

HW2 RBF 网络拟合

一、 代码实现

1. RBF 网络模型

本次作业的 RBF 网络是在 miniconda3 的虚拟环境(Python 3.7)中搭建, 并使用 PyTorch 框架编写, 网络模型的主要代码截取如下:

```
def model(t, print_cost=False):
    # data processing: dict t(containg xi, yi, H, Learning_rate, num_ite
    # rations) comes from cpp, transform them into torch.tensor

    # optimizer = torch.optim.SGD([a, b, w, w0], lr=Learning_rate)
    optimizer = torch.optim.Adam([a, b, w, w0], lr=learning_rate)
    for t in range(num_iterations):
        y_pred = torch.sum(torch.exp(-(x.mm(a) + b)**2 / 2.)
                            * w, dim=1, keepdim=True) + w0
        loss = (y_pred - y).pow(2).sum()

        loss.backward()
        optimizer.step()
        with torch.no_grad():
            # Manually zero the gradients after updating weights
            a.grad.zero_()
            b.grad.zero_()
            w.grad.zero_()
            w0.grad.zero_()

    return (a[0].numpy().tolist(), b[0].numpy().tolist(), w[0].numpy().
            tolist(), [w0.item()], [loss.item()])
```

可以看到, 网络采用了单隐藏层结构 (默认 64 节点), 总的前向传播函数为 $f(x) = \omega_0 + \sum_{i=1}^n \omega_i g_{0,1}(a_i x + b_i)$, 损失函数 $l(\hat{y}, y)$ 则采用平方误差进行度量。为加速训练进程, 反向传播使用了 Adam 优化器代替传统的 SGD 优化器进行自动梯度下降, 并在实验中取得了良好效果。完整代码详见与 CanvasSystem.cpp 同目录下的 Python 脚本 MyRBFNetwork1.py。

2. C++调用本地 Python 脚本

为在无境框架中调用上述网络模型脚本进行参数训练，作业的 CanvasSystem.cpp 源代码中包含了 Python.h 头文件，对环境进行了初始化，并对工程属性作了相应配置（操作步骤见同目录下的 readme.txt）。环境初始化和 Python 脚本调用的主要代码截取如下：

```
Py_SetPythonHome(L"D:\\miniconda3");
Py_Initialize(); //init python compiler

PyRun_SimpleString("import sys");
PyRun_SimpleString("sys.path.append('D:/GAMES102/homeworks/project/src/hw1/Systems/')"); //file location of 'MyRBFNetwork.py'

PyObject* pModule = PyImport_ImportModule("MyRBFNetwork1"); //file name of the python script
PyObject* pFunc = PyObject_GetAttrString(pModule, "model"); //func name to be called

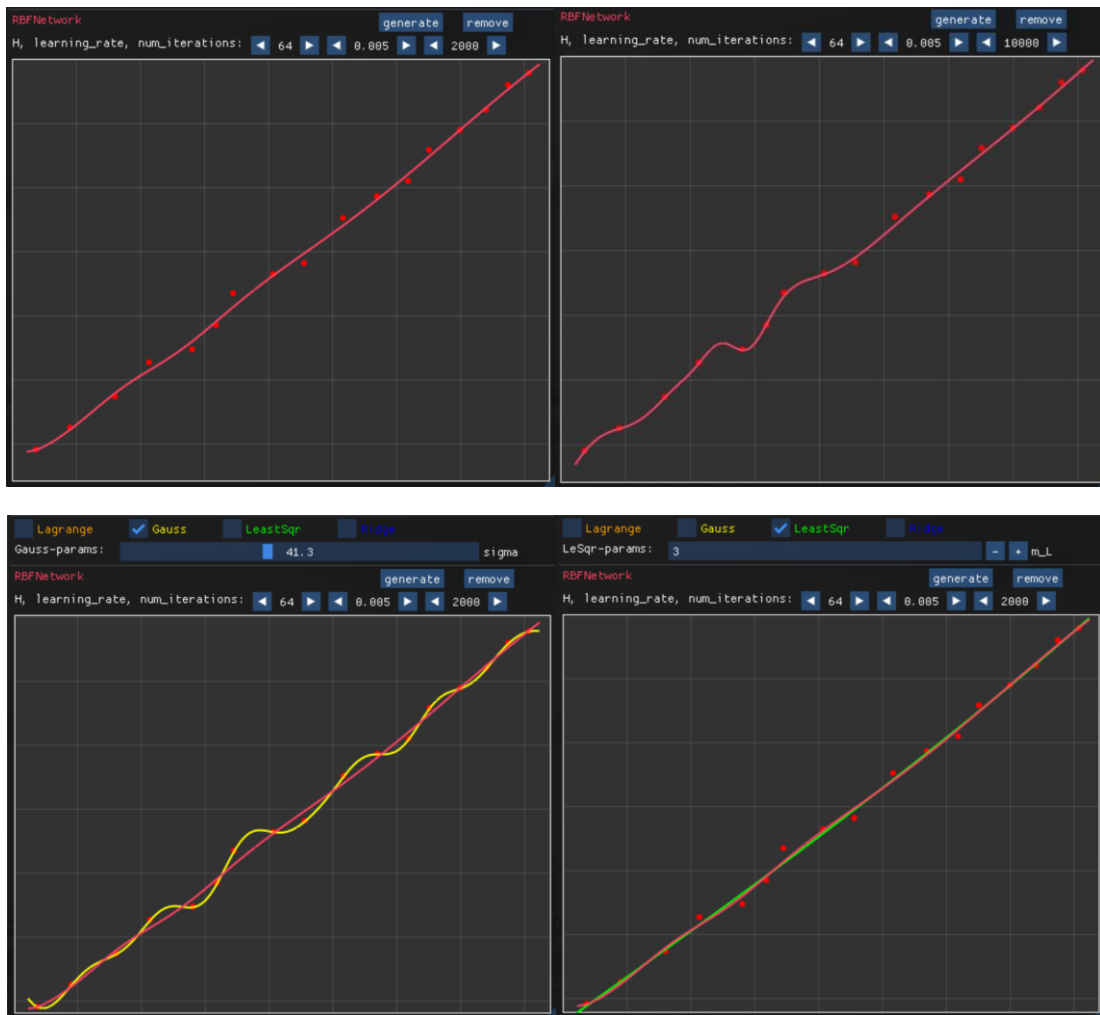
//construct params(type:dict) for the python func "model()"
PyObject* pArgsT = PyTuple_New(1);
PyObject* pArgsD = PyDict_New();
PyDict_SetItemString(pArgsD, "H", Py_BuildValue("i", data->RBF_H));
PyTuple_SetItem(pArgsT, 0, pArgsD);

//call the python func "model()" and receive return values
PyObject* pReturnTuple = PyObject_CallObject(pFunc, pArgsT);
```

二、分析与结果

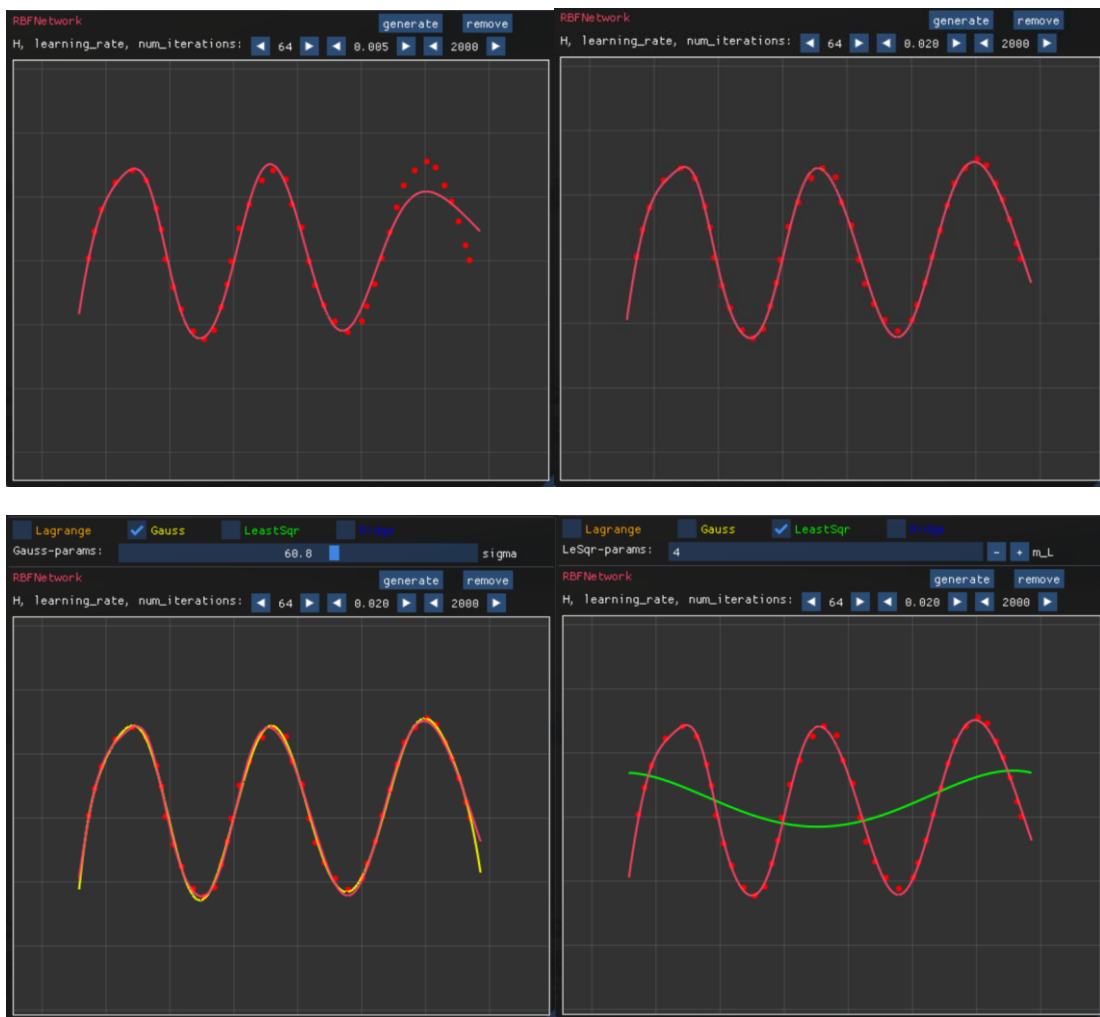
1. 简单曲线拟合

下图示出了在不同超参数条件下 RBF 网络对一组带有噪声的直线采样点的拟合结果，以及与高斯插值、最小二乘逼近的拟合效果对比：



可以看到, RBF 网络超参数的选取对拟合结果有着重要影响, 当迭代次数取 2k 时拟合效果已较好, 而取到 10k 时则发生了过拟合现象。相比而言, RBF 网络和最小二乘法对直线的拟合效果明显优于高斯插值, 图中的高斯插值是经多次调整 σ 后获得相对较优的结果, 但仍与理想有较大差距。

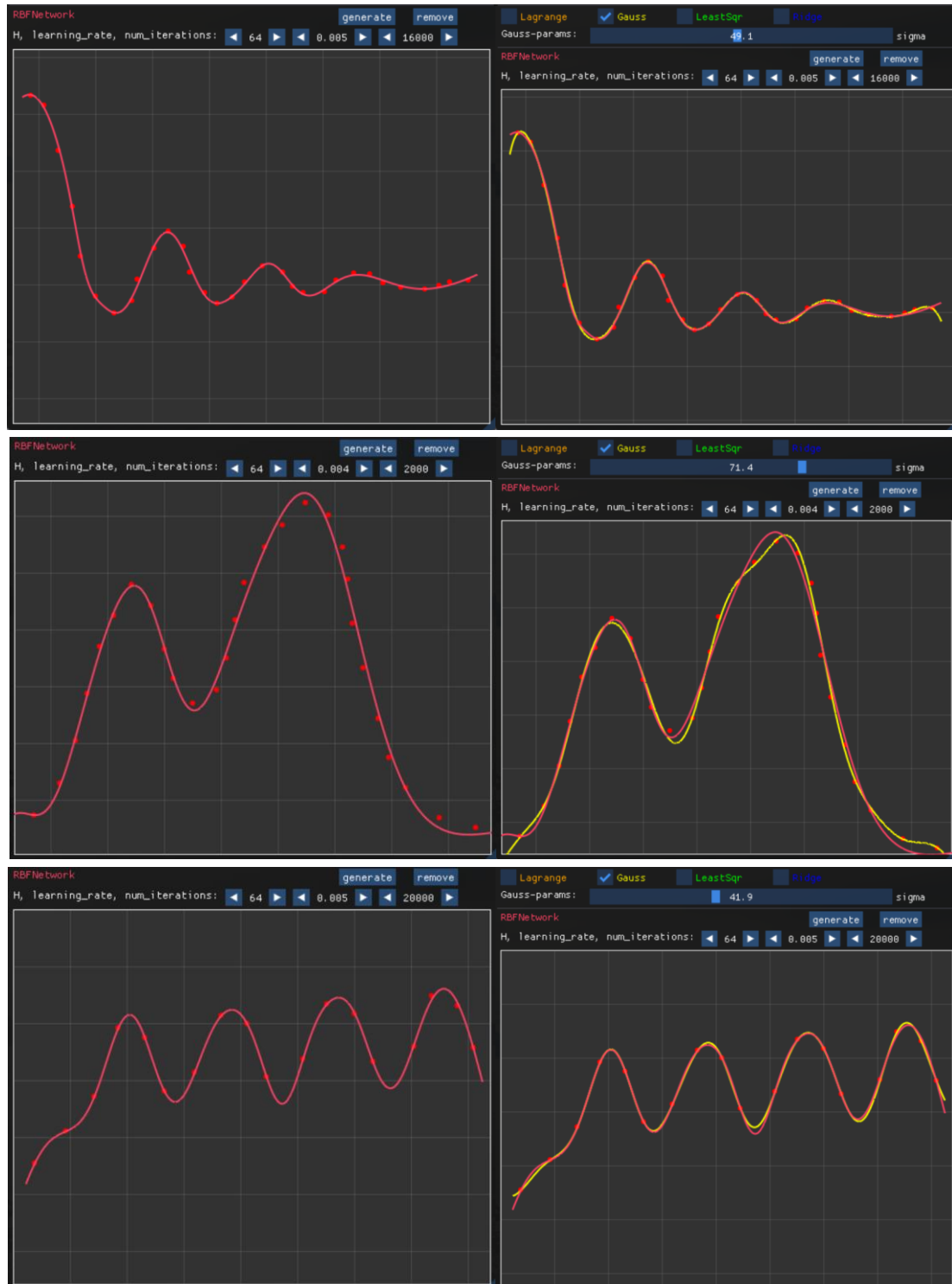
下图示出了对一组类似正弦曲线采样点的拟合结果:



对于采样点数较多且具有一定梯度变化的曲线，显然对 RBF 网络作 2k 次的迭代仍未能达到收敛（欠拟合），此时可通过增加迭代次数或学习率来减小损失、达到拟合，图中采用 0.02 的学习率代替原来的 0.005 以加速梯度下降过程，此时在 2k 次的迭代下即可取得较好的拟合结果且未发生超调。同时可以看到，对这种具有一定规律的梯度变化曲线，RBF 网络和高斯插值的拟合效果明显优于最小二乘逼近（多次调参也只能达到上图所示效果）。

2. 复杂曲线拟合

下图示出了 RBF 网络对多组不同曲线采样点的拟合结果及其与高斯插值拟合的对比 (所有结果都经过一定的超参数调整):



3. 结论

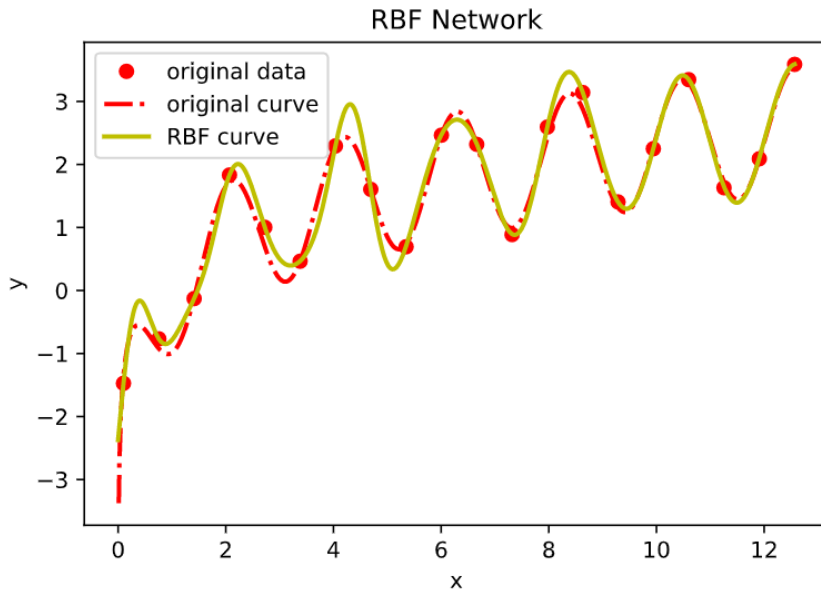
- a) 当偏差 bias 较大时, 网络欠拟合, 可通过增加迭代次数 epoch (训练更长时间)、调整学习率、构建更深、神经元更多的大型网络等方法加以修正;
- b) 当方差 variance 较大时, 网络过拟合, 可通过数据增广、添加正则约束、简化网络模型等方法加以修正;
- c) 最小二乘逼近对直线和次数较低、梯度变化较小的曲线拟合效果较好, 高斯插值对梯度变化较大的高次曲线拟合效果较好; 而通过不断调整、选取合适的超参数, RBF 网络可以对以上二者都达到理想的拟合结果。

附 1: 在 Python 脚本中用 matplotlib 库绘制拟合曲线

(以下两图各采样点的纵坐标都附加了一定的高斯随机噪声)

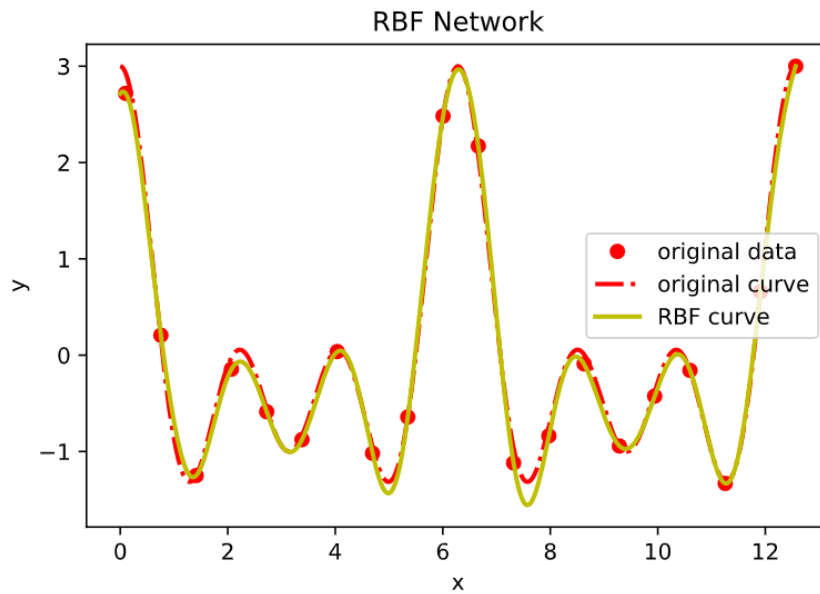
1. $f(x) = \ln x + \cos(3x)$

超参数: $H=64$, $\text{learning_rate}=0.005$, $\text{num_iterations}=15000$



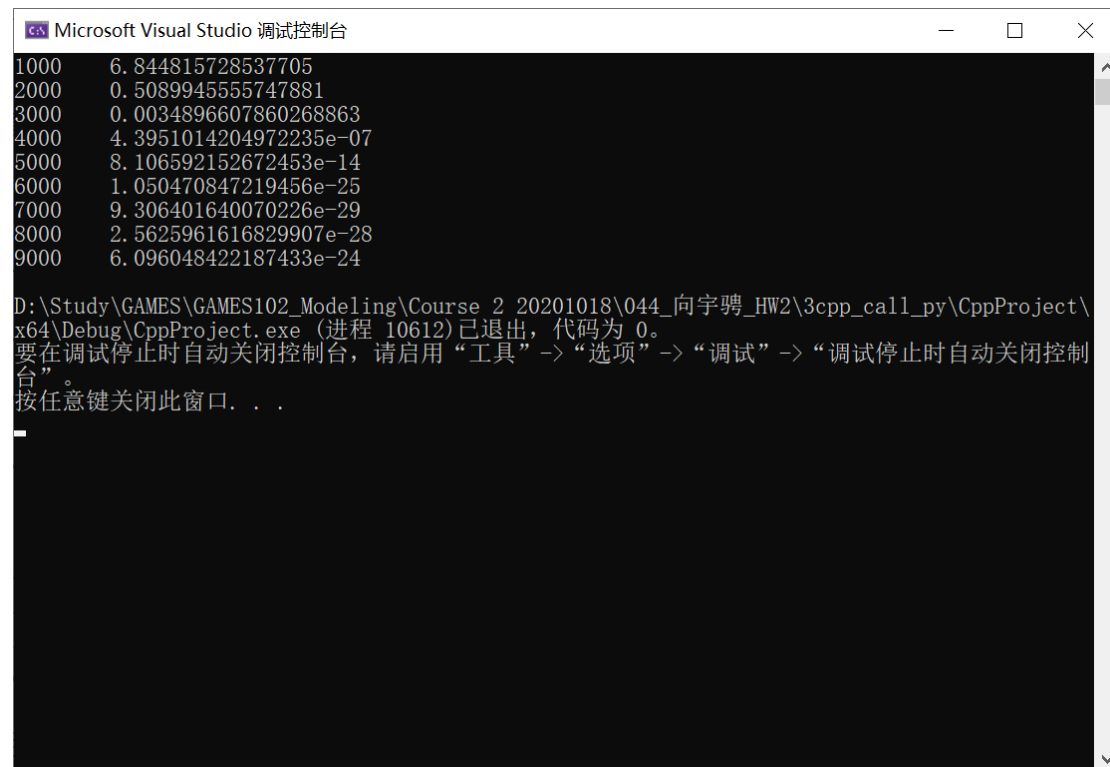
2. $f(x) = \cos x + \cos 2x + \cos 3x$

超参数: $H=64$, $\text{learning_rate}=0.005$, $\text{num_iterations}=20000$



附 2: C++调用 Python 脚本训练网络

下图示出了在 C++测试工程中调用 Python 脚本训练网络并在控制台上打印各 epoch 下损失函数的值:



The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio 调试控制台". The console output displays the loss function value for each epoch from 1000 to 9000. The values are as follows:

Epoch	Loss Function Value
1000	6.844815728537705
2000	0.5089945555747881
3000	0.0034896607860268863
4000	4.3951014204972235e-07
5000	8.106592152672453e-14
6000	1.050470847219456e-25
7000	9.306401640070226e-29
8000	2.5625961616829907e-28
9000	6.096048422187433e-24

Below the table, the console shows the following messages:

```
D:\Study\GAMES\GAMES102_Modeling\Course 2 20201018\044_向宇骋_HW2\3cpp_call_py\CppProject\x64\Debug\CppProject.exe (进程 10612) 已退出, 代码为 0。  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口. . .
```