文章编号:1671-4598(2019)12-0124-05 DOI:10.16526/j.cnki.11—4762/tp.2019.12.028 中图分类号:TP183 文献标识码:A

基于 BP 神经网络的某防空导弹发射机构故障分析

许弘毅[1]，郝 建[1]，邓思宇[1]，高 扬[2]
（1. 中国人民解放军 66294 部队，北京 100042；
2. 中国人民解放军陆军装备部信息保障室三室，北京 100042）

摘要：在武器装备维修与技术保障领域中，普遍存在故障模式复杂、分析定位繁琐等问题，一定程度上影响了装备维修工作的效率；为解决上述问题，以陆军某型便携防空导弹发射机构为典型研究对象，设计了一种基于人工神经网络的故障分析方法，研究了神经网络在装备技术保障领域中的应用；该方法以 BP 神经网络为基础，利用历史维修数据确定网络的正反向传播矩阵，在后期采取一定措施对预测准确率进行了优化，整套方法利用 Python 环境进行了实现；通过实验验证，该方法对发射机构故障进行定位时，速度快、效率高，准确率超过 96%，因此可知该方法对于多种故障现象影响下的故障模式分析具有较高的预测准确率，能满足装备维修中故障模式的快速分析定位，并且具有较强的通用性。

关键词：神经网络；防空导弹；故障分析

Fault Analysis of Launching Mechanism of an Air Defense Missile Based on BP Neural Network

Xu Hongyi[1], Hao Jian[1], Deng Siyu[1], Gao Yang[2]
(1. Unit 66294 of the PLA, Beijing 100042, China; 2. Third Section of Information Support Bureau of PLA Army Equipment Department, Beijing 100042, China)

Abstract: In the field of weapon equipment maintenance and technical support, there are many problems, such as complex failure modes and complicated analysis and location, which affect the efficiency of equipment maintenance to a certain extent. In order to solve the above problems, a fault analysis method based on artificial neural network is designed on the typical platform of a portable air defense missile launching mechanism of the Army. The application of neural network in the field of equipment technical support is studied. This method is based on BP neural network and uses historical maintenance data to determine the forward and backward propagation matrix of the network. In the later stage, some measures are taken to optimize the classification accuracy. The whole method is implemented in Python environment. Experiments show that this method has high speed, high efficiency and accuracy over 96% when locating the fault of launching mechanism. Therefore, this method has high classification accuracy for fault mode analysis under the influence of various fault phenomena, and can satisfy the rapid analysis and location of fault mode in equipment maintenance, and has strong versatility.

Keywords: BP neural network; air defense missile; fault analysis

0 引言

在武器装备维修与技术保障领域中，普遍存在故障模式复杂、分析定位繁琐等问题。以陆军某型便携式防空导弹为例，通过 FMEA 分析可知，故障现象与故障模式大量交错，尤其对于电气性能故障，现象与模式一对多、多对一的情况非常普遍。目前，对于装备在维修过程中的故障分析及定位，仍然以人工为主，人的经验水平对分析和定位的准确性影响很大。遇到经验不足的维修人员，无法准确定位故障部位，只能以分步代换的方法进行故障排除，导致工作效率低下、资源大量浪费。

收稿日期：2019-05-23； **修回日期**：2019-06-25。

作者简介：许弘毅(1987-)，男，河北省石家庄市人，工学学士，工程师，主要从事陆军导弹装备技术保障方向的研究。

目前，随着人工智能的发展进步，人工神经网络等方法大量应用在模式识别、自动控制、测试估计等多个领域。本文针对上述问题，以某型便携防空导弹发射机构为典型研究对象，以近五年的维修历史数据制作了训练和测试数据集，引入 BP 神经网络算法进行故障分析及定位，探讨神经网络在装备故障分析中的应用。

1 BP 神经网络及 Python 简介

1.1 BP 神经网络

BP 神经网络（back propagation neural network）是一种按误差反向传播训练的多层前馈网络，其基本思想是利用梯度搜索，使网络的实际输出和期望输出的误差均方差最小。

基本的 BP 网络包括信号的前向传播和误差的反向传播两个过程。在信号前向传播过程中，输入信号通过中间层

作用于输出层，经过非线性变换，产生输出信号，若实际输出与期望输出不相符，则转入误差反向传播过程，逐层获取误差信号以更新每个节点的联接强度以及阈值，以上即为神经网络的训练过程。经过大量样本的多次迭代，传播矩阵被不断修正，各节点间的权值进一步趋于合理，最终对数据归类的准确率也不断提高[1]。在训练结束后，以最终确定的各节点权值构建完整网络。

1.2 Python

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言，由于其简洁、易读的特性，有着大规模的用户群体。同时，Python 支持众多开源的科学计算库，如著名的 OpenCV、VTK 等等，得益于此，Python 大量应用于科学计算以及人工智能领域[2]。

2 某型便携防空导弹发射机构故障分析算法实现

2.1 构建 BP 神经网络

2.1.1 地面设备的主要电气组成及性能参数

该型地面设备内部主要由 3 块线路板组成，根据具体功能，可分为主要有以下 9 个功能模块：

1）供电电路：对系统主电源进行变换，为装备各部分提供二次电源。常见的故障模式主要是二次电源失效，所表现出的现象主要为相关部分的电源最大消耗电流以及稳定工作电流异常，出现故障后会对全局造成影响。

2）单片机核心及外围接口电路：系统的核心部分，接收各类外部信号，计算导弹发射条件，输出各种控制指令，出现故障后会对全局造成影响。

3）起转控制电路：控制导引头陀螺部分起转与停止，常见的故障模式主要是导引头起转失败或转速异常，表现出的现象主要为导引头不起转、起转断开时间或断开频率异常，出现故障后会对全局造成影响。

4）声音信号控制电路：控制输出声音信号，主要用于在截获目标以及分析目标角速度时给出声音指示，故障模式和现象比较简单，主要是有声（正常）/无声（异常），出现故障后不会对系统主体功能产生影响。

5）光信号控制电路：和声音信号控制电路类似，主要用于在截获目标以及分析目标角速度时给出灯光指示，故障模式和现象比较简单，主要是有光（正常）/无光（异常），出现故障后不会对系统主体功能产生影响。

6）解锁电路：控制导引头陀螺部分偏移的锁定以及解锁，常见的故障模式主要是导引头陀螺部分不解锁，无法正常跟踪目标，表现出的故障现象主要是对目标角飞行状态分析异常，无法给出正确的发射条件。其故障现象不直观，需要依据发射条件以及分析时间等多个量综合判断，该部分出现故障后会对全局造成影响。

7）电源转换电路：在导弹进入发射程序后，激活弹上电源，同时将工作电源从地面电源转换为弹上电源，常见的故障模式主要是弹上电源无法激活以及电源转换失败，表现出的故障现象主要是导弹起飞后姿态异常。其故障现象不直观，需要根据测试结果综合判断，该部分出现故障后会对全局造成影响。

8）气源激活电路：与电源转换电路类似，主要是激活弹上致冷气源。

9）点火电路：用于输出导弹发动机点火信号，常见的故障模式和现象主要是导弹进入发射程序后发射失败，该部分出现故障后会对全局造成影响。

10）IFF 电路：敌我识别功能电路，主要用于对目标进行敌我识别，在判定目标为友机时，及时停止发射程序。主要的故障模式和现象是无法进行敌我判定，该部分出现故障后不会影响系统主体功能。

对应各功能模块电路，主要的电气性能测试项目表 1 所列。

表 1　发射机构主要电气性能测试项目

项目名称	单位
电源 1 最大消耗电流	A
电源 2 最大消耗电流	A
断开时间	s
断开频率	Hz
电源 1 稳定工作电流	mA
电源 2 稳定工作电流	mA
电源 3 稳定工作电流	mA
声音指示信号	
光指示信号	
自动分析	s
1 级时间	s
2 级时间	s
3 级时间	s
光闪频率	Hz

2.1.2 神经网络模型

发射机构故障分析相对简单，为减少计算量及计算时间，在构建神经网络时采用常见的 3 层架构，即除输入层及输出层外，只建立 1 层中间层。

对于每层的节点数，根据以下规则确定：

1）输入层节点数由发射机构需要测试的性能参数决定，即 14 个节点；

2）输出层节点数由需要定位故障的发射机构功能模块电路数量确定，即 9 个节点；

3）中间层节点数按照 $m=\sqrt{n \cdot l}$（m 为中间层节点数，n 为输入层节点数，l 为输出层节点数）确定，即 11 个节点[3]。

确定了以上参数，在程序中建立 neuralNetwork 类，该类主要对输入层、中间层、输出层的节点数量和各层间的权重矩阵以及梯度下降算法的学习率进行初始化，其代码如下：

```
classneuralNetwork:
    初始化
def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
    根据初始化参数设置输入、隐藏、输出节点数量
self.inodes = inputnodes
self.hnodes = hiddennodes
self.onodes = outputnodes
    设置学习率
self.lr = learningrate
    以随机正态分布的方式确定两个权重矩阵
self.wih = numpy.random.normal(0.0, pow(self.hnodes, 0.5),(self.hnodes,self.inodes))
self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5),(self.onodes,self.hnodes))
    以 lambda 表达式的方式引用 Sigmoid 函数,expit()即为 S 函数
self.activation_function = lambda x: scipy.special.expit(x)
```

2.2 信号的前馈

在信号前馈过程中，重点需要关注由每个节点的权重组成的权重矩阵以及信号在中间层传播时应用的激活函数。在本项目中，权重矩阵的初始化以随机数的方式生成。同时，如果没有激活函数，每一层的输出和输入都为线性关系，无论中间有多少层，都没有意义[2]。在激活函数的选择上，采用了类似阶跃函数但比阶跃函数更加平滑、更接近现实的 S 函数（Sigmoid Function）：

$$y=\frac{1}{1+e^{-x}} \tag{1}$$

在本神经网络中，中间层的输入信号符合：

$$\boldsymbol{X}_h=\boldsymbol{W}_{ih}\cdot\boldsymbol{I}=\begin{pmatrix}w_{ih1}\\ \cdots\\ w_{ih11}\end{pmatrix}\cdot\begin{pmatrix}i_1\\ \cdots\\ i_{14}\end{pmatrix} \tag{2}$$

式中，$\boldsymbol{I}$ 为输入矩阵，$\boldsymbol{W}_{ih}$ 为输入层和中间层之间的权重矩阵。

中间层的输出为：

$$O=sigmoid(X_h)=\frac{1}{1-e^{-W_h\cdot I}} \tag{3}$$

输出层的信号符合：

$$\boldsymbol{X}o=\boldsymbol{W}_{ho}\cdot O=W_{ho}\cdot sigmoid(X_h)=\frac{1}{1-e^{-W_h\cdot I}} \tag{4}$$

式中，$\boldsymbol{I}$ 为输入矩阵，$\boldsymbol{W}_{ih}$、$\boldsymbol{W}_{ho}$ 分别为输入层至中间层以及中间层至输出层的权重矩阵，$\boldsymbol{X}_h$ 为中间层输出矩阵，$\boldsymbol{X}_o$ 为输出层输出矩阵）

2.3 误差反向传播以及权重的更新

误差的反向传播过程可以用矩阵乘法表示：

$$error_h=W_{ho}^T\cdot error_o \tag{5}$$

考虑到神经网络的规模，在权重更新过程中采用了梯度下降法，该方法计算效率高，但存在两个风险。一是有可能停止于局部极小值而不是全局最小值，导致分类不准确；二是超调越过最小值，系统不断振荡，导致无法有效收敛。为了尽量减少上述风险，需要选择适当的误差函数。在网络中，对于第 k 个节点，确定误差函数为：

$$e_k=(t_k-o_k)^2 \tag{6}$$

选择这一误差函数，主要基于两点原因[4]：

1）使用该函数，可以比较方便的计算出梯度下降的斜率；

2）误差函数平滑连续，不会出现下降过程中突然跳跃的情况；

3）误差越接近最小值，梯度越小，超调风险相应减小。

相应的，在该处的斜率为：$\partial e/\partial w$。

设输入层某一节点为 i，中间层某一节点为 j，输出层某一节点为 k，期望值为 t，实际输出值为 o。则对于任意 j —k 链路，可计算斜率为：

$$\frac{\partial E}{\partial W_{jk}}=\frac{\partial}{\partial W_{jk}}\sum\nolimits_n(t_n-o_n)^2 \tag{7}$$

由于 t 为常数，则有：

$$\frac{\partial E}{\partial W_{jk}}=-2(t_k-o_k)\cdot\frac{\partial o_k}{\partial W_{jk}}=$$

$$-2(t_k-o_k)\cdot\frac{\partial}{\partial W_{jk}}sigmoid\left(\sum\nolimits_j W_{j,k}\cdot o_j\right) \tag{8}$$

最终，中间层与输出层间误差函数斜率为：

$$\frac{\partial E}{\partial W_{jk}}=-(t_k-o_k)\cdot sigmoid\left(\sum\nolimits_j W_{j,k}\cdot o_j\right)$$

$$\left(1-sigmoid\left(\sum\nolimits_j W_{j,k}\cdot o_j\right)\right)\cdot o_j \tag{9}$$

输入层与中间层误差函数斜率为：

$$\frac{\partial E}{\partial W_{i,j}}=-(e_j)\cdot sigmoid\left(\sum\nolimits_j W_{j,k}\cdot o_j\right)$$

$$\left(1-sigmoid\left(\sum\nolimits_j W_{j,k}\cdot o_j\right)\right)\cdot o_j \tag{10}$$

最终的权重更新函数为：

$$W_{new}=W_{old}-\alpha\cdot\frac{\partial E}{\partial W_{i,j}} \tag{11}$$

式中，α 为学习率。在梯度下降算法中，学习率决定了参数每次更新的幅度，即下降过程中每一步的步长。

误差计算与权重更新源代码如下：

```
    计算误差
output_errors = targets - final_outputs
hidden_errors = numpy.dot(self.who.T, output_errors)
    更新权重
self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))
self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))
```

权重的初始化，遵循正态分布下的随机采样，其中均值为 0，标准差为该节点传入链接数的开方，即 $1/\sqrt{nodes}$。权重初始化源代码如下：

以随机正态分布的方式确定两个权重矩阵

```
self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5),(self.hnodes,self.inodes))
self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5),(self.onodes,self.hnodes))
```

2.4　数据预处理

考虑到激活函数以及整个神经网络的使用范围，需要输入数据进行预处理。由于网络特征所限，输入数据必须在 0.0～1.0 范围内，因此需要对输入数据进行归一化。同时，当输入数据为 0 时，权重更新函数会输出 0，导致网络丧失学习能力，因此在遇到输入为 0 时，需要加入一个微量的偏移 0.001。特别的，对于声音信号、光信号这类只存在正常和异常（非 0 即 1）两种状态的测试结果数据，为了使网络能正常进行判断，同样需要加入微量偏移，使其正常状态输出 0.99、异常状态输出 0.001。

2.5　对网络进行训练

为了使神经网络能正常工作，需要对其进行训练。训练数据由历史测试数据经预处理后综合而成，共 15 位，其中第 2～15 位为测试数据，第 1 位为故障判定结果。部分训练数据如图 1 所示。

```
4,0.50,0.50,0.27,0.92,0.15,0.18,0.10,0.001,0.99,0.51,0.58,0.14,0.11,0.4
3,0.42,0.48,0.41,0.90,0.18,0.15,0.09,0.99,0.99,0.51,0.56,0.12,0.14,0.5
2,0.45,0.49,0.34,0.85,0.19,0.19,0.10,0.99,0.99,0.61,0.56,0.16,0.15,0.3
```

图 1　部分训练数据

网络训练的实现过程中，需要先将输入矩阵转化为二维数组，之后通过激活函数，进行正向计算，在计算完成后，根据与目标的误差，反向更新权重矩阵，完整方法代码如下[5-7]：

```
def train(self, inputs_list, targets_list):
    将输入转换成 2D 数组
    inputs = numpy.array(inputs_list, ndmin =2).T
    targets = numpy.array(targets_list, ndmin =2).T
    hidden_inputs = numpy.dot(self.wih, inputs)
    hidden_outputs = self.activation_function(hidden_inputs)
    final_inputs = numpy.dot(self.who, hidden_outputs)
    final_outputs = self.activation_function(final_inputs)
    计算误差
    output_errors = targets - final_outputs
    hidden_errors = numpy.dot(self.who.T, output_errors)
    更新权重
    self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0
    - final_outputs)), numpy.transpose(hidden_outputs))
    self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0
    - hidden_outputs)), numpy.transpose(inputs))
```

2.6　测试实现

利用训练数据集以及测试数据集对训练完毕的网络进行训练、测试。收集五年的历史测试数据，其中 80％用于制作训练数据集，剩余 20％用于制作测试数据集。训练数据集总量为 520 条，测试数据集总量为 130 条。在学习率为 0.2、中间层节点数量为 11 的条件下，得到预测准确率约为 94.7％，程序总执行时间约为 8.18 秒。

3　算法优化

3.1　调整学习率

在网络中，如果学习率较小，则训练时间会增加，相应达到收敛所需要迭代的次数就高，反之，虽然效率提高，但有可能超过局部最小值，造成系统振荡不停，使其无法收敛[8]。对于目前网络，经过多次实验，发现在其他条件不变的情况下，当学习率低于 0.05 以下时，预测准确率明显下降，可知网络有可能在非全局最小值出现收敛；当学习率高于 0.2 时，程序执行时间明显增加，并且预测准确率出现下降趋势，可知网络出现超调振荡现象，分类效率受到影响；当学习率高于 0.4 时，预测准确率下降显著，可知网络在部分数据训练中有可能出现无法收敛的情况；在学习率设置为 0.1 左右时，得到最佳预测准确率约为 95.2％，用时约 7.36 s。整体结果如图 2 所示。

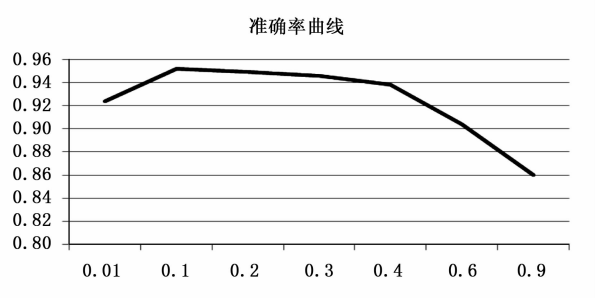

图 2　不同学习率下的准确率曲线

（横轴为学习率，纵轴为准确率）

3.2　调整中间层节点数量

网络中间层是执行学习过程的主要层次，如果节点过少，则没有足够的空间使网络完成学习[9]。经过实验，发现对于目前网络，在其他条件不变的情况下，当中间层节点数量减少为 7 个时，学习率约为 70％左右，当中间层数量为 100 时，预测准确率约为 95.6％，用时约 8.86 秒，当中间层节点数量设置为 200 时，得到预测准确率约为 96.2％，用时约 12.41 秒。此后，随着节点数量增加，程序执行时间增加明显，但准确率未发现明显提升，结果如图 3 所示。

4　实验结果分析与运用

通过分析结果，在最初的网络结构中，预测准确率约为 94.7％；在其他条件不变的情况下，预测准确率在学习率为 0.1 左右时达到最高，约 95.2％，同时学习率过高或过低时，准确率会出现明显下降；进一步将中间层节点增加至 200 个时，识别准确率又提升约 1％，达到 96.2％，而

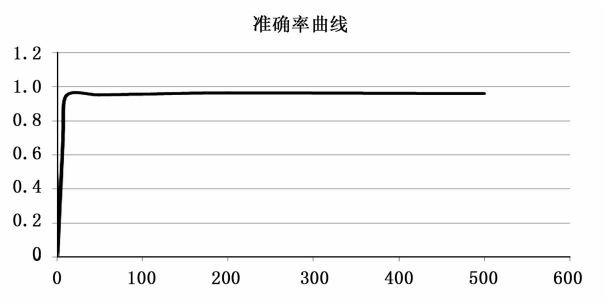

图 3 不同中间层节点下的准确率曲线
（横轴为中间层节点数，纵轴为准确率）

后随着节点数量增加，训练及预测时间成倍增长，但预测准确率并无明显变化。由此可知，对于简单的三层 BP 神经网络，设置适当的学习率以及中间层节点数量，可以有效提升最终结果的准确程度，反之，则有可能面临梯度消失或超调的风险。同时，在武器装备的维修及技术保障工作中，超过 95% 的准确率已经可以满足基本的故障分析和定位。

5 对神经网络进一步改进的探讨

5.1 改进激活函数

本文中，选择 S 函数作为神经网络的激活函数。该函数连续、并且在整个定义域间单调，是常见的激活函数。但是，在实际运用中仍然存在两个问题，一是 S 函数输出均值不为 0，存在偏移现象，在多层神经网络中，下层的输入会受到上层输出的影响；二是对该函数求导后发现，在变量 x 取值非常大或非常小的时候，其斜率会趋近于 0，使梯度更新缓慢，有可能导致梯度消失，即如果初始值很大的话，神经元会出现饱和现象，整个网络学习困难。

在人工神经网络中，常见的激活函数还可以选择 tanh 函数、ReLU 函数以及 Leaky－ReLU 函数。

tanh 函数原型如下：

$$f(x)=\frac{1-e^{-2x}}{1+e^{-2x}} \tag{12}$$

与 S 函数相比，tanh 函数的最大优点是输出均值为 0，但依然存在软饱和性，对于某些初始值，有梯度消失的风险。

ReLU 函数（线性整流函数）是斜坡函数的变种函数，原型如下：

$$f(x)=\max(0,x) \tag{13}$$

其在 x 小于 0 时，存在硬饱和现象，在 x 大于 0 时，不存在饱和问题，可以避免梯度消失现象。同时，若梯度过大，会导致“神经元死亡”的现象。因此，在使用时需要适当控制学习率。

Leaky－ReLU 函数是对 ReLU 函数的改进，原型如下：

$$f(x)=\begin{cases}x,\text{if} & x\geqslant 0\\ \alpha(e^{x}-1),\text{if} & x<0\end{cases} \tag{14}$$

函数中，α 为一较小常数，可以比较有效的解决变量小于 0 时，函数存在硬饱和区的问题，是比较理想的激活函数。同时，无论 ReLU 或 Leaky－ReLU 函数，都存在输出均值不为 0 的情况。

5.2 增加中间层数量

对于输入与输出节点数量多、对应关系复杂的系统，适当增加中间层数量，可以提升网络预测准确率。但是，对于简单系统，中间层数量增加不但对改进预测准确率贡献不大，而且还会使分析时间大量增加。

根据以上两点，对于以文中提到的某型便携防空导弹发射机构这类结构较为简单，故障现象、故障模式以及故障部位对应较为单一的装备，不存在极端的输入数据，选择单一中间层网络并且以 S 函数作为激活函数，在实现中较为简单，准确率和网络工作效率较高，同时不会出现梯度消失或者偏移量过大的问题。如果将此神经网络应用于故障模式复杂的大型装备，则要考虑根据输入输出的复杂程度，选择效率高、工作中冗余度大的激活函数，适当设置中间层数量，以达到整个网络的最优配置。

6 结语

本文以某型便携防空导弹发射机构为研究平台，研究了 BP 神经网络在装备故障分析中的应用，构建了基于神经网络的故障分析方法，并在 Python 环境下进行了实现，同时探讨了网络方法的进一步改进策略。通过实验验证，可知该方法具有速度快、应用简单、准确率高的特点，可以满足典型目标的故障分析及定位，并且可以快速扩展应用于其他装备的技术保障工作中。

参考文献：

[1] Steinbach M，等. 数据挖掘导论 [M]. 北京：人民邮电出版社，2013.
[2] Rashid T. Python 神经网络编程 [M]. 北京：人民邮电出版社，2018.
[3] Raschka S. Python 机器学习 [M]. 北京：机械工业出版社，2018.
[4] 彭 伟. 深度强化学习 [M]. 北京：中国水利水电出版社，2018.
[5] 李德毅. 人工智能导论 [M]. 北京：中国科学技术出版社，2018.
[6] Chun W. Python 核心编程 [M]. 北京：人民邮电出版社，2018.
[7] McKinney W. 利用 Python 进行数据分析 [M]. 北京：机械工业出版社，2018.
[8] Beazley D. Python Cookbook [M]. 北京：人民邮电出版社，2015.
[9] 邱 静，等. 装备测试性建模 [M]. 北京：科学出版社，2012.