# rBAS 使用文档

王江宇

2018-09-04

# 目录

# 表格

# 插图

# 前言

## 手册内容概述

本手册是为了大家更好地使用 **rBAS** (**?**) 包而撰写，内容如下:

- 第 一 章介绍了如何安装 R 语言的环境，来使用 rBAS 包。不用担心，R 的语法很简单，各种功能是按照自身的需要安装各种 packages，所以比 matlab 体积更小，入门时间成本也较低。**哪怕你无意于 R 的学习，也可以看看本手册的原理篇 (第 二 章)，应用篇 (第六章) 以及后续的更新计划 (第 八 章)，来了解算法的原理，出现了哪些变种，以及有着什么样的工程应用**。
- 第 二 章介绍了 **BAS 算法**以及在其基础上出现的各种**改进算法**的**原理**，当然，随着算法的不断改进和发展，这个文档还需要随之不断更新。
- 第 四 章讲述了如何在 R 中使用 rBAS 包**调用收录的算法的对应函数**，以及一些简单的**案例**（大部分是 BAS 相关文献中的算例和 benchmark functions）。每一句出现的代码我都会尽我所能去注释，让大家了解每一步的意义，以及 R 的简单易用。我也希望，自己的语言能尽力通俗，对于其他工具的使用者来说。
- 第 六 章主要介绍的是 BAS 及变体算法**在各种领域的应用**，当然，少不了对应的案例描述和代码。**可能有些涉及到各位作者的研究，不会做到完全的开源**，但在**李帅**老师的组织下，我相信这会是最完善全面的 BAS 应用大全。
- 第 五 章介绍了 rBAS 中的用户界面的调用，以及运行。

- 第 八 章讲述了 rBAS 包的**开发和使用手册更新的计划**。因为算法总是会不断地推陈出新，所以 rBAS 包也必须和目前的研究保持一致。如果你有好的想法，可以看此章的内容，然后把自己的建议传达给我们。

  好了，冗长的章节介绍完毕。大家可以开始浏览正文了。

## 夹带私货

如果你对这本手册本身的撰写环境感兴趣的话，那我可能还要啰嗦两句。

第一句：照搬 Yihui[1] 的一句话：我用了两个 R 包编译这本书，分别是 **knitr** (?) 和 **bookdown** (?)。

第二句：感谢 Yihui。嗯...，因为这个男人，R 用户的读书笔记，文章，学位论文，个人网站等等都可以在 R 里面撰写或者开发。不得不感慨他的天才和对需求的把握。

## 致谢

感谢提倡者李帅老师，以及姜向远博士。他们是 BAS 的提出者，也在算法原理与改进上，给了我这个做暖通的门外汉以启发。

此外，还感谢李晓晓, 王甜甜, 莫小娟, 阮月同学贡献自己的算法代码和应用案例，他 (她) 们改进了算法，并且让其应用部分变得更加丰富。

当然，还得感谢 Yihui 的 bookdown。

老实讲，2018/07, 也就是一个月以前，我刚开始用 R 编写这个算法，然后用在自己的建筑系统辨识研究中，没想到 ... 所以，这个手册是比较仓促的产物，再加之自身关于优化算法理论水平较低，如果大家发现了本手册的各种问题，欢迎在 QQ 群 (437958608) 内留言，或者是在 rBAS 的 github 上提出 issues[2]。

总之，谢谢上述老师及同学，也谢谢未来给我提供问题或建议的同学，你们的帮助，让手册更加完善。

[1] http://yihui.name/

[2] https://github.com/jywang2016/rBAS/issues

王江宇
2018/08/18
华中科技大学

# 作者简介

- 包作者
  - 王江宇：BSAS 算法，创建维护 rBAS 包。Github[3]
  - 李帅：提出 BAS 以及 BAS-WPT 算法。个人主页[4] & 谷歌学术[5]
  - 姜向远：提出 BAS 以及 BAS-WPT 算法。
- 贡献者
  - 李晓晓：二阶 BAS
  - 王甜甜：天牛群体优化算法 BSO
  - 阮月：Binary-BAS
  - 莫小娟：多杆机构优化问题

[3] https://github.com/jywang2016/rBAS

[4] http://www4.comp.polyu.edu.hk/~cssli/

[5] http://scholar.google.com/citations?hl=zh-CN&user=H8UOWqoAAAAJ

# 第一章　R 以及 rBAS 安装

R 以及其集成开发环境 (IDE)Rstudio 加起来都不到 200M，所以大家放心下载安装，不会是需要 10+G 的庞然大物。当然，matlab 也是很好的科学计算软件，这里仅仅是说明安装的大小。

总体来说，R 的安装十分简单，类似于把大象装进冰箱只需要三步。

## 1.1　R 安装

Step1: 进入 R 的网站 https://www.r-project.org/ ，然后点击左上角 Download 底下的 CRAN。

Step2: 选择并点击 China 底下的镜像网址，方便下载。然后点击 Download R for windows，出现的界面右上角有 install R for the first time，点击即可下载。

Step3: 安装，不需要各种复杂的配置，按照给定提示操作即可。

但是，打开 R，你会发现是如图1.1这样过于简洁的界面。

这并不符合新手的操作和开发习惯。因此，你可能需要一个集成开发环境，最好是有函数、变量的提示，方便浏览代码和结果等等优势的软件。那么，我想你说的应该是 Rstudio。

## 1.2　Rstudio 安装

Step1: 进入下载页面 https://www.rstudio.com/products/rstudio/download/ 。

Step2: 选择 free 版本的下载。

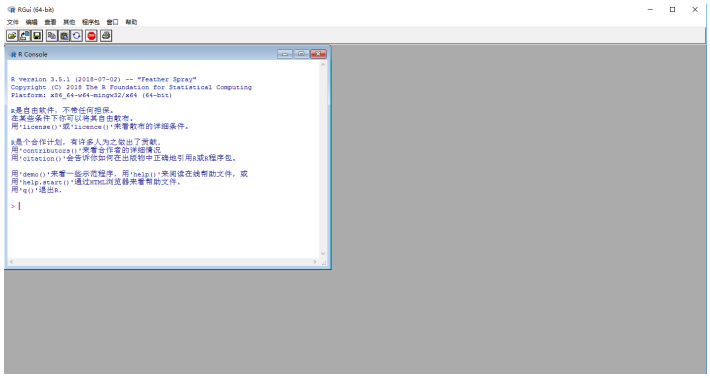

图 1.1: R 界面

Step3: 安装，无需配置特别的环境变量等。

那么，打开 Rstudio 后，会看到如图1.2这样的界面。

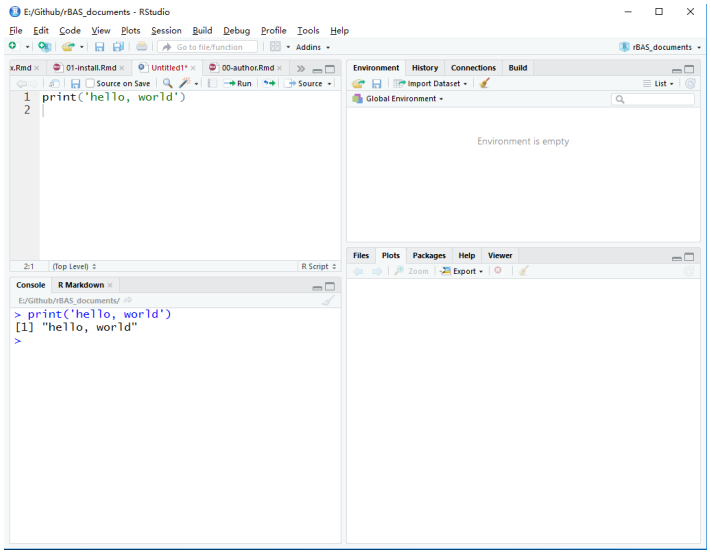


图 1.2: Rstudio 界面

左上角是撰写代码脚本的区域，左下角是结果输出的窗口。右下角的 files 可以查看工作路径下的文件，和 matlab 左侧的栏目是类似的；plots 用于查看使用代码绘制的图像，packages 可以用于安装 CRAN 上发布，或者是本地的 packages，也就类似 matlab 的 toolbox；help 则是用来显示各个函数的帮助文档；Viewer 则是用来预览 R 生成的交互图像 (比如 plotly 绘制的图)，生成的网页 (比如我现在正在使用

bookdown 包来写本手册，那就可以预览生成的 gitbook 电子书的内容) 等等。右上角的 Environment 显示被加载进来的函数，变量等信息，和 matlab 的 workspace 是类似的。剩下的和本手册无关，可以在后面的开发中慢慢了解。

## 1.3 rBAS 安装

在 Rstudio 的 Console 框内输入:

```
install.packages('devtools')
```

因为目前 rBAS 包还不在 CRAN 内，所以需要通过 devtools 包，来从 github 上安装。所以我们先在本地安装 devtools 包。如果觉得代码敲的累，那么有个更直观的方式，如图1.3:

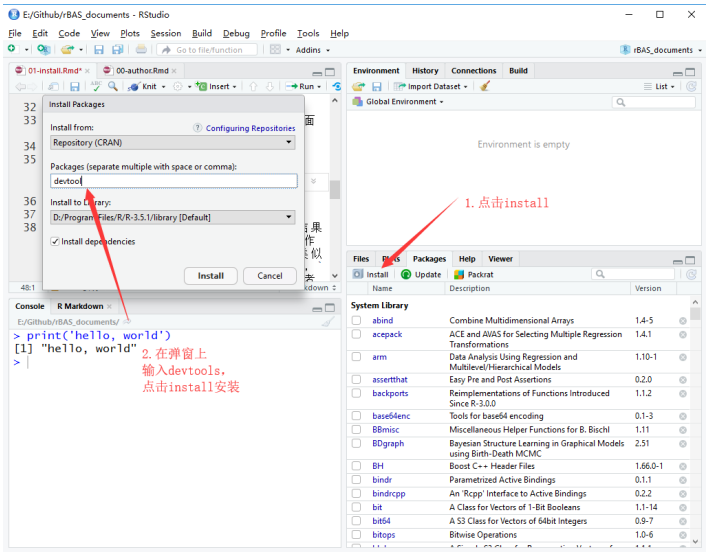


图 1.3: devtools 手动安装示意图

最后，有了 devtools 包，我们可以从 github 上安装 rBAS 包了。

```
# 不加载 devtools，只调用其中的函数
devtools::install_github("jywang2016/rBAS")
```

接下来，我们可以使用 rBAS 的函数了。

# 第二章　算法原理

本章讲述目前 rBAS 集成的三种算法，即 BAS，BSAS，BSAS-WPT 的原理。如有错漏，还请指出。此外，本章所述的算法，在原始的 BAS 基础上，并没有过多地改变根本上的东西。第三章的算法，则是 BAS 和其他一些算法的结合，又或者，有更多根本性改动。如王糖糖同学提出的 BSO，很典型地 BAS 和 PSO 的结合，则不在此章节。

## 2.1　BAS

关于 BAS，主要的参考资料为姜向远博士和李帅老师在 arXiv 上的论文，BAS: beetle antennae search algorithm for optimization problems[1]。而我是在知乎上看到一篇文章[2]后，才开始复现 BAS 算法。

### 2.1.1　算法流程

1. 随机生成方向向量，标准化

$$\overrightarrow{\mathbf{b}} = \frac{\mathrm{rnd}(n,1)}{\|\mathrm{rnd}(n,1)\|} \tag{2.1}$$

其中，$n$ 是待优化参数的维度。

2. 计算左右须的坐标

$$\begin{aligned} \mathbf{x}_r &= \mathbf{x}^t + d^t \overrightarrow{\mathbf{b}} \\ \mathbf{x}_l &= \mathbf{x}^t - d^t \overrightarrow{\mathbf{b}} \end{aligned} \tag{2.2}$$

[1] https://arxiv.org/abs/1710.10724

[2] https://zhuanlan.zhihu.com/p/30742461

其中，$\mathbf{x}^t$ 为 $t$ 时刻天牛的位置，$d^t$ 则是 $t$ 时刻，质心到须的距离。

3. 根据两须对应函数值，决定天牛下一时刻移动位置

$$\mathbf{x}^t = \mathbf{x}^{t-1} + \delta^t \overrightarrow{\mathbf{b}} \operatorname{sign}(f(\mathbf{x}_r) - f(\mathbf{x}_l)) \tag{2.3}$$

其中，$\delta^t$ 为 t 时刻的步长,$f$ 为待优化目标函数。

4. 步长与搜索距离更新

$$d^t = \eta_d d^{t-1} + d_0 \tag{2.4}$$

$$\delta^t = \eta_\delta \delta^{t-1} \tag{2.5}$$

其中，$d_0$ 是人为设定的距离的常数，$\eta_d$ 与 $\eta_\delta$ 分别是搜索距离和步长的更新衰减系数。

为了避免参数过多，姜向远博士在 BAS-WPT 算法中是按照式(2.6)来更新搜索距离和步长的。其中，$c_2$ 是人为设定的常数。

$$\begin{aligned} \delta^t &= \eta_\delta \delta^{t-1} \\ d^t &= \frac{\delta^t}{c_2} \end{aligned} \tag{2.6}$$

### 2.1.2 不足与改进

在对 BAS 算法的复现与案例应用中，我个人认为，其可能存在如下的缺点。

- 步长更新策略（反馈）
  - 缺点：无论每一步得到的结果是否变得更优，步长总会衰减；
  - 改进：带有反馈的步长更新，在无法找到更优的位置时，才进行步长的更新；
  - 关键：反馈
- 初始步长选取（参数标准化）

  - 缺点：对于多参数且量纲相差较大的问题，步长 $\delta$ 的初始值并不好选取；
  - 改进：标准化参数后，再进行调节，这也是 BAS-WPT 的技巧所在；
  - 关键：标准化

- 群体寻优
  - 缺点：1 只天牛在随机方向上搜索更优的位置，容易迷失；
  - 改进：多只天牛寻优，设定的回合内无法找到更优位置，再考虑步长更新；
  - 关键：群体智能

- 约束处理能力不足
  - 缺点：在约束边界上优化目标突变问题的处理上表现不佳
  - 改进：二阶 BAS
  - 关键：暂时没有能力归纳，有待学习二阶 BAS

## 2.2 BSAS

在2.1.2节中提及，BAS 可能在**步长更新**和**群体寻优**两个方面的策略上有一定的不足。因此，我比较莽撞地改出一个粗糙的算法，那就是所谓的 BSAS，即 beetle swarm antennae search。在 BSAS: Beetle Swarm Antennae Search Algorithm for Optimization Problems[3]中，我给出了更为详细的材料。至于具体和王甜甜同学的 BSO，即 beetle swarm optimization 有何不同，我需要进一步研究她的论文材料。

### 2.2.1 与 BAS 不同之处

此部分没有公式，因为和 BAS 算法核心公式思路是一致的。而图2.1与图2.2描述了一种假设的寻优场景，能比较清晰地体现 BSAS 与 BAS 之间的不同。

假定，天牛要找到图中**最蓝的点**。图2.1 中，天牛的起点在距离最优点较远处。由于位置更新只与时间有关，也就是每一步，天牛的步长都会缩

[3] https://arxiv.org/abs/1807.10470

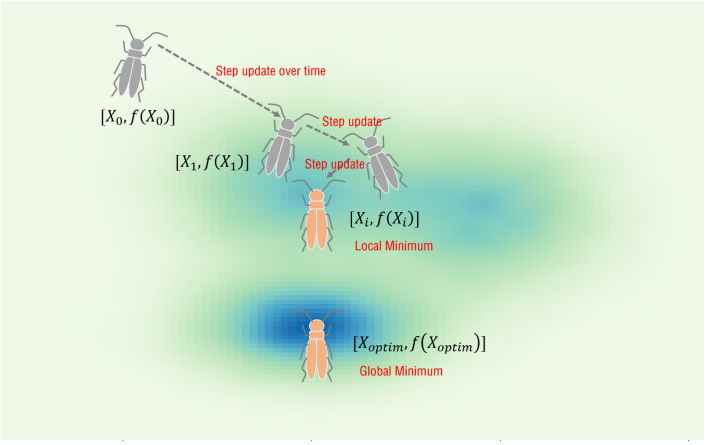


图 2.1: BAS 寻优过程示意

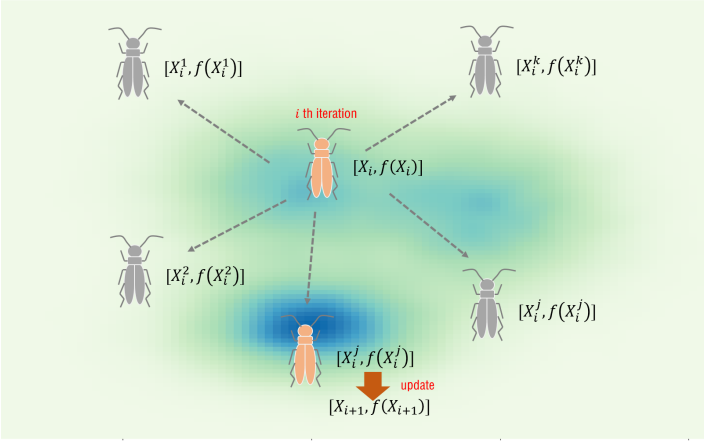


图 2.2: BSAS 寻优过程示意

减（为了可视化效果，天牛的大小我并没有缩放）。如果初始位置距离最优点较远，那在给定的步长缩减情况下，天牛只能在一个**局部最优点**处收敛。而图2.2中，每回合天牛会派出 $k$ 只天牛在外试探，如果有更优的点，那么更新天牛位置。这样天牛可以更好地到达**全局最优点**。

### 2.2.2 不足与改进

虽然解决了步长更新和群体寻优的策略问题，但是还有两点并未解决。

- 初始步长选取（参数标准化）
  - 缺点：对于多参数且量纲相差较大的问题，步长 $\delta$ 的初始值并不好选取；
  - 改进：标准化参数后，再进行调节，这也是 BAS-WPT 的技巧所在；
- 约束处理能力不足
  - 缺点：在约束边界上优化目标突变问题的处理上表现不佳
  - 改进：二阶 BAS

好的是，在 rBAS 0.1.5 中，我们吸收了 BAS-WPT 中**参数标准化**的想法，加入了 BSAS-WPT 算法，来解决步长调参的问题，并取得了一定的改进效果。

## 2.3 BAS-WPT

相比于2.1.1节中描绘的 BAS，Beetle Antennae Search without Parameter Tuning (BAS-WPT) for Multi-objective Optimization[4]一文给出了改进后的 BAS 是如何处理**步长调节**和**约束问题抽象**的。

### 2.3.1 与 BAS 不同之处

BAS-WPT 的小尾巴 without parameter tunning 已经说明了两者之间的区别，即 BAS-WPT 是不需要进行参数调节的。当然，按照我现在的理解，是 BAS-WPT 一方面简化了**每回合搜索距离** (质心到须的距离) 的**更新**，不需要再**额外设定与调节**诸如 $d_0$，$\eta_d$ 等参数，用户只需要按照式(2.6)来设置 $c_2$ 便可；另一方面，参数标准化，让**存在量级差异**的参数之间不必再像 BAS 一样，共享一个你不知道该怎么设定的步长 $\delta^t$（步长过大，小的参数可能经常处于在边界的状态；步长过小，大的参数可能搜索范围达不到）。

那么上述两方面的优势归纳起来是什么呢，那就是你可以设置一个在 1 附近 $\delta$ ，然后设定一个衰减率 $\eta_\delta$，以及步长与搜索距离之比 $c_2$，那么你的天牛就不会出太大的岔子，并且方便调整调节。也就是说，WPT 不是让你不用调参，而是减轻了调参的负担。

[4] https://arxiv.org/abs/1711.02395

> “不必就纠结归一化处理，之所以这么处理，仅仅是为了调参方便”
>
> — 姜向远

果然，偷懒催生了这一技巧的诞生。不过，我还得再次啰嗦一句标准化的好（是不是我没有接触这个领域，所以喜欢大惊小怪……）。我们在之后，压力容器约束问题 (**混合整型规划**) 中，可以看到，待优化参数存在量级差异时，标准化技巧下的步长会比原始的 `BAS` 步长设定要更加合理。

### 2.3.2 约束问题抽象形式

此外，`BAS-WPT` 还为 `BAS` 引入了约束问题处理的手段。不过，这和我做模型预测控制时候看到的抽象方式是相同的。我觉得 `BAS` 的用户们应该都早已了解，此处就照本宣科。

#### 2.3.2.1 约束问题一般形式

$$\begin{aligned} &\frac{\text{Minimize}}{\text{Maximize}} f(\mathbf{x}) \\ s.t. &g_j(\mathbf{x}) \leq 0, j = 1, \cdots, K \\ &x_i^{\max} \leq x_i \leq x_i^{\min}, i = 1, \cdots N \end{aligned} \tag{2.7}$$

$g_j(\mathbf{x}) \leq 0$ 和 $x_i^{\max} \leq x_i \leq x_i^{\min}$ 表示了参数本身的范围和更为精细具体的不等式约束控制。在 `rBAS` 包中，我们会有很**直观和简便**的方式，来设置这些约束。

#### 2.3.2.2 惩罚函数

$$F(\mathbf{x}) = f(\mathbf{x}) + \lambda \sum_{j=1}^{K} h_j(\mathbf{x}) g_j(\mathbf{x}) \tag{2.8}$$

$$h_j(\mathbf{x}) = \begin{cases} 1, & g_j(\mathbf{x}) > 0 \\ 0, & g_j(\mathbf{x}) \leq 0 \end{cases} \tag{2.9}$$

其中，式(2.8)中的 $\lambda$ 表示约束违背的惩罚因子，选取尽量大的正数。而后的 $h_j(\mathbf{x})$ 为 Heaviside 函数，即不等式约束满足时，该函数为 0，反之为 1。

### 2.3.3 不足与改进

- 约束处理能力不足
  - 缺点：在约束边界上优化目标突变问题的处理上表现不佳
  - 改进：二阶 BAS

此处的不足，还需要考虑步长反馈和群体搜索的问题。不过，既然 BSAS 把姜博的 WPT 给窃来了，摇身变为了 BSAS-WPT，那就不说上述两个问题了。等他日有闲，再去整合李晓晓同学的二阶 BAS。

# 第三章　混合算法

在第二章中说明过，在基本 BAS 算法及变体的基础上，做出了大的改动的算法在本章说明。

## 3.1　BSO

关于 BSO，主要的参考资料为王糖糖同学的论文 Beetle Swarm Optimization Algorithm:Theory and Application[1]。作为一个捣鼓了两天算法的能源人，论文对我而言很是吃力。因为我此前只听说过粒子群算法，却从未了解过其原理。在囫囵看过几篇入门粒子群算法的博文后，再阅读上述的论文，就看到了很多有意思的地方。

首先，假设大家都阅读过第二章，对 BAS 有了一定的了解。那么，接下来就能看看，BSO 是如何把 BAS 和 PSO 结合起来的。此处，只把我的认知写在此处，如有问题，还请大家指出。

### 3.1.1　粒子群算法流程

我们先说一下标准的粒子群算法的几个核心更新关系。

粒子在第 $k+1$ 回合中，其速度与第 $k$ 回合的速度关系如式(3.1):

$$v_{i,k+1} = \omega v_{i,k} + c_1 * rand_1 * (pbest_i - x_{i,k}) + c_2 * rand_2 * (gbest_k - x_{i,k}) \quad (3.1)$$

那么第 $k+1$ 回合中粒子位置如式(3.2)更新。

[1] https://arxiv.org/abs/1808.00206

$$x_{i,k} = x_{i,k} + v_{i,k} \tag{3.2}$$

其中，$i = 1, 2, \cdots, S$，$S$ 为设定的粒子数目。而 $k = 1, 2, \cdots, n$，$n$ 为设定的最大循环数。$\omega$ 表示粒子速度的惯性，也就是本回合速度对下回合速度更新的影响。$c1, c2$ 则是学习因子，$pbest_i - x_{i,k}$ 表示粒子 $i$ 当前的位置和其历史最优的位置的差距，而 $gbest_k - x_{i,k}$ 则表示，粒子 $i$ 当前的位置和历史全局最优的粒子的位置（所有粒子在历史时间跨度中最好的位置）的差距。

那么，这样更新了粒子的位置之后，自然要计算这一回合的适应度，也就是目标函数的值。如果**存在适应度比自身历史或全局历史更优**，那么就相应地更新 $pbest$ 和 $gbest$。然后，就重复上述过程，直到设定的终止条件满足。

$\omega$ 动态变化，能取得好的效果，一般，它如式(3.3)所示变化。

$$\omega = \omega_{max} - (\omega_{max} - \omega_{min})\frac{k}{n} \tag{3.3}$$

当然，如果 $k$ 取 0，也就是初始回合，那么 $\omega = \omega_{max}$。如果 $k$ 取 $n$（也就是最终的回合），那么 $\omega = \omega_{min}$。所以，这也是个权重线性递减的过程。

### 3.1.2 与 BAS 的结合

我们假想一下，要把 PSO 和 BAS 结合，从什么方向入手呢？

作为门外汉，我可能会想，那就多放置几只天牛，让它们像粒子一样，信息共享，但是本身的更新寻优还按照 BAS 的来（后面会讲，这类似于 BSAS，但不是 BSO）。

不过呢，BSO 给了一种很丰满和交错的方式，让 BAS 和 PSO 不是那么的泾渭分明。

首先，位置更新时，按照 PSO 和 BAS 的方式各自更新后加权得到新的位置。也就是，利用了 BAS 的触须方向和步长，同时也利用了粒子（在 BSO 中就是天牛）的速度。如式(3.4):

$$X_s^{k+1} = X_s^k + \lambda V_s^k + (1-\lambda)\xi_s^k \tag{3.4}$$

第一项 $X_s^k$ 自然是表明粒子 $s$ 在 $k$ 回合的位置了，第二项 $\lambda V_s^k$ 也就是 PSO 中的天牛速度乘以权重 $\lambda$。那么第三项中的 $\xi_s^k$ 应该是 BAS 中的步长乘以方向向量才对。但是，**方向并不是 BAS 中随机生成的方向**。

我们看看 $\xi_s^k$ 怎么得到。在文中，给出了式(3.5)。

$$\xi_s^{k+1} = \delta^k V_s^k sign(f(X_{rs}^k) - f(X_{ls}^k)) \tag{3.5}$$

这就是我们熟悉的 BAS 更新的方式，不过，把随机生成的天牛方向，换为了粒子速度 $V_s^k$。在 BSO 中，并没有 BAS 中成方向的操作。有同学可能会问，**既然没有随机方向，那么天牛的左右须坐标怎么来的呢？**和式(3.5)中使用**速度替代方向**的操作相同，计算左右须坐标时，也是用的速度，来代替方向。即，如式(3.6)。

$$\begin{aligned} X_{rs}^k &= X_s^k + V_s^k \frac{d}{2} \\ X_{ls}^k &= X_s^k - V_s^k \frac{d}{2} \end{aligned} \tag{3.6}$$

这样，就完成了 BSO 算法的更新过程。里面的速度，和粒子群算法的更新方式相同。

> 注明，原文中使用的上一回合的左右须坐标，更新这回合的左右须坐标。但是根据代码来看，是文章存在了脚标的小错误，应该是用天牛坐标，更新须坐标。

王糖糖同学还给出了一个动态的（BAS）步长更新系数，如式(3.7):

$$\eta = \omega_{\delta_1} (\frac{\omega_{\delta_1}}{\omega_{\delta_0}})^{1/(1+10k/n)} \tag{3.7}$$

默认参数为 $\omega_{\delta_1} = 0.4$，$\omega_{\delta_0} = 0.9$。即，步长在回合初始，衰减最快，回合临终时，衰减最慢。

### 3.1.3 总结

此前，在写 BSAS 算法说明的时候，我把 BSO 加入到了待完成列表内。看了双方的英文名，是在是太巧了。BSAS 是 `Beetle Swarm Antennae Search`，而 BSO 是 `Beetle Swarm Optimization`。彼时，我还不知晓 PSO 是啥含义。现在看来，这样的名称所包含的意义是千差万别的。后来，在复现 BSO 的过程中（此句为马后炮，毕竟是有了王糖糖同学提供的代码），也深刻地体会到两者的不同，下面我们来做一下罗列。

- 区别
  - BSAS：是 BAS 的衍生，移动和搜索的方式也很 BAS。一言蔽之，BSAS 就像天牛有了 k 对须而不是一对，在每回合，它必须要沿着这些须的方向都探索一遍。所谓的 `Swarm`，是因为每回合有 k 只（虚拟的）天牛在探索这 k 对须的方向。
  - BSO：在粒子群算法的大框架下，由速度的更新（PSO）和步长的更新（BAS）共同决定下一步的天牛位置。这其中的 `Swarm`，可以视作是真正独立的群体，它们间共享信息。
- 联系
  - BSO 和 BSAS 保留着 BAS 的搜寻和移动方式
  - BSO 和 BSAS 都是意图通过每回合搜索更多的位置（也可以视为方向），达到更优的效果。

### 3.1.4 算法调用说明

```
BSOoptim(fn, init = NULL, constr = NULL,
         lower = c(-50, -50), upper = c(50, 50), n = 300,
         s = floor(10 + 2 *sqrt(length(lower))),
         w = c(0.9, 0.4),
         w_vs = 0.4,
         step = 10,
         step_w = c(0.9, 0.4),
         c = 8,
```

```
          v = c(-5.12, 5.12),
          trace = T,
          seed = NULL,
          pen = 1e+06)
```

在本节写函数调用，是因为函数内有部分参数和本节的公式变量名称有出入。上面的代码是函数的默认调用形式，其中 $s$ 表示设定的天牛或者粒子数目，$w$ 也就是式(3.3)中提到的 $\omega_{max}$ 和 $\omega_{min}$。`w_vs` 是 $\lambda$，也就是式(3.4)中天牛速度和移动步长之间的权重，直观地理解为 `weight between velocity and step-size`。`step` 和 `c` 仍然是表示步长和步长与须距离之比。而 $step_w$ 为一个向量，表示的是式(3.7)中的 $\omega_{\delta_1} = 0.4$，$\omega_{\delta_0} = 0.9$。

一个简单的例子如下：

```
library(rBAS)
mich <- function(x){
y1 <- -sin(x[1])*(sin((x[1]^2)/pi))^20
y2 <- -sin(x[2])*(sin((2*x[2]^2)/pi))^20
return(y1+y2)
}
result <-
 BSOoptim(fn = mich,
           init = NULL,
           lower = c(-6,0),
           upper = c(-1,2),
           n = 100,
           step = 5,
           s = 10,seed = 1, trace = F)
result$par; result$value
```

```
## [1] -4.965998  1.570796
```

```
## [1] -1.967851
```

# 第四章　函数使用

首先, 加载 rBAS 包，然后在4.1节到4.3节中，我们详细讲述每个参数的含义。如果可能的话，我会加上调参时的经验（可能只对我的问题有用）。

```
library(rBAS)
```

打开网址[1]，可以看到托管在 github 上的 rBAS 文档。大家可以通过 Reference 来访问里面所有函数的帮助文档，通过 Changelog 来看每次包的更新及 bugs 修复记录。

> 文档网页是由 pkgdown[2]包制作而成，logo 由 hexSticker[3]包制作。

## 4.1　BASoptim

除了通过访问函数文档网站外，还可以在 R 中输入下面的命令，来查看文档。

```
help(BASoptim)
```

[1] https://jywang2016.github.io/rBAS/

[2] http://pkgdown.r-lib.org/

[3] https://github.com/GuangchuangYu/hexSticker

### 4.1.1 BASoptim 参数说明

BASoptim 函数[4](对应 BAS 算法) 调用的格式如下：

```
BASoptim(fn,
         init = NULL,
         lower = c(-6, 0), upper = c(-1, 2),
         constr = NULL, pen = 1e+05,
         d0 = 0.001, d1 = 3, eta_d = 0.95,
         l0 = 0,l1 = 0, eta_l = 0.95,
         step = 0.8, eta_step = 0.95,
         n = 200,steptol = 0.01,
         seed = NULL, trace = T )
```

由于英文蹩脚，所以大家看起包自带的文档会比较吃力。因此，在此处给出中文说明。

- 已知条件：目标函数与约束
  - fn 待优化的目标函数
  - init 参数初始值，默认为 NULL，即在上下限内随机选取，也可以自行指定
  - constr 不等式约束
  - lower/upper 上下限
  - pen 惩罚因子 $\lambda$
- BAS 待调参数
  - d0 参见式(2.4)中所述的搜索距离（也就是质心到须的距离）参数，一个比较小的值，默认为 0.001
  - d1 初始的搜索距离，默认为 3
  - eta_d 搜索距离的衰减系数
  - l0/l1/eta_l 这一系列关于 $l$ 的参数，来源于 **BAS** (**?**) 论文中给出的 matlab 代码。其作用在于每回合位置更新时，产生一个**随机抖动**$x = x - step * dir * sign(fn(left) - fn(right)) + l * random(npars)$

[4] https://jywang2016.github.io/rBAS/reference/BASoptim.html

  - step/eta_step 步长以及步长的衰减率
  - steptol 停止更新的步长临界值
  - n 回合数或者迭代次数

- 其他
  - seed 给定随机种子，用来固定寻优结果。不同的种子，对结果的影响**非常大**。
  - trace 是否显示寻优过程信息

### 4.1.2 BASoptim 简单案例

这里采用 **BAS** (**?**) 一文中给出的测试函数，即 `Michalewicz function` 与 `Goldstein-Price function`。

#### 4.1.2.1 Michalewicz function

$$f(x) = \sum_{i=1}^{d=2} sin(x_i)[sin(\frac{ix_i^2}{\pi})]^{20}$$

图4.1为 `Michalewicz` 函数在给定的约束范围的三维示意图。可以看到，最小值在 $x = -5, y = 1.5$ 的附近。

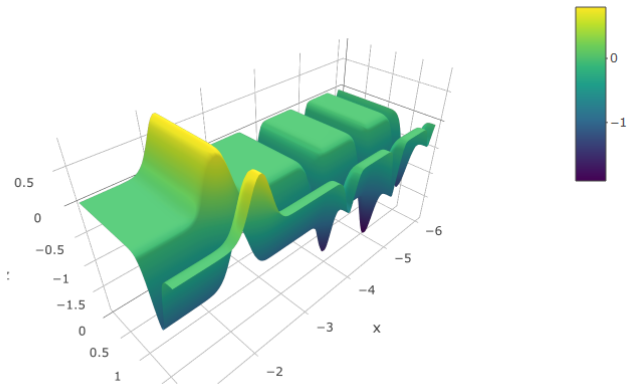


图 4.1: Michalewicz 函数示意

我们先在 `R` 的脚本中构建出函数：

```
# <- 可以视作 = 即用等于号在此处也是可以的
mich <- function(x){
  y1 <- -sin(x[1])*(sin((x[1]^2)/pi))^20
  y2 <- -sin(x[2])*(sin((2*x[2]^2)/pi))^20
  return(y1+y2)
}
```

然后利用 rBAS 包中的 BASoptim 函数求解：

```
# 把 BASoptim 的寻优结果赋值给 test
test<-
  BASoptim(fn = mich,
           lower = c(-6,0), upper = c(-1,2),
           seed = 1, n = 100,trace = FALSE)

test$par
```

```
## [1] -4.964687  1.575415
```

```
test$value
```

```
## [1] -1.966817
```

可以看到，BAS 在 100 个回合内找到了全局的最小值。非 R 用户可能对上下限的声明有点陌生，c(-6,0) 中 c()，其实是声明了一个向量，这也是 R 里面最基本的数据类型，和 matlab 里面的 [-6 0] 效果类似。整体看来，代码还是很简洁的。

#### 4.1.2.2 Goldstein-Price function

$$
\begin{aligned}
f(x) =&[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2 \\
&+6x_1x_2+3x_2^2)][30+(2x_1-3X_2)^2(18-32x_1 \\
&+12x_1^2+48x_2-36x_1x_2+27x_2^2)]
\end{aligned}
$$

图4.2为 `Goldstein-Price` 函数在给定的约束范围的三维示意图。可以看到，最小值在 $x=-5, y=1.5$ 的附近。图4.1与4.2均使用 `plotly`[5]绘制。

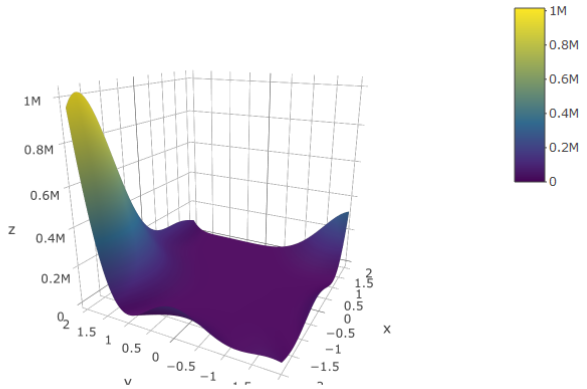


图 4.2: Michalewicz 函数示意

函数构造：

```
gold <- function(x){
  x1 <- x[1]
  x2 <- x[2]
  y1 <- 1 + (x1 + x2 + 1)^2*
    (19 - 14*x1+3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  y2 <- 30 + (2*x1 -3*x2)^2*
    (18 - 32*x1 + 12*x1^2+48*x2-36*x1*x2 + 27*x2^2)
```

[5] https://plot.ly/r/

```
  return(y1*y2)
}
```

其中，x[1] 表示向量 x 的第一个元素。举例，x = c(1,2)，那么 x[1] 等于 1，x[2] 等于 2。索引从 1 开始，并不是从 0 开始（python 和 C++ 用户可能需要在此处注意）。

优化代码：

```
test<-
  BASoptim(fn = gold,
           lower = c(-2,-2), upper = c(2,2),
           seed = NULL, n = 100,trace = F)

test$par
```

```
## [1]  0.001870855 -0.996496153
```

```
test$value
```

```
## [1] 3.004756
```

同样，结果也是给出了全局最优点（或在此附近，继续迭代下去，可能会有更精确更小的值）。

## 4.2 BSASoptim

BSASoptim 函数[6](对应 BSAS 算法)，在 BAS 的基础上，加入了步长反馈和群体策略。调用的格式如下：

[6] https://jywang2016.github.io/rBAS/reference/BSASoptim.html

```
BSASoptim(fn,
          init = NULL, constr = NULL,
          lower = c(-6, 0), upper = c(-1, 2),
          k = 5, pen = 1e+05,
          d0 = 0.001, d1 = 3, eta_d = 0.95,
          l0 = 0, l1 = 0, eta_l = 0.95,
          step = 0.8, eta_step = 0.95,steptol = 0.01,
          n = 200, seed = NULL, trace = T,
          p_min = 0.2,p_step = 0.2, n_flag = 2)
```

### 4.2.1 BSASoptim 参数说明

与 BAS 相比，BSAS 在下面几处不同参数：

- k 每回合的外出试探的天牛数目，越多结果会越稳定 (多次执行，结果更接近)，但是计算时长会相应增长。适当选取天牛数目，有助于避免随机的初始值和方向带来影响的同时，计算时长也可以接受。
- p_min 当 k 只外出的天牛存在超过 1 只找到了更优的位置，也就是比当前的最佳值要更小。那是否需要**更新到那 k 只天牛中最优的那一只所在的位置呢**？经过一些尝试，我片面地认为，未必是每次都最佳，最后的位置一定最佳。因此，给定一个概率 $p_{min}$。当有 2 只或以上的天牛找到更好的位置时，会在 [0,1] 间生成一个随机数，如果大于 $p_{min}$，那么就选 k 只天牛里**最优天牛**作为下次的更新位置牛；如果小于 $p_{min}$，那么就在找到了更好的位置的天牛里面，**随机选出**一只天牛，作为下次的更新位置。
- p_step 想法与 p_min 类同，用于**控制步长反馈策略**。在 k 只天牛找不到更优位置时，算法认为是步长过大，下一回合天牛位置不更新，且会减小步长。反之，则更新天牛位置，并保持当前步长直至不能找到更优位置。**那么，是否存在由于随机方向的原因，或者是 k 过小，导致在当前步长条件下，存在更优位置，但是找不到**。这个时候，我们设置一个更新概率 $p_{step}$，即在找不到更优的天牛位置下，步长有 $p_{step}$ 概率不更新，继续寻找。
- n_flag 为了防止设定过大的 p_step，让数次产生的随机数都小于 p_step，影响迭代的效率。我们给定了这个参数，默认为 2，只要

在同一个步长上的无效搜索 (因为找不到更优位置而反复搜索) 次数保持 3 次及以上，则会强制更新步长。

### 4.2.2 BSASoptim 取值摸索

好吧，用中文说明都这么绕口，何况是我撰写的可怜的英文文档。有同学会问了，为什么要后面那几个概率和什么次数的参数，这不是画蛇添足吗？回答是，这几个参数**来源于生活 •••**

我在做建筑阻容模型系统辨识时，每回合的寻优，都是在用龙哥库塔法求解一次常微分方程组 (ODEs)。在我的问题规模下，每回合纯粹的 R 代码要**耗费 0.25s 左右**来求解一次这样的 ODEs。也就是说，在求解目标函数上，程序耗费的时间就有 $k*n*0.25$，还不算其他的计算开销。（换言之，用遗传算法，会带来更大的计算开销，因为每回合至少计算 10* 参数个数次的目标函数）

所以，我必须要结果较好的同时，尽量减少不必要的计算。因此，k 不能太大，但是这又会在随机方向的影响下，**错失一些优化的位置**，那就需要 p_step 参数了。但是初始位置或者说中间位置附近的最优，**不代表在这附近或方向上，有全局最优**，所以我还需要 p_min 来保证，我有那么**一丝可能**，跳出**每次都找最优，可是收敛结果与全局最优背离**的怪圈。至于 n_flag，是因为我之前设置了 p_step 为 0.5，所以算法效率极低，几乎每个找不到更优的夜，这些天牛都悲伤地多做数次运行，所以我设置了这个参数。

> 还是需要强调，在我的问题里，这些参数起到了较好的效果。但是换成大家的研究，这些参数可能就是被害妄想症的产物了。有意思的是，我在默认参数下执行 50 次 Michalewicz 函数的寻优，效果并没有 BASoptim 好。但在 RC 模型辨识上，BSASoptim 远好于 BASoptim。

接下来就是这几个参数的调节的一些小技巧了。

- 设置 k 为 1，那就是带步长反馈的 BAS 了
- 如果求解目标函数速度快，可以设置较大的 k
- p_step 设置为 0，只要 k 只天牛找不到最优位置，步长就会更新；不存在不更新继续找的可能

- p_step 设置为 1，那算法会在一个步长下一直执行，直到找到更优的位置，才会更新步长
- p_min 设置为 0，在 k 只出去试探的天牛中找到了更优的位置时，那么当前时刻的天牛，总会选择这 k 只中最好的一只的位置来作为下一时刻的位置
- p_min 设置为 1，下一时刻的位置是 k 只中更优天牛的位置的随机选择
- 为了求解效率，p_step 会选择较小的值；p_min 我也没有摸清楚个规律，但是在我的研究对象中，为 0 得到的结果在多次试验中，整体看来没有为较小值 0.2 好。

上述是我在自身研究方向上摸出的规律，可能问题的类型不同，需要做的取舍也不同。大家可以保持默认参数，然后进行符合自身情况的微调。更为详细的结果可以参见 **BSAS** (**?**) 论文。

### 4.2.3 BSASoptim 案例

#### 4.2.3.1 Michalewicz function

不做过多的阐述对于此案例，可以参看4.1.2.1节。

```
library(rBAS)
mich <- function(x){
   y1 <- -sin(x[1])*(sin((x[1]^2)/pi))^20
   y2 <- -sin(x[2])*(sin((2*x[2]^2)/pi))^20
   return(y1+y2)
}
result <- BSASoptim(fn = mich,
                    lower = c(-6,0), upper = c(-1,2),
                    seed = 1, n = 100,k=5,step = 0.6,
                    trace = FALSE)
result$par
```

```
## [1] -4.970202  1.578791
```

```
result$value
```

```
## [1] -1.963534
```

#### 4.2.3.2 Pressure Vessel function

使用 **BAS-WPT**(**?**) 论文中压力容器优化函数来测试 `BSASoptim` 处理约束的能力。问题背景如下：

$$
\begin{aligned}
\text{minimize} f(\mathbf{x}) =& 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 \\
&+ 3.1661 x_1^2 x_4 + 19.84 x_1^2 x_3 \\
s.t.\ \ g_1(\mathbf{x}) =& -x1 + 0.0193 x_3 \leq 0 \\
g_2(\mathbf{x}) =& -x_2 + 0.00954 x_3 \leq 0 \\
g_3(\mathbf{x}) =& -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \\
g_4(\mathbf{x}) =& x_4 - 240 \leq 0 \\
x_1 \in& \{1, 2, 3, \cdots, 99\} \times 0.0625 \\
x_2 \in& \{1, 2, 3, \cdots, 99\} \times 0.0625 \\
x_3 \in& [10, 200] \\
x_4 \in& [10, 200]
\end{aligned}
\tag{4.1}
$$

构造一个列表，也就是 `list()`。其中包含有 2 个函数，一个是我们的目标函数 `obj`，一个是我们的不等式约束函数 `con`。为了方便起见，我并没有写每一个函数的返回值，那么，`R` 会自动返回计算的最后一个对象。比如，在 `obj` 函数中，是 `result` 变量（标量）被返回。而在 `con` 函数中，是由 `c()` 声明的向量被返回。

```
pressure_Vessel <- list(
  obj = function(x){
    x1 <- floor(x[1])*0.0625
```

```
    x2 <- floor(x[2])*0.0625
    x3 <- x[3]
    x4 <- x[4]
    result <- 0.6224*x1*x3*x4 +
      1.7781*x2*x3^2 +
      3.1611*x1^2*x4 +
      19.84*x1^2*x3
  },
  con = function(x){
    x1 <- floor(x[1])*0.0625
    x2 <- floor(x[2])*0.0625
    x3 <- x[3]
    x4 <- x[4]
    c(# 把所有的不等式约束，全部写为小于等于 0 的形式
      0.0193*x3 - x1,
      0.00954*x3 - x2,
      750.0*1728.0 - pi*x3^2*x4 - 4/3*pi*x3^3
    )
  }
)
```

使用 BSASoptim 函数进行优化。需要注意的是,pressure_Vessel 是一个列表，对于其中包含的元素，使用 $ 符号进行访问。也可以使用 [[符号，即 pressure_Vessel$obj 等价于 pressure_Vessel[[1]]。

```
result <- BSASoptim(fn = pressure_Vessel$obj,
                    k = 5,
                    lower =c( 1, 1, 10, 10),
                    upper = c(100, 100, 200, 200),
                    constr = pressure_Vessel$con,
                    n = 200,
                    step = 100,
                    d1 = 5,
```

```
                    pen = 1e6,
                    steptol = 1e-6,
                    n_flag = 2,
                    seed = 2,trace = FALSE)

result$par
```

```
## [1]  14.92195   7.87620  43.51377 159.87104
```

```
result$value
```

```
## [1] 6309.406
```

可以看到结果与论文 **BAS-WPT(?)** 中 TABLE 1 给出的优化值还是有一定的差距。不过，这也让我意识到了，对于**复杂的优化问题，调试其中的参数是个困难的活**。歧路亡羊呀！

好在，改进后的 BSAS-WPT 能够比较好地得到不逊于 **BAS-WPT(?)** 中的结果（在4.3.2节可以看到）。更多更优地结果，等待你去调参，如果你还有勇气的话。

#### 4.2.3.3 Himmelblau function

$$
\begin{aligned}
\text{minimize} f(\mathbf{x}) =& 5.3578547x_3^2 + 0.8356891x_1x_5 \\
&+ 37.29329x_1 - 40792.141 \\
s.t.\quad g_1(\mathbf{x}) =& 85.334407 + 0.0056858x_2x_5 \\
&+ 0.00026x_1x_4 - 0.0022053x_3x_5 \\
g_2(\mathbf{x}) =& 80.51249 + 0.0071317x_2x_5 \\
&+ 0.0029955x_1x_2 + 0.0021813x_3^2 \\
g_3(\mathbf{x}) =& 9.300961 + 0.0047026x_3x_5 \\
&+ 0.0012547x_1x_3 + 0.0019085x_3x_4 \\
g_1(\mathbf{x}) \in& [0, 92] \\
g_2(\mathbf{x}) \in& [90, 110] \\
g_3(\mathbf{x}) \in& [20, 25] \\
x_1 \in& [78, 102] \\
x_2 \in& [33, 45] \\
x_3 \in& [27, 45] \\
x_4 \in& [27, 45] \\
x_5 \in& [27, 45]
\end{aligned}
\tag{4.2}
$$

构造优化目标函数和约束：

```
himmelblau <- list(
  obj = function(x){
    x1 <- x[1]
    x3 <- x[3]
    x5 <- x[5]
    result <- 5.3578547*x3^2 +
      0.8356891*x1*x5 +
      37.29329*x[1] -
```

```
      40792.141
  },
  con = function(x){
    x1 <- x[1]
    x2 <- x[2]
    x3 <- x[3]
    x4 <- x[4]
    x5 <- x[5]
    g1 <- 85.334407 + 0.0056858*x2*x5 +
      0.00026*x1*x4 - 0.0022053*x3*x5
    g2 <- 80.51249 + 0.0071317*x2*x5 +
      0.0029955*x1*x2 + 0.0021813*x3^2
    g3 <- 9.300961 + 0.0047026*x3*x5 +
      0.0012547*x1*x3 + 0.0019085*x3*x4
    c(
      -g1,
      g1-92,
      90-g2,
      g2 - 110,
      20 - g3,
      g3 - 25
    )
  }
)
```

使用 BSASoptim 函数进行优化：

```
result <- BSASoptim(fn = himmelblau$obj,
                    k = 5,
                    lower =c(78,33,27,27,27),
                    upper = c(102,45,45,45,45),
                    constr = himmelblau$con,
                    n = 200,
```

```
                    step = 100,
                    d1 = 10,
                    pen = 1e6,
                    steptol = 1e-6,
                    n_flag = 2,
                    seed = 11,trace = FALSE)
result$par
```

```
## [1] 78.01565 33.00000 27.07409 45.00000 44.95878
```

```
result$value
```

```
## [1] -31024.17
```

这个结果，比 **BAS-WPT**(?) 中 TABLE 2 记载的结果都要好。但只要你愿意调差，嘿嘿，总有更好的。

## 4.3 BSAS-WPT

在进行 BSAS-WPT 参数讲解的这一部分前，我想问个问题。在式(4.1)和式(4.2)中，我们可以看到，有些 $x_i$ 的约束范围较小，有的较大。比如，压力容器中，$x_1$ 和 $x_2$ 就偏小，只是经过提取出 0.0625，勉强能达到 $x_3$ 和 $x_4$ 的一半。那么，如果某些优化问题，其参数约束范围之间，相差了量级，该**如何选择步长**呢？这就是 WPT 的便捷之处了。

BSAS-WPT 函数[7](对应 BSAS-WPT 算法) 调用的格式如下：

```
BSAS_WPT(fn,
         init = NULL,
         lower = c(-6, 0), upper = c(-1, 2),
```

[7] https://jywang2016.github.io/rBAS/reference/BSAS_WPT.html

```
          k = 5, constr = NULL, pen = 1e+05,
          c2 = 5,
          step = 1, eta_step = 0.95,steptol = 0.001,
          n = 200, seed = NULL, trace = T,
          p_min = 0.2,p_step = 0.2, n_flag = 2)
```

### 4.3.1 BSAS-WPT 参数说明

与 BSAS 相比，除去我人为略去的抖动部分，减少了搜索距离 d 相关的参数，这些用 c2 来替代。而初始步长 step，我们可以设定为一个在 1 附近的数。由于算法先标准化了参数，然后根据式(2.3)在计算位置后，再根据上下限进行反标准化，而后导入目标函数。所以，你可以认为，BSAS 中，把 step 变成一个 $n$ 维的向量，假设 $n$ 是参数个数，每个步长元素都根据参数的约束范围大小来设定，那么算法就会变成 BSAS-WPT。

总之，现在要调节的参数，主要有 2 个，即 c2 和 step。

### 4.3.2 BSAS-WPT 案例

我们使用和 BSASoptim 函数相同的例子来对比效果。但是，这些效果都是不固定的，即给定不同的参数，结果也会不同，所以不能根据一次结果评价算法的优劣。

#### 4.3.2.1 Pressure Vessel function

```
result <- BSAS_WPT(fn = pressure_Vessel$obj,
                   k = 8,
                   lower =c( 1, 1, 10, 10),
                   upper = c(100, 100, 200, 200),
                   constr = pressure_Vessel$con,
                   c2 = 10, n = 200, step = 2,
                   seed = 1,
```

```
                    n_flag = 3,
                    trace = FALSE,
                    steptol = 1e-6)
result$par
```

```
## [1]  13.882270   7.434164  42.094999 176.932890
```

```
result$value
```

```
## [1] 6065.478
```

#### 4.3.2.2 Himmelblau function

```
result <- BSAS_WPT(fn = himmelblau$obj,
                    k = 10,
                    lower =c(78,33,27,27,27),
                    upper = c(102,45,45,45,45),
                    constr = himmelblau$con,
                    c2 = 5, n = 200, step = 1.6,
                    pen = 1e5,trace = FALSE,seed = 11)
```

```
## ----step < steptol---------stop the iteration------
```

```
result$par
```

```
## [1] 78.00000 33.00000 27.07176 45.00000 44.96713
```

```
result$value
```

```
## [1] -31025.47
```

BSAS-WPT 没有做过多的参数调节，即可获得更畅快地优化体验。举例，在对 Himmelblau 函数进行优化时，我仅仅设定了随机种子 seed，然后把 step 从 1 调到了 2，看了看效果的变化。发现都不错，最后每隔 0.1 选取 step，试探最好的效果在哪里，于是就成了上面的例子。如果把这一套，放在 BSASoptim 函数上，对于复杂的优化问题，就**成了一种折磨**。

## 4.4 BSOoptim

### 4.4.1 BSO 参数说明

由于 BSO 参数与原理中的公式较为复杂。因此，在讲述其原理时，对函数的参数也进行了说明。故大家可以参考3.1.4节。此处，仅仅复制该节内容。

```
BSOoptim(fn, init = NULL, constr = NULL,
         lower = c(-50, -50), upper = c(50, 50), n = 300,
         s = floor(10 + 2 *sqrt(length(lower))),
         w = c(0.9, 0.4),
         w_vs = 0.4,
         step = 10,
         step_w = c(0.9, 0.4),
         c = 8,
         v = c(-5.12, 5.12),
         trace = T,
         seed = NULL,
         pen = 1e+06)
```

上面的代码是函数的默认调用形式，其中 $s$ 表示设定的天牛或者粒子数目，$w$ 也就是式(3.3)中提到的 $\omega_{max}$ 和 $\omega_{min}$。w_vs 是 $\lambda$，也就

是式(3.4)中天牛速度和移动步长之间的权重，直观地理解为 `weight between velocity and step-size`。`step` 和 `c` 仍然是表示步长和步长与须距离之比。而 $step_w$ 为一个向量，表示的是式(3.7)中的 $\omega_{\delta_1} = 0.4$，$\omega_{\delta_0} = 0.9$。

### 4.4.2 BSO 案例

#### 4.4.2.1 Michalewicz function

一个简单的例子（`Michalewicz function`）如下：

```
library(rBAS)
mich <- function(x){
y1 <- -sin(x[1])*(sin((x[1]^2)/pi))^20
y2 <- -sin(x[2])*(sin((2*x[2]^2)/pi))^20
return(y1+y2)
}
result <-
 BSOoptim(fn = mich,
           init = NULL,
           lower = c(-6,0),
           upper = c(-1,2),
           n = 100,
           step = 5,
           s = 10,seed = 1, trace = F)
result$par; result$value
```

```
## [1] -4.965998  1.570796
```

```
## [1] -1.967851
```

#### 4.4.2.2 Pressure Vessel function

```
pressure_Vessel <- list(
obj = function(x){
  x1 <- floor(x[1])*0.0625
  x2 <- floor(x[2])*0.0625
  x3 <- x[3]
  x4 <- x[4]
  result <- 0.6224*x1*x3*x4 + 1.7781*x2*x3^2 +3.1611*x1^2*x4 + 19.84*x1^2*x3
},
con = function(x){
  x1 <- floor(x[1])*0.0625
  x2 <- floor(x[2])*0.0625
  x3 <- x[3]
  x4 <- x[4]
  c(
    0.0193*x3 - x1,#<=0
    0.00954*x3 - x2,
    750.0*1728.0 - pi*x3^2*x4 - 4/3*pi*x3^3
  )
}
)
```

```
result<-
BSOoptim(fn = pressure_Vessel$obj,
         init = NULL,
         constr = pressure_Vessel$con,
         lower = c( 1, 1, 10, 10),
         upper = c(100, 100, 200, 200),
         n = 1000,
         w = c(0.9,0.4),
         w_vs = 0.9,
```

```
          step = 100,
          step_w = c(0.9,0.4),
          c = 35,
          v = c(-5.12,5.12),
          trace = F,seed = 1,
          pen = 1e6)
result$par
```

```
## [1]  13.955250   7.550179  42.098446 176.636596
```

```
result$value
```

```
## [1] 6059.131
```

得到的结果十分好（甚至比论文 (?) 还要高出那么一点点）。但是，这是调参调出来的结果。

总的来说，我自己的经验是：

- 调节最大迭代次数 `n`
- 调节步长 `step` 和步长与须距离比值 `c`（让搜索距离的尺度尽量在迭代后不要太大）
- 调节 `w_vs`，该值越大，粒子群算法更新方式所占的比重越大。

而王糖糖同学给出的建议是：

> “在试验的时候发现，当维度提高时，步长和迭代次数也要相应的提高”
>
> —王糖糖

最后呢，我们为什么要改算法！

> 开个玩笑，祝大家周末愉快，学业顺心。


图 4.3: 为什么我们孜孜不倦改算法

# 第五章　用户界面

用户界面基于 shiny[1]包开发。核心思想是，把目标函数或者约束在 R 的脚本中预先定义好，然后调用 `run_BAS_App(func = ..., constr = ...,theme = ...)` 函数来人为调节参数，运行以及可视化。

## 5.1　调用语句

```
run_BAS_App(func = Your-objective-func,
            constr = Your-constraint-func,
            theme = 'united')
```

其中：

- `func` 指的是，需要优化的目标函数
- `constr` 则是不等式约束，至于上下限约束，可以直接在用户界面里面定义
- `theme` 也就是界面主题，即**皮肤**，默认的主题是 `united`。可以使用 `help(run_BAS_App)` 语句来看可选的主题样式都有哪些，如 `superhero`,`cosmo`,`paper`,`journal` 等主题。这些主题是调用 shinythemes[2]而来。

[1] https://github.com/rstudio/shiny

[2] https://github.com/rstudio/shinythemes

## 5.2 使用案例

### 5.2.1 Michalewicz function

先在代码中预定义函数：

```
mich <- function(x){
  y1 <- -sin(x[1])*(sin((x[1]^2)/pi))^20
  y2 <- -sin(x[2])*(sin((2*x[2]^2)/pi))^20
  return(y1+y2)
}
```

然后调用用户界面：

```
run_BAS_App(func = mich)
#run_BAS_App(func = mich, theme = 'paper') 可以尝试下 paper 或是其他的主题
```

出现了如图5.1的界面：

左边是固定的参数调节栏，最上方有目前的收录的三种算法可供选择，以及本包的作者信息。右侧也有三个选项，分别是**优化过程信息**，**优化参数结果**以及**优化结果可视化**。

按照你的需要，调节好左边的参数信息（第一个参数，也就是初始值 `init`，默认为空，也可以指定），然后点击左下方的 `Run BAS` 键，即可看到如图5.2的内容：

由于回合数较大，因此只截取了部分内容显示。

分别点击 `Optimization Parameters` 和 `Progress Plot` 键，可以看到最后的结果，以及可视化信息，分别如图5.3与 5.4所示。

可以看到，窗口的 `$par` 显示的是参数的优化结果，而 `$value` 则是对应的目标函数值。

`BAS` 与其他两种算法有着不同的可视化结果，其并不是基于反馈来控制步长的。因此，图中的两条曲线，红色的为每一回合的目标函数值，而蓝色的为此前回合中最优的目标函数值。

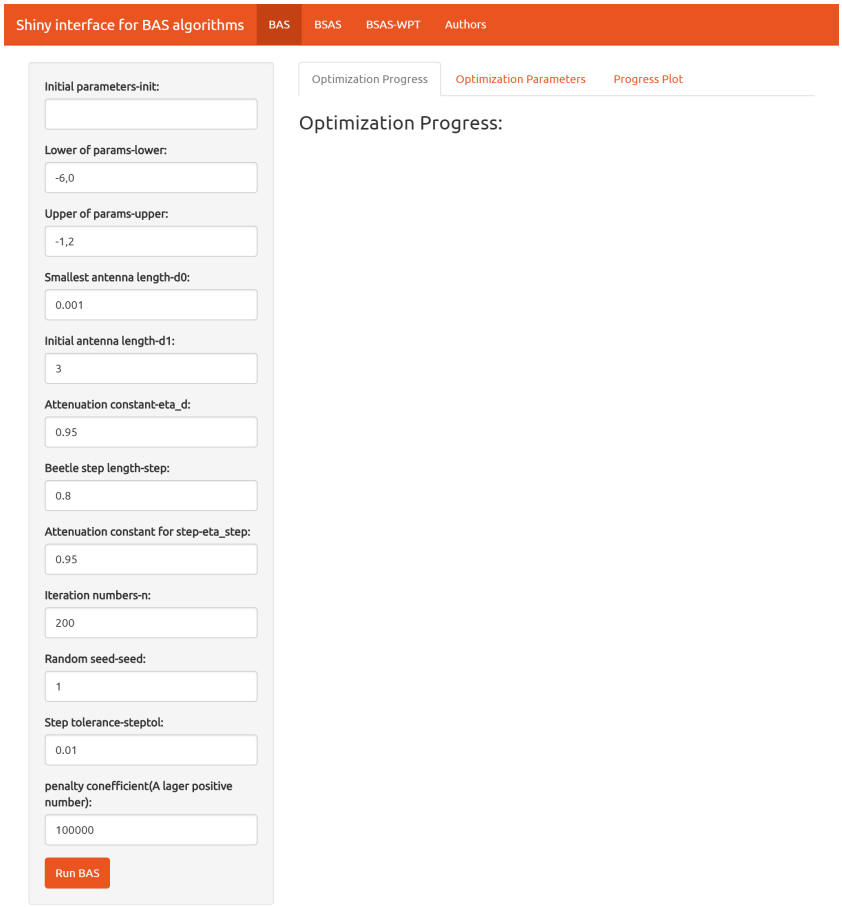


图 5.1: shiny interface

### 5.2.2 Pressure Vessel function

这次，我们在 BSAS-WPT 栏下进行界面使用。先在代码中预定义目标函数和约束：

```
pressure_Vessel <- list(
  obj = function(x){
    x1 <- floor(x[1])*0.0625
    x2 <- floor(x[2])*0.0625
    x3 <- x[3]
```

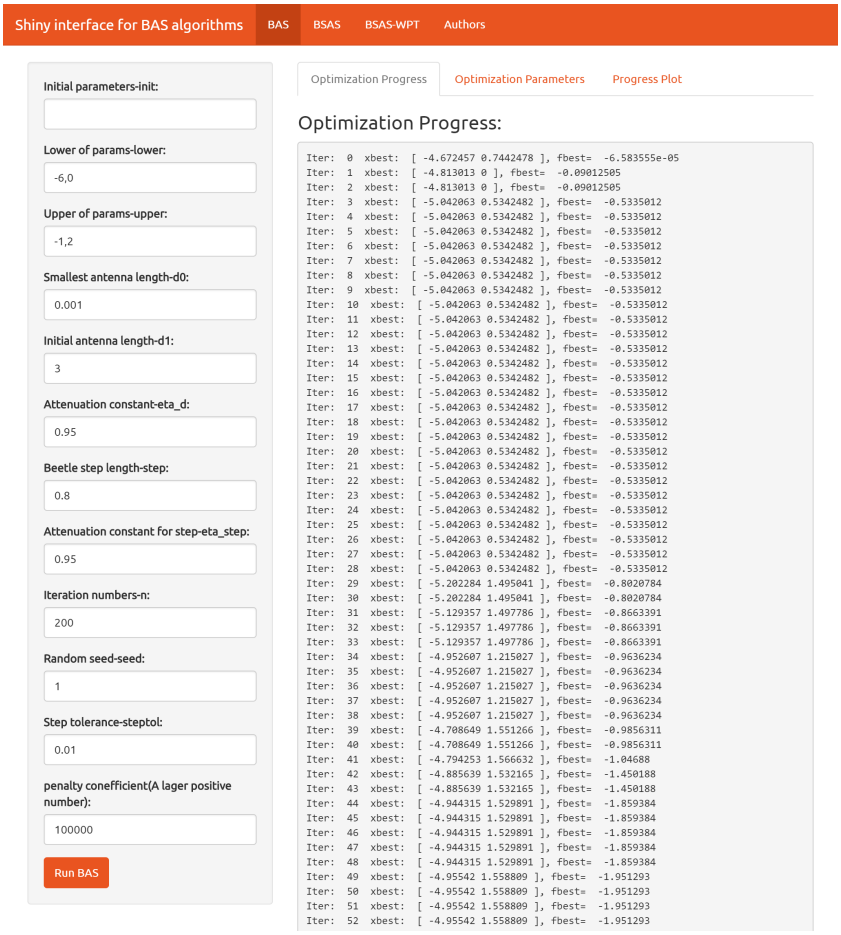


图 5.2: optimization progress 栏信息

```
  x4 <- x[4]
  result <- 0.6224*x1*x3*x4 +
    1.7781*x2*x3^2 +
    3.1611*x1^2*x4 +
    19.84*x1^2*x3
},
con = function(x){
  x1 <- floor(x[1])*0.0625
  x2 <- floor(x[2])*0.0625
```

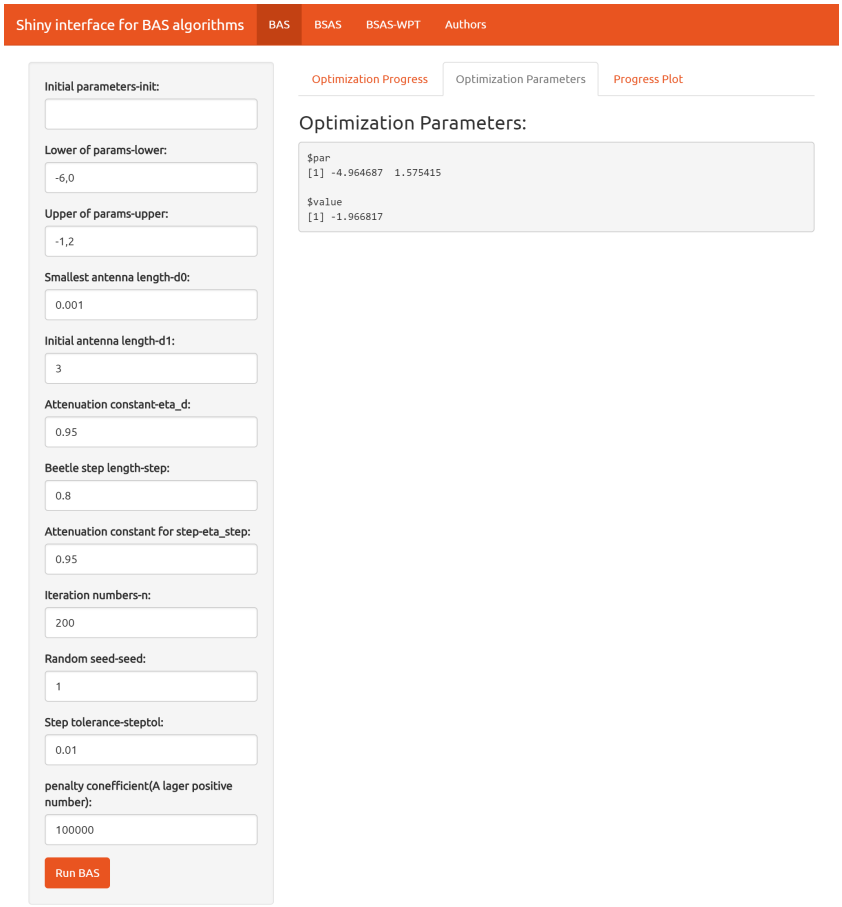


图 5.3: Optimization Parameters 栏信息

```
  x3 <- x[3]
  x4 <- x[4]
  c(
    0.0193*x3 - x1,#<=0
    0.00954*x3 - x2,
    750.0*1728.0 - pi*x3^2*x4 - 4/3*pi*x3^3
  )
}
```

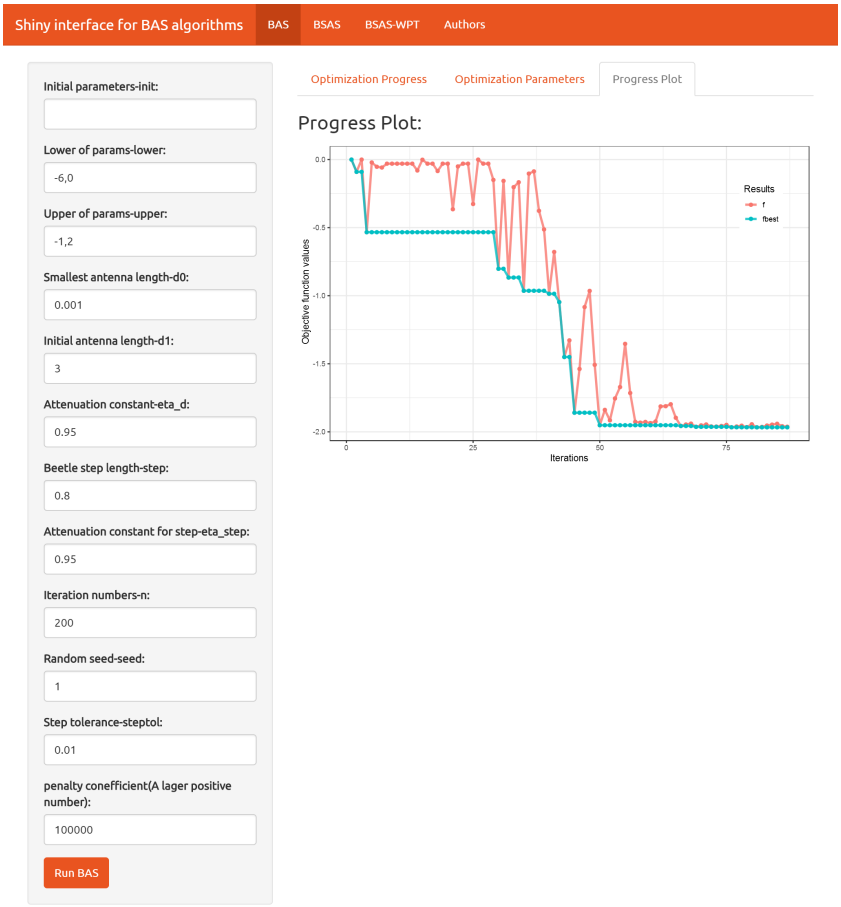


图 5.4: Progress Plot 栏信息

```
)
```

调用用户界面，注意此时多出了 `constr`，也就是约束函数，`$` 符号在索引列表中的元素时使用：

```
run_BAS_App(func = pressure_Vessel$obj,
            constr = pressure_Vessel$con)
```

自行调整参数后，用户界面如图5.5所示：

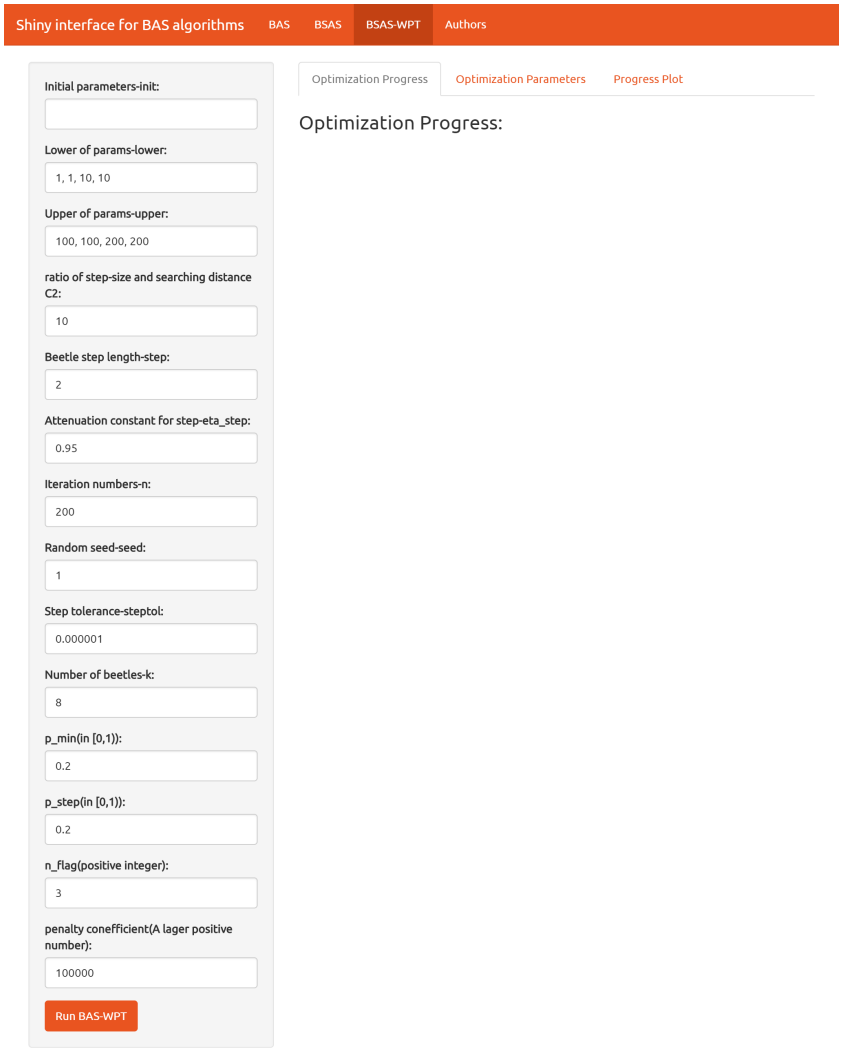


图 5.5: BSAS-WPT 参数调整

点击 Run BAS-WPT 之后，选择 optimization Parameters 栏目，可以看到优化结果如图5.6所示：

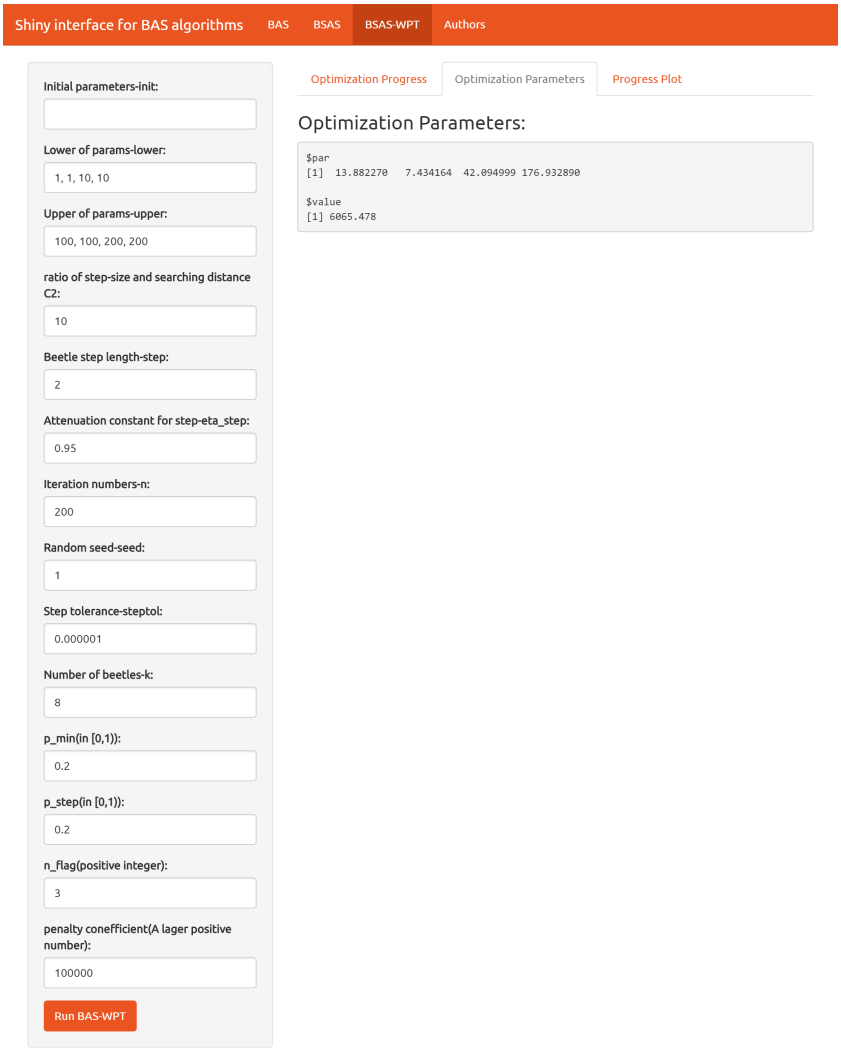


图 5.6: BSAS-WPT 优化结果

选择 Progress Plot 栏目，过程可视化如图5.7所示：

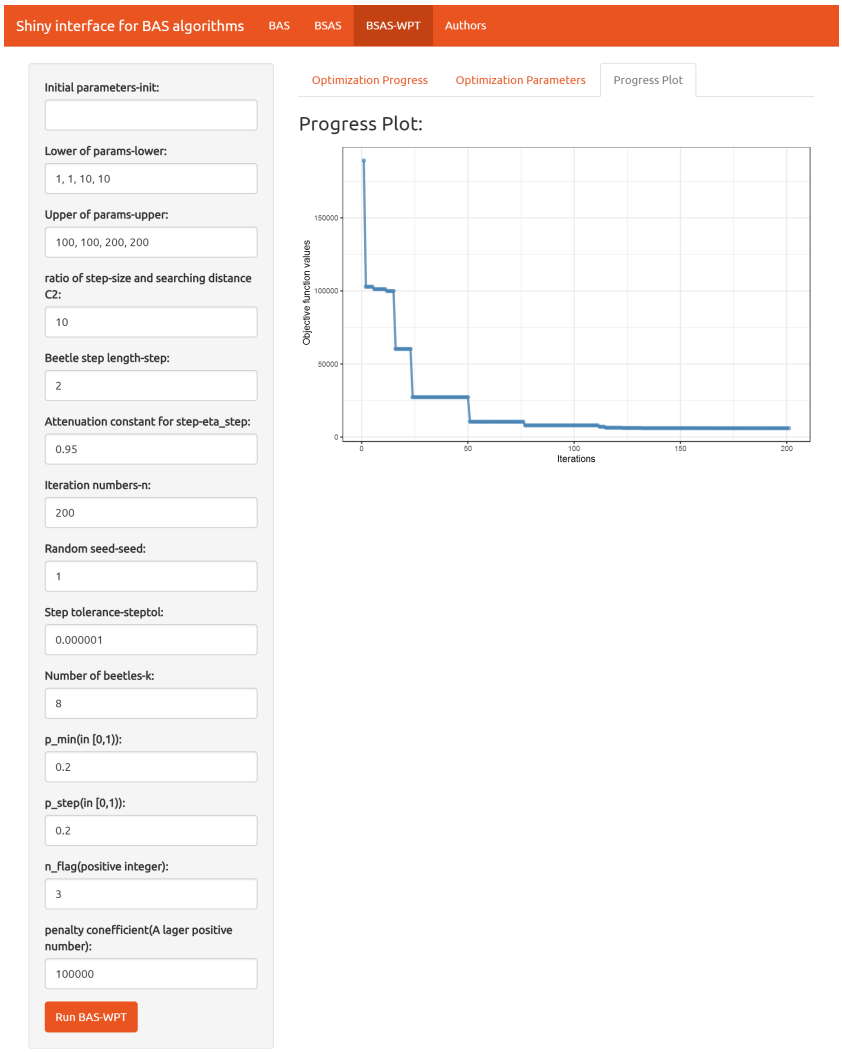


图 5.7: BSAS-WPT 优化过程可视化

## 5.3 Authors 界面

如果并不想执行任何函数优化，则可以不指定函数和约束。在 R 里面输入以下代码：

```
library(rBAS) # 加载 rBAS 包
run_BAS_App() # 直接调用函数
```

可以看到 rBAS 的用户界面，里面有关于 rBAS 的作者信息，如图5.8所示。

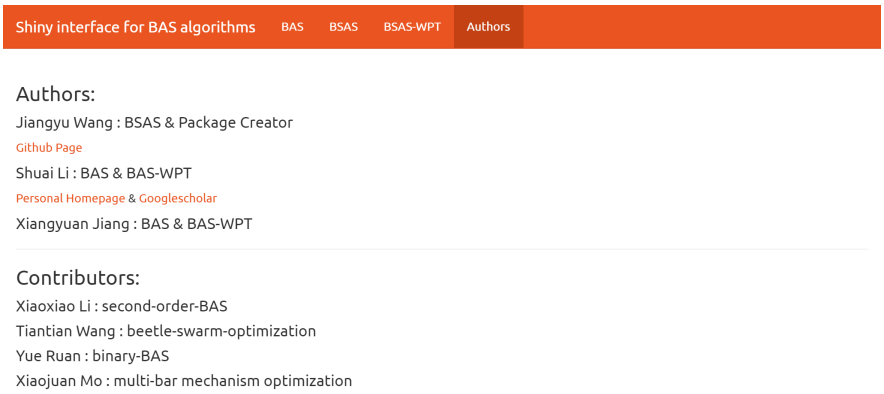


图 5.8: 用户界面作者信息

如果大家对该项目有兴趣，参与了包的开发，或者 BAS 算法应用案例的提出。我会在征得当事人同意的情况下，将名字加入该界面:)。

# 第六章　BAS 案例一: 多杆机构优化问题

说明：由于自身专业知识局限，在整理转述各位研究者或贡献者所提供的材料时，我可能无法准确地表达出对应领域的知识要点。避免言多必失，我仅仅做简要的翻译或者是介绍。对于读者而言，如果觉得并不能解你之疾，或没有**挠到痒处**，建议直接和对应的作者联系。对于贡献者而言，如果我表述有误，欢迎提出建议，或者在 github 上 `pull request`。

由群友莫小娟博士研究生提供案例。

由于案例均为各位热心的同学提供，均为自己的研究。因此，希望大家要引用其中的结果时，可以引用对应同学的文章。此外，也请大家转载时注明来源。

## 6.1　背景

### 6.1.1　四连杆机构 (Four-bar linkage mechanism)

四连杆机构如图6.1所示：

基于闭环矢量方程，推导出节点位置。推导过程如式(6.1)至式(6.4):

$$\text{Loop:} r_1 e^{i\theta_0} + r_4 e^{i\theta_4} - r_2 e^{i\theta_2} - r_3 e^{i\theta_3} = 0 \tag{6.1}$$

$$\begin{cases} r_1 cos(\theta_0) + r_4 cos(\theta_4) - r_2 cos(\theta_2) - r_3 cos(\theta_3) = 0 \\ r_1 sin(\theta_0) + r_4 sin(\theta_4) - r_2 sin(\theta_2) - r_3 sin(\theta_3) = 0 \end{cases} \tag{6.2}$$

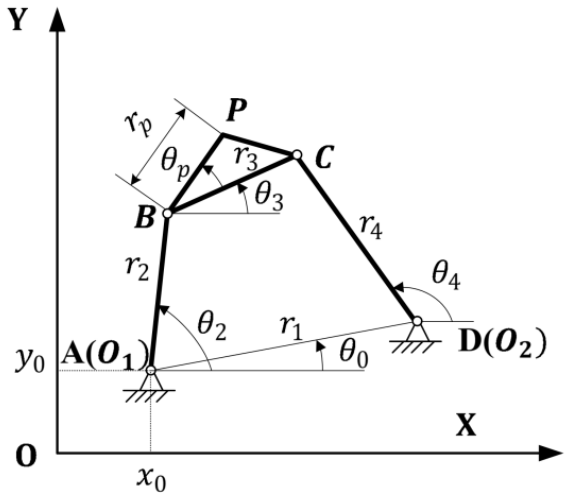


图 6.1: 四连杆机构示意

$$
\begin{aligned}
&\theta_3 = 2atan(\frac{-A \pm \sqrt{2}A^2 - 4BC}{2B}) + \theta_0 \\
&A = cos(\theta_2 - \theta_0) - K_1 + K_2 cos(\theta_2 - \theta_0) + K_3 \\
&B = -2sin(\theta_2 - \theta_0), F = K_1 + (K_2 - 1)cos(\theta_2 - \theta_0) + K_3 \\
&K_1 = r_1/r_2, K_2 = r_1/r_3, K_3 = (r_4^2 - r_1^2 - r_2^2 - r_3^2)/(2r_2r_3)
\end{aligned} \tag{6.3}
$$

$$
\begin{cases}
x_p = x_0 + r_2 cos(\theta_2) + r_p cos(\theta_3 + \theta_p) \\
y_p = y_0 + r_2 sin(\theta_2) + r_p sin(\theta_3 + \theta_p)
\end{cases} \tag{6.4}
$$

### 6.1.2 六连杆机构 (Stephenson III Six-bar linkage mechanism)

六连杆机构如图6.2所示：

基于闭环矢量方程，推导出节点位置和外部链接角度。推导过程如式(6.5)至式(6.6):

$$
\begin{aligned}
&\text{Loop1:} r_1 e^{i\theta_0} + r_4 e^{i\theta_4} - r_2 e^{i\theta_2} - r_3 e^{i\theta_3} = 0 \\
&\text{Loop2:} r_1' e^{i\theta_0'} + r_6 e^{i\theta_6} - r_2 e^{i\theta_2} - r_p e^{i(\theta_3 + \theta_p)} - r_5 e^{i\theta_5} = 0
\end{aligned} \tag{6.5}
$$

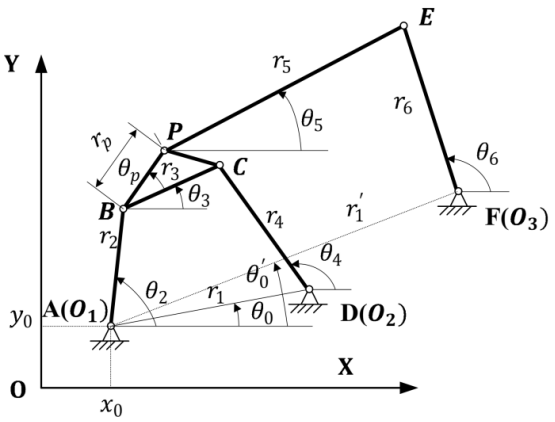


图 6.2: 六连杆机构示意

Loop1:

$$\alpha = r_2 cos(\theta_2) - r_1 cos(\theta_0), \beta = r_2 sin(\theta_2) - r_1 sin(\theta_0)$$
$$\gamma = (r_4^2 + \alpha^2 + \beta^2 - r_3^2)/(2r_4), \lambda = atan2(\alpha, \beta)$$
$$\theta_4 = atan(cos(\lambda)\gamma/\beta, \{1 - (cos(\lambda)\gamma/\beta)^2\}^{1/2}) - \lambda$$
$$\theta_3 = atan2(r_4 sin(\theta_4) - \beta, r_4 cos(\theta_4) - \alpha)$$

Loop2: (6.6)

$$\alpha_1 = r_2 cos(\theta_2) + r_p cos(\theta_3 + \theta_p) - r_6 cos(\theta_6)$$
$$\beta_1 = r_2 sin(\theta_2) + r_p sin(\theta_3 + \theta_p) - r_6 sin(\theta_6)$$
$$\gamma_1 = (r_6^2 + \alpha_1^2 + \beta_1^2 - r_5^2)/(2r_6), \lambda_1 = atan2(\alpha_1, \beta_1)$$
$$\theta_6 = atan2(cos(\lambda)\gamma_1/\beta_1, -\{1 - (cos(\lambda_1)\gamma_1/\beta_1)^2\}^{1/2}) - \lambda_1$$
$$\theta_5 = atan2(r_6 sin(\theta_6) - \beta_1, r_6 cos(\theta_6) - \alpha_1)$$

于是，可以得到节点位置如式(6.7)：

$$\begin{cases} x_A = x_0, y_A = y_0 \\ x_D = x_0 + r_1 cos(\theta_0), y_D = y_0 + r_1 sin(\theta_0) \\ x_F = x_0 + r_1' cos(\theta_0'), x_F = x_0 + r_1' sin(\theta_0') \\ x_p = x_0 + r_2 cos(\theta_2) + r_p cos(\theta_3 + \theta_p) \\ y_P = y_0 + r_2 sin(\theta_2) + r_P sin(\theta_3 + \theta_p) \end{cases} \tag{6.7}$$

## 6.2 优化问题

### 6.2.1 四连杆机构

四杆机构的优化问题可以用式(6.8)表示。

$$\begin{aligned} &\min \sum_{i=1}^{N}[(P_{Xd}^i - P_X^i)^2 + (P_{Yd}^i - P_Y^i)^2] + M_1 h_1(x) + M_2 h_2(x) \\ &\text{where } x_i \in [l_{min}^i, l_{max}^i] \quad \forall x_i \in X, \\ &\qquad X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^N] \end{aligned} \tag{6.8}$$

其中，

$$h_1(x) = \begin{cases} 1, & \text{the Grashof condition false} \\ 0, & \text{the Grashof condition true} \end{cases} \tag{6.9}$$

$$h_2(x) = \begin{cases} 1, & \text{the sequence condition of the crank angle false} \\ 0, & \text{the sequence condition of the crank angle true} \end{cases} \tag{6.10}$$

$h_1(X)$ 和 $h_2(X)$ 分别用于评估曲柄存在条件（`Grashof Condition`）以及曲柄角度（顺时针或逆时针）的顺序情况。$M_1$ 和 $M_2$ 分别是对应的惩罚系数。$X$ 为设计参数。

### 6.2.2 六连杆机构

$$\begin{aligned}\min\ & \sum_{i=1}^{N}[(P_{Xd}^{i}-P_{X}^{i})^{2}+(P_{Yd}^{i}-P_{Y}^{i})^{2}]+\sum_{i=1}^{M}[(\theta_{6d}^{i}-\theta_{6}^{i})^{2}]\\ &+M_1h_1(x)+M_2h_2(x)+M_3h_3(X)\\ \text{where } & x_i\in[l_{min}^{i},l_{max}^{i}]\quad \forall x_i\in X,\\ & X=[r_1,r_2,r_3,r_4,r_5,r_6,r_p,\theta_p,\theta_0,x_0,y_0,\theta_2^1,\cdots,\theta_2^N]\end{aligned}\tag{6.11}$$

其中，$h_1(X)$ 与 $h_2(X)$ 同式(6.9)与(6.10)，$h_3(X)$ 如式(6.12)：

$$h_3(x)=\begin{cases}1, & \text{non-violation of transmission angle false}\\ 0, & \text{non-violation of transmission angle true}\end{cases}\tag{6.12}$$

$h_3(X)$ 所表示的是，是否没有违背传动角（超过 20°）的约束。同样地，$M_3$ 为对应的惩罚系数。

## 6.3 优化理论

案例所使用的算法是**标准化的群体天牛算法**。有意思的是，此处群体天牛算法，和 `rBAS` 包的 `BSASoptim` 的算法极为类似。这也说明，加入群体智能策略，会使得 `BAS` 对于复杂问题寻优能力增强。

联系在于：此案例使用的群体天牛，是在每回合，对于天牛探索方向数的提升。即，每回合生成多个随机的方向，在这些方向上，派出天牛进行试探。这点和 `BSAS` 保持一致，可以理解为，**如果天牛不止有一对须，而是有多对，那每回合探索的方向也会有多个**。大致的原理如式(6.13)：

$$\begin{aligned}x_{ri}&=x_t+d^t\overrightarrow{b_i},\quad i=1,\cdots,q\\ x_{li}&=x_t-d^t\overrightarrow{b_i},\\ x_{ti}&=x_{t-1}-\delta^t\overrightarrow{b_i}sign(f(x_{ri})-f(x_{li}))\end{aligned}\tag{6.13}$$

区别在于：并未使用基于结果反馈的步长调节策略。

对于轨迹优化问题，莫小娟同学给出的参数建议是，$d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9998, c_2 = 0.5, q = 40, T_{max} = 50000$。部分同学可能看过手册的二节，对于步长 $d$ 和须到质心距离 $\delta$ 的更新，即式(2.6)中有所提及。此处，参数的含义如式(6.14)所示。部分参数与式(2.6)类同，也有同名但含义冲突的，大家复现时需要注意这些地方。

$$\begin{aligned} d^t &= c_1 d^{t-1} \\ \delta^t &= c_2 d^t \end{aligned} \tag{6.14}$$

## 6.4 优化结果

此处，莫小娟同学提供了 8 个案例，并且与其他的经典算法（指多杆机构优化问题中多用的算法）进行了优化效果的对比。

### 6.4.1 Case1 无规定时间内轨迹生成 (Path generation without prescribed timing)

本案例是四杆机构的路径（6 个点）在一条垂直的线上（没有规定时间）。通过式(6.8)计算得到误差。

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^6]$$

目标点坐标：

$$\{C_d^i\} = \{(20, 20), (20, 25), (20, 30), (20, 40), (20, 45)\}$$

参数约束：

$$r_1, r_2, r_3, r_4 \in [0, 60] \quad r_p, x_0, y_0 \in [-60, 60] \quad \theta_0, \theta_1, \cdots, \theta_2^6, \theta_p \in [0, 2\pi]$$

算法参数为 $d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9998, c_2 = 0.5, q = 40, T_{max} = 50000$，动画结果如图**??**。

img/case1.gif

与其他算法对比结果如图6.3和表6.1。

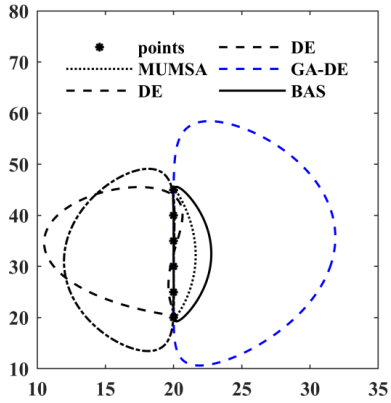


图 6.3: 各算法优化轨迹

### 6.4.2 Case2 有规定时间的轨迹生成 (with prescribed timing)

本案例四杆机构的路径为 5 个没有对齐的点（规定时间）。通过式(6.8)计算得到误差。

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p]$$

目标点坐标：

$$\begin{aligned}&\{C_d^i\} = \{(3,3),(2.759,3.363),(2.759,3.363),(1.890,3.862),(1.355,3.943)\}\\&\{\theta_2^1,\theta_2^2,\theta_2^3,\theta_2^4,\theta_2^5\} = \{\pi/6,\pi/4,\pi/3,5\pi/12,\pi/2\}\end{aligned} \tag{6.15}$$

参数约束：

表 6.1: case1 各算法结果对比

| | MUMSA | GA | DE | GA.DE | BAS |
|---|---|---|---|---|---|
| $r_1$ | 31.788264 | 28.771330 | 35.020740 | 13.2516000 | 19.4920810 |
| $r_2$ | 8.204647 | 5.000000 | 6.404196 | 5.9407800 | 6.2644716 |
| $r_3$ | 24.932131 | 35.365480 | 31.607220 | 58.3118000 | 20.1001631 |
| $r_4$ | 31.385926 | 59.136810 | 50.599490 | 53.7207000 | 19.0219161 |
| $r_p$ | 37.108246 | 14.850370 | 46.461261 | 61.3011560 | 39.8538048 |
| $ _p$ | 0.398977 | 1.570796 | 1.106544 | -1.3002600 | 0.3679660 |
| $ _0$ | 4.015959 | 5.287474 | 0.000000 | 0.1960760 | 4.4245619 |
| $x_0$ | -6.366519 | 29.913290 | 60.000000 | -35.3621000 | -13.0300715 |
| $y_0$ | 56.836760 | 32.602280 | 18.077910 | 36.7704000 | 51.1796666 |
| $ _2^1$ | 1.366547 | 6.283185 | 6.283185 | 1.6601500 | 5.9699819 |
| $ _2^2$ | 2.330773 | 0.318205 | 0.264935 | 2.0468400 | 0.4554995 |
| $ _2^3$ | 2.871039 | 0.638520 | 0.500377 | 2.4281100 | 1.0202707 |
| $ _2^4$ | 3.394591 | 0.979950 | 0.735321 | 2.8090100 | 1.5550696 |
| $ _2^5$ | 3.970960 | 1.412732 | 0.996529 | 3.1900900 | 2.1117409 |
| $ _2^6$ | 4.963490 | 2.076254 | 1.333549 | 3.5737900 | 2.8839248 |
| $Error$ | 0.002057 | 1.101697 | 0.122738 | 0.0000172 | 0.0000124 |

$$r_1, r_2, r_3, r_4 \in [0, 5] \quad r_p \in [-5, 5] \quad \theta_p \in [0, 2\pi]$$

算法参数为 $d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9998, c_2 = 0.5, q = 40, T_{max} = 10000$，动画结果如图**??**。

img/case2.gif

与其他算法对比结果如图6.4和表6.2。

### 6.4.3 Case3 规定时间内路径生成（Path generation with prescribed timing）

本案例四杆机构需要在规定时间通过一个闭环。通过式(6.8)计算得到误差。

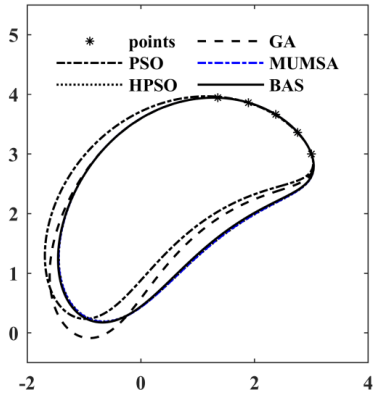


图 6.4: 各算法优化轨迹

表 6.2: case2 各算法结果对比

| | PSO | HPSO | GA | MUMSA | BAS |
|---|---|---|---|---|---|
| $r_1$ | 3.0576200 | 3.7877200 | 3.0630424 | 3.7732686 | 3.7135255 |
| $r_2$ | 1.8618400 | 1.9984200 | 1.9959624 | 2.0000040 | 1.9978745 |
| $r_3$ | 3.8459100 | 4.1331300 | 3.3058230 | 4.1169710 | 4.0469527 |
| $r_4$ | 2.9706300 | 2.7451300 | 2.5247060 | 2.7461567 | 2.7192391 |
| $r_p$ | 2.4968270 | 2.3697220 | 2.3721479 | 2.3684374 | 2.3702974 |
| $ _p$ | 0.7243557 | 0.7841479 | 0.8044176 | 0.7831567 | 0.7865371 |
| $Error$ | 0.0005860 | 0.0009860 | 0.0000018 | 0.0000018 | 0.0000007 |

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^6]$$

目标点坐标：

$$
\begin{aligned}
\{C_d^i\} = \{&(0.5, 1.1), (0.4, 1.1), (0.3, 1.1), (0.2, 1.0), (0.1, 0.9), (0.05, 0.75), \\
&(0.02, 0.6), (0, 0.5), (0, 0.4), (0.03, 0.3), (0.1, 0.25), (0.15, 0.2), \\
&(0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.5, 0.7), (0.6, 0.9), (0.6, 1.0)\} \\
\{\theta_2^i\} = \{&\theta_2^1, \theta_2^1 + 20 \cdot i/\pi\}, \quad i = 1, \cdots, 17
\end{aligned}
\tag{6.16}
$$

参数约束：

$$
r_1, r_2, r_3, r_4 \in [0, 5] \quad r_p, x_0, y_0 \in [-5, 5] \quad \theta_0, \theta_2^1, \theta_p \in [0, 2\pi]
$$

算法参数为 $d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9998, c_2 = 0.5, q = 8, T_{max} = 50000$，动画结果如图**??**。

`img/case3.gif`

与其他算法对比结果如图6.5和表6.3。

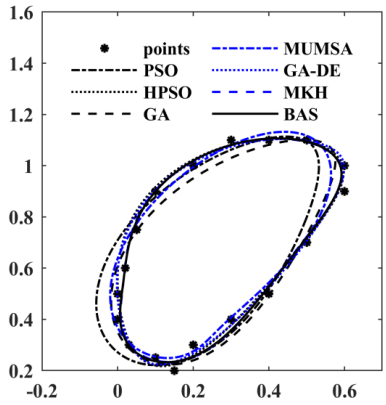


图 6.5: 各算法优化轨迹

### 6.4.4 Case4 规定时间路径生成问题

第四个案例同样是一个规定时间的路径生成问题。六个优化点由一个 `semi-archer` 弧构成，问题定义如下。

表 6.3: case3 各算法结果对比

| | PSO | HPSO | GA | MUMSA | GA.DE | MKH | |
|---|---|---|---|---|---|---|---|
| $r_1$ | 2.926100 | 2.850000 | 3.057878 | 4.453772 | 47.4379000 | 1.0042900 | 1 |
| $r_2$ | 0.487700 | 0.370000 | 0.237803 | 0.297057 | 0.3247700 | 0.4218000 | 0 |
| $r_3$ | 2.909900 | 2.904800 | 4.828954 | 3.913095 | 0.4728570 | 0.8782100 | 0 |
| $r_4$ | 2.150300 | 0.500000 | 2.056465 | 0.849372 | 47.3093000 | 0.5801300 | 0 |
| $r_p$ | 1.493947 | 1.973700 | 2.003475 | 2.651983 | 0.3412513 | 0.5234000 | 0 |
| $ _p$ | -0.332546 | 1.027396 | 1.177913 | 2.464734 | -1.2153827 | 0.8147728 | 0 |
| $ _0$ | 0.719000 | 0.760000 | 1.002168 | 2.738736 | 3.3202900 | 0.2929400 | 0 |
| $x_0$ | -0.384600 | 0.940000 | 1.776808 | -1.309243 | 0.5269880 | 0.2688600 | 0 |
| $y_0$ | -0.675200 | -1.171200 | -0.641991 | 2.806964 | 0.7239300 | 0.1771500 | 0 |
| $ _2^1$ | 0.215900 | 0.513400 | 0.226186 | 4.853543 | 3.5123300 | 0.8859500 | 1 |
| $Error$ | 0.049200 | 0.011100 | 0.033700 | 0.019600 | 0.0108613 | 0.0091100 | 0 |

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0]$$

目标点坐标：

$$\{C_d^i\} = \{(0,0), (1.9098, 5.8779), (6.60989.5106), (13.09, 9.5106), (18.09, 5.8779), (20, 0)\}$$
$$\{\theta_2^1, \theta_2^2, \theta_2^3, \theta_2^4, \theta_2^5\} = \{\pi/6, \pi/3, \pi/2, 2\pi/3, 5\pi/6, \pi\} \tag{6.17}$$

参数约束：

$$r_1, r_2, r_3, r_4 \in [0, 50] \quad r_p, x_0, y_0 \in [-50, 50] \quad \theta_0, \theta_p \in [0, 2\pi]$$

算法参数为 $d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9997, c_2 = 0.5, q = 40, T_{max} = 30000$，动画结果如图**??**。

img/case4.gif

表 6.4: case4 各算法结果对比

| | MUMSA | GA | PSO | GA.DE | BAS |
|---|---|---|---|---|---|
| $r_1$ | 50.0000000 | 50.0000000 | 50.0000000 | 50.000000 | 47.2345017 |
| $r_2$ | 5.0000000 | 5.0000000 | 5.0000000 | 5.000000 | 8.8473992 |
| $r_3$ | 7.0310470 | 6.9700900 | 7.0310200 | 5.905343 | 25.0474709 |
| $r_4$ | 48.1341830 | 48.1993000 | 48.1342000 | 50.000000 | 50.0000000 |
| $r_p$ | 21.3533558 | 21.2191204 | 21.3532819 | 18.819312 | 50.0000000 |
| $ _p$ | 0.6517294 | 0.6380060 | 0.6517238 | 0.000000 | 5.7107187 |
| $ _0$ | 0.0428247 | 0.0508453 | 0.0428286 | 0.463633 | 0.8225945 |
| $x_0$ | 12.1974940 | 12.2377000 | 12.1975000 | 14.373772 | 16.5531171 |
| $y_0$ | -15.9982030 | -15.8332000 | -15.9981000 | -12.444295 | -48.1473786 |
| $Error$ | 2.5803500 | 2.5828600 | 2.5803600 | 2.349649 | 0.7863680 |

与其他算法对比结果如图6.6和表6.4。

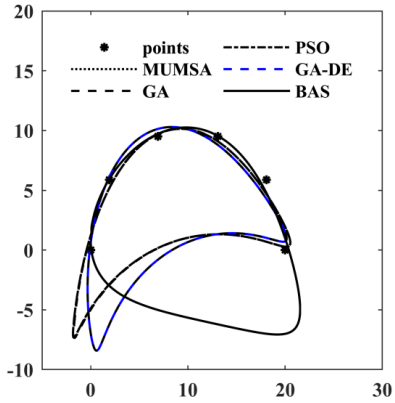


图 6.6: 各算法优化轨迹

### 6.4.5 Case5 规定时间内路径生成问题

这个例子是一个椭圆路径生成问题，没有规定的时间，其中轨迹是由 10 个点定义的。问题定义如下。

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^{10}]$$

目标点坐标：

$$\{C_d^i\} = \{(20, 10), (17.66, 15.142), (11.736, 17.878), (5, 16.928), (0.60307, 12.736), (0.60307, 7.2638), (5, 3.0718), (11.736, 2.1215), (17.66, 4.8577), (20, 0)\} \tag{6.18}$$

参数约束：

$$r_1, r_2, r_3, r_4 \in [0, 80] \quad r_p, x_0, y_0 \in [-80, 80] \quad \theta_0, \theta_2^1, \cdots, \theta_2^{10}, \theta_p \in [0, 2\pi]$$

算法参数为 $d_0 = 0.10, \delta_0 = 0.05, c_1 = 0.9998, c_2 = 0.5, q = 40, T_{max} = 40000$，动画结果如图**??**。

img/case5.gif

与其他算法对比结果如图6.7和表6.5。

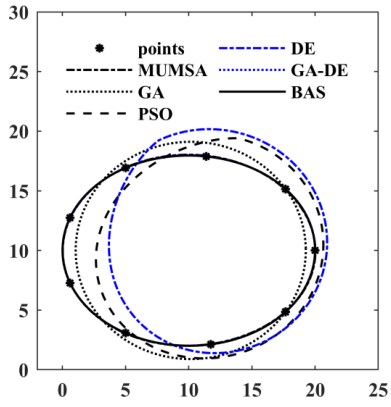


图 6.7: 各算法优化轨迹

表 6.5: case5 各算法结果对比

| | MUMSA | GA | PSO | DE | GA.DE | BAS |
|---|---|---|---|---|---|---|
| $r_1$ | 79.5160680 | 79.981513 | 52.535162 | 54.360893 | 80.0000000 | 71.8681225 |
| $r_2$ | 9.7239730 | 9.109993 | 8.687886 | 8.683351 | 8.4203200 | 9.2618623 |
| $r_3$ | 45.8425240 | 72.936511 | 36.155078 | 34.318634 | 51.3426000 | 44.4542963 |
| $r_4$ | 51.4384800 | 80.000000 | 80.000000 | 79.996171 | 42.4532000 | 43.0533508 |
| $r_p$ | 8.7289390 | 0.000000 | 1.481055 | 1.465250 | 10.6530404 | 8.7820722 |
| $ _p$ | -0.3452264 | 0.000000 | 1.570796 | 1.570669 | 2.6465448 | 1.6362575 |
| $ _0$ | 5.5969445 | 0.026149 | 1.403504 | 2.129650 | 4.2817700 | 1.6011655 |
| $x_0$ | 2.0211090 | 10.155966 | 11.002124 | 10.954397 | 5.5337200 | 16.7540360 |
| $y_0$ | 13.2165878 | 10.000000 | 11.095585 | 11.074534 | 0.4771830 | 15.2986682 |
| $ _2^1$ | 0.6376873 | 6.283185 | 6.282619 | 6.283185 | 2.0935000 | 0.1258416 |
| $ _2^2$ | 1.3255329 | 0.600745 | 0.615302 | 0.616731 | 2.8129100 | 0.8167208 |
| $ _2^3$ | 2.0080339 | 1.372812 | 1.305421 | 1.310254 | 3.5160500 | 1.5351326 |
| $ _2^4$ | 2.6955659 | 2.210575 | 2.188053 | 2.193570 | 4.2063800 | 2.1811489 |
| $ _2^5$ | 3.3845794 | 2.862639 | 2.913049 | 2.917170 | 4.8905100 | 2.8753839 |
| $ _2^6$ | 4.0829376 | 3.420547 | 3.499313 | 3.490746 | 5.5739800 | 3.5728072 |
| $ _2^7$ | 4.7984548 | 4.072611 | 4.125586 | 4.132017 | 6.2645800 | 4.2854058 |
| $ _2^8$ | 5.5117056 | 4.910373 | 4.919977 | 4.922075 | 0.6761980 | 5.0016208 |
| $ _2^9$ | 6.2127919 | 5.682440 | 5.685021 | 5.695372 | 1.3830700 | 5.7133417 |
| $ _2^{10}$ | 0.6371866 | 6.283185 | 6.282323 | 6.282970 | 2.0934800 | 0.1258290 |
| $Error$ | 0.0047000 | 2.281273 | 1.971004 | 1.952326 | 0.0006022 | 0.0004252 |

### 6.4.6 Case6 六杆机构路径生成

这个案例我也看不懂。大概意思是，六杆问题，在规定时间内让轨迹耦合目标点，并且 `output link` 在停顿位置（`dwell portion`? 我翻译不下去了……）保持在一个精确的角度。大家看底下的原文靠谱一点。

> "This case is a path and function combined synthesis problem with prescribed timing in which the coupler of six-bar mechanism has to the precision points and its output link has to

maintain an accuracy angle in the dwell portion."

— Xiaojuan Mo

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_5, r_6, r_p, \theta_p, r_1', \theta_0, \theta_0', x_0, y_0, \theta_2^1]$$

目标点坐标：

$$\begin{aligned}
\{C_d^i\} = \{&(-0.5424, 2.3708), (0.2202, 2.9871), (0.9761, 3.4633),\\
&(1.0618, 36380), (0.8835, 3.7226), (0.5629, 3.7156),\\
&(0.1744, 3.6128), (-0.2338, 3.4206), (-0.6315, 3.1536),\\
&(-1.0, 2.8284), (-1.3251, 2.4600), (-1.5922, 2.0622),\\
&(-1.7844, 1.6539), (-1.8872, 1.2654), (-1.8942, 0.9448),\\
&(-1.8096, 0.7665), (-1.6349, 0.8522), (-1.1587, 1.6081)\}\\
\{\theta_2^i\} = \{&0, 15, 40, 60, 80, 100, 120, 140, 160, 180,\\
&200, 220, 240, 260, 280, 300, 320, 345\}\\
&\rightarrow \theta_2^i = \theta_2 + \delta_2^i
\end{aligned} \tag{6.19}$$

在停顿位置处的输入输出角度的关联如下：

$$\begin{aligned}
\theta_2^i = \theta_2^1 + \{160, 180, 200, 220\} \rightarrow \theta_6^i = 210\\
\theta_2^i = \theta_2^1 + \{345, 0, 15\} \rightarrow \theta_6^i = 225
\end{aligned} \tag{6.20}$$

注意，上述涉及到角度的数值单位均为 deg，而非弧度。

算法参数为 $d_0 = 0.05, \delta_0 = 0.025, c_1 = 0.9999, c_2 = 0.5, q = 10, T_{max} = 50000$，动画结果如图**??**。

img/case6.gif

与其他算法对比结果如图6.8, 图6.9和表6.6。

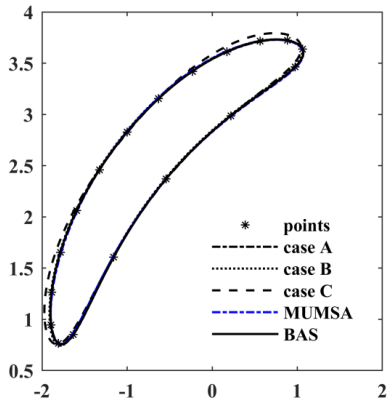


图 6.8: 各算法优化轨迹

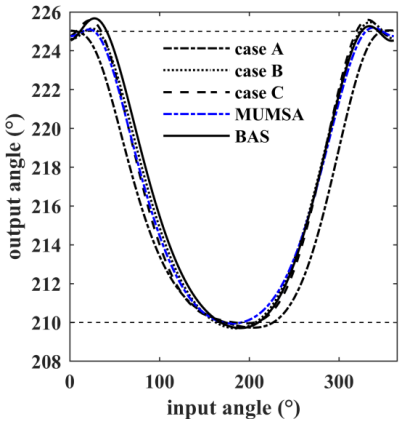


图 6.9: 各算法优化角度

### 6.4.7 Case7 无规定时间的路径生成

本案例为一个'8' 型路径生成问题，轨迹由 12 个点规定。

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^{12}]$$

目标点坐标：

表 6.6: case6 各算法结果对比

| | DE.A | DE.B | DE.C | MUMSA | BAS |
|---|---|---|---|---|---|
| $r_1$ | 1.8145 | 1.8065 | 2.0926 | 1.713529 | 1.8380584 |
| $r_2$ | 0.9911 | 0.9826 | 1.1464 | 0.92602 | 1.0060976 |
| $r_3$ | 1.9995 | 2.0177 | 1.989 | 1.991373 | 1.9978006 |
| $r_4$ | 2.0315 | 2.0009 | 1.9727 | 1.848672 | 2.0084721 |
| $r_5$ | 4.3674 | 5.7769 | 6.6633 | 5.35498 | 6.1065867 |
| $r_6$ | 2.4924 | 2.5296 | 2.5517 | 2.54979 | 2.5514921 |
| $r_p$ | 2.8174 | 2.8711 | 2.7178 | 2.975936132 | 2.8150412 |
| $ _p$ | 0.7776 | 0.783712 | 0.845246 | 0.831651258 | 0.7838698 |
| $ _0$ | 6.269879 | 0.011582 | 6.242260827 | 0.067677 | 6.2774277 |
| $r_1'$ | 4.4158 | 5.2817 | 6.2907 | 4.87374 | 5.6382822 |
| $ _0'$ | 0.235595 | 0.0170309 | 6.18118303 | 0.0967027 | 6.2578638 |
| $x_0$ | 0.0115 | 0.0415 | -0.2729 | 0.175257 | -0.013849 |
| $y_0$ | 0.0157 | -0.0377 | 0.0931 | -0.1187032 | 0.0115919 |
| $ _2^1$ | 0.00052 | 0.003648 | -0.04062 | 6.222361 | 6.2817682 |
| $Evaluations$ | 53310 | 93405 | 93405 | 93405 | 50000 |
| $Error$ | 0.000250876 | 0.005653158 | 0.03537533 | 0.0014 | 0.0001952 |

$$\{C_d^i\} = \{(4.15, 2.21), (4.50, 2.18), (4.53, 1.83), (4.13, 1.68), (3.67, 1.58), (2.96, 1.33), (2.67, 1.06), (2.63, 0.82), (2.92, 0.81), (3.23, 1.07), (3.49, 1.45), (3.76, 1.87)\} \tag{6.21}$$

参数约束：

$$\begin{aligned} &r_1 \in [0, 5] \quad r_2, r_3, r_4 \in [0, 10] \quad r_p \in [0, 14], \\ &x_0, y_0 \in [-80, 80] \quad \theta_0, \theta_2^1, \cdots, \theta_2^{12}, \theta_p \in [0, 2\pi] \end{aligned} \tag{6.22}$$

表 6.7: case7 各算法结果对比

| | PSO | HPSO | MKH | BAS |
|---|---|---|---|---|
| $r_1$ | 4.550300 | 4.535900 | 2.8764500 | 2.8162769 |
| $r_2$ | 1.101300 | 1.113300 | 1.1464400 | 1.1382629 |
| $r_3$ | 3.955800 | 14.738100 | 4.4077300 | 4.0509922 |
| $r_4$ | 3.933000 | 16.801700 | 4.7571300 | 4.1957758 |
| $r_p$ | 3.941572 | 3.941866 | 2.6665746 | 2.6682801 |
| $ _p$ | -0.539171 | -1.521104 | -1.1363513 | 5.1910788 |
| $ _0$ | 0.000000 | 0.000000 | 0.1650200 | 0.2070603 |
| $x_0$ | 0.000000 | 0.000000 | 1.1426500 | 1.1226995 |
| $y_0$ | 0.000000 | 0.000000 | 0.4823700 | 0.5246662 |
| $ _2^1$ | -0.201400 | -0.181600 | -0.3139900 | 6.1191060 |
| $Error$ | 0.171600 | 0.096400 | 0.0001597 | 0.0000994 |

算法参数为 $d_0 = 0.05, \delta_0 = 0.025, c_1 = 0.9995, c_2 = 0.5, q = 40, T_{max} = 20000$，动画结果如图**??**。

img/case7.gif

与其他算法对比结果如图6.10和表6.7。

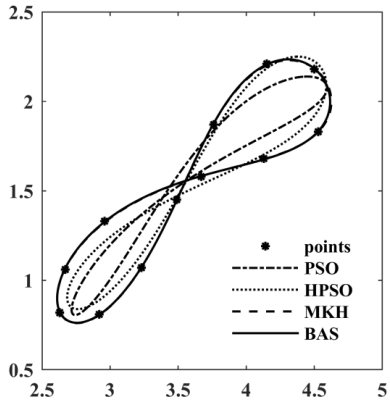


图 6.10: 各算法优化轨迹

### 6.4.8 Case8 无规定时间的路径生成

本案例为一个叶形路径生成问题（无规定时间），轨迹由 25 个点规定。

设计参数为：

$$X = [r_1, r_2, r_3, r_4, r_p, \theta_p, \theta_0, x_0, y_0, \theta_2^1, \cdots, \theta_2^{25}]$$

目标点坐标：

$$\begin{aligned}\{C_d^i\} = \{&(7.03, 5.99), (6.95, 5.45), (6.77, 5.03), (6.4, 4.6), (5.91, 4.03),\\ &(5.43, 3.56), (4.93, 2.94), (4.67, 2.6), (4.38, 2.2), (4.04, 1.67),\\ &(3.76, 1.22), (3.76, 1.97), (3.76, 2.78), (3.76, 3.56), (3.76, 4.34),\\ &(3.76, 4.91), (3.76, 5.47), (3.8, 5.98), (4.07, 6.4), (4.53, 6.75),\\ &(5.07, 6.85), (5.05, 6.84), (5.89, 6.83), (6.41, 6.8), (6.92, 6.58)\}\end{aligned} \tag{6.23}$$

参数约束：

$$r_1, r_2, r_3, r_4 \in [0, 5] \quad r_p, x_0, y_0 \in [-5, 5] \quad \theta_0, \theta_2^1, \cdots, \theta_2^{25}, \theta_p \in [0, 2\pi]$$

算法参数为 $d_0 = 0.05, \delta_0 = 0.025, c_1 = 0.99975, c_2 = 0.5, q = 40, T_{max} = 50000$，动画结果如图**??**。

img/case8.gif

与其他算法对比结果如图6.11和表6.8。

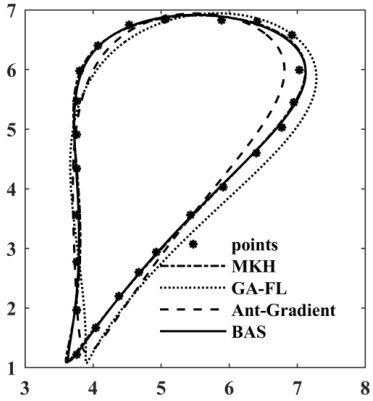


图 6.11: 各算法优化轨迹

表 6.8: case8 各算法结果对比

| | MKH | GA.FL | Ant.Gradient | BAS |
|---|---|---|---|---|
| $r_1$ | 9.99432 | 9 | 13.08 | 9.6132193 |
| $r_2$ | 1.93027 | 3.01 | 1.89 | 1.9459840 |
| $r_3$ | 4.57242 | 8.8 | 8.41 | 4.9080817 |
| $r_4$ | 7.36674 | 8.8 | 6.75 | 6.6768279 |
| $r_p$ | 8.04254 | 11.0995 | 14.45 | 8.6409226 |
| $ _p$ | 0.187430 | -0.681 | 0.195 | 0.1694076 |
| $ _0$ | -0.61763 | 0.489 | -0.3815 | 5.7260733 |
| $x_0$ | -2.31301 | -2.4 | -8.77 | -2.9214626 |
| $y_0$ | 2.86189 | -4 | 1.20 | 2.7190256 |
| $ _2^1$ | * | * | * | 0.4529787 |
| $Error$ | 0.03916 | 0.9022 | 0.5504 | 0.0397777 |

# 第七章　Matlab 用户

之前一直有同学在群里面反映，有没有 matlab 用户可以使用 rBAS 的方法。尽管我曾经也是 matlab 的忠实用户，但现在能力退化严重，实在是写不动代码。加之，我认为学习 R 的成本很低，而且更加便捷。因此，也就熄了再去做一个 matlab toolbox 的心思。

不过，我还是决定尝试下是否存在两者的接口，可以完成参数的传递，让使用 matlab 的同学也可以使用 rBAS。于是有了下面几点思路。

- 使用 matlab，通过 R(D)COM 服务器来与 R 通讯，其配置对于 R 用户而言尚且显得麻烦，何况 matlab 用户。不过在看 mathwork 的网站时，倒是得知他们把 R interface 作为了未来的开发计划之一。所以，日后可以采用在 matlab 中调用 R 的形式。目前尚无此功能。
- 使用 R 中的 R.matlab 包，来与 matlab 通讯。这个办法咋一看很简单，尝试也成功了。最大的问题是**慢**，其次看源码应该是要配置 **Java**(我电脑上原本有，执行无碍，但没有删除 Java 后尝试)。这种慢，可以称为很慢，后面我们能见识到。
- 使用**第三方语言**，如 C/C++。别害怕，因为 matlab 提供了很强大的工具，即 `Matlab Coder` 来把你的 matlab 代码转为 C 或者 C++。

下面我们讲第二点和第三点是如何操作的。

## 7.1　使用 R.matlab 包

这一节更类似于娱乐和探索。因为该方法用于优化求解的速度，真的让人难以接受（也有可能是我没有找对方法）。如果无意于此，可以跳过。

首先，可以按照下面的代码从 CRAN 安装该包。

```
install.packages('R.matlab')
```

当然，也可以利用 devtools 从 github 上按照该包。

```
devtools::install_github('HenrikBengtsson/R.matlab')
```

接下来，按照包的文档[1]进行操作。

首先，加载包。

```
library('R.matlab')
```

在 R 中打开 matlab server。

```
Matlab$startServer()
```

效果就是可以看到 matlab 窗口以最小化的形式出现，不过里面没有视图，大家可以不必理会。后续操作均在 R 中完成。

其次创建 matlab 对象，用于与 Matlab 通讯。

```
matlab <- Matlab()
```

然后，把要优化的函数（**使用 matlab 定义**），写入 R。

```
setFunction(matlab, " \
  function z=test(x) \
            %test function \
            z = 100 *(x(2) - x(1).^2).^2+(x(1)-1).^2;\
            ")
```

[1] https://cran.r-project.org/web/packages/R.matlab/R.matlab.pdf

这段代码，在 matlab 的本地服务器创建了一个 M 函数，如下所示。注意用\号的使用。

```
function z =test(x,y)
  %test function
  z=100*(y-x.^2).^2+(x-1).^2;
```

接下来我们创建目标函数。

```
func_MR <- function(x){
  setVariable(matlab,x = x) # 把 R 里面的 x，赋值到 matlab 里面的 x 变量
  evaluate(matlab,'z = test(x)') # 在 matlab 里面执行 z = test(x) 命令
  res <- getVariable(matlab,c('z')) # 从 matlab 里面读取 z 变量
  return(res$z) # 返回 z 值
}
```

这段代码块思路是，既然利用的是 rBAS 包，那么每次更新，都利用上一回合 R 计算的值，传递到 matlab 里面。然后，在 matlab 里面执行用户定义的函数。最后再用 R 把 matlab 里面算出来的变量值读取出来。

这就是我们新的优化函数，虽然绕一点，但是是容易想到和理解的办法。接下来就是使用 rBAS 包来优化这个目标函数了。

```
system.time({
  fit<-
    BSOoptim(fn = func_MR,
             init = NULL,
             s = 2,
             lower = c(-50,-50),
             upper = c(50,50),
             n = 1,
             w = c(0.9,0.4),
             w_vs = 0.4,
```

```
            step = 10,
            step_w = c(0.9,0.4),
            c = 8,
            v = c(-5.12,5.12),
            trace = T,seed = 1)
})
```

我们使用了一段 BSO 算法的优化代码，但是，粒子数目设置为 2，回合数只设置为 1。结果，这么简单的计算，要花费 **13s**。两个软件来回通讯占用了大量的时间，我相信，应该不会有用户愿意等待这么长久的时间。这个完整的优化过程，之后会有纯 R 代码，求解过程不到 0.1s 的时间。

最后呢，需要使用下面的代码关闭 matlab 服务器。

```
close(matlab)
```

R 代码使用 BSO 解这个优化问题是这样的效果。

```
library(rBAS)

test <- function(x){
  z <- 100*(x[2]-x[1]^2)^2 + (x[1] - 1)^2
}

system.time({
  fit<-
    BSOoptim(fn = test,
             init = NULL,
             lower = c(-50,-50),
             upper = c(50,50),
             n = 300,
             w = c(0.9,0.4),
             w_vs = 0.4,
```

```
            step = 10,
            step_w = c(0.9,0.4),
            c = 8,
            v = c(-5.12,5.12),
            trace = F,seed = 1)
})
```

```
##    user  system elapsed
##    0.07    0.00    0.08
```

```
fit$par
```

```
## [1] 1 1
```

```
fit$value
```

```
## [1] 0
```

时间短，而且能求出理论的最小值。这也反映了第二个方法过于低效的问题，我们看看借助第三方语言能否行得通。

## 7.2 中转站：C/C++

matlab 的 `matlab Coder` 可以让用户把自己的代码变成 C 或者 C++，前提是，需要有编译器。不过，如果你使用过 R，那么一般都会安装 Rtools[2]，这就意味着你安装了 MinGw。下面的过程需要用到 Rtools，还请大家自行下载安装。

打开 matlab，输入下面的指令：

[2] https://cran.r-project.org/bin/windows/Rtools/

```
mex -steup
```

如果电脑中只安装过 Rtools，而没有 VS 之类的编译器，一般会显示图7.1的界面。

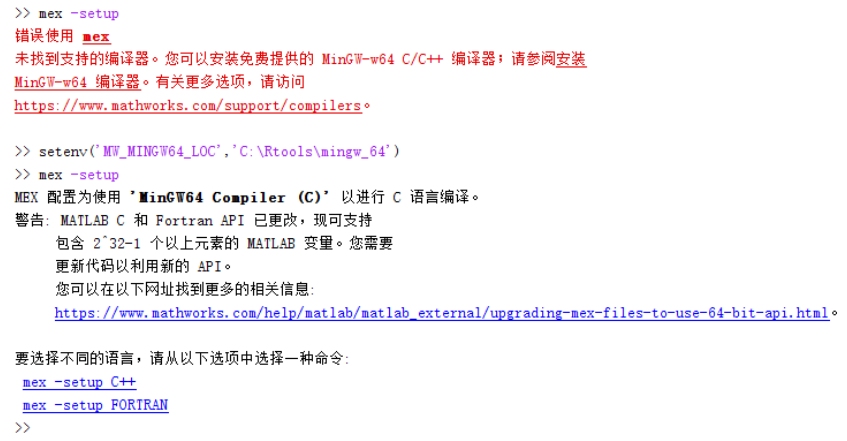


图 7.1: mex -setup

第一句报错，是因为我并没有把 Rtools 中的 mingw_64 文件夹加入环境变量。因此，在执行第二句之后，再执行 `mex -setup` 就不会报错。

好了，加入你成功了，那么抛开前面所有的知识不看。现在说 Coder 工具箱。

假设你的路径下有 2 个 M 文件，一个是我们之前所说的测试函数 `test.m`，还有一个是我们所说的，用来测试 `test` 函数的文件 `test_main.m` 文件。它们的代码如下。

```
% -------------test.m-----------------%

function z = test(x)
z = 100 *(x(2) - x(1).^2).^2+(x(1)-1).^2;
```

```
% -------------test_main.m-----------------%

z = test([1 1]);
```

接下来我们点击 Matlab 菜单栏上的 APP，在里面找到 Matlab Coder 工具箱打开，如图7.2。

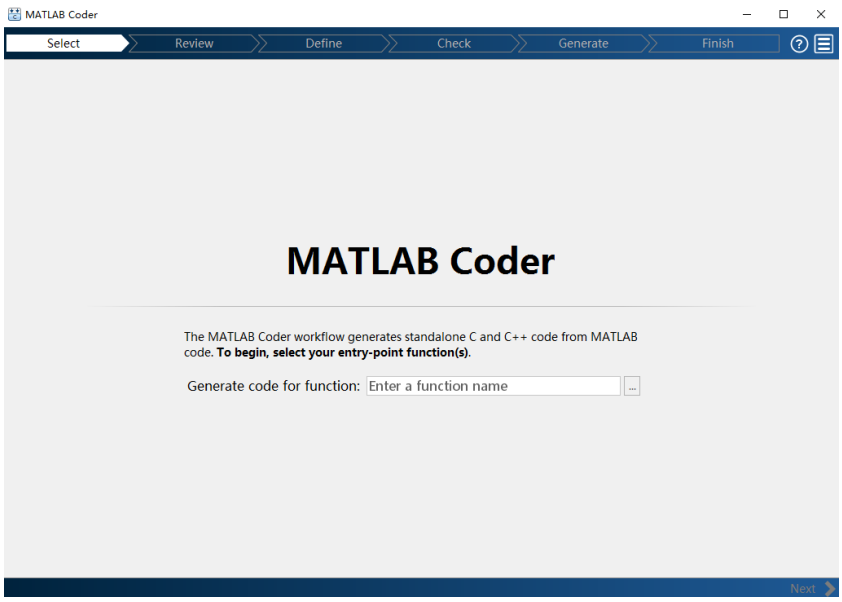


图 7.2: Matlab Coder

点击 Enter a function name 右边的 ··· 符号，然后选中 `test.m` 文件，可以看到图7.3的效果。

点击右下角 `Next` 按钮，可以看到如图7.4的界面。

点击图7.4红色箭头所指处，并选中 `test_main.m` 文件，点击 `Autodefine Input types`，可以看到如图7.5的界面，接下来就是一路的 `Next`。

直到图7.7，点击 `Generate` 按钮，生成 C 代码。

而后工具箱会给出 `test.c` 代码的预览，如图7.8。

我相信，大家还是能看懂这段代码。对于简单的优化问题，我们照猫画

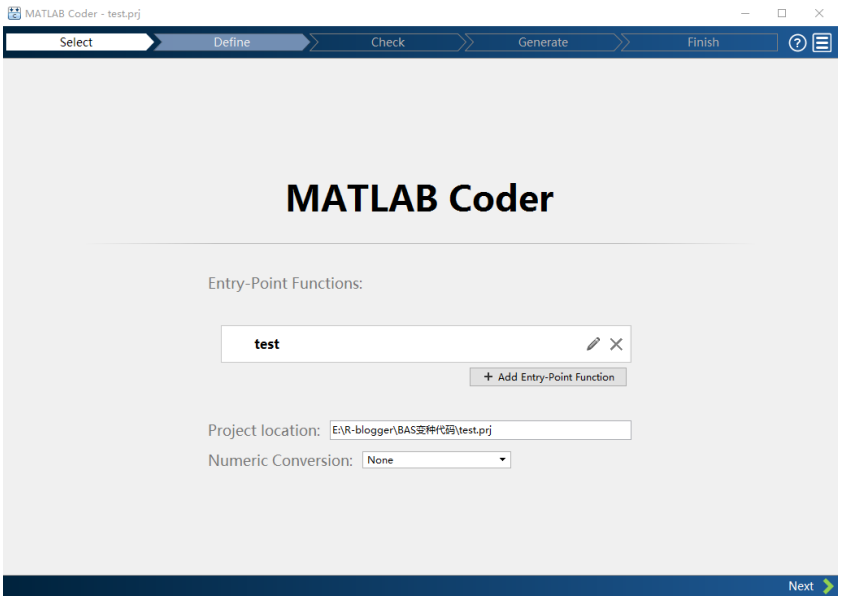


图 7.3: Matlab Coder

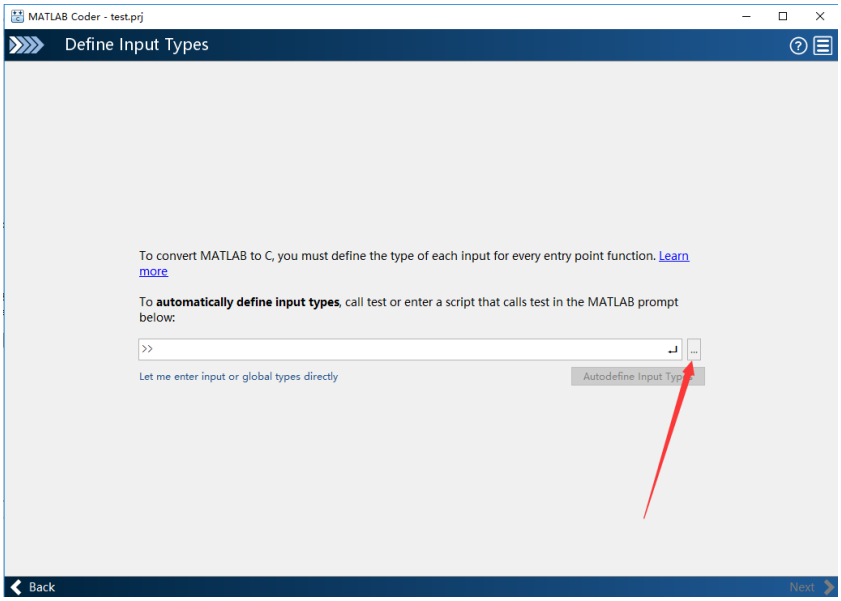


图 7.4: Matlab Coder

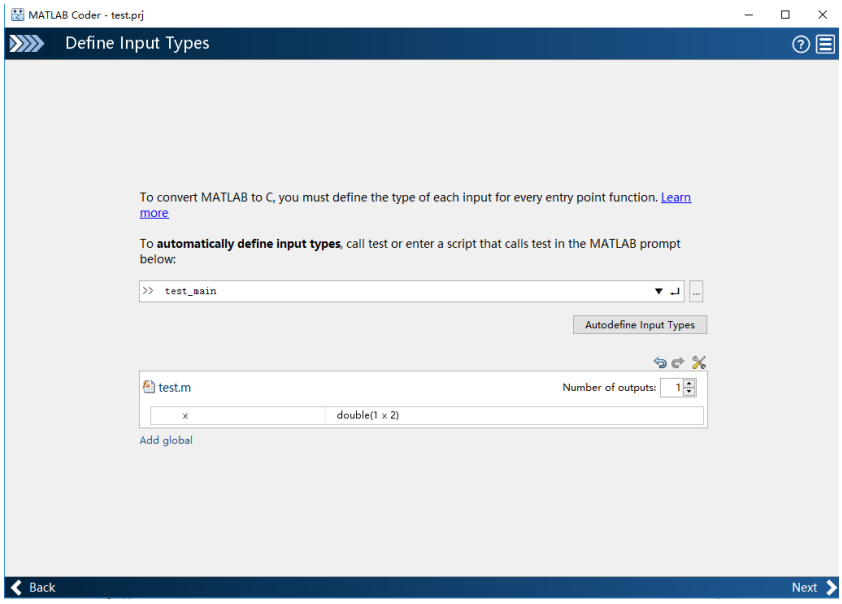


图 7.5: Matlab Coder

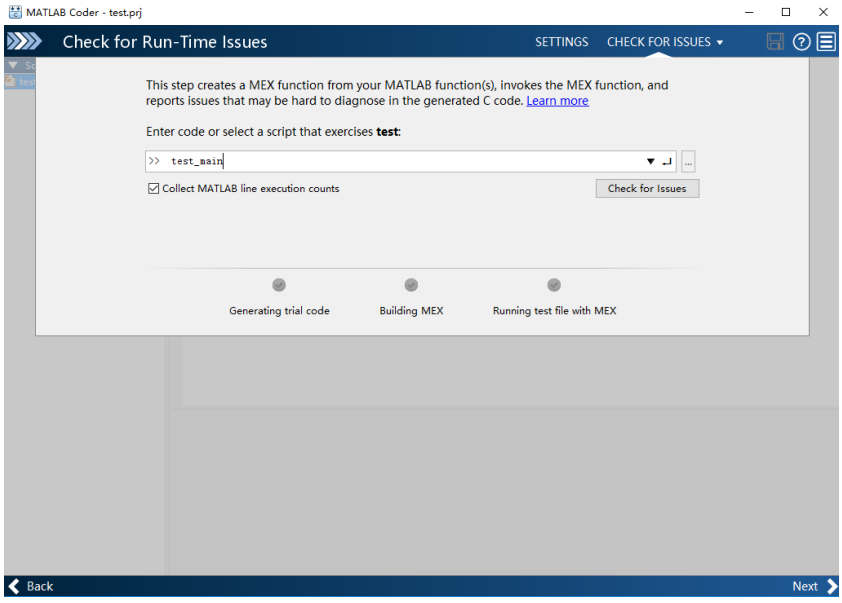


图 7.6: Matlab Coder

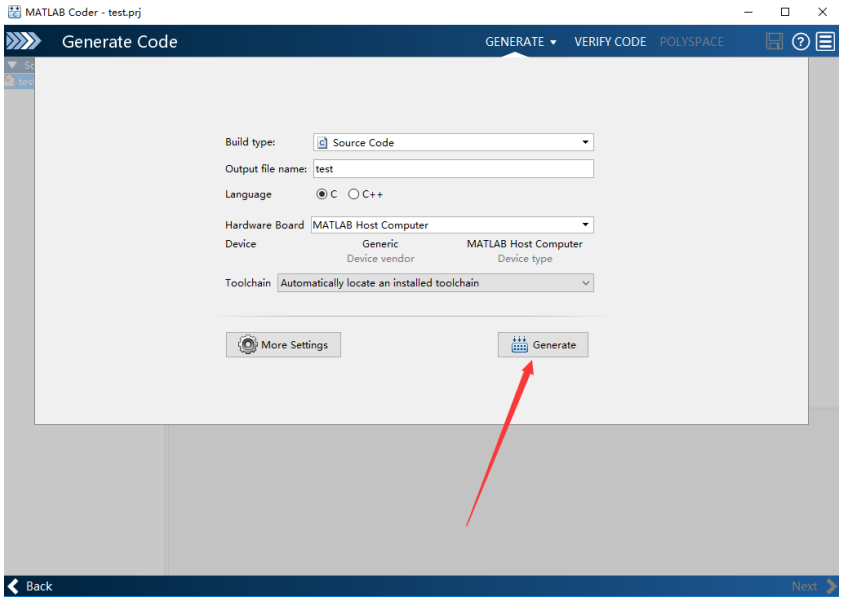


图 7.7: Matlab Coder

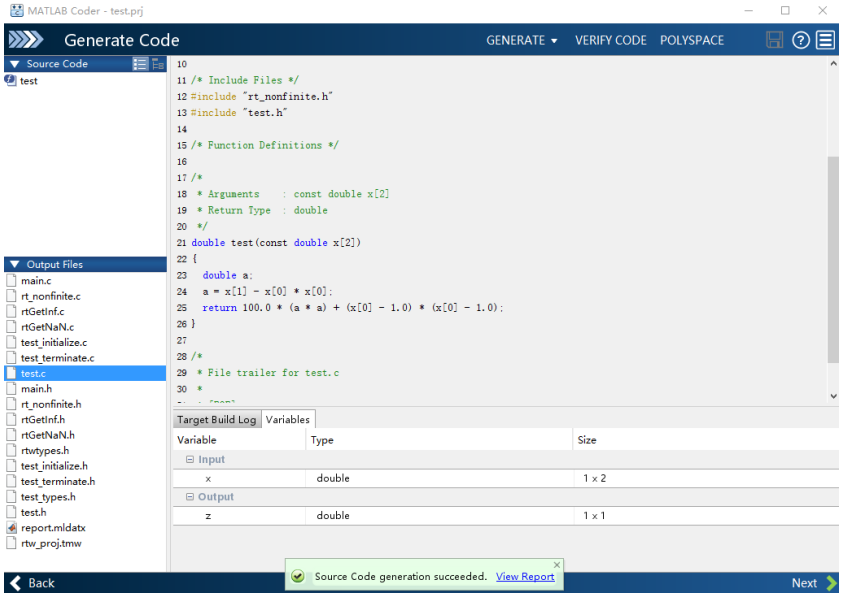


图 7.8: Matlab Coder

虎，就可以写出对应的 C 或者 C++ 的代码。如果不想写，还可以按照上述的方法生成。

那现在问题来了，如果我们有了 C 代码，如何在 R 语言中用了。

由于 matlab 生成的代码很多库都是 matlab 依赖的（我这么认为，不一定正确），所以我们需要简单地调整一下，把代码变得简洁。

原始的代码是这样的。

```
/* Include Files */
#include "rt_nonfinite.h"
#include "test.h"

/* Function Definitions */

/*
 * Arguments    : const double x[2]
 * Return Type  : double
 */
double test(const double x[2])
{
  double a;
  a = x[1] - x[0] * x[0];
  return 100.0 * (a * a) + (x[0] - 1.0) * (x[0] - 1.0);
}

/*
 * File trailer for test.c
 *
 * [EOF]
 */
```

我们进行**微调**（大家肯定是能理解并模仿的），变成 R 可以正确编译的。注意，由于 Rcpp 能让 R 和 C++ 无缝贴合的强大功能，我把代码调整为了 C++。后面的 `Michalewicz function`,`Pressure Vessel function`

均是用 C++ 所写。

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
double test(NumericVector x) {
  double a;
  a = x[1] - x[0] * x[0];
  return 100.0 * (a*a) + (x[0] - 1.0) * (x[0] - 1.0);
}
```

把这段代码保存成 Rcpptest.cpp 文件，然后在 R 里面执行代码：

```
library(Rcpp) # 加载 Rcpp 包
sourceCpp(' 你的 Rcpptest.cpp 文件路径') # 编译之前保存的 cpp 文件
```

```
# 执行 BSO 优化算法函数
system.time({
  fit<-BSOoptim(fn = test, # 调用了编译的 C++ 中的 test 函数
                init = NULL,
                lower = c(-50,-50),
                upper = c(50,50),
                n = 300,
                w = c(0.9,0.4),
                w_vs = 0.4,
                step = 10,
                step_w = c(0.9,0.4),
                c = 8,
                v = c(-5.12,5.12),
                trace = F,seed = 1)
})
```

```
##    user  system elapsed
##    0.09    0.00    0.09
```

```
# 查看优化参数
fit$par
```

```
## [1] 1 1
```

```
# 查看优化结果
fit$value
```

```
## [1] 0
```

执行速度很快，大家可能会有疑问，为啥看起来比纯 R 代码执行的时间还要长一点。这是因为你多次执行，耗时和电脑所处的状态有关系，这点误差是正常的。但是，如果是计算量较大的代码，利用 **Rcpp 会比 R 要节省大量的时间**。

接下来我们看看另外两个例子，也就是 `Michalewicz function` 和 `Pressure Vessel function` 函数的优化问题。

`Michalewicz function` 也可以先用 matlab 生成 C 或者 C++ 代码，然后稍作改编，保存为 `Rcppmich.cpp` 文件，代码如下：

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double michcpp(NumericVector x) {
  double x0;
  double x1;
  x0 = -sin(x[0])*pow(sin((pow(x[0],2))/M_PI),20);
  x1 = -sin(x[1])*pow(sin((2*pow(x[1],2))/M_PI),20);
  return x0+x1;
}
```

和 matlab 或者 R 代码比起来，也是十分易懂的。

```
library(Rcpp) # 加载 Rcpp 包
sourceCpp(' 你的 Rcppmich.cpp 文件路径') # 编译之前保存的 cpp 文件
```

```
# 执行 BAS 优化算法函数
system.time({
  fit<-BASoptim(fn = michcpp, # 利用了 C++ 中的 michcpp 函数
                lower = c(-6,0), upper = c(-1,2),
                seed = 1, n = 100,trace = FALSE)
})
```

```
##    user  system elapsed
##       0       0       0
```

```
# 查看优化参数
fit$par
```

```
## [1] -4.964687  1.575415
```

```
# 查看优化结果
fit$value
```

```
## [1] -1.966817
```

对于 Pressure Vessel function，保存为 RcppPV.cpp 文件，代码如下：

```
#include <Rcpp.h>
using namespace Rcpp;
```

```
// [[Rcpp::export]]
double PVobj(NumericVector x) {
  double x0 = floor(x[0])*0.0625;
  double x1 = floor(x[1])*0.0625;
  double x2 = x[2];
  double x3 = x[3];

  double result = 0.6224*x0*x2*x3 + 1.7781*x1*pow(x2,2)
    +3.1611*pow(x0,2)*x3 + 19.84*pow(x0,2)*x2;
  return result;
}

// [[Rcpp::export]]
NumericVector PVcon(NumericVector x) {
  double x0 = floor(x[0])*0.0625;
  double x1 = floor(x[1])*0.0625;
  double x2 = x[2];
  double x3 = x[3];

  NumericVector constraint(3);
  constraint[0] = 0.0193*x2 - x0;
  constraint[1] = 0.00954*x2 - x1;
  constraint[2] = 750.0*1728.0 - 3.141593*pow(x2,2)*x3
    - (4.0/3.0)*3.141593*pow(x2,3);
  return constraint;
}
```

其中，PVobj 是目标函数，PVcon 是约束函数。

```
library(Rcpp) # 加载 Rcpp 包
sourceCpp(' 你的 RcppPV.cpp 文件路径') # 编译之前保存的 cpp 文件
```

```
# 执行 BSO 优化算法函数
system.time({
  fit<-BSOoptim(fn = PVobj, #C++ 中的 PVobj 函数
          init = NULL,
          constr = PVcon, #C++ 中的 PVcon 函数
          lower = c( 1, 1, 10, 10),
          upper = c(100, 100, 200, 200),
          n = 1000,
          w = c(0.9,0.4),
          w_vs = 0.9,
          step = 100,
          step_w = c(0.9,0.4),
          c = 35,
          v = c(-5.12,5.12),
          trace = F,seed = 1,
          pen = 1e6)
})
```

```
##    user  system elapsed
##     1.5     0.0     1.5
```

```
# 查看优化参数
fit$par
```

```
## [1]  13.948846   7.541387  42.098446 176.636570
```

```
# 查看优化结果
fit$value
```

```
## [1] 6059.131
```

结果很好，耗时也可以接受。不过，在写 BSOoptim 函数的时候，我使用了 2 层的循环来嵌套，因此，BSOoptim 函数比 rBAS 包中其他的函数要慢上一些。后续我打算使用 Rcpp 把 BSOoptim 重写一遍，这样对大型的优化问题耗时会更容易接受。

下面是 BSASoptim 对该问题的优化结果，同样也是采用 C++ 编译的方式。

```
system.time({
  result <- BSASoptim(fn = PVobj,
                      k = 5,
                      lower =c( 1, 1, 10, 10),
                      upper = c(100, 100, 200, 200),
                      constr = PVcon,
                      n = 1000,
                      step = 100,
                      d1 = 5,
                      pen = 1e6,
                      steptol = 1e-6,
                      n_flag = 2,
                      seed = 2,trace = FALSE)
})
```

```
## ----step < steptol---------stop the iteration------

##    user  system elapsed
##    0.32    0.00    0.31
```

```
result$par
```

```
## [1]  14.908531   7.652175  43.541638 159.543000
```

```
result$value
```

```
## [1] 6305.57
```

# 第八章　更新及维护计划

## 8.1　待加入的功能

算法:

- ~~加入 BSAS~~
- ~~加入 BSAS-WPT~~
- 加入 binary BAS (阮月)
- 加入二阶 BAS (李晓晓)
- ~~add BSO(Beetle Swarm Optimizaiton)~~ (王甜甜)
- …

应用:

- 工程应用:
  - ~~多杆件机构优化~~ (莫小娟)
  - 建筑系统阻容模型辨识
  - 装配路径规划
  - 批量问题 (binary BAS)
  - …
- 基准测试
  - 计划超过 20 个基准测试
  - …

用户界面:

- 基本界面
  - ~~基本的 shiny 界面~~
  - ~~更新子约束处理功能~~
  - ...
- 自动文档
  - 基于 rmarkdown[1]的自动文档报告生成
  - 文档导出
  - ...

算法部分与用户界面将会在 rBAS 包中不断更新。应用方面虽然有计划，但是除却基准测试外，更多的需要各位同学们的贡献。这部分暂时会选择几个重要的应用集成在 rBAS 包的案例库中，全部内容则会在本手册中更新。

## 8.2 联系方式

1. 大家可以加入 QQ 群 (437958608) 来讨论涉及 BAS 算法的各种问题。
2. 更进一步，如果大家有意愿将自己的研究纳入 rBAS 包或者是手册的应用案例上，欢迎大家给我发送邮件 (jywang2016@hust.edu.cn) 或者群内私信。具体的代码（如果大家愿意开源的话）或者文档形式（没有代码也是十分欢迎的）都可以具体商议。我也会尽量尝试将大家的 matlab 或者 python 代码复现为 R，所以“语言阻碍”暂时还不是问题。
3. 如果对 rBAS 有什么建议，或者 bugs，欢迎大家在 issues[2]上发表评论。

[1] https://github.com/rstudio/rmarkdown

[2] https://github.com/jywang2016/rBAS/issues

# 结语

暂无