

## 第二章 基于实验设计、代理模型、AI 增强的优化方法

层流机翼及增升装置的全局气动优化问题的特点是设计参数多，响应面多峰值，非线性，这也是许多复杂气动设计优化问题的典型特点。使用传统的代理模型更新方法，即使使用大量的样本点，由于高维度带来的问题，很难提高代理模型的准确度，优化结果和 CFD 验证结果不匹配。

本章主要介绍基于代理模型的遗传算法优化方法，以及基于 AI 增强的决策加点方法，同时使用 Python 代码构建了整个优化的工具箱 saopy<sup>1</sup>，并使用多种测试函数进行验证，为层流机翼及增升装置的优化建立了基础。

saopy 优化工具箱是使用面向对象的编程方法实现的，各个模块之间相互独立，同时可以相互调用，方便代码的维护和使用。本章所使用的解释说明代码及算法的流程图图例如图2-1所示。矩形代表一个模块，类，或子类；椭圆形为输入或输出的数据；平行四边形为类的方法，并且可以互相调用；白色尖头为输入数据流，黑色尖头为输出数据流。整个优化工具箱的流程图如图2-2所示（该图为矢量图，可以任意放大查看细节），简化流程图如图2-3所示，下面将详细介绍每个模块。

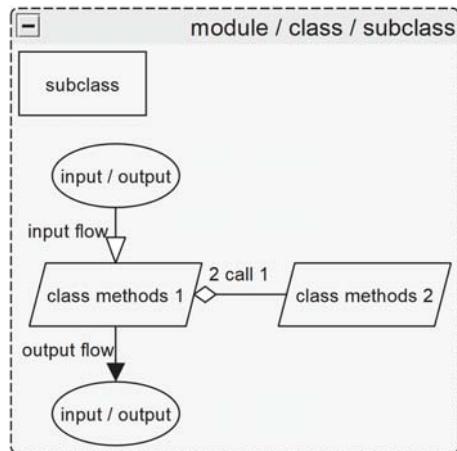


图 2-1 代码流程图图例

Figure 2-1 Flow chart legend

<sup>1</sup><https://github.com/smallball-ljj/saopy>

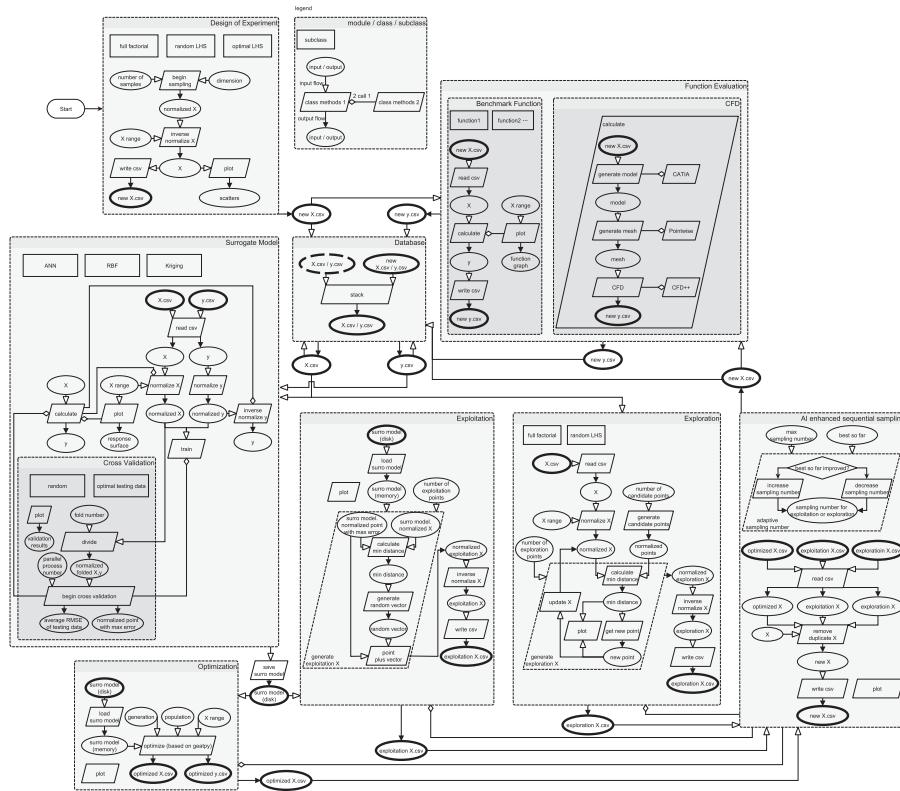


图 2-2 saopy 优化工具箱流程图

Figure 2-2 Flow chart of saopy toolbox

## 2.1 实验设计方法

实验设计方法模块的流程图如图2-4所示，目前提供三种生成样本点的方法：full factorial<sup>[152]</sup>, random LHS<sup>[152]</sup>, optimal LHS<sup>[153]</sup>。模块有两个输入参数，分别是样本点个数和维数，根据所选择的生成样本点的方法，使用对应的 begin sampling 方法，生成归一化的样本点，然后根据给定的参数的范围，反归一化（inverse normalize）后得到最后的样本点 X，同时会生成.csv 文件，方便其他模块的调用。此外，还提供了样本点可视化的方法，使用 plot 方法，可以输出样本点的散点图，更加直观的显示样本点分布情况。

full factorial 方法将每个变量平均分成 n 份，假设维数是 D，那么总共会生成  $n^D$  个样本点。图2-5显示了二维情况生成的样本点示例。这个方法实现简单，并且样本点在设计空间是完全均匀分布的，但是缺点也很明显，随着维数的增加，样本点的个数会成指数增加，无法控制样本点个数，所以此方法只适合维数较低，且样本点计算成本较低的问题。

random LHS 方法首先将每个变量平均分成 n 份，然后进行随机排序，最后生

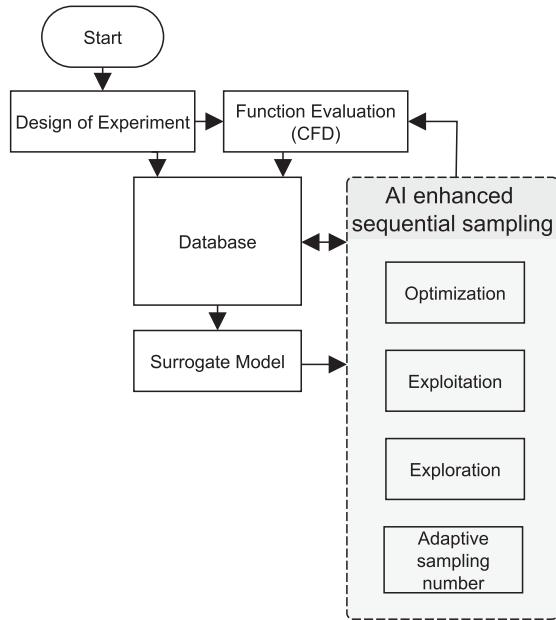


图 2-3 saopy 优化工具箱简化流程图

Figure 2-3 Simplified Flow chart of saopy toolbox

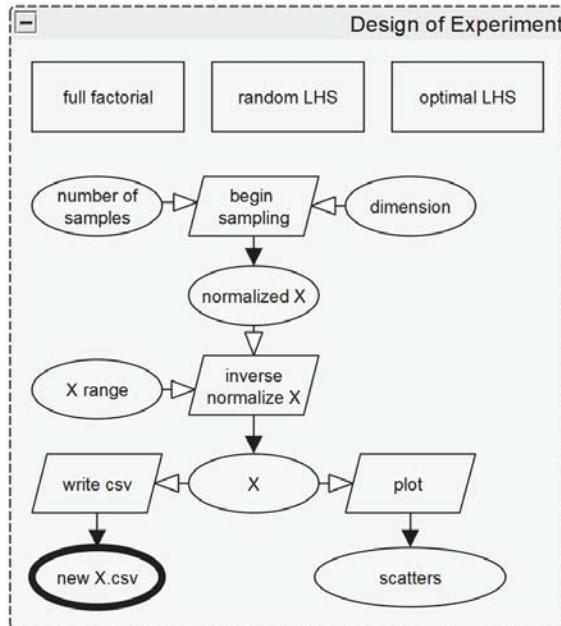


图 2-4 实验设计方法模块流程图

Figure 2-4 Flow chart of Design of Experiment module

成 n 个样本点。图2-6显示了二维情况生成的样本点示例。这个方法相比于使用完全随机的生成样本点的方法，可以使样本点更好的充满整个设计空间，同时也能

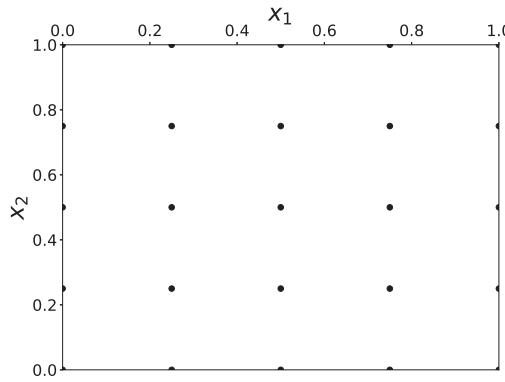


图 2-5 full factorial 方法生成的样本点示例

Figure 2-5 Example of generated samples using full factorial

随意指定所需要的样本点个数。这个方法的缺点是，由于算法本身仍然具有一定的随机性，样本点在设计空间的分布并不会非常均匀，可能会出现某个区域样本点非常密集，而某个区域样本点非常稀疏的情况。

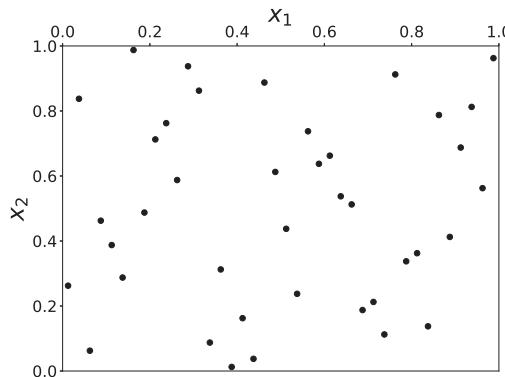


图 2-6 random LHS 方法生成的样本点示例

Figure 2-6 Example of generated samples using random LHS

optimal LHS 针对 random LHS 存在缺点进行了改进，使样本点在设计空间的分布更加均匀，其核心思想是使用最小距离最大的方法，即尽可能使到每个样本点最近的点的距离越大越好。具体实现方法是通过优化算法，使  $\phi$  最小，其中  $d(x_i, x_j)$  表示样本点  $x_i$  和  $x_j$  之间的距离。图2-7显示了二维情况生成的样本点示例，明显比 random LHS 生成的样本点更加均匀。由于需要用到优化算法，对于维

数较高的问题，需要较长的优化时间才能得到较好的效果。

$$\phi = \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{d^k(x_i, x_j)} \right\}^{1/k} \quad (2-1)$$

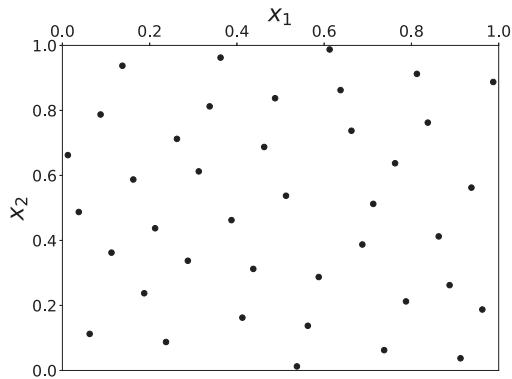


图 2-7 optimal LHS 方法生成的样本点示例

Figure 2-7 Example of generated samples using optimal LHS

## 2.2 函数计算模块

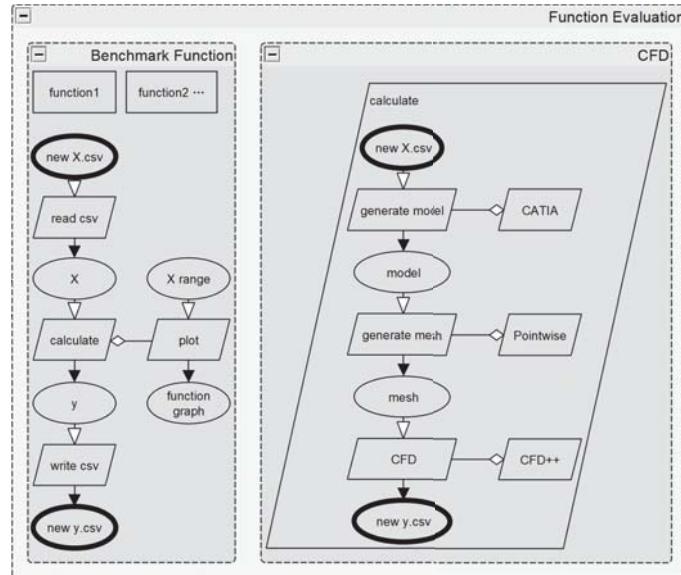


图 2-8 函数计算模块流程图

Figure 2-8 Flow chart of function evaluation module

函数计算模块的流程图如图2–8所示，包括标准测试函数模块（Benchmark Function），及 CFD 计算模块。此模块的输入为实验设计模块生成的样本点 X.csv，或者可以使用其他方法或者人工生成的样本点，然后使用 calculate 方法计算得出每个样本点对应的函数值 y，并生成.csv 文件，方便其他模块的调用。其中标准测试函数模块还具有画出标准函数图像的功能，方便和之后使用代理模型生成的响应面进行对比。CFD 计算模块中会调用 CATIA 生成模型，调用 Pointwise 画网格，调用 CFD++ 进行计算，这些功能都通过 Python 及各个软件的脚本实现自动化完成。

## 2.3 代理模型方法

代理模型模块的流程图如图2–9所示，一共提供三种代理模型方法：人工神经网络（ANN），径向基函数（RBF），和 Kriging。此模块的输入为样本点 X.csv 和通过函数计算模块得到的对应的函数值 y.csv。首先会对 X 和 y 进行归一化，然后根据选择的代理模型，使用对应的方法进行训练，训练完成后会将代理模型保存下来，方便其他模块调用。同时，此模块还具任意维数响应面的可视化功能，例如在画 5 维响应面时，任意选择两个参数作为变量，其余参数默认固定为参数范围的中间值（也可以指定特定值），这样参数两两组合，一共会画出 10 个响应面。图2–33显示了一个 5 维响应面的例子。

另一个子模块是交叉验证模块（Cross Validation）。此模块的功能是将所有数据平均分成 n 份（n 为输入参数 fold number），选择 n-1 份作为训练集，1 份作为测试集，一共会做 n 次不同的训练，最后得到所有测试集的均方根误差（RMSE）的平均值，作为评判代理模型精度的一个标准。同时会输出误差最大的一个样本点，方便后续模块（Exploitation）调用。RMSE 公式如2–2所示， $y_i$  表示真值， $\hat{y}_i$  表示代理模型的预测值。由于 n 次不同的训练之间互不影响，所以这些训练可以并行进行，此模块同样提供了这个功能，根据输入的并行进程数（parallel process number）并行训练，大大提高了运行速度。此外，此模块还提供两种将数据平均分成 n 份的方法，最简单的是随机分成 n 份（random），这个方法运行速度快，但是其缺点是，如果测试集的样本和训练集非常接近，那么得到的均方根误差就会较小，反之，则会较大，不能非常好的反应出模型的真实性能。另一种方法是测试集样本均匀分布的方法（optimal testing data），这个方法的实现过程和实验设计模块中的 optimal LHS 一样，这样可以更好的反应模型的性能。最后 plot 功能可以将所有训练集和测试集的真值和代理模型预测值进行对比，如图2–10所示，这

些点越靠近 45 度直线就代表预测值和真值越接近。

$$RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2} \quad (2-2)$$

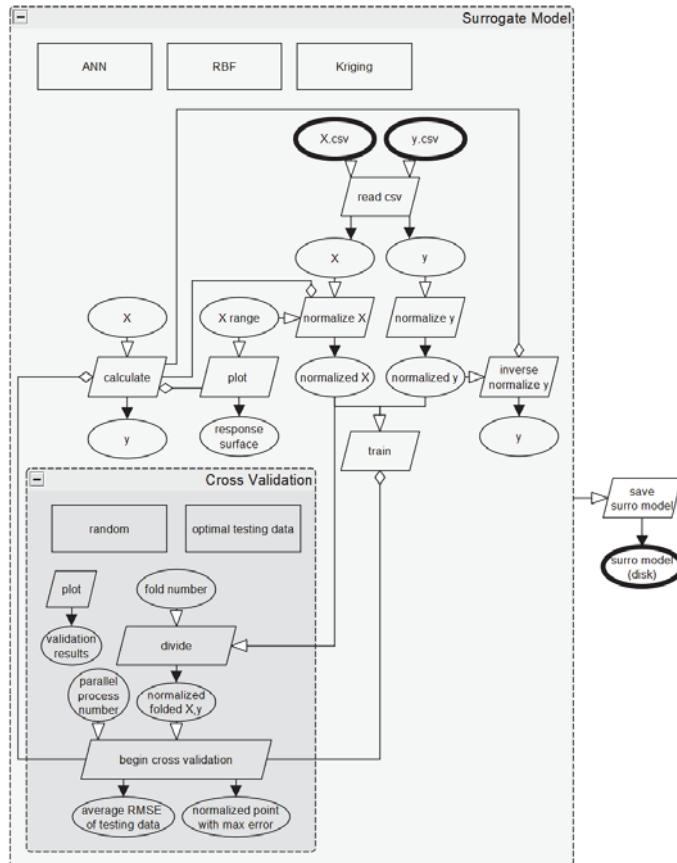


图 2-9 代理模型模块流程图

Figure 2-9 Flow chart of surrogate model module

ANN 是基于 PyTorch<sup>1</sup>建立的，PyTorch 是一个开源的机器学习框架，允许用户利用其高级应用程序编程接口（API）高效的创建和训练不同类型的神经网络。隐含层数和隐含层神经元个数是人工神经网络难以确定的一些参数，目前还没有标准步骤来构建最有效的网络。隐含节点个数固然很重要，因为它们直接关系到模型的性能。然而，隐含节点过多会导致模型泛化能力差，而隐藏节点太少则会导致模型失效<sup>[101]</sup>。所以在训练 ANN 时，首先会测试不同隐含层数和隐含层神经元个数的性能，简单起见，这里使用的每层神经元的个数是相同的（图2-11显示了神

<sup>1</sup><https://pytorch.org/>

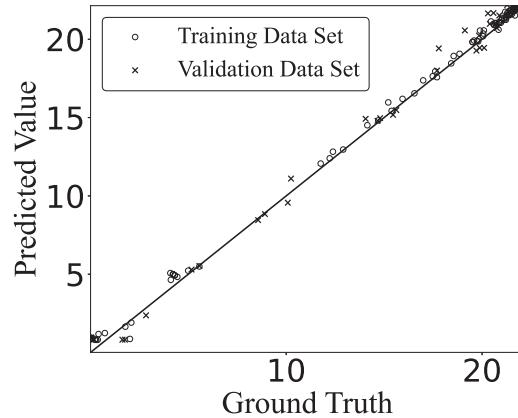


图 2-10 交叉验证训练集和测试集的真值和代理模型预测值对比

Figure 2-10 Comparison between ground truth and predicted value of cross validation

经网络结构的示例, 它有 2 个输入, 2 层隐含层, 每层有 3 个神经元, 1 个输出), 激活函数默认使用 ReLU (程序中也提供了其他激活函数), 训练默认 epoch=10000, 可以保证 RMSE 完全收敛。然后使用交叉验证 (默认将所有数据分成 3 份), 得到所有测试集的均方根误差 (RMSE) 的平均值, 作为评判模型精度的标准, 最后选择 RMSE 最小的神经网络的结构进行使用, 图2-12显示了对于某个测试函数, 使用不同神经网络结构的性能。

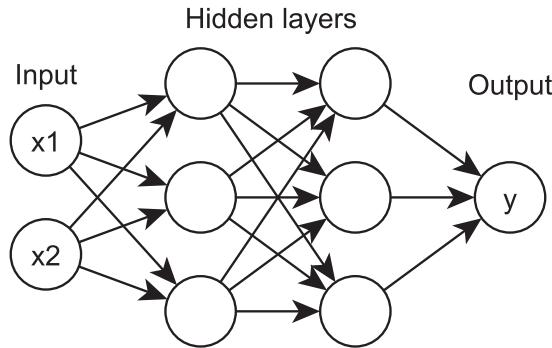


图 2-11 神经网络结构示例

Figure 2-11 Example of ANN architecture

RBF 同样基于 PyTorch 建立, 公式2-3<sup>[105]</sup> 为 RBF 的函数表达式, 其中  $c_i$  为 RBF 的中心点,  $J$  为中心点个数,  $\Phi$  为基函数, 本文默认使用高斯函数 (2-4) 作为基函数 (程序中提供了多种基函数)。如果选择所有训练集作为中心点, 则 RBF 为插值函数, 否则 RBF 为拟合函数。目前 PyTorch 版本的 RBF 为拟合函数, 给定

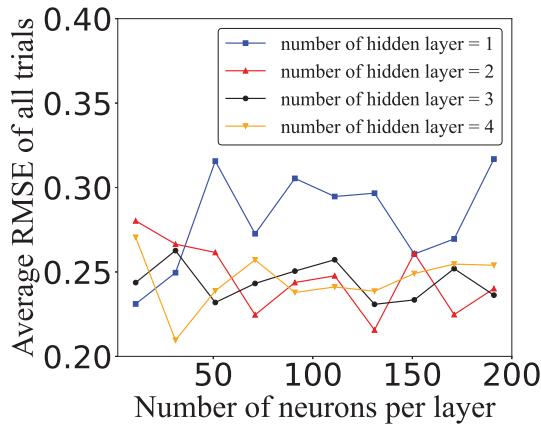


图 2-12 不同神经网络结构的性能

Figure 2-12 Performance of different ANN architecture

中心点的个数后，中心点具体值可以通过训练得到。图2-13显示了 RBF 结构的示例，它与神经网络类似，但是只有一层 RBF 层。

$$y = \sum_{i=1}^J w_i \Phi(||\mathbf{x} - \mathbf{c}_i||) \quad (2-3)$$

$$\Phi(r) = e^{-r^2/2\sigma^2} \quad (2-4)$$

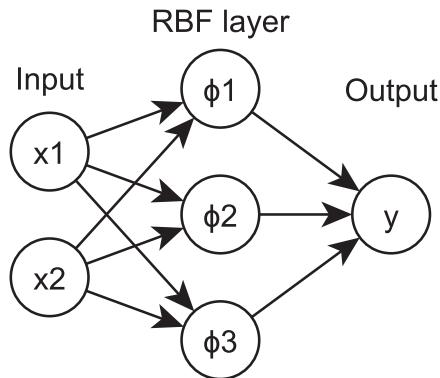


图 2-13 径向基函数结构示例

Figure 2-13 Example of RBF architecture

Kriging<sup>[115]</sup> 的定义方法如下，设  $\{\mathbf{x}^i, y_i, i=1, 2, \dots, n\}$  为训练集，其中  $\mathbf{x} \in \mathbb{R}^d$  为  $d$  维输入， $y \in \mathbb{R}^d$  为输出，对于一个新的点  $\mathbf{x}^*$ ，Kriging 代理模型预测值表达式可

以写为

$$\hat{y}(\mathbf{x}^*) = \beta + \mathbf{r}(\mathbf{x}^*)^T R^{-1}(\mathbf{y} - \mathbf{1}\beta) \quad (2-5)$$

其中

$$R(\mathbf{x}^i, \mathbf{x}^j) = \prod_{k=1}^d \exp(-\theta_k |x_k^i - x_k^j|^{p_k}) \quad (2-6)$$

$$\mathbf{r}(\mathbf{x}^*) = \{R(\mathbf{x}^*, \mathbf{x}^1), \dots, R(\mathbf{x}^*, \mathbf{x}^n)\} \quad (2-7)$$

显然 Kriging 是插值函数， $\theta$  和  $p$  为超参数，对于大多数问题来说，使用  $p=2$  就可以为函数提供足够的光滑程度和非线性程度。Kriging 直接使用现有的开源代码<sup>1</sup>，并对其进行封装，整合到代理模型模块中，方便其他模块调用。

为了针对特定的问题选择较为合适的代理模型，有许多研究对比了不同代理模型之间的性能，包括多项式、Kriging、RBF 和 ANN<sup>[95-110]</sup>。最终的结论是 Kriging、RBF 和 ANN 相比于二次多项式具有更高的精度。然而，对于 Kriging、RBF 和 ANN 之间的优缺点还没有明确的结论，不仅因为其性能可能取决于需要构建代理模型的问题的本身，而且还因为需要考虑多个评价标准。

本节的最后使用了 3 个不同的二维测试函数<sup>[154]</sup> 对比了 3 种上文提到的代理模型的效果，如图2-14-图2-16所示。训练 3 种代理模型的样本都是相同的（以黑点表示），对于 ANN 选择使 RMSE 最小的神经网络的结构，对于 RBF 默认选择 100 个中心点。可以看出，对于相对比较简单的函数 Easom 和 Michaelwicz，三种代理模型都有较好的拟合测试函数的能力，另一方面也验证了这部分代码的可靠性。而对于复杂的多峰函数 Rastrigin，三种代理模型的拟合效果都较差，因此对于实际问题，单纯使用初始样本点建立代理模型是不够的，需要根据模型的性能增加新的样本点，具体方法将在 2.5 节详细介绍。由于 Kriging 的训练和预测时间比 ANN 要慢一个数量级，本文后面部分都使用 ANN 作为代理模型。

## 2.4 遗传算法优化方法

遗传算法优化模块基于现有的开源工具箱 Geatpy<sup>[155]</sup> 建立。Geatpy 是一个高性能遗传算法工具箱，并提供一个高度模块化的面向对象的算法框架，可用于求解单目标优化、多目标优化等问题。Geatpy 提供了各种遗传算法，本文对于单目标使用最简单、最经典的遗传算法 SGA，对于多目标使用效果较好的 NSGA-III<sup>[156]</sup> 算

<sup>1</sup><https://github.com/capaulson/pyKriging>

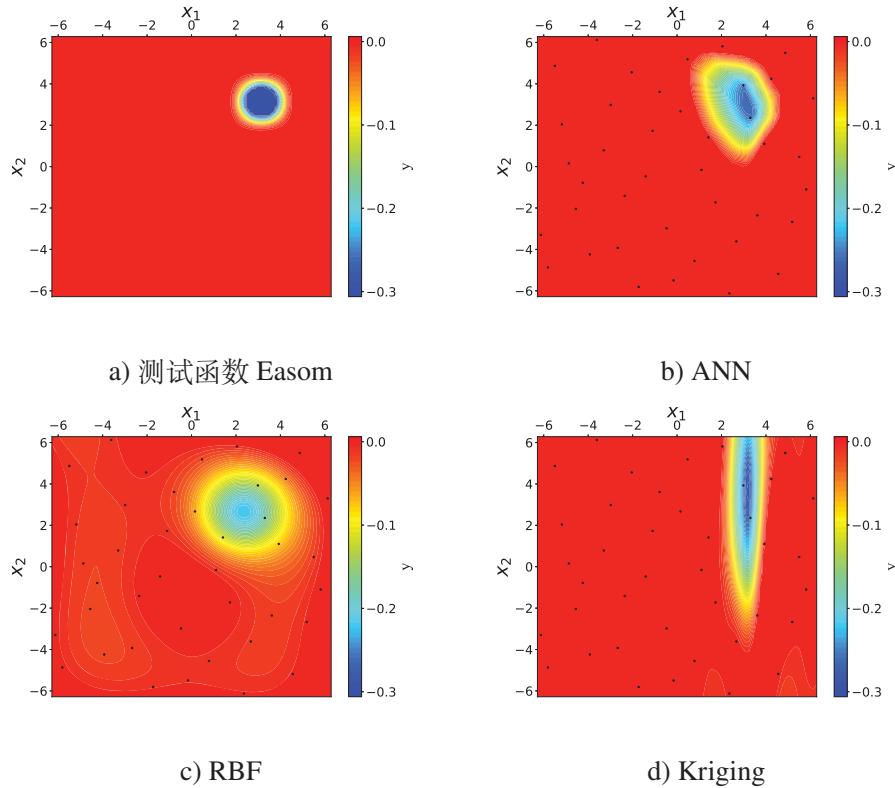


图 2-14 测试函数 Easom 和代理模型对比

Figure 2-14 Comparison between benchmark function Easom and surrogate models

法。遗传算法优化模块流程图如图2-17所示，它的功能是求解代理模型的最优值，同时也能求解具有解析式的测试函数的最优值，用于对比验证。它的输入是代理模型模块中保存下来的已经训练好的代理模型 (surro model)，遗传代数 (generation)，种群规模 (population)，和参数优化范围 (X range)。交叉概率和变异概率使用默认值 0.7 和 0.025。优化完成后会保存最优值 y.csv 和对应的设计变量 X.csv (对于单目标优化会保存最优值；对于多目标优化会保存 Pareto front 上的一系列值，其总数为种群规模)，方便其他模块调用。优化完成后，使用 plot 功能可以画出最优值随遗传代数收敛曲线，用于确定合适的遗传代数，由于代理模型或者测试函数的计算时间几乎可以忽略不计，所以推荐使用较大的遗传代数 (默认使用 5000)，保证优化的收敛。

## 2.5 基于 AI 增强的决策加点方法

2.3节中提到对于实际问题，单纯使用初始样本点建立代理模型是不够的，需要根据模型的性能，和使用遗传算法优化的效果，增加新的样本点。本节介绍基

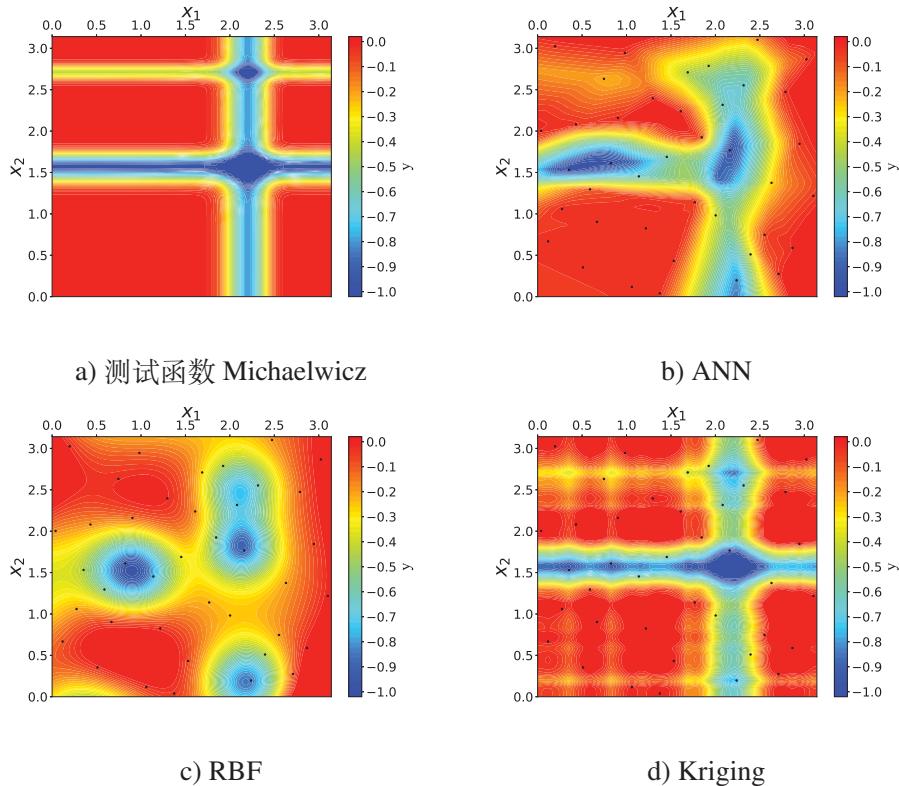


图 2-15 测试函数 Michaelwicz 和代理模型对比

Figure 2-15 Comparison between benchmark function Michaelwicz and surrogate models

于 Exploitation 和 Exploration 的加点方法，以及动态调整加点个数的方法。

### 2.5.1 基于 Exploitation 的加点方法

Exploitation<sup>[113]</sup> 的核心思想是，对于函数非线性程度较大的区域或者说梯度变化较大的区域，代理模型往往不能较好的进行拟合，需要在这些区域加入更多的样本点才能提高代理模型在这些区域的拟合精度。对于有解析表达式的函数，可以使用数值的方法求出梯度变化较大的区域，而实际问题是沒有函数表达式的，无法求出梯度，所以需要换一种思路。由于代理模型在这些区域的拟合精度较低，可以通过 RMSE 反应出来，所以可以在 RMSE 较大的点的附近区域加点。在 2.3 节的 cross validation 中介绍过会输出 RMSE 最大的一个样本点，作用就在此。下面介绍 Exploitation 具体加点实现方法。

Exploitation 模块流程图如图 2-18 所示，首先将代理模型中 RMSE 最大的一个样本点和需要加点的个数作为输入，然后求出到这个样本点最近的一个样本点的距离，之后生成一个长度在这个距离范围内的随机向量，将 RMSE 最大的样本点

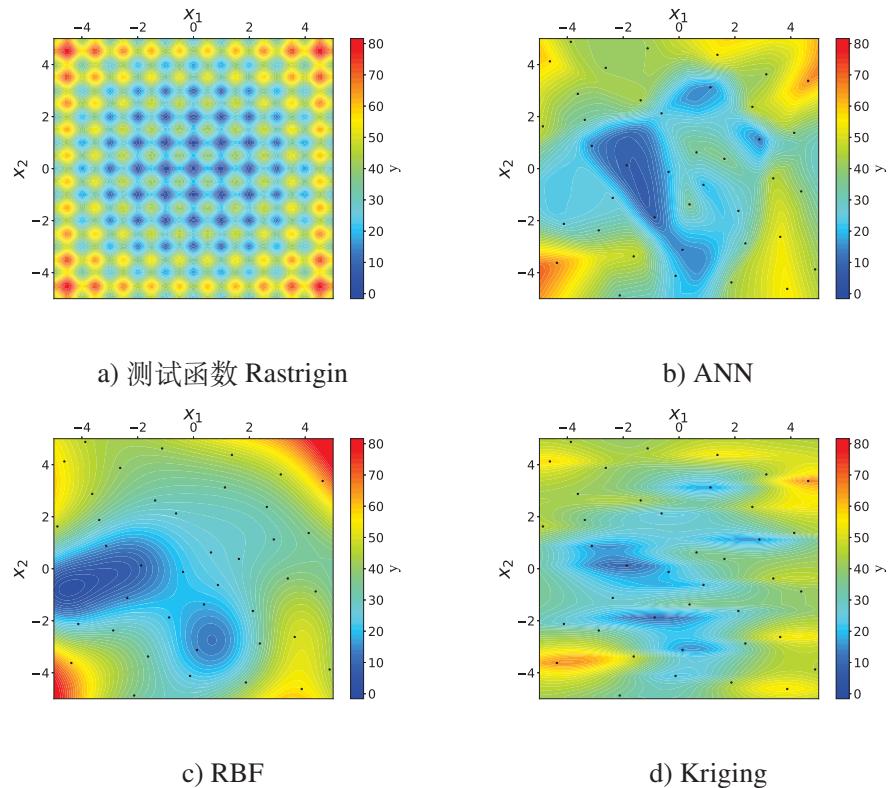


图 2-16 测试函数 Rastrigin 和代理模型对比

Figure 2-16 Comparison between benchmark function Rastrigin and surrogate models

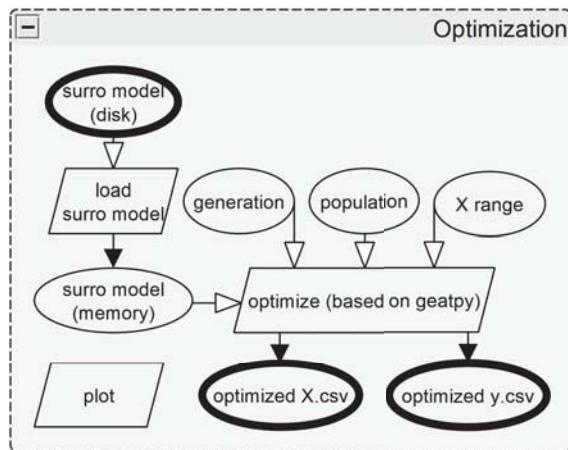


图 2-17 遗传算法优化模块流程图

Figure 2-17 Flow chart of Optimization module

加上这个向量就得到了这个样本点附近的随机的一个点，当这个点超出了参数的边界范围，就会舍弃它，重新生成一个，最后将这些点反归一化后输出为.csv 文

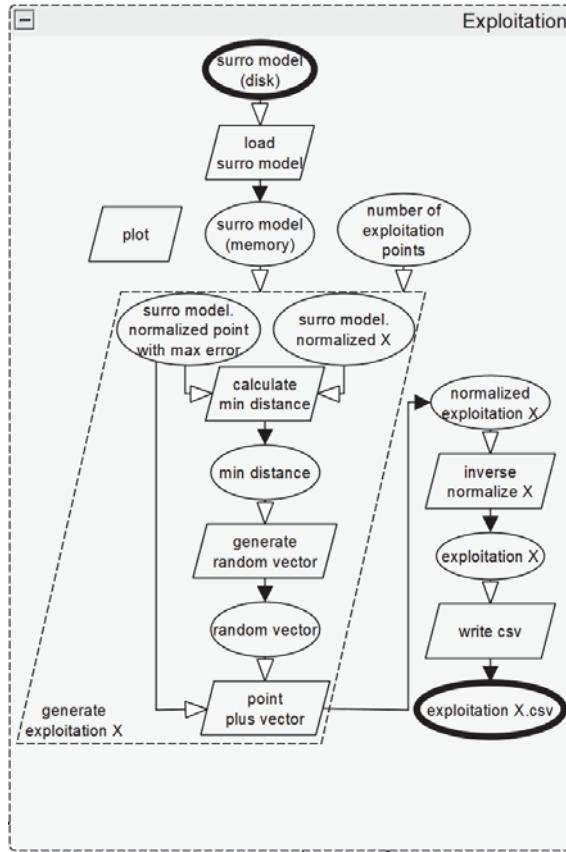


图 2-18 Exploitation 模块流程图

Figure 2-18 Flow chart of Exploitation module

件，方便其他模块调用。图2-19显示了加点个数为 5 的一个例子。

### 2.5.2 基于 Exploration 的加点方法

Exploration<sup>[113]</sup> 的核心思想是，对设计空间的各个区域给与同等的重视程度，尽可能使样本点均匀的充满整个设计空间，所以会在样本密度较低的区域加点。Exploration 模块流程图如图2-20所示，与 Exploitation 的一个区别在于 Exploration 不需要代理模型作为输入，只需要样本点 X.csv 作为输入。Intersite distance<sup>[112]</sup> 可以用来衡量设计空间中样本的密度，intersite distance 越大就代表样本密度越低。具体实现方法是将设计空间划分网格，计算每个网格节点到所有样本点的最小距离作为 intersite distance，当网格足够密时，就可以画出整个设计空间的 intersite distance 的云图。这里划分网格的方法类似于实验设计方法，目前提供两种方法 full factorial 和 random LHS，和实验设计模块中的一致。对于维数较高的问题，使用 full factorial 网格节点个数会成指数增加，需要大量计算时间，所以推荐使用 random LHS。在得

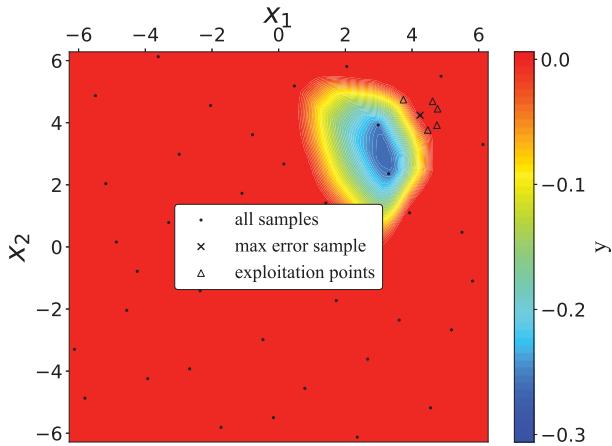


图 2-19 Exploitation 加点示例

Figure 2-19 Example of Exploitation points

到所有网格节点的 intersite distance 后，就可以选择 intersite distance 最大的网格节点，作为需要增加的新的样本点。由于增加新的样本点后，整个设计空间的 intersite distance 分布会发生变化，所以每加一个点后需要重新计算 intersite distance。最后同样将这些点反归一化后输出为.csv 文件，方便其他模块调用。图2-21显示了加点个数为 6 的一个例子。

### 2.5.3 动态调整加点个数方法

有了上述两种加点方法，再将遗传算法优化得到的最优的点也作为新的样本点加入进来，与原始的所有样本点组成新的数据库，重新建立代理模型，由此开始新一轮的优化，如此循环迭代多次，直到得到收敛的结果。这其中存在的一个问题是在每次迭代过程中 Exploitation 和 Exploration 的加点个数是人为指定的，那么是否能够根据迭代过程中代理模型精度的改善和优化迭代收敛的情况来动态调整加点的个数呢？最终的目标，一是尽可能加快收敛速度，即减小迭代次数，因为每次迭代之间是串行的，相邻两次迭代生成的新的样本点只能串行计算（每次迭代中的样本点可以并行计算），这样可以减少需要的总的优化时间。二是尽可能减少新加的样本点的个数，即减少所需增加的函数计算次数。因为实际情况中函数计算的成本可能非常大，例如进行一次 CFD 计算可能需要花费 10 元，这样可以减少优化所需要的费用，有实际应用价值。下面介绍动态调整加点个数实现方法。

AI 增强的动态加点模块流程图如图2-22所示。首先需要给定一个最大允许加

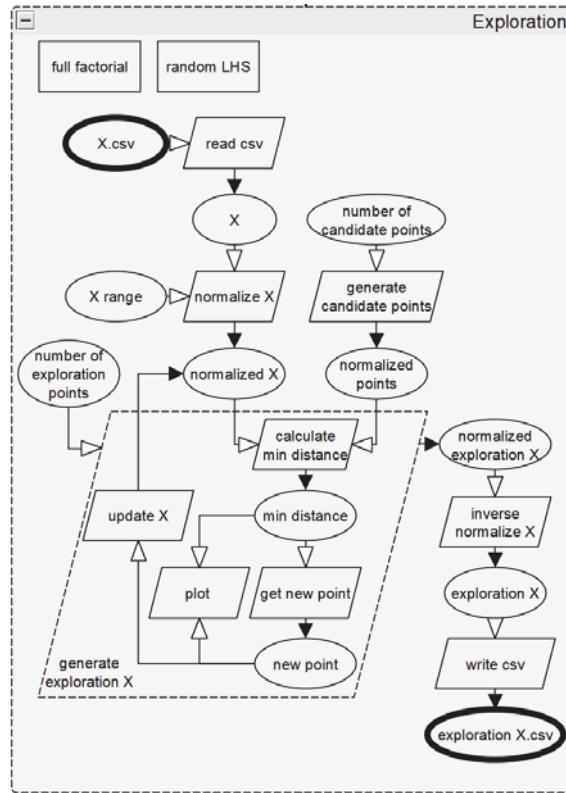


图 2-20 Exploration 模块流程图

Figure 2-20 Flow chart of Exploration module

点个数（最小加点个数默认为 1），这个可以根据实际情况具有的预算和优化时间的限制来确定，通常来说，加点个数越多，收敛速度越快，优化时间越短，但是计算成本也越大。默认初始加点个数为  $(\text{最大允许加点个数} + 1) / 2$ 。当前后两次迭代 best so far 有改进时，代表目前优化还没有收敛，需要加更多的点来加快收敛速度，以满足第一个目标，所以加点个数增加；当前后两次迭代 best so far 没有改进时，代表目前优化可能已经开始收敛，可以适当减小加点个数，以满足第二个目标。目前增加或者减少加点个数的速度是线性的，默认从初始加点个数开始，连续 5 次增加或者减少加点个数后达到最大或者最小加点个数。在得到加点个数之后，会调用 Optimization, Exploitation 和 Exploration 模块，得到 3 个模块各自所加的点，然后去掉重复的点，输出新加的样本点 new X.csv，方便其他模块调用。图2-23显示了一次迭代后添加的新的样本点的一个例子。

## 2.6 测试函数验证算例

本节使用多种测试函数对整个工具箱的优化流程进行验证。

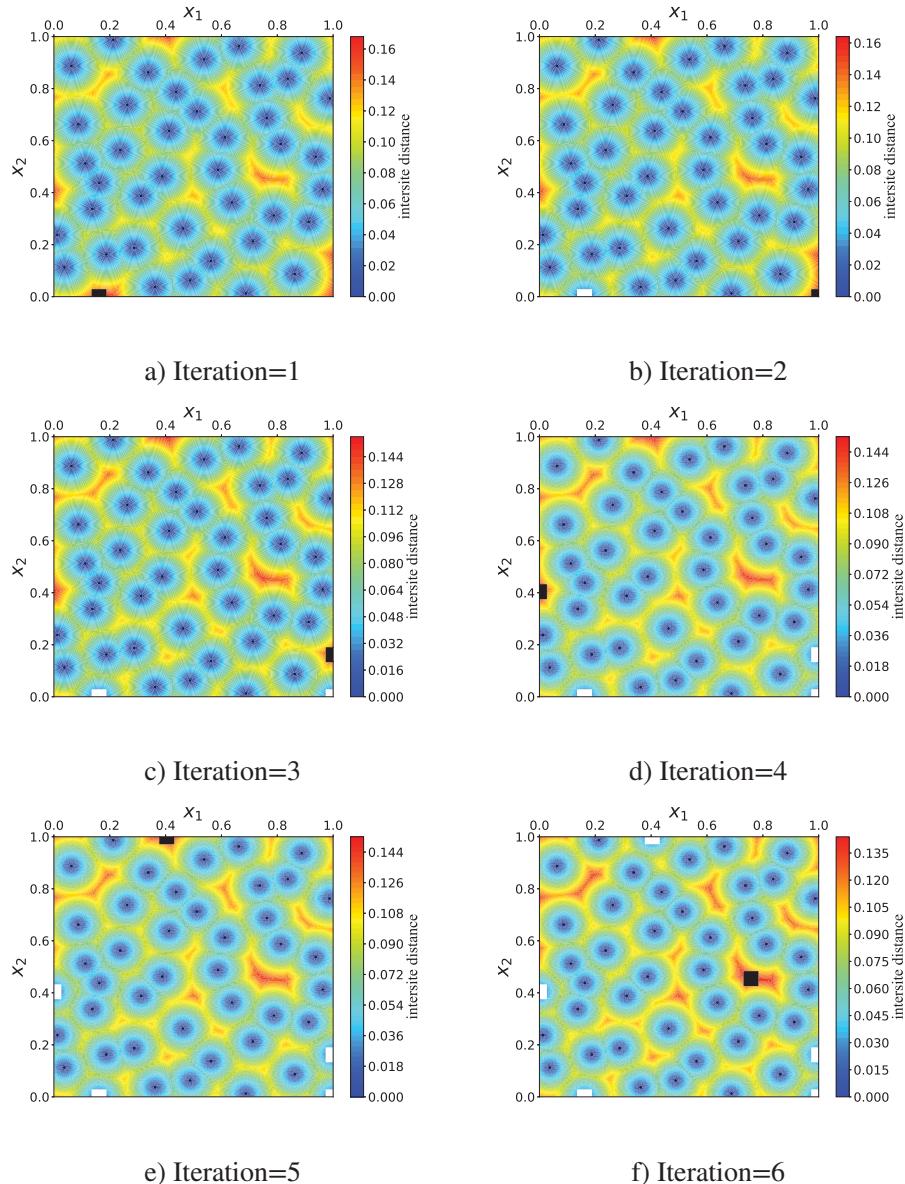


图 2-21 Exploration 加点示例 (黑点为原始样本点, 黑方块为当前迭代所加的点, 白方块为已经加的点)

Figure 2-21 Example of Exploration points

### 2.6.1 单目标优化

对于单目标优化问题, 一共使用 6 种测试函数, 包括 2 维, 5 维, 和 20 维的情况, 并且对比了使用动态加点个数和固定加点个数方法的优化结果。表2-1显示了优化结果和测试函数理论最小值对比, 对于较为简单的测试函数, 动态加点和固定加点的优化结果都达到了理论最小值, 对于较为复杂的函数, 固定加点的优

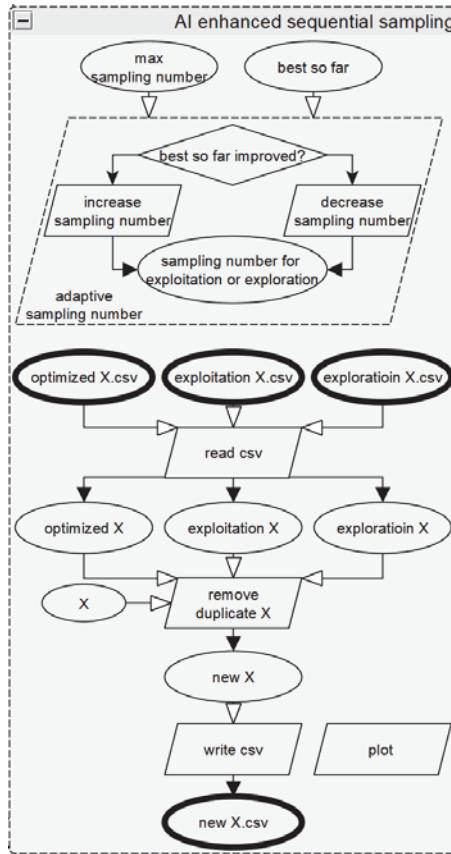


图 2-22 AI 增强的动态加点模块流程图

Figure 2-22 Flow chart of AI enhanced sequential sampling

化结果略优于动态加点，并且误差都在可以接受范围内。表2-2给出了动态加点和固定加点函数计算总数（Number of function evaluation, NOFE）的对比，在迭代次数相同，即总的优化时间相同的情况下，使用动态加点相比固定加点能减少一半的计算量。

### 2.6.2 多目标优化

3 目标优化问题使用测试函数 DTLZ1<sup>[157]</sup>:

$$\underset{\mathbf{x}}{\text{Minimize}} \quad F(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), F_3(\mathbf{x})]^T \quad (2-8)$$

其中

$$F_1(\mathbf{x}) = \frac{1}{2}x_1x_2(1 + g(x_3, x_4, x_5)) \quad (2-9)$$

$$F_2(\mathbf{x}) = \frac{1}{2}x_1(1 - x_2)(1 + g(x_3, x_4, x_5)) \quad (2-10)$$

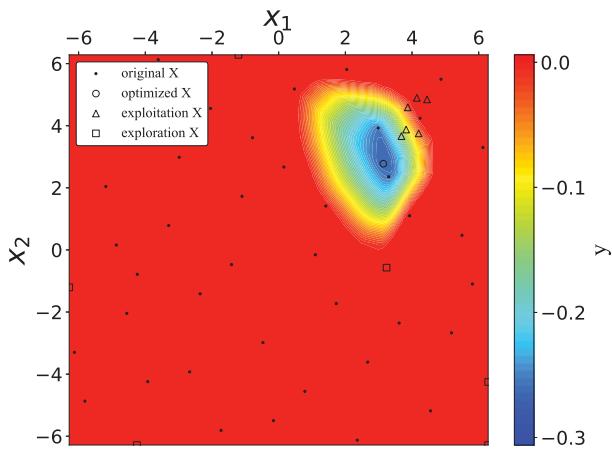


图 2-23 一次迭代后添加的新的样本点示例

Figure 2-23 Example of new samples after one iteration

表 2-1 测试函数验证结果 (1)

Table 2-1 Validation results of benchmark functions

Name	Theoretical min	Optimized min (fixed)	Optimized min (adaptive)
ackley_2D	0.00	0.01	0.01
easom_2D	-1.00	-1.00	-1.00
michaelwicz_2D	-1.80	-1.80	-1.80
ackley_5D	0.00	0.43	0.64
michaelwicz_5D	-4.67	-3.58	-3.36
ackley_20D	-4.67	-3.58	-3.36

表 2-2 测试函数验证结果 (2)

Table 2-2 Validation results of benchmark functions

Name	Iteration (fixed)	Iteration (adaptive)	NOFE (fixed)	NOFE (adaptive)
ackley_2D	30	30	1270	<b>622</b>
easom_2D	30	30	1270	<b>774</b>
michaelwicz_2D	30	30	1270	<b>718</b>
ackley_5D	30	30	1330	<b>1102</b>
michaelwicz_5D	60	60	2560	<b>1104</b>
ackley_20D	60	60	2880	<b>2608</b>

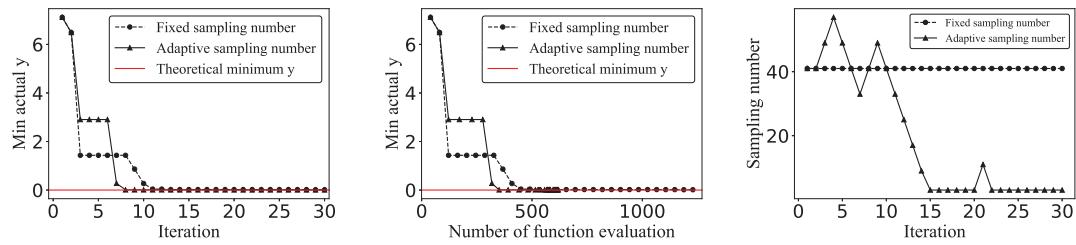


图 2-24 优化收敛曲线 (ackley\_2D)

Figure 2-24 Convergence of optimization

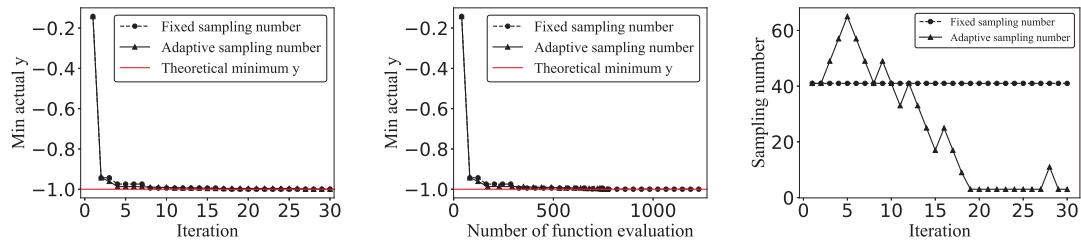


图 2-25 优化收敛曲线 (easom\_2D)

Figure 2-25 Convergence of optimization

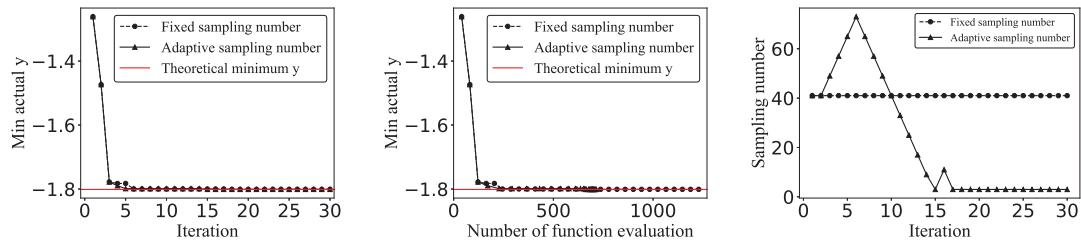


图 2-26 优化收敛曲线 (michaelwicz\_2D)

Figure 2-26 Convergence of optimization

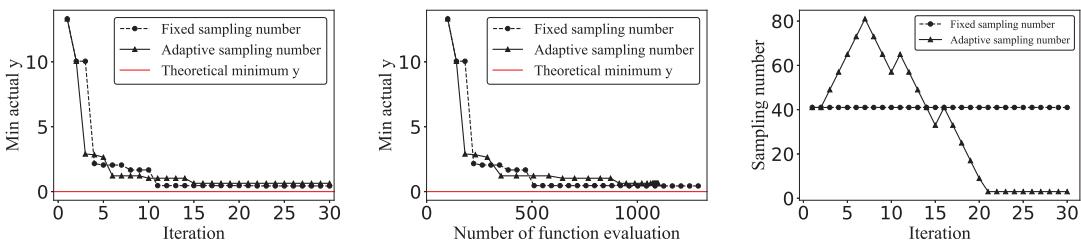


图 2-27 优化收敛曲线 (ackley\_5D)

Figure 2-27 Convergence of optimization

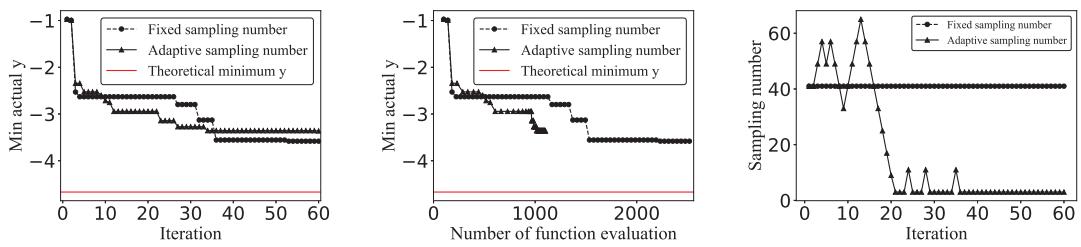


图 2-28 优化收敛曲线 (michaelwicz\_5D)

Figure 2-28 Convergence of optimization

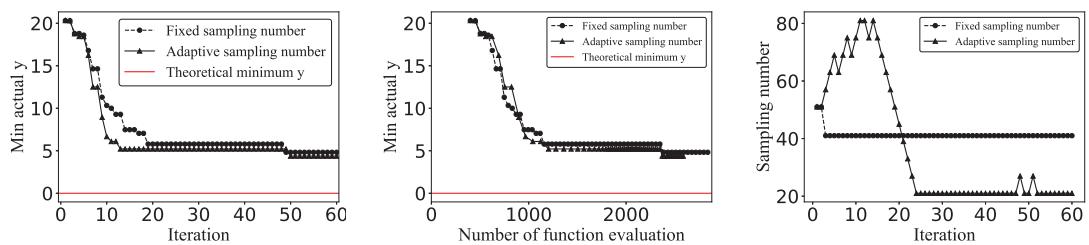
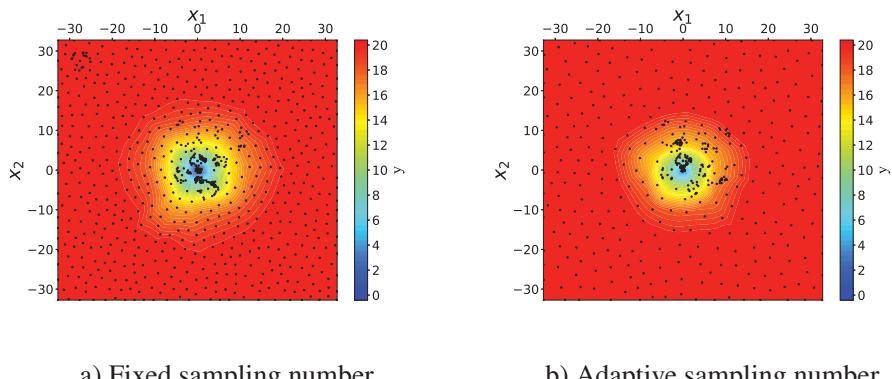


图 2-29 优化收敛曲线 (ackley\_20D)

Figure 2-29 Convergence of optimization



a) Fixed sampling number

b) Adaptive sampling number

图 2-30 固定加点个数和动态加点个数最后一代代理模型响应面对比 (ackley\_2D)

Figure 2-30 Comparison of response surface of the last generation surrogate model with fixed and adaptive sampling number

$$F_3(\mathbf{x}) = \frac{1}{2}(1 - x_1)(1 + g(x_3, x_4, x_5)) \quad (2-11)$$

$$g(x_3, x_4, x_5) = 100 \left( 3 + \sum_{i=3}^5 (x_i - 0.5)^2 \right) \quad (2-12)$$

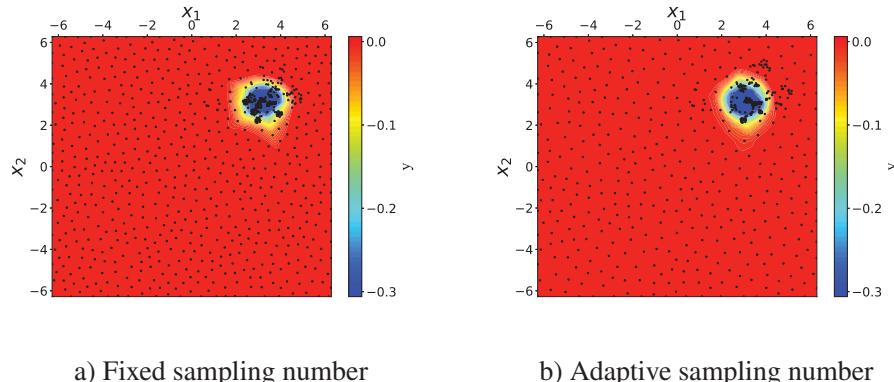


图 2-31 固定加点个数和动态加点个数最后一代代理模型响应面对比 (easom\_2D)

Figure 2-31 Comparison of response surface of the last generation surrogate model with fixed and adaptive sampling number

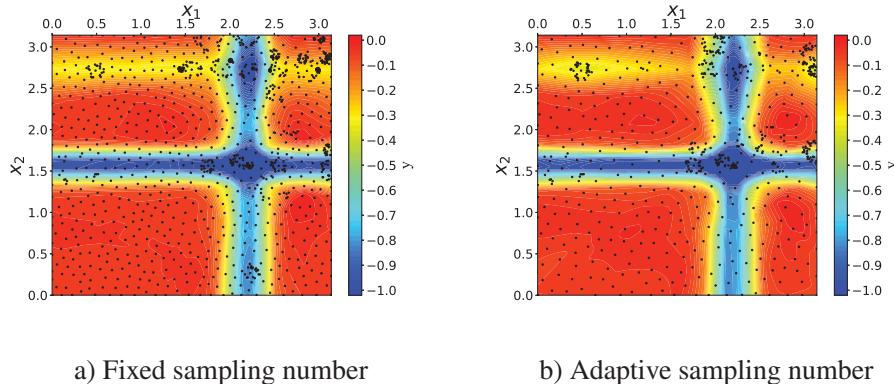


图 2-32 固定加点个数和动态加点个数最后一代代理模型响应面对比 (michaelwicz\_2D)

Figure 2-32 Comparison of response surface of the last generation surrogate model with fixed and adaptive sampling number

s.t.

$$0 \leq x_i \leq 1, i = 1, 2, 3, 4, 5 \quad (2-13)$$

图2-35显示了 Pareto front 上三个目标的真实函数的最小值随迭代的收敛曲线。经过 20 次迭代后优化终止，因为已经超过了 10 次以上的迭代不能进一步改善优化结果。图2-36显示了 Iteration=20 的优化结果，其中虚线将 Pareto front 上代理模型的值与实际函数值连接起来。基于代理模型优化得到的 Pareto front 与理论 Pareto front 之间的误差是可以接受的。图2-37对比了 ANN 响应面和理论函数等值线图。虽然 ANN 不能拟合测试函数的所有细节，但它具有良好的全局拟合性能。

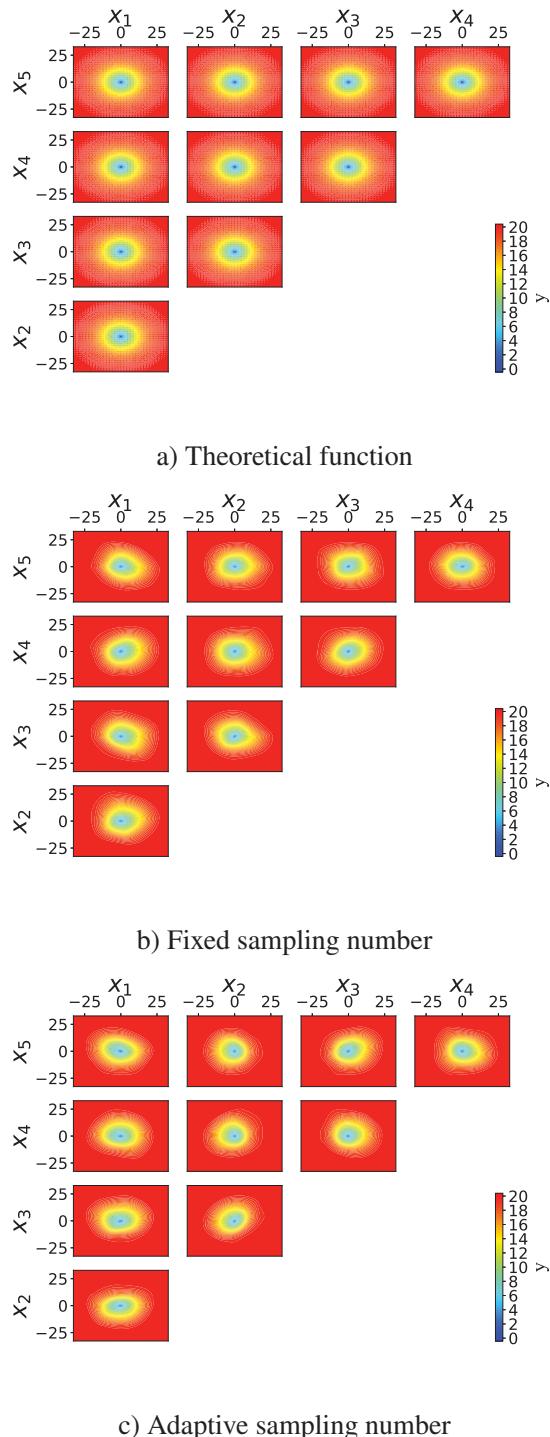
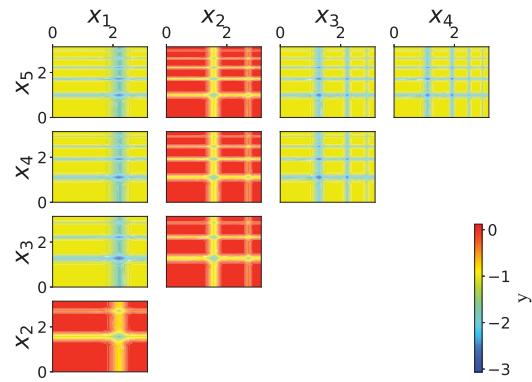
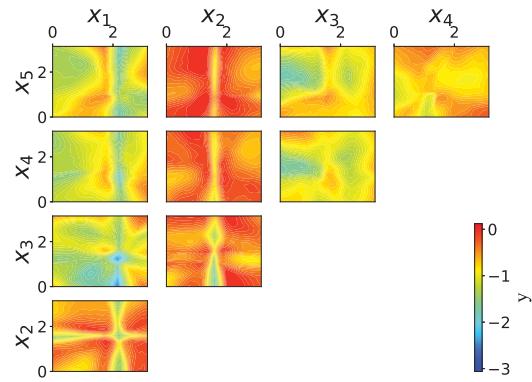


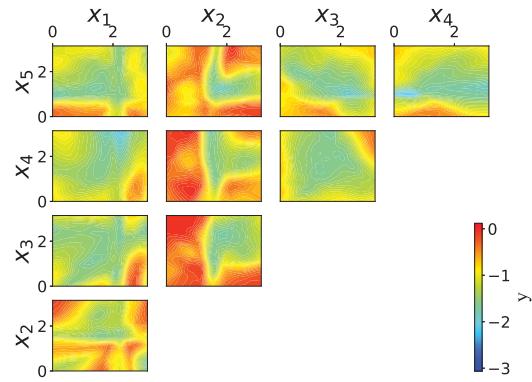
图 2-33 固定加点个数和动态加点个数最后一代代理模型响应面对比 (ackley\_5D)  
 Figure 2-33 Comparison of response surface of the last generation surrogate model with fixed and adaptive sampling number



a) Theoretical function



b) Fixed sampling number



c) Adaptive sampling number

图 2-34 固定加点个数和动态加点个数最后一代代理模型响应面对比 (michaelwicz\_5D)  
 Figure 2-34 Comparison of response surface of the last generation surrogate model with fixed and adaptive sampling number

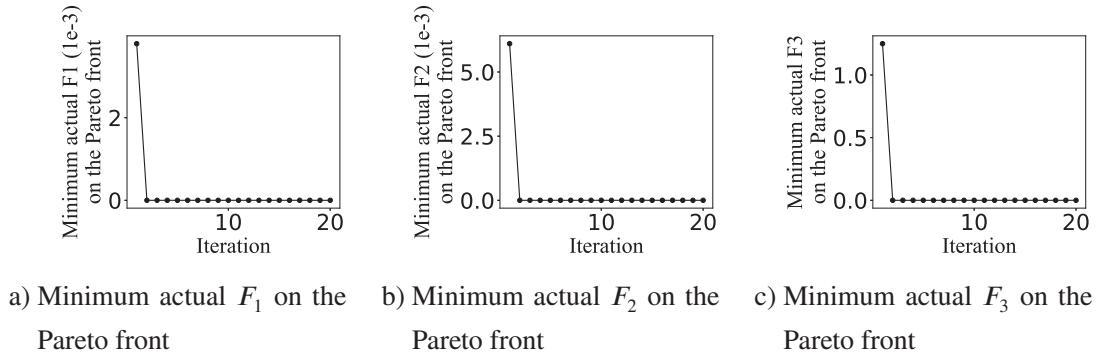


图 2-35 Pareto front 上真实函数的最小值收敛曲线

Figure 2-35 Minimum actual objective values on the Pareto front.

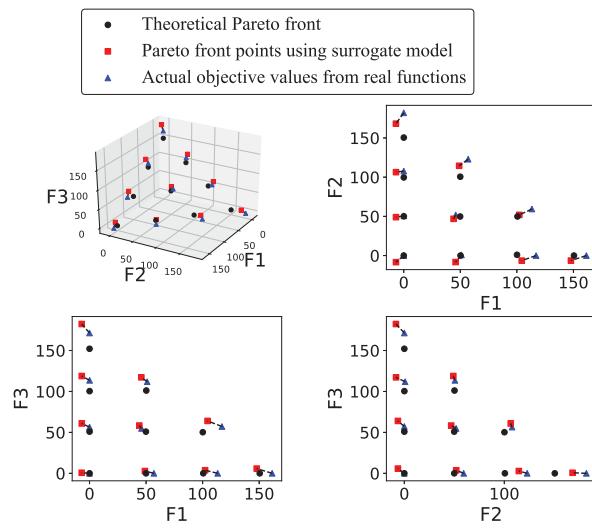


图 2-36 使用代理模型和真实函数值的 Pareto front 对比

Figure 2-36 Comparison of Pareto front using surrogate models and actual function values.

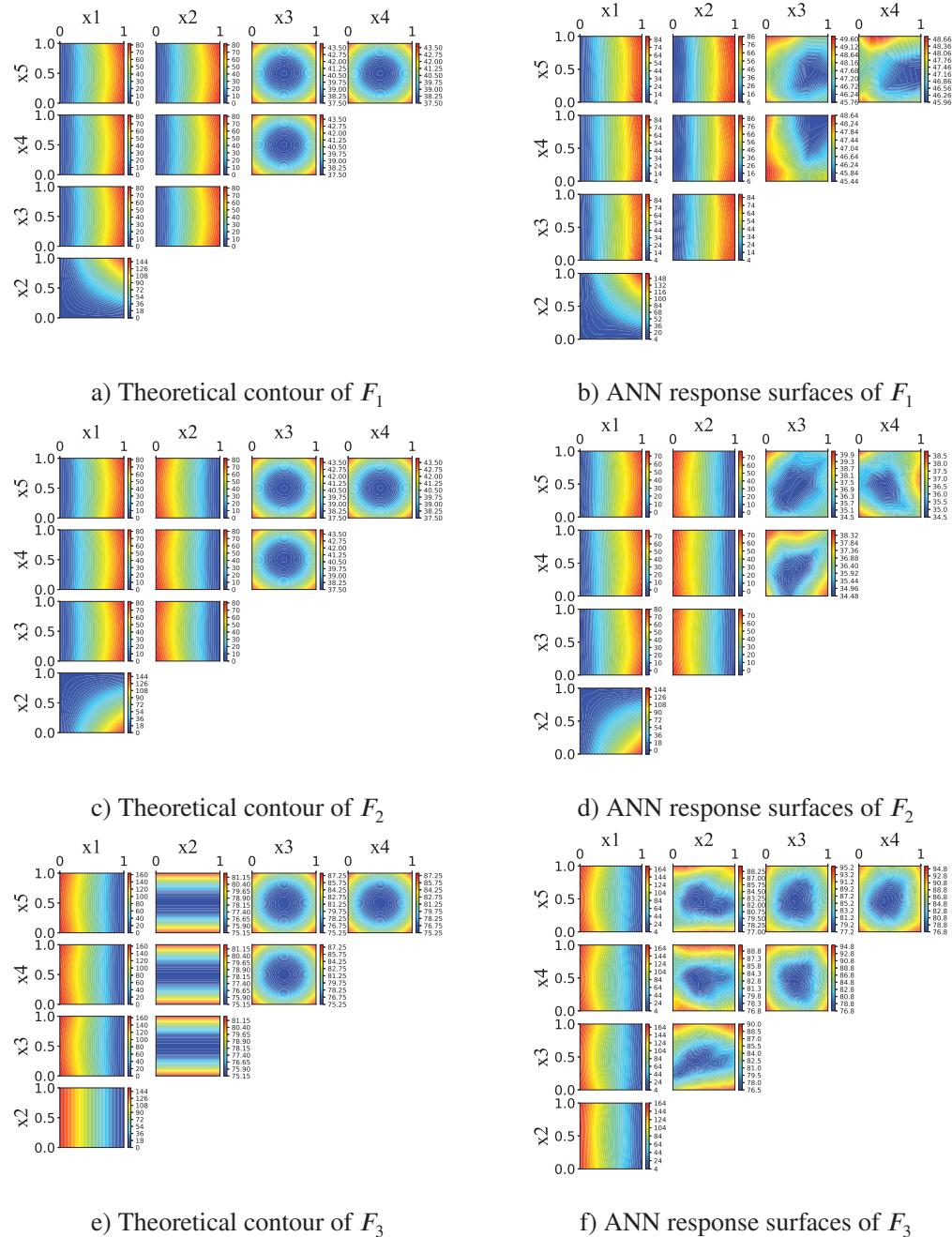


图 2-37 ANN 响应面和理论函数等值线图对比

Figure 2-37 Comparison of theoretical contours and ANN response surfaces of benchmark functions.