AutoMR使用说明文档(第二版)

编者：邱祺

email：1292396438@qq.com

简介： AutoMR是一个基于粒子群算法（PSO）自动求解蜕变关系（MR）的科研项目，当前读者看到的版本为修改版，原版和当前版本的git地址如下：

- 原版git地址：https://github.com/bolzzzz/AutoMR
- 当前版本的git地址：https://github.com/quiqi/AutoMR_le

原论文地址如下：

- 原论文地址： Automatic Discovery and Cleansing of Numerical Metamorphic Relations | IEEE Conference Publication | IEEE Xplore

相较于原版，该版本有如下不同：

1. bug: setting.py 中 map_index_func的键值错误（有两个值为15的键值）
2. bug: setting.py 中 更详细的配置了每个函数的定义域，见get_input_range函数的修改
3. bug: Phase3_RemoveRedundancy.py 中，427行缩进错误

读者可以在对应文档中搜索字符串 "QUQI"，可直接定位到被修改的部分。如果读者希望使用原版AutoMR，同样建议修改这三处代码。

添加了：

1. 脚本 individual_laboratory.py，对每个函数的每个参数进行独立的实验。
2. 脚本 get_input_and_output.py，用于生成随机的输入输出数据。
3. 脚本 npz2md.py，用于将npz文件提取成json文件和markdown文件。
4. 数据 Associated_research_paper/OurResults。
5. 文档 AutoMR使用说明文档(第二版)，更详细的介绍了AutoMR的使用方法。

该文档的目的是希望帮助读者快速的熟悉该项目，并可以基于该项目完成一些简单的实验。该项目逻辑清晰，依次完成本文设置的6个任务，相信读者可以很快的上手这个项目！

本文结构如下：

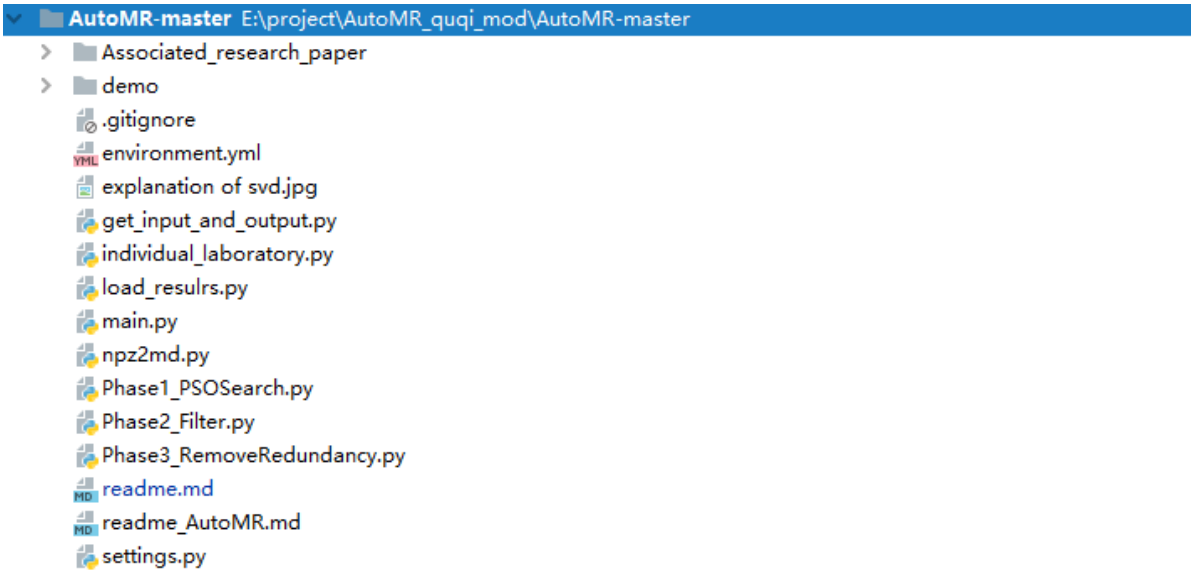
1. **项目结构：** 介绍项目中文件和文件夹的含义和用途。
2. **环境配置：** 介绍如何让项目跑起来。
3. **阶梯任务：** 依次完成这些任务即可上手该项目
    1. **运行 demo.py：**判断环境配置是否成功
    2. **理解demo.py的输出**
    3. **介绍setting变量**
    4. **求解特定函数的蜕变关系**
    5. **对单个函数进行独立重复实验**
    6. **将npz文件读取为markdown**

希望该文档可以帮到您，如果有任何疑问，欢迎联系编者，联系方式在文档开头。

AutoMR使用说明文档(第二版)

一、项目结构

该部分将会对项目中重要的文件按[图1]顺序逐一进行介绍，读者可以将之当成一个字典，也就是说现在可以粗略浏览一遍，有个印象即可，待以后任务中有需要，或可再来查阅，项目中比较重要的文件(夹)已用黑体标出。



[图1]：项目结构<\center>

除去一些控制文件，比较重要的文件和文件夹如下：

文件(夹)名	文件(夹)介绍
Associated_research_paper	实验结果文件夹，包含原论文给出的实验结果(PaperResults)[注释1]、编者自己跑出来的实验结果(OurResults)[注释2]、原论文(Automatic……pdf)三个部分。
demo	一个文件夹，包含一个运行样例。如果展开该目录，我们会发现这其实是一个根目录的副本，用于读者测试，实验，或者自己在原有代码上修改实现新的功能。
doc	项目文档，包括论文中提到的函数的具体介绍以及本文档的md 版本和pdf版本。
environment.yml	用conda快速配置环境时需要使用的配置文件，但在接下来的说明中，我们将避免使用Anconda，故不重要。（因为Anconda软件有10个G的大小，也非常难以配置，所以我们在第二节中直接使用pip来配置环境。）
*get_input_and_output.py	与本项目没有直接关联，用于随机生成有关联的输入输出数据。
***individual_laboratory.py**	对每个函数、在每个不同的参数配置下独立重复的求解MR关系。
load_resulrs.py	用于加载保存到磁盘的保存为npz格式的实验结果，当我们运行demo.py时看到命令行的一些输出就是从该文件中的代码打印出来的。
main.py	用于测试的代码文件，该文件可以帮助用户理解该项目的调用和使用方式。
***npz2md**	一个将npz提取为markdown的脚本。
Phase1_PSOSearch.py	python代码文件，第一阶段程序，功能为使用粒子群算法（PSO）搜索可能的蜕变关系，一般不需要被改动。
Phase2_Filter.py	python代码文件，第二阶段程序，用于过滤掉不正确的蜕变关系，一般不需要被改动，亦没有需要调整的参数，且对程序运行时间影响不大。

文件(夹)名	文件(夹)介绍
Phase3_RemoveRedundancy.py	python代码文件，第三阶段程序，用于消除线性相关的蜕变关系[注释3]。在第三阶段，需要使用奇异值分解去除这些多余的蜕变关系，一般不需要被改动。
readme.md	描述当前版本与原版本的不同之处。
readme_AutoMR.md	原项目介绍文件，**建议优先阅读**，其中介绍了AutoMR的原理、用途，以及环境配置方式。
settings.py	其他所有py文件的参数来源，是项目作者为我们留下的输入接口，在接下来的实验中，我们将主要修改此处的代码。

> - **[注释0]** 带星号(*)表示该文件(夹)在原项目中本来没有，为编者自行添加。
> - **[注释1]** 原论文声称在8个Apache的函数和29个NumPy的函数进行了实验，但在此处，只包含8个Apache的函数和29个NumPy的函数的运行结果，而且由于种种问题，该文件夹中的npz文件中包含的一些错误的蜕变关系。
> - **[注释2]** 编者在修改完项目中的一些bug之后重新在4.arcsin、5.arcsinh、12.exp、14.hppot、15.log、21.sin六个函数上进行了实验，实验结果可见文件夹中的"np25_MR.pdf" 文件。
> - **[注释3]** 线性相关的蜕变关系是指某组蜕变关系中的某条蜕变关系可以由组类的其他蜕变关系线性表示，如：
>
> $$MR1: sin(x) + sin(-x) = 0$$
> $$MR2: sin(x) - sin(2\pi + x) = 0$$
> $$MR3: sin(-x) + sin(2\pi + x) = 0$$
>
> 中，MR3可以由MR1和MR2推导出来，也就是说MR3是多余的。

二、环境配置：

在**readme_AutoMR.md**中有详细的文件配置信息,但由于需要使用Anconda，且该软件难以安装，或可以使用python自带的pip进行环境配置。

> 有anconda的读者可以直接使用readme.md中的配置方式，并跳过 2.1、2.2 节，直接完成2.3节的环境测试即可。

2.1 pip安装与常见问题：

pip是Python最常用的包下载工具，一般Python安装时都会自动安装pip，读者可以通过在cmd中输入如下指令判断pip是否已经被安装：

```
python -m pip --version
```

```
终端:  本地
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS F:\项目\MR项目\AutoMR-master\AutoMR-master> python -m pip --version
pip 22.0.3 from C:\Users\QuQi\AppData\Local\Programs\Python\Python38\lib\site-packages\pip (python 3.8)
PS F:\项目\MR项目\AutoMR-master\AutoMR-master>
```

如图表示安装完成。

若未有安装pip，请找到附件中的get-pip.py，并在对应目录下用如下语句运行：

```
python get-pip.py
```

在接下来的环境配置中，如果出现下载慢，或者报错中出现TimeOutError的错误，请关闭电脑中的vpn软件重试，如果仍然没用，请对pip换源，参考如下博客：windows下Python更换国内pip源

2.2 安装依赖包：

在readme.md中可以得知，AutoMR项目中依赖的包如下：

0. Environemt settings

Tested on Windows 10 1809, Ubuntu 18.04, macOS Mojave with the following packages:

- python 3.6.3
- numpy 1.14.5
- pandas 0.23.1
- sympy 1.1.1
- scipy 1.1.0
- z3-solver 4.8.5.0
- scikit-learn 0.19.2

依次在命令行中运行如下命令安装依赖：

```
pip install numpy
pip install pandas
pip install sympy
pip install scipy
pip install z3-solver
pip install scikit-learn
```

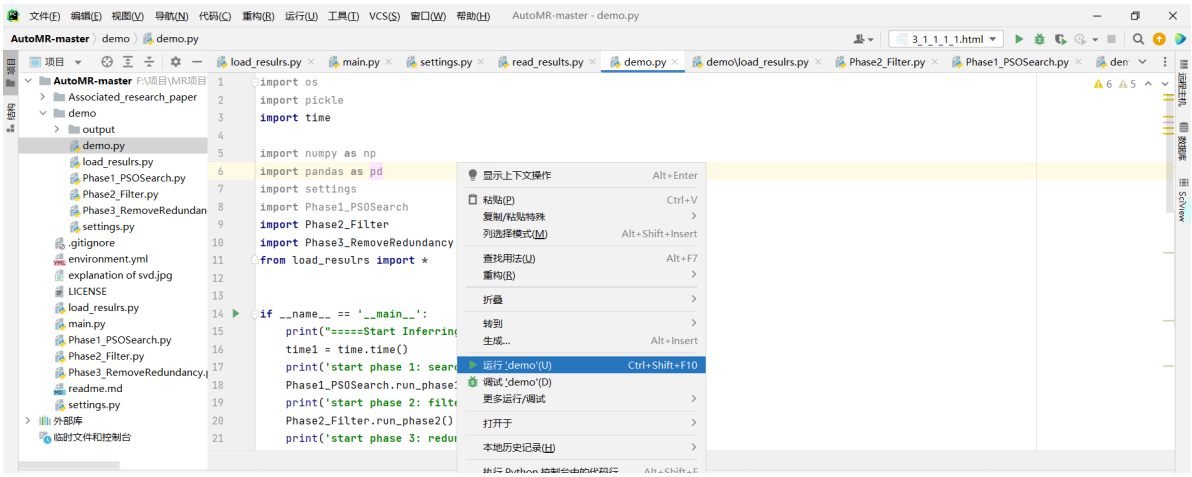
注意： 如果在安装scipy包时出现问题，请参考如下博客：如何用pip安装scipy

一个包安装好之后的命令行如下图所示（以 pip install z3-solver为例）：

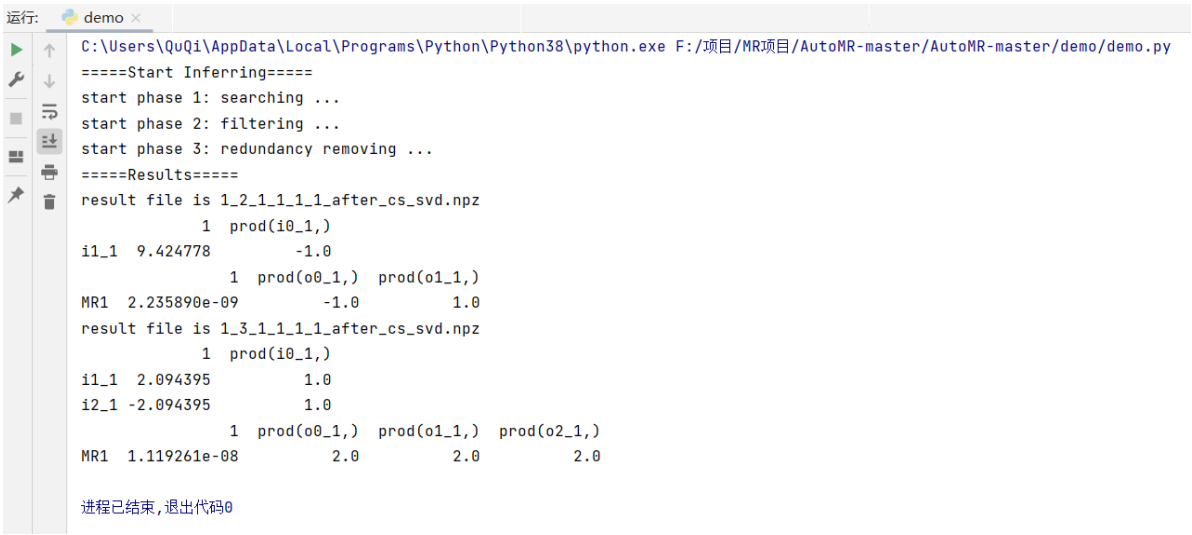
```
PS F:\项目\背包问题\pop_algorithm> pip install z3-solver
Collecting z3-solver
  Downloading z3_solver-4.8.16.0-py2.py3-none-win_amd64.whl (57.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.0/57.0 MB 8.6 MB/s eta 0:00:00
Installing collected packages: z3-solver
Successfully installed z3-solver-4.8.16.0
```

2.3 测试依赖是否安装成功

打开demo/demo.py右键点击运行：



时间可能会比较久，一般约170秒，若运行结果类似下图，表示环境配置成功：



```
C:\Users\QuQi\AppData\Local\Programs\Python\Python38\python.exe F:/项目/MR项目/AutoMR-master/AutoMR-master/demo/demo.py
=====Start Inferring=====
start phase 1: searching ...
start phase 2: filtering ...
start phase 3: redundancy removing ...
=====Results=====
result file is 1_2_1_1_1_1_after_cs_svd.npz
             1  prod(i0_1,)
i1_1  9.424778         -1.0
                1  prod(o0_1,)  prod(o1_1,)
MR1  2.235890e-09         -1.0          1.0
result file is 1_3_1_1_1_1_after_cs_svd.npz
             1  prod(i0_1,)
i1_1  2.094395          1.0
i2_1 -2.094395          1.0
                1  prod(o0_1,)  prod(o1_1,)  prod(o2_1,)
MR1  1.119261e-08          2.0          2.0          2.0

进程已结束,退出代码0
```

出现类似上图输出，表示环境配置成功。

三、阶梯任务：

在该部分中，我们需要利用第二节中配置到的环境，在demo文件夹中进行一些实践，读者之后需要完成的任务正是这些实践，或者这些实践的组合。所以，如果读者可以顺利完成如下实践任务，就约等于掌握了当前项目，当然，如果需要更进一步的理解AutoMR项目，需要读者阅读论文并更仔细地阅读项目代码。

3.1 运行demo.py

任务同 2.3，在此解释一下demo.py 中的15~29行：

```python
if __name__ == '__main__':
    print("=====Start Inferring=====")
    print('start phase 1: searching ...')
    Phase1_PSOSearch.run_phase1()    # 运行第一阶段
    print('start phase 2: filtering ...')
    Phase2_Filter.run_phase2()       # 运行第二阶段
    print('start phase 3: redundancy removing ...')
```

```
    Phase3_RemoveRedundancy.run_phase3()     # 运行第三阶段(见下文注释1)
    print("=====Results=====")
    MRs_path = settings.output_path
    func_indices = settings.func_indices
    # 得到运行第三阶段后保存的所有npz文件
    result_files = os.listdir(f"{MRs_path}/phase3")
    for result_file in result_files:
        print(f'result file is {result_file}')
        # 获得文件名中的第一个数字，这个数字表示这是哪个函数的蜕变关系
        func_index = int(result_file[0:result_file.find('_')])
        # 将这个文件加载到内存，这个返回值MRs就是当前文件中保留的蜕变关系（见注释2）
        MRs = load_phase3_results(f"{MRs_path}/phase3/{result_file}")
        # 剩下的部分是将MRs写成一个html文件，用于在网页中显示（见注释3）
```

[注释1] 运行第 i 阶段会将第 i 阶段运行得到的结果自动存储到指定路径中，该保存路径由setting.py中的output_path变量给出。

[注释2] load_phase3_results 是 load_resulrs.py 中的一个函数，用来加载被 Phase3_RemoveRedundancy.run_phase3() 保存在磁盘上的结果；另外，下图这些信息都是在这个函数中被打印出来的：

```
=====Results=====
result file is 1_2_1_1_1_1_after_cs_svd.npz
              1  prod(i0_1,)
i1_1  9.424778         -1.0
                1  prod(o0_1,)  prod(o1_1,)
MR1  2.235890e-09         -1.0          1.0
```

[注释3] 这个html的显示如下，我们可以从这段生成html的代码中知道如何解析从磁盘读到的MRs。

func index is 1

Mode of Input Relation is "="

Mode of Output Relation is "="

Input Relation is:

	1	prod(i0_1)
i1_1	9.42	-1.0

Output Relation is:

	1	prod(o0_1)	prod(o1_1)
MR1	0.0	-1.0	1.0

3.2 读懂demo.py的输出

在 3.1 中，我们得到了如下的输出，读懂这些输出非常重要，当前任务不需要编程，重点是理解这些输出。

```
=====Results=====
result file is 1_2_1_1_1_1_after_cs_svd.npz
                1  prod(i0_1,)
i1_1  9.424778         -1.0
                1  prod(o0_1,)  prod(o1_1,)
MR1  2.235890e-09         -1.0          1.0
```

如图是demo.py的一次输出，求解的是numpy.sin(x)函数的蜕变关系，从Results往下：

1. **第一行**表示当前打印出的蜕变关系保存在1_2_1_1_1_1_after_cs_svd.npz中。

   > 该文件名的前6个数字表示该文件中保存的MR对应的参数设置信息。
   >
   > 1. 文件名中第一个 “1” 表示这是第一个函数的蜕变关系（ 也就是sin(x) ）
   > 2. 文件名中的第二个 “2” 表示是两个输入输出之间的关系（在之后会更具体的说明）
   > 3. 文件名中的第三个和第四个 “1”：用于控制产生的MR是否为相等关系。
   > 4. 文件名中的最后两个 “1” 表示需要拟合的阶数，阶数越多，求解花费的代价越大。

2. **第二行**和**第三行**是一个表格，表格的表头是 **[1, prod(i0_1)]**，第一行数据为 **[i1_1, 9.424778, -1.0]**，如下表：

	1	**prod(i0_1)**
i1_1	9.424778	-1.0

   其表达的含义为 **i1_1 = 9.424778 + (-1.0)*i0_1**，这里的i0_1表示0号输入，i1_1表示一号输入，由于文件名中的第二个数字为 “2”，故有 第0次 和 第1次 两次输入。

3. **第三行** 和 **第四行** 也是一个表格：

	1	**prod(o0_1,)**	**prod(o1_1,)**
MR1	0	-1.0	1.0

   > 2.235890e-09 就是 $2.235890*10^{-9}$ 可视为 0；

   MR1行表达的是一个蜕变关系：**0*1 + (-1)o0_1 + 1*o1_1**，其中o0_1的含义是i0_1对应的输出，o1_1是i1_1对应的输出，以上表达式共同表示了一个如下的MR关系：

   $$i_1 = 9.424778 - i_0$$
   $$o_0 - o_1 = 0$$

   也就是：$sin(x) - sin(3\pi - x) = 0$

   > 9.424778约等于3π

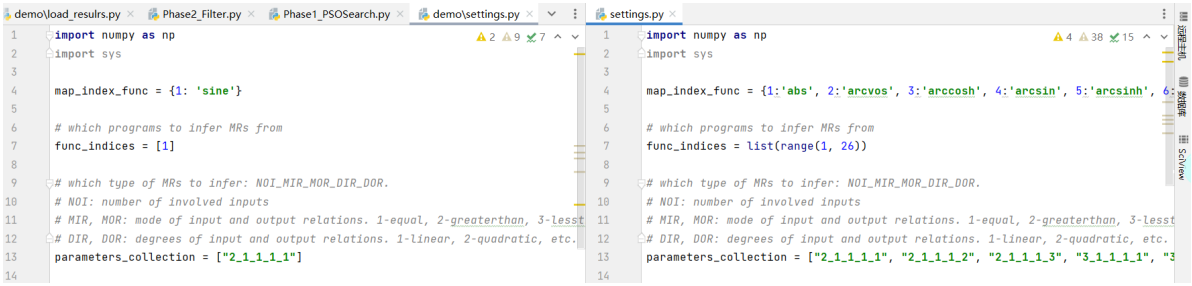
任务：基于以上的解释，请分析下面输出表示的是一个怎样的蜕变关系：

```
result file is 1_3_1_1_1_1_after_cs_svd.npz
             1  prod(i0_1,)
i1_1 -8.377574         -1.0
i2_1 -8.377565          1.0
             1  prod(o0_1,)  prod(o1_1,)  prod(o2_1,)
MR1  0.000023         -2.0          2.0         -2.0
```

> 答案：$-2 \times sin(x) + 2 \times sin(-\frac{8}{3}\pi - x) - 2 \times sin(-\frac{8}{3}\pi + x)$。经过验证，这条蜕变关系是正确的。

3.3 熟悉setting.py中的变量：

setting.py 是这个项目为使用者留下的参数输入口，项目中一共有两份setting.py，一份在根目录下，一份在demo包下。



[图2]左边为 demo/setting.py，右边为根目录下的setting.py

从两份 setting.py 的第4行可以看到，根目录下的 setting.py 内置了25个需要求蜕变关系的np中的函数，在demo/setting.py中只有一个。但两份文件的结构和内容是基本相同的。在接下来的介绍中，我们将直接以根目录下的setting.py进行介绍。

对于使用者而言，以下的一些变量和函数比较重要：

1. **map_index_func：**每个编号对应的函数名，这个字典是用于读者对照每个编号分别是什么函数用的，仅仅用于提高程序的可读性。其对程序并没有太大的意义，没有任何程序使用到这个变量，所以这个字典中甚至有个没被发现的低级bug，有两个为 15 的键值：

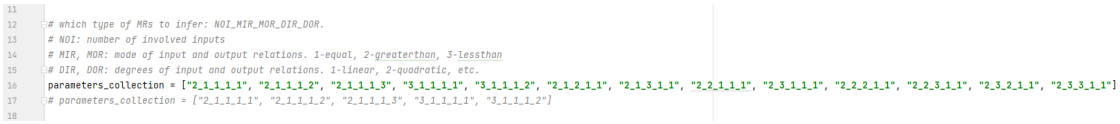
```
map_index_func = {1:'abs', 2:'arcvos', 3:'arccosh', 4:'arcsin', 5:'arcsinh', 6:'arctan', 7:'arctan2', 8:'arctanh',
                  9:'ceil', 10:'cos', 11:'cosh', 12:'exp', 13:'floor', 14:'hypot', 15:'log', 15:'log1p', 17:'log10',
                  18:'amax', 19:'amin', 20:'round', 21:'sin', 22:'sinh', 23:'sqrt', 24:'tan', 25:'tanh',
                  26: 'sin', 27: 'abs', 28:'asinh', 29:'atan', 30:'cos', 31:'log1p', 32:'log10', 33:'tan'}
```

> 上图中26~33为编者所加，为apache中的八个函数，但apache的实验是在java上完成的，所以这里并不会运行到。

2. **func_indices：** 一个列表，表示需要被寻找蜕变关系的函数，此处的 **list(range(1, 26))** 表示一个依次包含1~25的整数列表，表示map_index_func中的前25个函数都会被使用到。
3. **parameters_collection：** 一个列表，表示需要寻找的蜕变关系的模式，该列表的每一个元素都是用"_"隔开的五个数字，**这五个数字是AutoMR中最重要的五个参数**，根据代码给出的注释，对于一个参数串**NOI_MIR_MOP_DIR_DOR** ，在项目中将被解释为：
    1. **NOI：**表示生成的蜕变关系是NOI次输入之间的蜕变关系；
    2. **MIR、MOR：** MIR表示输入之间是相等关系、大于关系还是小于关系，MOR表示输出之间是相等关系、大于关系还是小于关系，他们等于1时表示相等关系，等于2时为大于关系、等于3为小于关系。从parameters_collection的取值情况可以看出，当MIR于MOR中的任何一个设

置为不相等关系时，NOI都等于2，也就是说，不相等关系只会是两次输入之间的不相等关系。

3. **DIR、DOR**：DIR表示输入关系的最高阶数，DOR表示输出关系的最高阶数，这两个参数越大，生成的蜕变关系就越多、越复杂，且将花费更多的时间和内存。



[图三] 蜕变关系参数解释

在此特别说明程序执行的过程：

1. 在第一阶段，依次取得 **func_indices** 中的每一个编号，然后对这个编号对应的函数依次求解**parameters_collection**中指定的每一种蜕变关系，等所有函数的所有类型的蜕变关系的第一阶段结束，**Phase1_PSOSearch.run_phase1()**将第一阶段所有的中间结果打包成npz文件，放入指定文件夹的phase1子文件夹中
2. 第二阶段程序将依次处理所有phase1子文件夹的npz文件，并将结果放入phase2子文件夹中。
3. 第三阶段程序将依次处理phase2子文件中的npz文件，并将处理结果放入phase3子文件夹中。

也就是说，程序是先进行所有待测函数的所有求解模式的第一阶段，然后执行第二阶段，在进行程序运行时间统计的时候需要注意到这一点。

4. **program(i, func_index)**：共程序调用的函数接口，如果你需要运行自己选定的函数，需要在此处添加之，比如要添加np.log2函数，只需添加如下代码：

```
def program(i, func_index):
    if func_index == 1:
        o = np.abs(i)
    elif func_index == 2:
        o = np.arccos(i)
    elif func_index == 3:
        o = np.arccosh(i)
    ……
    # 读者需要添加的代码start
    elif func_index == 26:
        o = np.log2(i)
    # 读者需要添加的代码end
    ……

    return 0
```

然后再修改func_indices变量：

```
func_indices = [26]
```

就可以求解 **np.log2** 中包含的蜕变关系了。

5. **getNEI(func_index)：** 返回待测函数一共有几个输入，比如 14.hypot(x,y)[注释1]就有两个输入。

```
# the number of elements of the input for the programs
def getNEI(func_index):
    if func_index in [7, 14, 18, 19]:
        return 2
    else:
        return 1
```

6. **output_path：** 输出文件的位置，AutoMR的任何一个阶段的程序都没有返回值，其数据直接以npz的格式存回磁盘，默认为根目录下：./output/np25
7. **pso_runs：** 运行几次粒子群算法（PSO），运行次数越多，找到蜕变关系的就越多，程序展示出来的水平也越稳定，默认3次。
8. **pso_iterations：** 粒子群算法（PSO）粒子群算法的迭代次数，迭代次数越多，找到蜕变关系的概率越大，花费的时间也越多。

> **[注释1]** 用于计算欧几里得范数：$hypot(x, y) = \sqrt{x^2 + y^2}$

3.4 求解特定函数的蜕变关系：

假设我们希望求解10号函数（cos）的二元相等线性蜕变关系，那么，我们需要：

1. 修改 **setting.py** 中的 **func_indices** 变量：

```
# func_indices = list(range(1, 26))
func_indices = [10]
```

> tips: 对源码的修改永远是注释即可，不要删除。

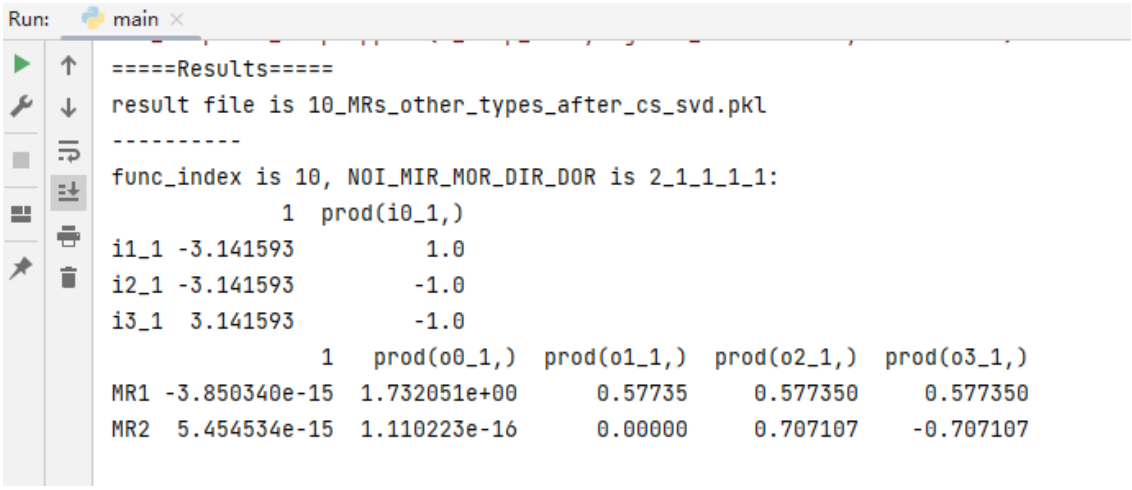
2. 然后修改**parameters_collection**：

```
# DIR, DOR: degrees of input and output relations. 1-linear, 2-quadratic, etc.
# parameters_collection = ["2_1_1_1_1", "2_1_1_1_2", "2_1_1_1_3", "3_1_1_1_1", "
parameters_collection = ["2_1_1_1_1"]
```

3. 第三步修改输出文件地址**output_path**，这个修改非常有必要，若不修改，可能会覆盖掉之前的运行结果。

```
# path to store results
# output_path = "./output/np25"
output_path = "./output/cos"
```

4. 第四步运行main.py: 运行完毕之后会发现output下出现了cos文件夹，终端会打印出这些找到了蜕变关系：



任务： 求解8号函数的三元相等蜕变关系。

3.5 对单个函数进行独立重复实验：

我们常常需要对单个函数进行独立重复实验，举例来说，我们想知道 np.sin 函数在['2_1_1_1_1', '2_1_1_1_2', '2_1_1_1_3']三组不同参数设置下的运行时间，根据学界惯例，我们需要至少运行30次，取得平均值，首先，我们不能直接将 setting.py 中的 **parameters_collection** 变量直接设定为['2_1_1_1_1', '2_1_1_1_2', '2_1_1_1_3']，这样会先依次运行这三组参数配置下的第一阶段，然后依次运行第二阶段……这样我们无法依次统计不同参数下分别的运行时间。

为了让使用者更方便的进行实验，编者写了一个用于对单个函数进行独立重复实验的脚本 **individual_laboratory.py** ，下面是该脚本的详细注释：

```
import Phase2_Filter
import Phase3_RemoveRedundancy
from load_resulrs import *
import time
import csv


if __name__ == '__main__':
    # 此脚本用于实现多次独立重复实验，可以生成npz格式的实验数据和一张表格，表格记录每个函数在每个参数下在每次独立运行时花费的时间
    repet = 5        # 重复实验次数
    pars = ["2_1_1_1_1"]     # 参数设置
    fun_ids = [21]           # 待实验函数
    save_root = 'output/{}/'.format(time.strftime("%Y_%m_%d_%Hh%Mm%Ss",
time.localtime()))  # 保存地址，默认保存到output下的 “当下时间” 文件夹
    if not os.path.isdir(save_root):
        os.makedirs(save_root)
    for fun_id in fun_ids:  # 每次跑一个函数
        with open(save_root + 'times_{}.csv'.format(fun_id), 'a', newline='',
encoding='utf-8') as f:
            cw = csv.writer(f)
            for par in pars:     # 每次跑一个参数
                settings.func_indices = [fun_id]
                settings.func_indices = [par]
                for i in range(repet):       # 每种情况运行repet次
                    save_root_times = save_root + '{}/{}/{}/'.format(fun_id,
par, i)
                    if not os.path.isdir(save_root_times):
```

```
                    os.makedirs(save_root_times)
                settings.output_path = save_root_times
                s = time.time()
                print('{}-{}-{}起始时间：{}s'.format(i,
settings.map_index_func[fun_id], par, s))
                print('start phase 1: searching ...')
                Phase1_PSOSearch.run_phase1()   # 第一阶段
                print('start phase 2: filtering ...')
                Phase2_Filter.run_phase2()      # 第二阶段
                print('start phase 3: redundancy removing ...')
                Phase3_RemoveRedundancy.run_phase3()    # 第三阶段
                e = time.time()
                print('{}-{}-{}花费时间：{}s'.format(i,
settings.map_index_func[fun_id], par, e - s))# 时间花费保存到save_root +
'times_{}.csv'表格中
                cw.writerow([settings.map_index_func[fun_id], i, par, e -
s])
```

任务： 统计 AutoMR对np.sin(x) 在参数分别设定为 ['2_1_1_1_1', '2_1_1_1_2', '2_1_1_1_3'] 时的求解蜕变关系的平均时间花费。

3.6 将npz文件读取为markdown：

AutoMR的 load_resulrs.py 脚本中包含了提取结果的方法，其中 load_phase3_results(result_path):可以读取并返回第三阶段的蜕变关系信息，该函数将返回一个迭代器，每次迭代抛出两个数据，第二个数据为蜕变关系（用两个panda表格表示），第一个数据当前蜕变关系对应的parameters_collection参数。下面这段代码或许可以更清晰的描述之：

```
MRs = load_phase3_results(npz_path)
        for k, v in MRs.items():
            print(k) # 打印参数
            # 接下来两行打印的结果如3.4中第四步的结果。
            print(v[0]) # 打印输入关系对应的panda表格
            print(v[1]) # 打印蜕变对应的panda表格
```

为了更直观的观察AutoMR求解的蜕变关系，编者写了一个名为 npz2md.py的脚本，读者使用该脚本时，只需要修改 json_name 和 markdown_name 两个参数就好了：

```
if __name__ == '__main__':
    json_name = 'np'    # 将json文件保存成np.json
    markdown_name = 'npmd'      # 将markdown文件保存成npmd.md
    npz_root = 'npz_root/phase3/'  # npz文件目录，请最后一定要以/结尾。
    npz_to_json(npz_root, '{}.json'.format(json_name))  # 将npz文件转换为json文件
    json_to_md('{}.json'.format(json_name), '{}.md'.format(markdown_name))# 将
json文件转化为md文件
```

任务： 将在3.6中得到的蜕变关系提取成 md 格式。

> 文档结束，希望对您有所帮助！如有任何问题欢迎批评，联系方式在文档顶部，感激不尽！