    Values:
    
    CONFERENCE_NORMAL
        normal conference design
         
    CONFERENCE_DIAGONAL
        conference design with zeros only on diagonal
         
    DCONFERENCE
        double conference design

Public Functions

conference_t ()
    Structure representing the type of conference designs

conference_t (int N, int k, int j1zero)
    Structure representing the type of conference designs

Parameters

• N: Number of rows
• k: Number of columns
• j1zero: If True then require the J1-characteristics to be zero

conference_t (const conference_t &rhs)
    Structure representing the type of conference designs

std::string idstr () const

array_link create_root () const
    create the unique representative of the 2 column conference design in LMC0 form

array_link create_root_three_columns () const
    create the unique representative of the 3 column conference design in LMC0 form

arraylist_t createDoubleConferenceRootArrays () const
    create the root arrays with 1 column for the double conference matrices

arraylist_t createRootArrays () const
    return the list of root arrays for the class of conference designs

std::string __repr__ () const
    return string representation of the object
**Public Members**

```cpp
rowindex_t N
   number of runs
```

```cpp
colindex_t ncols
   total number of columns (factors) in the design
```

```cpp
conference_type ctype
   defines the type of designs
```

```cpp
matrix_isomorphism_t itype
   defines the isomorphism type
```

```cpp
bool j1zero
   if true then J1 values should be zero
```

```cpp
bool j3zero
   if true then J3 values should be zero
```

```cpp
class CandidateGeneratorBase
   #include <conference.h>
   Class to generate candidate extensions with caching
```

We assume that the designs to be extended are run ordered, so that the caching has maximal effect.

The key idea used is that any valid extension of a design A with k columns is a permutation of a valid extension of the design B obtained by taking the first l < k columns of A. The permutations that are allowed are called the symmetry inflations. All the J2 checks performed for the extension of B do not have to be repeated for the permutations of this extension.

Subclassed by `CandidateGeneratorConference, CandidateGeneratorDouble`

**Public Functions**

```cpp
CandidateGeneratorBase (const array_link &al, const conference_t &ct)
```

```cpp
void showCandidates (int verbose = 1) const
   Show the candidate extensions for each column
```

```cpp
conference_column_list candidates (int k)
   return all candidates for the kth column
```

**Public Members**

```cpp
conference_t ct
   type of designs to generate
```

```cpp
int verbose
   verbosity level
```

```cpp
array_link al
   last array analyzed
```

```cpp
int last_valid
   index of last valid column
```
Protected Functions

int startColumn (const array_link &alx, int verbose = 0) const
    Find the starting column for the extension of a design
    The static variable START_COL is the number of columns for which is the starting point if the cache
    is empty. Therefore for a design with initial columns the same, START_COL+1 is the first number of
    columns with valid entries.
    For startcol k the elements in candidate_list[k] are valid, e.g. we can start with extensions valid for index
    k-1

Protected Attributes

std::vector<conference_column_list> candidate_list
    list of candidate extensions. the elements of candidate_list[k] correspond to columns with index k-1

Protected Static Attributes

const int START_COL = 2

class CandidateGeneratorConference : public CandidateGeneratorBase
    #include <conference.h> Class to generate conference candidate extensions.

Public Functions

CandidateGeneratorConference (const array_link &al, const conference_t &ct)

const std::vector<conference_column> &generateCandidates (const array_link &al) const
    Generate a list of candidate extensions for the specified design.

std::vector<conference_column> generateCandidatesZero (const array_link &al, int kz) const
    generate all candidate extensions with a zero at the specified position

class CandidateGeneratorDouble : public CandidateGeneratorBase
    #include <conference.h> Class to generate double conference candidate extensions with caching.

Public Functions

CandidateGeneratorDouble (const array_link &al, const conference_t &ct)

const std::vector<conference_column> &generateCandidates (const array_link &al) const
    Generate a list of candidate extensions for the specified design.
    This method uses symmetry inflation, assumes j1=0 and j2=0. Optimal performance is achieved when the
    arrays to be extended have identical first columns.

class DconferenceFilter
    #include <conference.h> class to filter single or double conference designs
Public Functions

\texttt{DconferenceFilter (const array\_link \_als, int filter\_symmetry, int filterj2\_, int filterj3\_ = 1)}

\texttt{void show () const}
\hspace{1em} print object to stdout

std::vector<conference\_column> filterList (const std::vector<conference\_column> \_list, int verbose = 0) const
\hspace{1em} filter a list of columns using the filter method

std::vector<conference\_column> filterListJ2last (const std::vector<conference\_column> \_column\_list) const
\hspace{1em} filter a list of columns using the filterZero method

bool filter (const conference\_column \_c) const
\hspace{1em} return True if the extension satisfies all checks

bool filterJpartial (const conference\_column \_c, int maxrow) const
\hspace{1em} Filter on partial column (only last col)

Parameters

- \texttt{column}: Extension column
- \texttt{maxrow}: the number of rows that are valid

bool filterJ (const conference\_column \_column, int j2start = 0) const
\hspace{1em} return True if the extension satisfies all J-characteristic checks

bool filterJlast (const conference\_column \_c, int j2start = 0) const
\hspace{1em} return True if the extension satisfies all J-characteristic checks for the last columns

bool filterReason (const conference\_column \_column) const
\hspace{1em} return True if the extension satisfies all checks. prints the reason for returning True or False to stdout

bool filterJ3 (const conference\_column \_column) const
\hspace{1em} return True if the candidate satisfies the J3 check

bool filterJ3s (const conference\_column \_column, int idxstart) const
\hspace{1em} return True if the candidate satisfies the J3 check for specified pairs

bool filterJ3inline (const conference\_column \_column) const
\hspace{1em} return True if the candidate satisfies the J3 check

bool filterSymmetry (const conference\_column \_column) const
\hspace{1em} return True of the candidate satisfies the symmetry check

bool filterJ2 (const conference\_column \_c) const
\hspace{1em} return True of the candidate extension satisfies the J2 check

bool filterJ2last (const conference\_column \_c) const
\hspace{1em} return True of the candidate extension satisfies the J2 check for the last column of the array checked against

bool filterZero (const conference\_column \_c) const
\hspace{1em} return True of the candidate extension satisfies the zero check

This means that the first entries of the extension do not contain a zero.
**Public Members**

*array_link als*

int filtersymm
   filter based on symmetry

int filterj2
   filter based on j2 value

int filterj3
   filter based on j3 value

int filterfirst
   filter only columns with first value >=0

int filterzero
   filter based on first occurrence of zero in a column

long ngood

int inline_row
   row at which infile filtering is performed

*symmdata sd*

**Private Members**

*array_link dtable*
   table of J2 vectors for J3 filter

*array_link inline_dtable*
   table of J2 vectors for inline J3 filter

std::vector<int> check_indices
   indices to check for symmetry check

int minzvalue
   used for filtering based on zero

### 7.5 Interface for even-odd designs

Contains functions to generate even-odd designs.

The generation is done by defining a special ordering in the set of designs. The primary ordering is based in the J5 value of 5-column designs, the secondary ordering is the regular LMC ordering.
Typedefs

typedef Pareto<mvalue_t<long>, array_link>::pValue (*pareto_cb) (const array_link&, int)  
callback function for Pareto calculations

typedef Pareto<mvalue_t<long>, array_link>::pValue (*pareto_cb_cache) (const array_link&, int,  
rankStructure &rs)  
callback function for Pareto calculations with cache

Enums

enum depth_alg_t  
Values:

DEPTH_DIRECT
DEPTH_EXTENSIONS

Functions

void processDepth (const arraylist_t &goodarrays, depth_alg_t depthalg, depth_extend_t &dextend,  
depth_extend_sub_t &dextendsublight, int extensioncol, int verbose = 0)  
Extend arrays using a depth-first or breadth-first approach

Parameters

• goodarrays: List of arrays to extend
• depthalg: Extend using depth-first or breadth-first
• dextend: Option structure for the extension
• dextendsublight: Data structure for the extensions
• extensioncol: Column to extend
• verbose: Verbosity level

void depth_extend_hybrid (const arraylist_t &alist, depth_extend_t &dextend, int extcol, const  
OAextend &oaextendx, int verbose)  
depth-first extension of arrays. depending on the symmetry group of the array to be extended a direct method is  
used or a method with caching of candidate columns

void depth_extend_direct (const arraylist_t &alist, depth_extend_t &dextend, int extcol, const  
OAextend &oaextendx, int verbose)  
variation of depth_extend for arrays with large symmetry groups

void depth_extend_array (const array_link &al, depth_extend_t &dextend, const arraydata_t &ad-  
full, int verbose, depth_extensions_storage_t *ds = 0, int = 0)  
depth extend a single array

template<class IndexType>  
Pareto<mvalue_t<long>, IndexType>::pValue calculateArrayParetoJ5Cache (const array_link  
&al, int verbose,  
rankStructure &rs)  

void addArraysToPareto (Pareto<mvalue_t<long>, array_link> &pset, pareto_cb paretofunction, const  
arraylist_t &arraylist, int jj, int verbose)  
add arrays to set of Pareto results

7.5. Interface for even-odd designs
void addArraysToPareto (Pareto<mvalue_t<long>, array_link> &pset, pareto_cb_cache paretofunction, const arraylist_t &arraylist, int jj, int verbose)
   add arrays to set of Pareto results

Jcounter readStatisticsFile (const char *numbersfile, int verbose)
   read statistics object from disk

void writeStatisticsFile (const char *numbersfile, const Jcounter &jc, int verbose)
   write statistics object to disk

Jcounter calculateJstatistics (const char *afile, int jj = 5, int verbose = 1)
   calculate J-value statistics

int compareJ54 (const array_link &lhs, const array_link &rhs)
   Return -1 if the first array is smaller in J54 ordering than the second array, 0 if equal and 1 otherwise

struct depth_path_t
   #include <evenodd.h> structure containing current position in search tree

Public Functions

depth_path_t ()

void updatePositionGEC (int k, int goodextensioncols)

void updatePosition (int k, int c, int m, int extensioncols, int goodextensioncols)

void show (int depth, int maxentries = 8) const

void init (int ncols, int _depthstart = 9)

Public Members

std::vector<int> ncurr
   vector with current position

std::vector<int> nmax
   vector with target

std::vector<int> necols
   number of extension columns

std::vector<int> ngecols
   number of good extension columns

int depthstart

struct counter_t
   #include <evenodd.h> structure to count and show number of arrays generated, the structure is thread safe
Public Functions

counter_t (int n)
void addNfound (int col, int num)
long nArrays () const
void addNumberFound (int n, int k)
void clearNumberFound ()
void addNumberFound (counter_t &de)
void showcountsc (const char *str, int first, int last) const

Public Members

std::vector<int> nfound

struct depth_extend_sub_t
#include <evenodd.h> Helper structure for dynamic extension
In this structure we keep track of pointers to valid column extensions

Public Functions

depth_extend_sub_t (int nn = 0)
void resize (int nn)
size_t n () const
std::vector<int> updateExtensionPointers (int extcol)
arraylist_t initialize (const arraylist_t &alist, const arraydata_t &adf, const OAextend &oaextend)
initialize the new list of extension columns
arraylist_t selectArraysZ (const arraylist_t &alist) const
select the arrays with are LMC and hence need to be written to disk
arraylist_t selectArraysXX (const array_link &al, const arraylist_t &elist) const
void info () const

7.5. Interface for even-odd designs
Public Members

std::vector<int> lmc_type
std::vector<int> lastcol
    last column changed in lmc check
std::vector<double> strengthcheck
std::vector<int> valididx
int verbose

struct depth_extend_t
    #include <evenodd.h>  Helper structure for dynamic extension.
    This structure allows for writing the generated arrays to disk. It also contains functions to print progress of the extension.
    Multiple copies of this class are made, but they all share the same counter_t and arraywriter_t object. Also t0 and tp are shared

Public Functions

depth_extend_t (const arraydata_t *ad_, double _logtime = 10000000, int _discardJ5 = -1)
depth_extend_t (const depth_extend_t &de)
~depth_extend_t ()
void show ()
void setNarraysMax (long n)
void maxArrayCheck ()
void showsearchpath (int depth) const
bool showprogress (int showtime = 1, int depth = 0, int forcelog = 0)
    show information about the progress of the loop
void info () const
void setPosition (int k, int c, int m, int extensioncols = -1, int goodextensioncols = -1)
    set the position in the dextend structure
void setPositionGEC (int k, int goodextensioncols)
    set the position in the dextend structure

Public Members

int verbose
OAextend oaextend
const arraydata_t *ad
int loglevelcol
double logtime
    print progress every x seconds
arraylist_t extension_column_list
int writearrays
    if set to true write arrays to disk
int discardJ5
long discardJ5number
    if true, then we discard the designs which have J5 maximal
arraywriter_t *arraywriter
counter_t *counter

Public Static Attributes

double t0

double tp

Private Members

long narraysmax
depth_path_t searchpath

struct depth_extensions_storage_t
#include <evenodd.h> Helper structure for the even-odd depth extension.

Public Functions

void resize (size_t s)

void set (int ai, const arraylist_t &goodarrays, const arraylist_t &extension_column_list,
    depth_alg_t depthalg, const depth_extend_sub_t &dextendsub)

Public Members

std::vector<arraylist_t> columnextensionsList
std::vector<arraylist_t> goodarrayslist
std::vector<depth_alg_t> depthalglist
std::vector<depth_extend_sub_t> dextendsubList

struct jindex_t
#include <evenodd.h> helper class for indexing statistics of designs

The index consists of the number of columns and the value for the J-characteristic

7.5. Interface for even-odd designs
Public Functions

jindex_t (int colindex, int jvalue)
bool operator<(const jindex_t &rhs) const
std::string toString() const

Public Members

int k
    number of columns
int j
    J-value.

class Jcounter
#include <evenodd.h> object to hold counts of maximum J_k-values

Public Functions

Jcounter()
Jcounter (int N, int jj = 5, int k = -1)
bool validData ()
bool hasColumn (int col) const
    return true if specified column is in the data
bool isOpen() const
void showPerformance() const
long narrays() const
void show() const
    show statistics of the object
int maxCols() const
long getCount (int k, int j) const
std::vector<long> getTotalsJvalue (int jval) const
std::vector<long> getTotals() const
void showcompact() const
    show statistics of the object

Jcounter &operator+=(Jcounter &jc)

void addArrays (const arraylist_t &arraylist, int verbose = 0)
    add list of arrays to object
void addArray (const array_link &al, int verbose = 0)
    add single array to statistics object
Public Members

int \textbf{N} \\
number of rows

int \textbf{jj}

std::vector<int> \textbf{fvals}

std::map<jindex_t, long> \textbf{maxJcounts}

double \textbf{dt} \\
time needed for calculation

Private Functions

void \textbf{init} (int \textit{N}, int \textit{jj}, int \textit{k} = -1)

7.6 Interface for extension of LMC designs

Contains functions to generate and extend orthogonal arrays.

Enums

\textbf{enum dfilter_t} \\
Values:

\begin{itemize}
\item \textbf{DFILTER_NONE} \\
no filtering on D-efficiency
\item \textbf{DFILTER_BASIC} \\
filtering on D-efficiency
\item \textbf{DFILTER_MULTI} \\
filtering on D-efficiency with multi column prediction
\end{itemize}

\textbf{enum dcalc_mode} \\
Values:

\begin{itemize}
\item \textbf{DCALC_ALWAYS} \\
always calculate efficiency
\item \textbf{DCALC_COND} \\
only calculate efficiency for LMC_LESS
\end{itemize}

Functions

\begin{itemize}
\item \textbf{arraylist_t extend_arraylist (const arraylist_t &array_list, arraydata_t &array_class, OAextend const &oaextend_options)} \\
Extend a list of orthogonal arrays
\item \textbf{Return} List of all generated arrays
\item \textbf{See} extend_array(const array_link &, arraydata_t &, OAextend const &)
\end{itemize}
• array_list: The list of arrays to be extended
• array_class: Class of arrays to generate
• oaextend_options: Parameters for the extension algorithm

arraylist_t extend_arraylist (const arraylist_t &array_list, const arraydata_t &array_class)
Extend a list of arrays with default options

See extend_array(const array_link &, arraydata_t &, OAextend const &)

int extend_arraylist (const arraylist_t &array_list, arraydata_t &array_class, OAextend const &oaextend_options, colindex_t extensioncol, arraylist_t &extensions)
Extend a list of orthogonal arrays

Return List of all generated arrays
See extend_array(const array_link &, arraydata_t &, OAextend const &)

Return Number of candidate arrays generated

Parameters
• array_list: The list of arrays to be extended
• array_class: Class of arrays to generate
• oaextend_options: Parameters for the extension algorithm

Parameters
• extensioncol: Index of column to be added to the designs
• extensions: List to append generated designs to

arraylist_t extend_array (const array_link &array, arraydata_t &array_class, OAextend const &oaextend)
Extend a single orthogonal array

Parameters
• array: The array to be extended
• array_class: Class of arrays to generate
• oaextend: Parameters for the extension algorithm

arraylist_t extend_array (const array_link &array, arraydata_t &array_class)
Extend a single orthogonal array with the default LMC algorithm

See extend_array(const array_link &, arraydata_t &, OAextend const &)

int extend_array (const array_link &array, const arraydata_t *array_class, const colindex_t extension_column, arraylist_t &extensions, OAextend const &oaextend)
Extend an orthogonal array with a single column

See extend_array(const array_link &, arraydata_t &, OAextend const &)

Return Number of candidate extensions generated

Parameters
• array: Array to extend
• arrayclass: Array data for the full array
• extension_column: Column to extend
• extensions: List to which generated valid extensions are added
• oaextend: Structure with options

arraylist_t runExtendRoot (arraydata_t arrayclass, int max_number_columns, int verbose = 0)
Run the LMC extension algorithm starting with the root array

See extend_array(const array_link &, arraydata_t &, OAextend const &)

class OAextend
#include <extend.h> Options for the extend code.
class containing parameters of the extension and LMC algorithm

Public Types

denum extendarray_mode_t
Specification of how to use the generated extensions.
Values:

APPENDEXTENSION
append extension column to extension list

APPENDFULL
append full array to extension list

STOREARRAY
store extension to disk

NONE
do not store generated extensions

Public Functions

OAextend ()
Options for the extension algorithm

OAextend (const OAextend &o)
Options for the extension algorithm

OAextend (arraydata_t &arrayclass)
Options for the extension algorithm

The algorithm is automatically determined from the specified arrayclass.

void setAlgorithm (algorithm_t algorithm, arraydata_t *ad = 0)
Set the algorithm to use for LMC checks.

void setAlgorithmAuto (arraydata_t *ad = 0)
Set the algorithm automatically.

algorithm_t getAlgorithm () const
Return algorithm used.

7.6. Interface for extension of LMC designs
std::string getAlgorithmName() const
    Return algorithm used (as string)

void updateArraydata(arraydata_t* arrayclass = 0) const
    update the options structure with the specified class of designs

void info(int verbose = 1) const
    print configuration to stdout

std::string __repr__() const

**Public Members**

double singleExtendTime
    time before printing progress of single extension, [seconds]

int nLMC
    number of arrays LMC tested before printing progress of single extension

int checkarrays
    perform LMC test after generation of array

int check_maximal
    if true then return at once if a single extension has been found

int use_row_symmetry
    adds a symmetry check to the extension algorithm based in symmetry of row permutations

int init_column_previous
    init column with previous column in extension (if in the same column group)

extendarray_mode_t extendarraymode
    determines how the extension arrays are stored

arrayfile_t storefile

j5structure_t j5structure

**Public Static Functions**

static algorithm_t getPreferredAlgorithm(const arraydata_t &arrayclass, int verbose = 0)
    Return preferred extension algorithm

    **Return** Algorithm selected to be used for this class

    **Parameters**
        • arrayclass: Class of designs to extend
        • verbose: Verbosity level
Private Members

algorithm_t algmode
    Algorithm mode.

struct dextend_t
    #include <extend.h> Structure for dynamic extension of arrays based on D-efficiencies.

Public Functions

dextend_t ()
    void resize(int nn)
    void DefficiencyFilter(double Dfinal, int k, int kfinal, double Lmax, int verbose = 1)
        perform filtering using D-efficiency
    std::vector<int> filterArrays(const array_link &al, const arraylist_t &earrays, arraylist_t &earraysout, std::vector<std::vector<double>> &edata, int verbose = 1)
        filter the arrays based on values in filter

Public Members

std::vector<lmc_t> lmctype
    results of minimal form calculations
std::vector<int> lastcol
    last column changed in lmc check
std::vector<double> Deff
    calculated efficiency values
std::vector<int> filter
    indices of filtered arrays
dfilter_t filtermode
dcalc_mode Dcheck
    int directcheck
        perform immediate LMC check in extension
long ntotal
    total number of arrays found
long nlmc
    total number of arrays found in LMC form
long n
    total number of arrays found passing all tests
double DmaxDiscard
long nmaxrnktotal
Public Static Attributes

\texttt{const \ int NO\_VALUE = 0}

### 7.7 Interface for graph tools

This file contains definitions and functions related to graphs and designs.

Author: Pieter Eendebak \ pieter.eendebak@gmail.com, (C) 2016

Copyright: See COPYING file that comes with this distribution

#### Enums

**enum matrix_isomorphism_t**

Isomorphism types for matrices

- **ISOTOPY**
  - isotopy: permute rows, columns and symbols
- **MATRIX\_ISOMORPHISM**
  - permute rows and columns
- **CONFERENCE\_ISOMORPHISM**
  - permute rows, columns and to row and column negations (values in 0, +1, -1)
- **OA\_ISOMORPHISM**
  - permutations of rows, columns and column symbol permutations

#### Functions

- **array\_link transformGraph(const \ array\_link \& graph, \ const std::vector<int> \& vertex\_permutation, int \ verbose = 1)**
  - Apply a vertex permutation to a graph.

- **array\_transformation\_t reduceOAnauty(const \ array\_link \& array, int \ verbose = 0)**
  - Reduce an orthogonal array to Nauty minimal form. The array transformation is returned.

- **array\_transformation\_t reduceOAnauty(const \ array\_link \& array, int \ verbose, \ const \ arraydata\_t \& ar- rayclass)**
  - Reduce an orthogonal array to Nauty minimal form. The array transformation is returned.

- **std::pair<array\_link, \ std::vector<int>\rangle array2graph(const \ array\_link \& array, int \ verbose = 1)**
  - Convert orthogonal array to graph representation
  
  The conversion method is as in Ryan and Bulutoglu. The resulting graph is bi-partite. The graph representation can be used for isomorphism testing.

- **std::pair<array\_link, \ std::vector<int>\rangle array2graph(const \ array\_link \& array, int \ verbose, \ const \ arraydata\_t \& arrayclass)**
  - Convert orthogonal array to graph representation
The conversion method is as in Ryan and Bulutoglu. The resulting graph is bi-partite. The graph representation can be used for isomorphism testing.

```cpp
array_transformation_t oagraph2transformation (const std::vector<int> &pp, const arraydata_t &arrayclass, int verbose = 1)
```

From a relabelling of the graph return the corresponding array transformation.

### Variables

```cpp
const matrix_isomorphism_t CONFERENCE_RESTRICTED_ISOMORPHISM = OA_ISOMORPHISM
```

isomorphism type for column and row permutations and column permutations

### namespace nauty

### Functions

```cpp
std::vector<int> reduceNauty (const array_link &graph, std::vector<int> colors, int verbose = 0)
```

Reduce a colored graph to Nauty minimal form

The transformation returned is from the normal form to the specified graph.

**Return** Relabelling of the graph vertices

**Parameters**

- `graph`: Graph in incidence matrix form
- `colors`: Colors of the graph nodes
- `verbose`: Verbosity level

### 7.8 Interface for LMC normal forms

This file contains definitions and functions to perform minimal form tests and reductions.

**Author:** Pieter Eendebak pieter.eendebak@gmail.com

**Copyright:** See LICENSE file that comes with this distribution

**Defines**

```cpp
ORDER_J5_SMALLER
ORDER_J5_GREATER
ORDER_J45_SMALLER
ORDER_J45_GREATER
stringify(name)
SDSMART
```
**Typedefs**

```cpp
typedef unsigned int rowsort_value_t

typedef rowindex_t rowsortlight_t

typedef larray<rowindex_t> rowpermtypelight

typedef larray<colindex_t> colpermtypelight

typedef std::vector<int> colpermtype

typedef std::vector<colpermtype> colpermset

typedef std::tr1::shared_ptr<symmdata> symmdataPointer

typedef double jj45_t
```

Value representing the ordered combination of J5 and the 5 J4-values in the J54 ordering.

**Enums**

```cpp
enum lmc_t
    Possible results for the LMC check
    Values:

    LMC_LESS
    Found a permutation which leads to a lexicographically smaller array.

    LMC_EQUAL
    Found a permutation which leads to a lexicographically equal array.

    LMC_MORE
    Found a permutation which leads to a lexicographically larger array.

    LMC_NONSENSE
    No valid result.

enum algorithm_t
    different algorithms for minimal form check
    Values:

    MODE_LMC
    LMC minimal form.

    MODE_J4
    LMC minimal form with J4 method.

    MODE_J5ORDER
    J5 minimal form.

    MODE_J5ORDERX
    J5 minimal form.

    MODE_INVALID

    MODE_AUTOSELECT
    Automatically select the algorithm.

    MODE_LMC_SYMMETRY
    debugging method
MODE_LMC_2LEVEL
LMC minimal form, specialized for 2-level arrays.

MODE_LMC_DEBUG
debugging method

MODE_J5ORDER_2LEVEL
J5 minimal form for 2-level arrays.

enum initcolumn_t
method used for initialization of columns

Values:

INITCOLUMN_ZERO
Initialize column with zeros.

INITCOLUMN_PREVIOUS
Initialize column with values of previous column.

INITCOLUMN_J5
Initialize column with values based on J5 value.

enum j5structure_t
variations of the J45 structures

Values:

J5_ORIGINAL
Ordering based in J5 in successive columns.

J5_45
Ordering based on J5 and the 5-tuple of J4 values.

enum REDUCTION_STATE
variable indicating the state of the reduction process

Values:

REDUCTION_INITIAL
the reduction is equal to the initial

REDUCTION_CHANGED
the reduction was changed

enum OA_MODE
main mode for the LMC routine: test, reduce or reduce with initialization

Values:

OA_TEST
test for minimal form

OA_REDUCE
reduce to minimal form

OA_REDUCE_PARTIAL
reduce to partial minimal form

enum INIT_STATE
initial state for reduction algorithm

Values:

INIT_STATE_INVALID
COPY
    copy from array argument

INIT
    initialized by user

SETROOT
    set initial state to root array

Functions

std::string algnames(algorithm_t m)
    return name of the algorithm

static bool operator<(const rowsort_t &a, const rowsort_t &b)
    Comparison operator for the rowsort_t structure.

static bool operator>(const rowsort_t &a, const rowsort_t &b)
    Comparison operator for the rowsort_t structure.

void apply_hadamard(array_link &al, colindex_t hcolumn)
    Apply Hadamard transformation to orthogonal array.

LMCreduction_helper_t *acquire_LMCreduction_object()
    return static structure from dynamic global pool, return with releaseGlobalStatic

void release_LMCreduction_object(LMCreduction_helper_t *p)

void clear_LMCreduction_pool()
    release all objects in the pool

template<class Type>
void insert_if_not_at_end_of_vector(std::vector<Type> &cp, const Type &value)
    Append element to vector if the element the element is not at the end of vector.

bool is_root_form(const array_link &array, int strength)
    Return True if the array is in root form

Parameters

    • array: Array to check
    • strength: Strength to use

LMCreduction_train(const array_link &al, const arraydata_t *ad, LMCReduction_t *reduction, const OAextend &oaextend)
    helper function for LMC reduction

LMCcheck(const array_t *array, const arraydata_t &ad, const OAextend &oaextend, LMCReduction_t &reduction)
    Perform LMC check or reduction on an array.

LMCcheck(const array_link &array, const arraydata_t &ad, const OAextend &oaextend, LMCReduction_t &reduction)
    Perform LMC check or reduction on an array.

LMCcheck(const array_link &array)
    Perform LMC check on an orthogonal array
Return Result of the LMC check

Parameters

- `array`: Array to be checked for LMC minimal form

`lmc_t LMCcheckOriginal (const array_link &array)`

Perform LMC check on a 2-level orthogonal array

The algorithm used is the original algorithm from “Complete enumeration of pure-level and mixed-level orthogonal arrays”, Schoen et al, 2009

Return Result of the LMC check

Parameters

- `array`: Array to be checked for LMC minimal form

`void reduceArraysGWLP (const arraylist_t &input_arrays, arraylist_t &reduced_arrays, int verbose, int dopruning = 1, int strength = 2, int dolmc = 1)`

reduce arrays to canonical form using delete-1-factor ordering

`array_transformation_t reductionDOP (const array_link &array, int verbose = 0)`

Calculate the transformation reducing an array to delete-on-factor normal

The normal form is described in “A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections”, Eendebak, 2014

Return The transformation that reduces the array to normal form

Parameters

- `array`: Orthogonal array
- `verbose`: Verbosity level

`array_link reduceDOPform (const array_link &array, int verbose = 0)`

Reduce an array to canonical form using delete-1-factor ordering

The normal form is described in “A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections”, Eendebak, 2014

Return The array transformed to normal form

Parameters

- `array`: Orthogonal array
- `verbose`: Verbosity level

`void selectUniqueArrays (arraylist_t &input_arrays, arraylist_t &output_arrays, int verbose = 1)`

select the unique arrays in a list, the original list is sorted in place. the unique arrays are append to the output list

`std::vector<GWLPvalue> projectionDOFvalues (const array_link &array, int verbose = 0)`

Calculate projection values for delete-of-factor algorithm

`array_link reduceLMCform (const array_link &array)`

reduce an array to canonical form using LMC ordering

`std::vector<int> LMCcheckLex (arraylist_t const &list, arraydata_t const &ad, int verbose = 0)`

Apply LMC check (original mode) to a list of arrays

7.8. Interface for LMC normal forms
Perform minimal form check with LMC ordering.

Perform minimal form check with J4 ordering.

Perform minimal form check for J5 ordering.

Print the contents of a rowsort structure.

Print the contents of a rowsort structure.

Print the contents of a rowsort structure.

Variables

const algorithm_t MODE_ORIGINAL = MODE_LMC

struct LMCreduction_helper_t

# include <lmc.h> Contains structures used by the LMC reduction or LMC check.

Public Functions

LMCreduction_helper_t ()

~LMCreduction_helper_t ()

void show (int verbose = 1) const

void init (const arraydata_t *adp)

void freeall ()

int update (const arraydata_t *adp)

update structure with new design specification

int needUpdate (const arraydata_t *adp) const

void init_root_stage (levelperm_t *&lperm_p, colperm_t *&colperm_p, const arraydata_t *adp)

void init_nonroot_stage (levelperm_t *&lperm_p, colperm_t *&colperm_p, colperm_t *&localcolperm_p, dyndata_t **&dynd_p, int &dynd_p_nelem, array_t * &colbuffer, const arraydata_t * &adp) const

void init_rootrowperms (int &totalperms, rowperm_t *&rootrowperms, levelperm_t *&lperm_p)

Static initialization of root row permutations.
void init_rootrowperms_full (int &totalperms, rowperm_t *rootrowperms, levelperm_t *lperm_p)
    Static initialization of root row permutations (full group)

Public Members

int LMC_non_root_init
int LMC_root_init
int LMC_reduce_root_rowperms_init
arraydata_t *ad
int LMC_root_rowperms_init
int nrootrowperms
    number of root row permutations
rowperm_t *rootrowperms
    pointer to row permutations that leave the root unchanged
int LMC_root_rowperms_init_full
int nrootrowperms_full
rowperm_t *rootrowperms_full
array_t *colbuffer
dyndata_t **dyndata_p
    buffer for a single column
colindex_t **colperm_p
dynamic data; row permutations
colindex_t **localcolperm_p
    column permutations
array_transformation_t *current_trans
    local column permutation

struct LMCreduction_t
    #include <lmc.h> Class to describe an LMC reduction.
    The most important variable is the transformation itself, contained in transformation. The state contains information about how the reduction was performed.

Public Functions

LMCReduction_t (const LMCReduction_t &at)

LMCReduction_t (const arraydata_t *arrayclass)
    copy constructor

~LMCReduction_t ()

LMCReduction_t &operator= (const LMCReduction_t &at)

array_link getArray () const
    Assignment operator.
void setArray(const array_link al)

void setArray(const array_t*array, int nrows, int ncols)

void updateSDpointer(const array_link al, bool cache = false)
update the pointer to the symmetry data based on the specified array

void releaseStatic()
release internal LMCReduction_helper_t object

void initStatic()
acquire a reference to a LMCReduction_helper_t object

LMCreduction_helper_t &getReferenceReductionHelper()
return a reference to a object with LMC reduction data

void reset()
reset the reduction: clears the symmetries and sets the transformation to zero

void show (int verbose = 2) const
std::string __repr__ () const

void updateFromLoop(const arraydata_t &ad, const dyndata_t &dynd, levelperm_t *lperms,
const array_t *original)
called whenever we find a reduction

void updateTransformation(const arraydata_t &ad, const dyndata_t &dynd, levelperm_t *
lperms, const array_t *original)

void updateLastCol (int col)

**Public Members**

array_t *array

array_transformation_t *transformation
pointer to transformation_t structure

**OA_MODE** mode

**REDUCTION_STATE** state

**INIT_STATE** init_state

int maxdepth
maximum depth for search tree

int lastcol
last column visited in algorithm

long nred
counter for number of reductions made

int targetcol
int mincol
int nrows
int ncols

**LMCreduction_helper_t** *staticdata**
symmdataPointer sd

**Private Functions**

void free()

class rowsorter_t
#include <lmc.h> Structure to sort rows of arrays.

**Public Functions**

rowsorter_t (int number_of_rows)
~rowsorter_t ()

**Public Members**

int number_of_rows
rowsort_t *rowsort

**Private Functions**

void reset_rowsort()

struct dyndata_t
#include <lmc.h> Contains dynamic data of an array.

The dynamic data are used in the inner loops of the LMC algorithm. In particular they keep track of the current row ordering and column permutation. By not applying these transformations to the array we can save calculation time.

We try to prevent copying the object, so it is re-used at different levels in the algorithm.

- N: static
  - col: changes at each column level
- rowsort: changes at each column level, used mainly in non-root stage
- colperm: changes at all levels

See arraydata_t

**Public Functions**

dyndata_t (int N, int col = 0)
dyndata_t (const dyndata_t *dd)
dyndata_t (const dyndata_t&)
~dyndata_t ()
dyndata_t &operator=(const dyndata_t&)

7.8. Interface for LMC normal forms
void show() const
void reset()
void setColperm(const colperm_t perm, int n)
void setColperm(const larray<colindex_t> &perm)
void setColperm(const std::vector<colindex_t> &perm)
void getRowperm(rowpermtypelight &rp) const
    get lightweight row permutation
void getRowperm(rowperm_t &rperm) const
    get row permutation

rowpermtypelight getRowperm() const
    return lightweight row permutation
colpermtypelight getColperm() const
    return column permutation
void getColperm(colpermtypelight &cp) const
    set column permutation

void allocate_rowsortl()
    allocate lightweight rowsort structure
void deleterowsortl()
void initrowsortl()
    initialize rowsortl from rowsort
void rowsortl2rowsort()
    copy rowsortl variable to rowsrt
void copydata(const dyndata_t &dd)

Public Members

colindex_t col
    active column
rowindex_t N
    number of rows
rowsort_t *rowsort
    ordering of rows
rowsortlight_t *rowsortl

colperm_t colperm
    current column permutation
Private Functions

```c
void initdata(const dyndata_t &dd)
```

### 7.9 Interface for MD5 sums

Contains functions to calculate MD5 sums.

**Functions**

```c
std::string md5 (void *data, int number_of_bytes)
```
calculate md5 sum of a data block in memory

```c
std::string md5 (const std::string &filename)
```
calculate md5 sum of a file on disk

### 7.10 Interface for Pareto optimality

Class for calculation the Pareto optimal elements from a set of multi-valued objects.

Copyright (c) 2013, Pieter Eendebak pieter.eendebak@gmail.com All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Defines**

```c
defines
```

```c
template<class ValueType, class IndexType>
struct pareto_element
#include <pareto.h> helper class for the Pareto class to hold elements
```
Public Types

typedef std::vector<

Public Functions

bool dominates(pValue v)
return true if the argument element dominates this value

bool isdominated(pValue v)
return true if the argument element is dominated by this value

bool equal(pValue v)
return true if the argument element is equal to this value

Public Members

pValue value
std::vector<IndexType> indices

template<class ValueType, class IndexType>
class Pareto
#include <pareto.h> Class to calculate Pareto optimal elements.
The class is templated by the type of values to be compared and an index type. The index type is used to index the elements.
For elements added to the Pareto structure larger is better.

Public Types

typedef std::vector<

typedef pareto_element<

Public Functions

Pareto()
Create an empty Pareto class.

~Pareto()

int number() const
return the total number of Pareto optimal values

int numberindices() const
return the total number Pareto optimal objects

std::string __repr__() const

void show(int verbose = 1)
show the current set of Pareto optimal elements
std::deque<IndexType> allindicesdeque() const  
return all indices of the Pareto optimal elements as a std::deque

std::vector<IndexType> allindices() const  
return all indices of the Pareto optimal elements

std::vector<pValue> allvalues() const  
return the values of all Pareto optimal elements

bool addvalue(const pValue value, const IndexType idx)  
add a new element

**Public Members**

int verbose
Verbosity level.

std::deque<pareto_element<ValueType, IndexType>> elements  
contains a list of all Pareto optimal elements

**Public Static Functions**

static void showvalue(const pValue p)  
show a Pareto element
REFERENCES
INDICES AND TABLES

- genindex
- modindex
- search
Copyright (c) 2011-2018, Pieter Eendebak <pieter.eendebak@gmail.com> Copyright (c) 2009, TNO Science & Industry

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Citation notice: if you use this work or any results of this work, please cite the work as (see the file README.md for details):

“OApackage: A Python package for generation and analysis of orthogonal arrays, optimal designs and conference designs”, https://doi.org/10.21105/joss.01097

For more information contact (pieter.eendebak@gmail.com or eric.schoen@tno.nl).

This package also uses code from other works:

- Eigen (see http://eigen.tuxfamily.org/, MPL2 license)
- InfInt (by Sercan Tutar, MPL2 license)
- bitarray (by Isaac Turner, MIT license)
- md5.cpp (by RSA Data Security)
- msst dint.h (by Alexander Chemeris, see https://code.google.com/p/msinttypes/)
- oapackage._scanf (from https://github.com/joshburnett/scanf, MIT license)
BIBLIOGRAPHY


<table>
<thead>
<tr>
<th>C++ Function/Member</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3A4</td>
<td>60</td>
</tr>
<tr>
<td>acquire_LMCReduction_object</td>
<td>116</td>
</tr>
<tr>
<td>addArraysToPareto</td>
<td>101</td>
</tr>
<tr>
<td>addConstant</td>
<td>66</td>
</tr>
<tr>
<td>addJMax</td>
<td>60</td>
</tr>
<tr>
<td>Aefficiencies</td>
<td>57</td>
</tr>
<tr>
<td>Aefficiency</td>
<td>57</td>
</tr>
<tr>
<td>algnames</td>
<td>116</td>
</tr>
<tr>
<td>algorithm_t</td>
<td>114</td>
</tr>
<tr>
<td>algorithm_t_list</td>
<td>116</td>
</tr>
<tr>
<td>analyseArrays</td>
<td>66</td>
</tr>
<tr>
<td>append_arrayfile</td>
<td>67</td>
</tr>
<tr>
<td>appendArrays</td>
<td>68</td>
</tr>
<tr>
<td>APPENDEXTENSION</td>
<td>109</td>
</tr>
<tr>
<td>APPENDFULL</td>
<td>109</td>
</tr>
<tr>
<td>apply_hadamard</td>
<td>116</td>
</tr>
<tr>
<td>array2eigenModelMatrix</td>
<td>69</td>
</tr>
<tr>
<td>array2eigenModelMatrixMixed</td>
<td>69</td>
</tr>
<tr>
<td>array2eigenX1</td>
<td>68</td>
</tr>
<tr>
<td>array2eigenX2</td>
<td>68</td>
</tr>
<tr>
<td>array2graph</td>
<td>112</td>
</tr>
<tr>
<td>array2modelmatrix</td>
<td>59</td>
</tr>
<tr>
<td>array2modelmatrix_sizes</td>
<td>59</td>
</tr>
<tr>
<td>array2rank_Deff_Beff</td>
<td>56</td>
</tr>
<tr>
<td>array2secondorder</td>
<td>58</td>
</tr>
<tr>
<td>array2xf</td>
<td>59</td>
</tr>
<tr>
<td>array2xfeigen</td>
<td>59</td>
</tr>
<tr>
<td>array_diff</td>
<td>66</td>
</tr>
<tr>
<td>array_link</td>
<td>73</td>
</tr>
<tr>
<td>array_link::at</td>
<td>77</td>
</tr>
<tr>
<td>array_link::setvalue</td>
<td>78</td>
</tr>
<tr>
<td>array_link::valid_index</td>
<td>80</td>
</tr>
<tr>
<td>array_link::~array_link</td>
<td>74</td>
</tr>
<tr>
<td>array_link::Aefficiency</td>
<td>75</td>
</tr>
<tr>
<td>array_link::array</td>
<td>79</td>
</tr>
<tr>
<td>array_link::array_link</td>
<td>74</td>
</tr>
<tr>
<td>array_link::at</td>
<td>78</td>
</tr>
<tr>
<td>array_link::atfast</td>
<td>77</td>
</tr>
<tr>
<td>array_link::CL2discrepancy</td>
<td>77</td>
</tr>
<tr>
<td>array_link::clear</td>
<td>78</td>
</tr>
<tr>
<td>array_link::clone</td>
<td>74</td>
</tr>
<tr>
<td>array_link::columnEqual</td>
<td>78</td>
</tr>
<tr>
<td>array_link::columnGreater</td>
<td>79</td>
</tr>
<tr>
<td>array_link::create_root</td>
<td>78</td>
</tr>
<tr>
<td>array_link::debug</td>
<td>78</td>
</tr>
<tr>
<td>array_link::deepcopy</td>
<td>77</td>
</tr>
<tr>
<td>array_link::Defficiencies</td>
<td>75</td>
</tr>
<tr>
<td>array_link::Defficiency</td>
<td>75</td>
</tr>
<tr>
<td>array_link::deleteColumn</td>
<td>75</td>
</tr>
<tr>
<td>array_link::DsEfficiency</td>
<td>75</td>
</tr>
<tr>
<td>array_link::Eefficiency</td>
<td>75</td>
</tr>
<tr>
<td>array_link::equal_size</td>
<td>80</td>
</tr>
<tr>
<td>array_link::equalsize</td>
<td>77</td>
</tr>
<tr>
<td>array_link::firstColumnDifference</td>
<td>78</td>
</tr>
<tr>
<td>array_link::firstDiff</td>
<td>78</td>
</tr>
<tr>
<td>array_link::foldover</td>
<td>76</td>
</tr>
<tr>
<td>array_link::Fvalues</td>
<td>75</td>
</tr>
<tr>
<td>array_link::FvaluesConference</td>
<td>75</td>
</tr>
<tr>
<td>array_link::getarraydata</td>
<td>78</td>
</tr>
<tr>
<td>array_link::getEigenMatrix</td>
<td>79</td>
</tr>
<tr>
<td>array_link::getModelMatrix</td>
<td>76</td>
</tr>
<tr>
<td>array_link::GWLP</td>
<td>76</td>
</tr>
<tr>
<td>array_link::index</td>
<td>79</td>
</tr>
<tr>
<td>array_link::INDEX_DEFAULT</td>
<td>79</td>
</tr>
<tr>
<td>array_link::INDEX_ERROR</td>
<td>79</td>
</tr>
<tr>
<td>array_link::INDEX_NONE</td>
<td>79</td>
</tr>
<tr>
<td>array_link::init</td>
<td>79</td>
</tr>
<tr>
<td>array_link::is2level</td>
<td>74</td>
</tr>
<tr>
<td>array_link::is_conference</td>
<td>74</td>
</tr>
<tr>
<td>array_link::is_mixed_level</td>
<td>74</td>
</tr>
<tr>
<td>array_link::is_orthogonal_array</td>
<td>74</td>
</tr>
<tr>
<td>array_link::isSymmetric</td>
<td>74</td>
</tr>
<tr>
<td>array_link::Jcharacteristics</td>
<td>76</td>
</tr>
</tbody>
</table>

INDEX
array_link::makeSymmetric (C++ function), 75
array_link::max (C++ function), 76
array_link::md5 (C++ function), 78
array_link::min (C++ function), 76
array_link::n_columns (C++ member), 79
array_link::n_rows (C++ member), 79
array_link::negateRow (C++ function), 78
array_link::nonzero_fraction (C++ function), 78
array_link::randomcolperm (C++ function), 76
array_link::randomperm (C++ function), 76
array_link::randomrowperm (C++ function), 76
array_link::randomizecolperm (C++ function), 76
array_link::randomperm (C++ function), 76
array_transformation_t::allocate_data_structures (C++ function), 84
array_transformation_t::apply (C++ function), 84
array_transformation_t::array_transformation_t (C++ function), 84
array_transformation_t::colperm (C++ function), 85
array_transformation_t::cperm (C++ member), 85
array_transformation_t::free_data_structures (C++ function), 85
array_transformation_t::inverse (C++ function), 84
array_transformation_t::isIdentity (C++ function), 84
array_transformation_t::lperm (C++ member), 85
array_transformation_t::lvlperm (C++ function), 85
array_transformation_t::operator* (C++ function), 84
array_transformation_t::operator== (C++ function), 84
array_transformation_t::operator= (C++ function), 84
array_transformation_t::print_transformed (C++ function), 84
array_transformation_t::randomize (C++ function), 84
array_transformation_t::randomizecolperm (C++ function), 84
array_transformation_t::randomizerowperm (C++ function), 84
array_transformation_t::reset (C++ function), 84
array_transformation_t::rowperm (C++ function), 84
array_transformation_t::rperm (C++ member), 85
array_transformation_t::setcolperm (C++ function), 85
array_transformation_t::setlevelperm (C++ function), 85
array_transformation_t::setrowperm (C++ function), 85
array_transformation_t::showarraycompact (C++ function), 79
array_transformation_t::show (C++ function), 84
arraydata_t (C++ class), 70
arraydata_t::arraydata_t (C++ function), 71
arraydata_t::arraydata_t (C++ function), 71
arraydata_t::calculate_oa_index (C++ function), 72
arraydata_t::colgroupindex (C++ member), 73
arraydata_t::colgroups (C++ member), 73
arraydata_t::complete_arraydata (C++ function), 72
arraydata_t::complete_arraydata_fixlast (C++ function), 72
arraydata_t::complete_arraydata_splitn (C++ function), 72
arraydata_t::create_root (C++ function), 72
arraydata_t::factor_levels (C++ function), 72
arraydata_t::fullidstr (C++ function), 72
arraydata_t::get_col_group (C++ function), 72
arraydata_t::getfactorlevel (C++ function), 72
arraydata_t::getS (C++ function), 72
arraydata_t::idstr (C++ function), 72
arraydata_t::idstrseriesfull (C++ function), 72
arraydata_t::is2level (C++ function), 71
arraydata_t::is_factor_levels_sorted (C++ function), 72
arraydata_t::ismixed (C++ function), 71
arraydata_t::latexstr (C++ function), 72
arraydata_t::lmc_overflow_check (C++ function), 72
arraydata_t::N (C++ member), 73
arraydata_t::ncolgroups (C++ member), 73
arraydata_t::ncols (C++ member), 73
arraydata_t::oaindex (C++ member), 73
arraydata_t::operator== (C++ function), 71
arraydata_t::operator== (C++ function), 71
arraydata_t::order (C++ member), 73
arraydata_t::randomarray (C++ function), 71
arraydata_t::reduceColumns (C++ function), 72
arraydata_t::reset_strength (C++ function), 72
arraydata_t::s (C++ member), 73
arraydata_t::set_colgroups (C++ function), 72
arraydata_t::show (C++ function), 72
arraydata_t::show_colgroups (C++ function), 72
arraydata_t::showstr (C++ function), 72
arraydata_t::strength (C++ member), 73
arraydata_t::writeConfigFile (C++ function), 71
arrayfile (C++ type), 87
arrayfile::A_AUTOMATIC (C++ enumerator), 88
arrayfile::A_AUTOMATIC_BINARY (C++ enumerator), 88
arrayfile::ABINARY (C++ enumerator), 88
arrayfile::ABINARY_DIFF (C++ enumerator), 88
arrayfile::ABINARY_DIFFZERO (C++ enumerator), 88
arrayfile::AERROR (C++ enumerator), 88
arrayfile::afilierw_t (C++ enum), 88
arrayfile::ALATEX (C++ enumerator), 88
arrayfile::arrayfile_t (C++ class), 88
arrayfile::arrayfile_t::~arrayfile_t (C++ function), 89
arrayfile::arrayfile_t::afread (C++ function), 91
arrayfile::arrayfile_t::afwrite (C++ function), 91
arrayfile::arrayfile_t::append_array (C++ function), 89
arrayfile::arrayfile_t::append_arrays (C++ function), 89
arrayfile::arrayfile_t::arrayfile_t (C++ function), 88
arrayfile::arrayfile_t::arrayNbits (C++ function), 91
arrayfile::arrayfile_t::barraysize (C++ function), 91
arrayfile::arrayfile_t::closefile (C++ function), 89
arrayfile::arrayfile_t::createfile (C++ function), 89
arrayfile::arrayfile_t::createfile_t (C++ function), 88
arrayfile::arrayfile_t::createfile_t (C++ function), 88
arrayfile::arrayfile_t::diffarray (C++ member), 92
arrayfile::arrayfile_t::filename (C++ member), 90
arrayfile::arrayfile_t::finisharrayfile (C++ function), 90
arrayfile::arrayfile_t::flush (C++ function), 89
arrayfile::arrayfile_t::getnbits (C++ function), 90
arrayfile::arrayfile_t::gzfid (C++ member), 90
arrayfile::arrayfile_t::hasrandomaccess (C++ function), 89
arrayfile::arrayfile_t::headersize (C++ function), 91
<table>
<thead>
<tr>
<th>Function/Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrayfile::arrayfile_t::isbinary</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::iscompressed</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::isopen</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::mode</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::narraycounter</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::narrays</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::NARRAYS_MAX</td>
<td>(C++ member), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::nbits</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::ncols</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::nfid</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::nrows</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::parseModeString</td>
<td>(C++ function), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::pos</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::read_array</td>
<td>(C++ function), 89, 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::read_array_binary</td>
<td>(C++ function), 89, 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::read_array_binary_zero</td>
<td>(C++ function), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::readarrays</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::readnext</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::rwmode</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::seek</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::setVerbose</td>
<td>(C++ function), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::showstr</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::swigcheck</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::updatenumbers</td>
<td>(C++ function), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::verbose</td>
<td>(C++ member), 90</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::write_array_binary</td>
<td>(C++ function), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::write_array_binary_diffzero</td>
<td>(C++ function), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::writearrays</td>
<td>(C++ function), 89</td>
</tr>
<tr>
<td>arrayfile::arrayfile_t::writeheader</td>
<td>(C++ function), 91</td>
</tr>
<tr>
<td>arrayfile::arrayfilemode_t</td>
<td>(C++ enum), 88</td>
</tr>
<tr>
<td>arrayfile::ATEXT</td>
<td>(C++ enumerator), 88</td>
</tr>
<tr>
<td>arrayfile::READ</td>
<td>(C++ enumerator), 88</td>
</tr>
<tr>
<td>arrayfile::READWRITE</td>
<td>(C++ enumerator), 88</td>
</tr>
<tr>
<td>arrayfile::WRITE</td>
<td>(C++ enumerator), 88</td>
</tr>
<tr>
<td>arrayfileinfo</td>
<td>(C++ function), 66</td>
</tr>
<tr>
<td>ArrayFloat</td>
<td>(C++ type), 62</td>
</tr>
<tr>
<td>arrayInFile</td>
<td>(C++ function), 69</td>
</tr>
<tr>
<td>arrayInList</td>
<td>(C++ function), 70</td>
</tr>
<tr>
<td>arraylink2arraydata</td>
<td>(C++ function), 65</td>
</tr>
<tr>
<td>arraylink2eigen</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arraylist_t</td>
<td>(C++ type), 63</td>
</tr>
<tr>
<td>arrayrank</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arrayrankColPivQR</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arrayrankFullPivLU</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arrayrankFullPivQR</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arrayrankInfo</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arrayrankSVD</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>arraywriter_t</td>
<td>(C++ class), 87</td>
</tr>
<tr>
<td>arraywriter_t::~arraywriter_t</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::afiles</td>
<td>(C++ member), 87</td>
</tr>
<tr>
<td>arraywriter_t::arraywriter_t</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::closeafiles</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::flush</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::initArrayFiles</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::nArraysWritten</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::nwritten</td>
<td>(C++ member), 87</td>
</tr>
<tr>
<td>arraywriter_t::verbose</td>
<td>(C++ member), 87</td>
</tr>
<tr>
<td>arraywriter_t::writeArray</td>
<td>(C++ function), 87</td>
</tr>
<tr>
<td>arraywriter_t::writearrays</td>
<td>(C++ member), 87</td>
</tr>
<tr>
<td>calculateArrayParetoJ5</td>
<td>(C++ function), 61</td>
</tr>
<tr>
<td>calculateArrayParetoJ5Cache</td>
<td>(C++ function), 101</td>
</tr>
<tr>
<td>calculateArrayParetoRankFA</td>
<td>(C++ function), 60</td>
</tr>
<tr>
<td>calculateJstatistics</td>
<td>(C++ function), 102</td>
</tr>
<tr>
<td>CandidateGenerator</td>
<td>(C++ class), 97</td>
</tr>
<tr>
<td>CandidateGeneratorBase::al</td>
<td>(C++ member), 92</td>
</tr>
</tbody>
</table>
CandidateGeneratorBase::candidate_list (C++ member), 98
CandidateGeneratorBase::CandidateGenerator (C++ function), 97
CandidateGeneratorBase::candidates (C++ function), 97
CandidateGeneratorBase::ct (C++ member), 97
CandidateGeneratorBase::last_valid (C++ member), 97
CandidateGeneratorBase::showCandidates (C++ function), 97
CandidateGeneratorBase::START_COL (C++ member), 98
CandidateGeneratorBase::startColumn (C++ function), 98
CandidateGeneratorBase::verbose (C++ member), 97
CandidateGeneratorConference (C++ class), 98
CandidateGeneratorConference::CandidateGenerator (C++ function), 98
CandidateGeneratorConference::generateCandidate (C++ function), 98
CandidateGeneratorConference::generateCandidate (C++ function), 98
CandidateGeneratorDouble (C++ class), 98
CandidateGeneratorDouble::CandidateGenerator (C++ function), 98
CandidateGeneratorDouble::generateCandidate (C++ function), 98
carray_p (C++ type), 63
carray_t (C++ type), 62
CL2discrepancy (C++ function), 58
clear_LMCreduction_pool (C++ function), 116
c5clone_array (C++ function), 65
colindex_t (C++ type), 63
colperm_t (C++ type), 63
colpermset (C++ type), 114
colpermtypelight (C++ type), 114
colpermtype (C++ type), 114
colpermtypelight (C++ type), 114
compareJ54 (C++ function), 102
compareLMC (C++ function), 65
compareLMC0 (C++ function), 95
conditionNumber (C++ function), 60
conf_t (C++ type), 63
conference2DSD (C++ function), 92
conference_column (C++ type), 63
conference_column_list (C++ type), 63
conference_design2modelmatrix (C++ function), 59
CONFERENCE_DIAGONAL (C++ enumerator), 96
CONFERENCE_ISOMORPHISM (C++ enumerator), 112
CONFERENCE_NORMAL (C++ enumerator), 96
CONFERENCE_RESTRICTED_ISOMORPHISM (C++ member), 113
conference_t (C++ class), 95
conference_t::__repr__ (C++ function), 96
conference_t::conference_t (C++ function), 96
conference_t::create_root (C++ function), 96
colcol2DSD (C++ function), 96
conference_t::createDoubleConferenceRootArrays (C++ function), 96
colindex_t::conference_t::createRootArrays (C++ function), 96
colltype (C++ member), 97
conference_t::idstr (C++ function), 96
colindex_t::itype (C++ member), 97
conference_t::j1zero (C++ member), 97
conference_t::j2zero (C++ member), 97
conference_t::j3zero (C++ member), 97
conference_t::N (C++ member), 97
conference_t::ncols (C++ member), 97
colindex_t::conference_t::ncols (C++ member), 97
colindex_t::conference_t::apply (C++ function), 86
conference_t::conference_t::conference_transformation_t::apply (C++ function), 86
conference_t::conference_t::conference_transformation_t::conference_transformation_t::apply (C++ function), 85
conference_t::conference_t::conference_transformation_t::cperm (C++ member), 86
colindex_t::conference_t::conference_transformation_t::cswitch (C++ member), 86
colindex_t::conference_t::conference_transformation_t::init (C++ function), 87
conference_t::conference_t::inverse (C++ function), 86
conference_t::conference_t::isIdentity (C++ function), 86
conference_t::conference_t::ncols (C++ member), 86
conference_t::conference_t::nrows (C++ member), 86
conference_t::conference_t::operator* (C++ function), 86
conference_t::conference_t::operator== (C++ function), 86
conference_t::conference_t::randomize (C++ function), 86
conference_t::conference_t::randomizecolflips (C++ function), 86
conference_t::conference_t::randomizecolperm (C++ function), 86
conference_t::conference_t::randomizerowflips (C++ function), 86
conference_t::conference_t::randomizerowperm (C++ function), 86
conference_t::conference_t::reset (C++
function), 86
conference_transformation_t::rperm (C++ member), 86
conference_transformation_t::rswitch (C++ member), 86
conference_transformation_t::setcolperm (C++ function), 86
conference_transformation_t::setrowperm (C++ function), 86
conference_transformation_t::show (C++ function), 85
conference_type (C++ enum), 96
const_colindex_t (C++ type), 63
convert_array_file (C++ function), 68
coordinate_exchange_method_t (C++ enum), 53
COPY (C++ enumerator), 115
copy_array (C++ function), 64
counter_t (C++ class), 102
counter_t::addNfound (C++ function), 103
counter_t::addNumberFound (C++ function), 103
counter_t::clearNumberFound (C++ function), 103
counter_t::counter_t (C++ function), 103
counter_t::nArrays (C++ function), 103
counter_t::nfound (C++ member), 103
counter_t::showcounts (C++ function), 103
counter_t::showcountscompact (C++ function), 103
create_array (C++ function), 64
create_root (C++ function), 66
cvalue2Dvalue (C++ function), 61

D
DAEefficiencyWithSVD (C++ function), 56
DCALC_ALWAYS (C++ enumerator), 107
DCALC_COND (C++ enumerator), 107
dcalc_mode (C++ enum), 107
DCONFERENCE (C++ enumerator), 96
DconferenceFilter (C++ class), 98
DconferenceFilter::als (C++ member), 100
DconferenceFilter::check_indices (C++ member), 100
DconferenceFilter::DconferenceFilter (C++ function), 99
DconferenceFilter::dtable (C++ member), 100
DconferenceFilter::filter (C++ function), 99
DconferenceFilter::filterfirst (C++ member), 100
DconferenceFilter::filterJ (C++ function), 99
DconferenceFilter::filterJ2 (C++ function), 99
DconferenceFilter::filterJ2last (C++ function), 100
DconferenceFilter::filterJ3 (C++ function), 99
DconferenceFilter::filterJ3 (C++ member), 100
DconferenceFilter::filterJ3inline (C++ function), 99
DconferenceFilter::filterJ3s (C++ function), 99
DconferenceFilter::filterJlast (C++ function), 99
DconferenceFilter::filterJpartial (C++ function), 99
DconferenceFilter::filterList (C++ function), 99
DconferenceFilter::filterListJ2last (C++ function), 99
DconferenceFilter::filterListZero (C++ function), 99
DconferenceFilter::filterReason (C++ function), 99
DconferenceFilter::filtersymm (C++ member), 100
DconferenceFilter::filterSymmetry (C++ function), 99
DconferenceFilter::filterZero (C++ function), 99
DconferenceFilter::filterzero (C++ member), 100
DconferenceFilter::inline_dtable (C++ member), 100
DconferenceFilter::inline_row (C++ member), 100
DconferenceFilter::minzvalue (C++ member), 100
DconferenceFilter::ngood (C++ member), 100
DconferenceFilter::sd (C++ member), 100
DconferenceFilter::show (C++ function), 99
D efficiencies (C++ function), 56
Deficiency (C++ function), 56
depth_alg_t (C++ enum), 101
depth_algm_t (C++ function), 101
depth_extend_array (C++ function), 101
depth_extend_direct (C++ function), 101
depth_extend_hybrid (C++ function), 101
depth_extend_sub_t (C++ class), 103
depth_extend_sub_t::depth_extend_sub_t (C++ function), 103
depth_extend_sub_t::info (C++ function), 103
depth_extend_sub_t::initialize (C++ function), 103
depth_extend_sub_t::lastcol (C++ member), 104
depth_extend_sub_t::lmctype (C++ member), 104
depth_extend_sub_t::n (C++ function), 103
depth_extend_sub_t::resize (C++ function), 103
depth_extend_sub_t::selectArraysXX (C++ function), 103
depth_extend_sub_t::selectArraysZ (C++ function), 103
depth_extend_sub_t::strengthcheck (C++ member), 104
depth_extend_sub_t::updateExtensionPointers (C++ function), 103
depth_extend_sub_t::valididx (C++ member), 104
depth_extend_sub_t::verbose (C++ function), 103
depth_extend_t::~depth_extend_t (C++ function), 104
depth_extensions_storage_t::dextendsubList (C++ member), 105
depth_extensions_storage_t::depthalglist (C++ member), 105
depth_extensions_storage_t::dextend_t::Dcheck (C++ member), 105
depth_extensions_storage_t::dmaxrnktotal (C++ member), 105
depth_extensions_storage_t::nlmc (C++ member), 105
depth_extensions_storage_t::nmaxrnktotal (C++ member), 105
depth_extensions_storage_t::nncols (C++ member), 105
depth_extensions_storage_t::show (C++ function), 104
depth_extensions_storage_t::writearrays (C++ member), 104
depth_path_t::depth_path_t (C++ function), 102
depth_path_t::depth_path_t::init (C++ function), 102
depth_path_t::lastcol (C++ member), 105
depth_path_t::maxArrayCheck (C++ function), 104
depth_path_t::maxarraysmax (C++ member), 105
depth_path_t::maxnarrays (C++ function), 104
depth_path_t::oaextend (C++ member), 104
depth_path_t::searchpath (C++ member), 105
depth_path_t::setNarraysMax (C++ function), 104
depth_path_t::setposition (C++ function), 104
depth_path_t::setpositionGEC (C++ function), 104
depth_path_t::showprogress (C++ function), 104
depth_path_t::showsearchpath (C++ function), 104
depth_path_t::t0 (C++ member), 105
depth_path_t::tp (C++ member), 105
depth_path_t::verbose (C++ member), 104
DEPTH_EXTENSIONS (C++ enumerator), 101
depth_extensions_storage_t::columnextensionsList (C++ member), 105
depth_extensions_storage_t::depthalglist (C++ member), 105
depth_extensions_storage_t::dextendsubList (C++ member), 105
depth_extensions_storage_t::goodarrayslist (C++ member), 105
depth_extensions_storage_t::narraysmax (C++ member), 105
depth_extensions_storage_t::nmaxrnktotal (C++ member), 105
depth_extensions_storage_t::set (C++ function), 105
depth_extensions_storage_t::validate (C++ function), 102
destroy_array (C++ function), 64
dextend_t::ad (C++ member), 105
dextend_t::arraywriter (C++ member), 105
dextend_t::counter (C++ member), 105
dextend_t::dextend_t::Dcheck (C++ member), 105
dextend_t::Dcheck (C++ member), 105
dextend_t::Dcheck (C++ member), 105
dextend_t::DefficiencyFilter (C++ function), 111
dextend_t::DefficiencyFilter (C++ function), 111
dextend_t::Dcheck (C++ member), 111
dextend_t::Deff (C++ member), 111
dextend_t::deficiencyFilter (C++ function), 111
dextend_t::dextend_t (C++ function), 111
dextend_t::directcheck (C++ member), 111
dextend_t::DmaxDiscard (C++ member), 111
dextend_t::filter (C++ member), 111
dextend_t::filterDiscard (C++ function), 111
dextend_t::filterDiscard (C++ function), 111
dextend_t::filtermode (C++ member), 111
dextend_t::lastcol (C++ member), 111
dextend_t::lmctype (C++ member), 111
dextend_t::n (C++ member), 111
dextend_t::nlmc (C++ member), 111
dextend_t::nmaxrnktotal (C++ member), 111

MODEL_CONSTANT (C++ enumerator), 56
MODEL_INTERACTION (C++ enumerator), 56
MODEL_INVALID (C++ enumerator), 56
MODEL_MAIN (C++ enumerator), 56
model_matrix_t (C++ enum), 56
MODEL_SECONDOORDER (C++ enumerator), 56
MPI_ARRAY_T (C macro), 62
myprintf (C macro), 123

N
nArrays (C++ function), 66
nauty (C++ type), 113
nauty::reduceNauty (C++ function), 113
NONE (C++ enumerator), 109
NRAUTO (C++ member), 70
numberModelParams (C++ function), 69

O
oa_file_exists (C++ function), 64
OA_ISOMORPHISM (C++ enumerator), 112
OA_MODE (C++ enum), 115
OA_REDUCE (C++ enumerator), 115
OA_REDUCE_PARTIAL (C++ enumerator), 115
OA_TEST (C++ enumerator), 115
OAextend (C++ class), 109
OAextend::__repr__(C++ function), 110
OAextend::algmode (C++ member), 111
OAextend::check_maximal (C++ member), 110
OAextend::checkarrays (C++ member), 110
OAextend::extendarraymode (C++ member), 110
OAextend::getAlgorithm (C++ function), 109
OAextend::getAlgorithmName (C++ function), 109
OAextend::getPreferredAlgorithm (C++ function), 110
OAextend::info (C++ function), 110
OAextend::init_column_previous (C++ member), 110
OAextend::j5structure (C++ member), 110
OAextend::nLMC (C++ member), 110
OAextend::OAextend (C++ function), 109
OAextend::setAlgorithm (C++ function), 109
OAextend::setAlgorithmAuto (C++ function), 109
OAextend::singleExtendTime (C++ member), 110
OAextend::storefile (C++ member), 110
OAextend::updateArraydata (C++ function), 110
OAextend::use_row_symmetry (C++ member), 110
oagraph2transformation (C++ function), 113
operator> (C++ function), 116
operator<< (C++ function), 116
optimDef (C++ function), 80
ORDER_J45_GREATER (C macro), 113
ORDER_J45_SMALLER (C macro), 113
ORDER_J5 (C++ enumerator), 63
ORDER_J5_GREATER (C macro), 113
ORDER_J5_SMALLER (C macro), 113
ORDER_LEX (C++ enumerator), 63
ordering_t (C++ enum), 63

P
Pareo (C++ class), 124
Pareo::__repr__(C++ function), 124
Pareo::~Pareo (C++ function), 124
Pareo::addvalue (C++ function), 125
Pareo::allindices (C++ function), 125
Pareo::allindicesdeque (C++ function), 125
Pareo::allvalues (C++ function), 125
Pareo::elements (C++ member), 125
Pareo::number (C++ function), 124
Pareo::numberindices (C++ function), 124
Pareo::Pareto (C++ function), 124
Pareo::pElement (C++ function), 124
Pareo::pValue (C++ type), 124
Pareo::show (C++ function), 124
Pareo::showvalue (C++ function), 125
Pareo::verbose (C++ member), 125
pareto_cb (C++ type), 101
pareto_cb_cache (C++ type), 101
pareto_element (C++ class), 123
pareto_element::dominates (C++ function), 124
pareto_element::equal (C++ function), 124
pareto_element::indices (C++ member), 124
pareto_element::isdominated (C++ function), 124
pareto_element::pValue (C++ type), 124
pareto_element::value (C++ member), 124
PARETOFUNCTION_DEFAULT (C++ enumerator), 56
PARETOFUNCTION_J5 (C++ enumerator), 56
paretomenotd (C++ enum), 56
parseArrayPareto (C++ function), 61
parsePareto (C++ function), 60
PECsequence (C++ function), 57
perform_column_permutation (C++ function), 65
perform_row_permutation (C++ function), 65
PICsequence (C++ function), 57
possible_F_values (C++ function), 64
print_column (C++ function), 92
print_column_rowsort (C++ function), 118
print_eigen_matrix (C++ function), 64
print_rowsort (C++ function), 118
printfstring (C++ function), 64
processDepth (C++ function), 101
projDeff (C++ function), 57
projectionDOFvalues (C++ function), 117
projectionGWLPs (C++ function), 58

rankStructure (C++ class), 61
rankStructure::alsub (C++ member), 62
rankStructure::decomp (C++ member), 62
rankStructure::EigenDecomp (C++ type), 61
rankStructure::id (C++ member), 62
rankStructure::info (C++ function), 61
rankStructure::ks (C++ member), 62
rankStructure::ncalc (C++ member), 62
rankStructure::nsub (C++ member), 62
rankStructure::nupdate (C++ member), 62
rankStructure::Qi (C++ member), 62
rankStructure::: (C++ member), 62
rankStructure::rankdirect (C++ function), 61
rankStructure::rankStructure (C++ function), 61
rankStructure::: (C++ function), 61
rankStructure::rankxf (C++ function), 61
rankStructure::rankxfdirect (C++ function), 61
rankStructure::updateStructure (C++ function), 61
rankStructure::verbose (C++ member), 62
readarrayfile (C++ function), 66, 67
readbinheader (C++ function), 68
readConfigFile (C++ function), 64
readStatisticsFile (C++ function), 102
reduceArraysGWLP (C++ function), 117
reduceConference (C++ function), 92
reduceConferenceTransformation (C++ function), 92
reduceDOPform (C++ function), 117
reduceLMCform (C++ function), 117
reduceOAnauty (C++ function), 112
REDUCTION_CHANGED (C++ enumerator), 115
REDUCTION_INITIAL (C++ enumerator), 115
REDUCTION_STATE (C++ enum), 115
reductionDOP (C++ function), 117
release_LMCredutction_object (C++ function), 116
removeElements (C++ function), 68
rowindex_t (C++ type), 63
rowperm_t (C++ type), 63
rowpermtype_ (C++ type), 114
rowsort_value_t (C++ type), 114
rowsorter_t (C++ class), 121
rowsorter_t::number_of_rows (C++ member), 121
rowsorter_t::reset_rowsort (C++ function), 121
rowsorter_t::rowsort (C++ member), 121
rowsorter_t::rowsorter_t (C++ function), 121
rowsortlight_t (C++ type), 114
runExtendRoot (C++ function), 109

scoreD (C++ function), 54
SDSMART (C macro), 113
selectArrays (C++ function), 67
selectArraysMask (C++ function), 68
selectConferenceIsomorpismClasses (C++ function), 93
selectConferenceIsomorpismIndices (C++ function), 93
selectLMC0 (C++ function), 94
selectLMC0doubleconference (C++ function), 94
selectUniqueArrays (C++ function), 117
SETROOT (C++ enumerator), 116
showArrayList (C++ function), 66
showCandidates (C++ function), 92
sizeof_array_t (C++ function), 64
sizeof_double (C++ function), 64
sortGWLP (C++ function), 58
sortLMC0 (C++ function), 95
STOREARRAY (C++ enumerator), 109
stringify (C macro), 113
symmda (C++ class), 81
symmda::: (C++ function), 81
symmda::: (C++ function), 81
symmda::orig (C++ member), 81
symmda::rowvalue (C++ member), 81
symmda::show (C++ function), 81
symmda::symmda (C++ function), 81
symmdaData (C++ type), 114

throw_runtime_exception (C++ function), 63
transformGraph (C++ function), 112

vector2doublebinfile (C++ function), 68
VectorFloat (C++ type), 62
vectorvector2binfile (C++ function), 68
VPEfficiency (C++ function), 56
vindex_t (C++ type), 63
vstack (C++ function), 65

write_array_format (C++ function), 68
write_array_latex (C++ function), 68
writearrayfile (C++ function), 67
writebinheader (C++ function), 68
writeStatisticsFile (C++ function), 102