参赛密码 _____

（由组委会填写）

“华为杯”第十四届中国研究生

数学建模竞赛

学　　校		电子科技大学
参赛队号		**10614023**
队员姓名	**1.**	张凤林
	2.	樊哿
	3.	张珺

参赛密码 _____

（由组委会填写）

“华为杯”第十四届中国研究生

数学建模竞赛

题 目 面向下一代光通信的 VCSEL 激光器仿真模型

摘 要

L-I 模型和带宽模型一直是 VCSEL 激光器研究的重点。对于 L-I 模型，引入了正则项、样本权重和线性整流函数对经典的 L-I 模型进行了改进。对于带宽模型，通过 Levenberg-Marquardt 算法估计出了模型参数并详细分析了各参数对 S21 曲线的影响，给出了增大带宽的方案，并提出了快速小信号等效带宽模型，用以降低模型复杂度。所提模型在不损失模型精度的情况下不仅提升了运算速度，而且在 3dB 和 10dB 处得到了更宽的带宽。

针对问题 1，首先推导了各个变量与模型参数之间的关系表达式，以实测数据为基础，利用最小二乘回归结合 Adagerd 算法求解得到了 L-I 模型的模型参数，检验了模型的 MSE 值为 4.9424E-6，误差率为 18.16%，证明了模型的可用性。之后通过得到的模型表达式绘制了不同温度下的 L-I 曲线。最后利用得到的 L-I 曲线求解出满足光功率条件下的激光器最高工作温度为 39.3 摄氏度。

针对问题 2，首先计算了 L-I 模型的精度值，采样点的精度最大值为 98.47%，最小值为 13.23%，波动范围相当大。分析产生误差的原因主要有三个：实测值的零值干扰、过拟合以及算法陷入局部最优。针对这三个问题，采用加入线性整流函数修正模型和加入权重参数的方法用以改善零值干扰；加入正则项用以改善过拟合现象；加入 Adam 算法和初始化偏差修正模型用以改善算法陷入局部最优

的现象。通过这 3 种方法得到了最新的 L-I 模型参数，利用新模型重新绘制了不同温度下的 L-I 曲线并与问题 1 中的曲线逐一进行了比较。重新考虑了问题 1 中提到的实际问题，在新模型下激光器的最高温度不能高于 29.2 摄氏度时，能够保证特定光功率下用户能够正常使用网络。模型结果分析表明，模型的 MSE 值为 4.7886E-8，误差率为 2.18%，相对于原始模型，误差率降低了 15.98%。同时模型推导了输入电流与输入电压间的关系，以避免模型对实测值的过度依赖。

针对问题 3，首先基于速率方程，建立了带宽模型，应用 Levenberg-Marquardt 算法求解得到了模型参数，绘制出了不同环境温度和不同偏置电流下的带宽响应曲线。通过绘制的曲线可分析得到：偏置电流一定的情况下，随着温度的升高（温度范围选取有局限），带宽响应曲线中的带宽越来越小，曲线的走势越来越陡峭。而在温度一定的情况下，随着偏置电流的增加（电流选取范围有局限），带宽响应曲线中的带宽会越来越大，曲线的走势越来越平坦。之后通过调整模型参数的方法获得了更宽的带宽，实验表明，当G_0或η_i变大时、ε或τ_P变小时，带宽就会越宽。同样的，通过调整参数的方法能够在不改变带宽的情况下获得更为平坦的曲线。结果表明有两个参数的性能最好，即：当G_0或η_i变大时，带宽变宽的同时曲线也更加平坦。

针对问题 4，提出了 Faster SSECM 模型，并给出了 4 种模型构造方式。通过 Levenberg-Marquardt 算法求解得到了模型参数，之后重点分析了 4 种构造方式下模型的运算时间与误差。结果表明，Faster SSECM-taoY 模型在 100%保持了原模型精度的同时，运行时间缩短至 0.001261 秒，加速比率达到了 50.65%。运行时间最短的是 Faster SSECM-taoYZ 模型，运行时间仅有 0.000746 秒，但其精度是原模型的 7.86%。结果分析中对 4 种模型的带宽也做了详细分析，结果表明，仅 Faster SSECM-taoYZ 模型的 3dB 带宽变小，其余模型无论是 3dB 带宽还是 10dB 带宽都变宽了。最后，选取了 4 种模型中综合性能最优的 Faster SSECM-taoY 作为改进带宽模型，绘制了不同温度下和不同偏置电流下的带宽响应曲线，并与问题 3 中得到的曲线逐一做了比较。

对模型评价后表明，改进的 L-I 模型以及提出的快速小信号等效带宽模型能够较为准确的反映 VCSEL 激光器的特性。文中提出的增加带宽及平坦度方案能够为通信设计提供一定的参考。

关键词：VCSEL；最小二乘回归；Adam 算法；Levenberg–Marquardt 算法；快速小信号等效带宽模型

目 录

一、问题重述与分析

通过阅读题目、附录相关文档以及相关参考文献，下面根据对题目的理解重新叙述该问题所要求做的工作，并对如何解决这些问题做了全面的分析。

1.1 问题重述

互联网技术的快速发展使得“光纤入户”逐步成为可能，而在进行光纤系统的设计之前，往往会通过计算机仿真的方式来研究系统设计的指标，以便找到最适合的解决方案。激光器是光纤通信系统的核心器件，考虑使用难度及功耗等问题，垂直腔面发射激光器（Vertical Cavity Surface Emitting Laser, VCSEL）最为常用，本题的主要任务就是需要得到准确反应 VCSEL 激光器特性的数学模型。

关于 VCSEL 的特性，最需要考虑的问题有两个：

- **激光器输出的功率强度与温度的关系**
- **如何设计激光器参数可以使激光器具有更大的传输带宽**

光功率强度与温度的关系到了激光器可以在多大的外界环境温度范围内使用。带宽也是现代生活中广泛的需求，参数设计显得尤为重要。如何建立模型来描述 VCSEL 的特性，并建立与两者之间的关系是本文的重点。

VCSEL 的温度模型在业界中得到了广泛的研究[1-2]，这些模型虽然可以较好的描述 VCSEL 的性能特性，但由于基于数值仿真，计算量巨大，并不适用于与其他电路模型相结合。Wipiejewski [3]提出了一种简单的模型，但只能计算器件的静态温度特性，并不能做动态仿真。由于模型简单、计算方便，Mena [4]等人提出的基于速率方程的模型在业界受到广泛的认可[5-7]。

由于 VCSEL 具有高速调制能力、低电流单纵模波操作能力[8-9]，VCSEL 的带宽模型也是业界研究的热点[10-12]。同时必须要考虑激光器输出光功率强度和器件温度的关系以及如何设计激光器参数可以使激光器具有更大的传输带宽。

下面重述题目中要求解决的问题。需要特别说明的是，下面的标题是该问题需要解决的关键问题，并不是全部。在问题的详细描述中，分析了需要解决的全部问题。

- **问题 1：求解 VCSEL 的 L-I 模型**
  通过给定的经验公式与实测数据，并结合附录中给出的参数经验值，重新确定模型的参数。以此为基础，重新绘制不同温度下的 L-I 曲线并解决一个实际问题，即：假定机房的 VCSEL 激光器在直流输入时输出的平均功率低于 2mW 时（此时用户不能接收正常的信号），根据得到的 L-I 模型来确定激光器的温度低于多少度时其工作正常。
- **问题 2：改进得到的 L-I 模型**
  分析得到的 L-I 模型精度与误差产生的原因，在正确分析的基础上，改进上一问题中得到的 L-I 模型，根据改进后的模型重新绘制不同温度下的 L-I 曲线，并与上一问题中得到的 L-I 曲线进行比较。
- **问题 3：求解 VCSEL 的带宽模型**
  建立激光器小信号幅频响应参数模型，给出参数构成以及确定方法（可以利用附录中所给的建模方法估计其参数），在此基础上，画出不同环境温度和不同偏置电流下的带宽响应曲线。利用该模型分析激光器的温度和偏置电流对器件带宽曲线的影响。并解决一些实际问题，诸如获得更宽带宽的方案、带宽曲线更加平坦的设计。

- **问题 4：对 VCSEL 带宽模型进行改进**
  建立运算速度更快的带宽模型，或者在相同的温度和偏置电流下，获得更宽的 10dB 带宽。并与问题 3 的模型进行比较。

1.2 问题分析

通过仔细研读相关资料，下面给出对以上四个问题的分析。从而为后面的模型建立与求解提供思路。

- **针对问题 1：求解 VCSEL 的 L-I 模型**
  问题 1 其实可以归结为三个小问题，即：确定模型参数；根据模型绘制 L-I 曲线；实际问题解决。
  (1)、确定模型参数是后两个小问题解决的关键，题目中给出了关于 VCSEL 的一些经验公式以及一组实测值，通过对公式的推导可以得到以模型参数和实测变量为未知数的一个公式。通过该公式并结合附件中所提供的一组实测数据，就可以应用最小二乘回归的方法对模型参数进行建模。在求解回归模型的算法选择上，梯度下降法等经典算法收敛速度较慢、误差较大。应采用改进后的新型算法对模型求解，这样得到的模型参数才更为准确。
  (2)、绘制不同温度下的 L-I 曲线需要根据最新的 L-I 模型，这一小问完全依赖于模型参数求解。特别需要注意的是，在已知最新 L-I 模型的情况下，方程的解可能存在不同情况，应对此进行必要的分析与解释。
  (3)、当电信机房里面的 VCSEL 激光器在直流输入时输出的平均光功率低于 2mW 时，无法进行正常的通信。根据建立的最新 L-I 模型推测，VCSEL 工作的环境温度应不高于多少度时，用户可以正常使用网络。这一问题应根据上一小问中绘制的不同温度下的 L-I 曲线来解决，曲线的纵坐标为光功率，只要做一条与横坐标平行的、穿过 2mW 坐标处的直线，就可以将是否能正常工作的温度曲线标记出来。在直线上方的温度曲线下，只要调整好电流强度就能够使 VCSEL 达到 2mW 的光功率，从而保证用户的光猫正常检测到信号。
- **针对问题 2：改进得到的 L-I 模型**
  这一问题要求首先对问题 1 中得到的模型进行误差分析，并分析产生误差的原因，进而改进 L-I 模型。最后重新绘制不同温度下的 L-I 曲线。
  (1)、将问题 1 模型得到的光功率值与实测值进行比较，能够得到模型的精度数据，而造成这种误差的原因很可能是在回归过程中存在过拟合、零值干扰等现象，所以应该重点对算法进行一些改进。
  (2)、将模型改进后重新绘制不同温度下的 L-I 曲线，并与问题 1 中的曲线进行比较，分析改进之后的模型效果是否优于问题 1 中得到的模型效果。
- **针对问题 3：VCSEL 的带宽模型**
  这一问题是题目的难点所在，问题的关键在于建立小信号幅频响应参数模型，或者以附录中所给模型为基础，通过参数估计的方法来得到不同温度下的带宽响应曲线。应解决的问题如下：
  (1)、将附录中所给小信号幅频响应参数模型进行参数估计，得到估计到的模型参数后，就能绘制在不同环境温度和不同偏置电流下的带宽响应曲线。
  (2)、利用得到参数后的模型通过改变激光器温度和偏置电流就可以评估这两项因素对带宽曲线的影响。
  (3)、由于影响带宽模型特性的主要是其参数，所以应从模型参数的调整入

手，尝试能否在限定的条件下获得更大的带宽。

(4)、该问主要是调整不同参数看图形结果，找到这些影响参数后，使其在可调范围内尽量接近能够带来平坦带宽的理想值附近。

- **针对问题 4：对 VCSEL 带宽模型进行改进**

  该题是开放性思考题。

  (1)、对于使模型运算速度更快这一问题，只能改变附录中的建模方法，以新的思路来建立模型。或者通过借鉴附录中的一些推导，加入一些新的建模思想来达到更好的效果。

  (2)、而想要达到相同温度和偏置电流下获得更宽的 10dB 带宽的效果，可以尝试改变模型的参数，或者增加一两个参数进行调节。

本文使用 MATLAB[14]和 Python[15-21]语言进行仿真实验，在 Jupyter[15]环境下运行代码。程序主要基于 PyTorch[16]框架编写，使用 Pandas[17]进行数据分析与清洗，通过 Numpy[17-18]进行矩阵运算，运用 SciPy[19]进行一些符号计算，最后应用 Matplotlib[20]与 Seaborn[21]输出图像。

二、模型假设与符号说明

由于该题目兼具理论性与现实性，一些现实因素应进行限制，以增加模型的准确性以及可行性。下面对模型做部分假设，对文章中出现的符号进行说明，以便读者清晰的了解本文的内容。

2.1 模型假设

下面提出几个问题假设，以便使其能够适用求解问题的模型。

- 假设环境温度不会突变，即 VCSEL 的工作环境比较稳定。
- 假设 VCSEL 激光器工作在直流条件下。
- 假设转换效率受温度影响较小，可以近似于常数。
- 对于一个激光器来说，只要在任一电流下其光功率可以超过光猫的最低检测标准，光猫便能正常工作。

2.2 重要符号说明

这一部分主要介绍正文中出现的较为重要的数学符号，以便读者能够读懂模型，从而整体把握文章思路。另外，虽然该部分列出了一些重要的数学符号，但正文部分针对较为重要的数学符号不免重复说明。

表 1 重要符号说明

符号	意义
S'	实测光功率与计算光功率间的误差
$\hat{m}_t$	梯度的第一时刻平均值
$\hat{v}_t$	梯度第二个时刻方差
φ	权重参数
P_{0j}	光功率实测数据值
$P_{0\max}$	数据的最大值
θ	权重主参数
N_s	载流子浓度的稳态理论值
S_s	光子数的稳态理论值
η_i	注入效率
N_0	透明载流子数
ε	增益压缩因子
β	受激辐射耦合系数

三、问题 1 模型的建立与求解

根据问题重述与分析中对问题 1 的分析，下面建立求解模型。

3.1 VCSEL 的 L-I 模型建立

该部分首先根据经验公式推导了 VCSEL 的驱动电流、电压和光功率三者的关系，得到了三者之间的关系式后（关系式中包含所有模型参数），利用已知数据进行回归分析，之后应用 Adagrad 算法[22]对回归模型进行求解计算。

3.1.1 基于经验公式的变量关系推导

一般情况下，影响 VCSEL 的 L-I 特性的各个物理量之间能够用下式表示。

$$P_0 = \eta(T)(I - I_{th}(N,T)) \tag{1}$$

其中，P_0是 VCSEL 的输出光功率，I 是注入到激光器的外部驱动电流，$\eta(T)$是与温度相关的斜坡效率，$I_{th}(N,T)$是与温度和载流子数目相关的阈值电流。

假设斜坡效率的温度依赖性对于输出功率影响很小，也就是将斜坡效率看成是常数。其次，模型中忽略了空间烧孔现象，这样阈值电流仅仅是温度的函数。即阈值电流可以表示为：

$$I_{th}(N,T) = I_{th0} + I_{off}(T) \tag{2}$$

其中I_{th0}为常数，$I_{off}(T)$是与温度相关的经验热偏置电流。综上，可以利用常值的斜坡效率和仅与温度相关的阈值电流来描述不同室温下的 L-I 曲线。此时，上面的公式(1)可以简化为，

$$P_0 = \eta(I - I_{th0} - I_{off}(T)) \tag{3}$$

通过以上的简化就可以避免过于偏重内部细节。为了进一步简化公式，可以用温度的多项式形式来模拟偏移电流。也就是将$I_{off}(T)$表示为：

$$I_{off}(T) = \sum_{n=0}^{\infty} a_n T^n \tag{4}$$

而上式中的温度T受外界环境温度T_0和自身的温度影响，自身的温度与器件产生的瞬时功率VI有关，也就是受 V-I 特性的影响。即：

$$T = T_0 + (VI - P_0)R_{th} - \tau_{th}\frac{dT}{dt} \tag{5}$$

其中，R_{th}为 VCSEL 的热力学阻抗，τ_{th}为热力学时间常数，T_0为环境温度，I和V分别是输入电流和输入电压。在直流条件下，

$$\frac{dT}{dt} = 0 \tag{6}$$

根据以上的推导与简化，将(4)、(5)两式带入到(3)式，可以得到各变量间的关系表达式，

$$P_0 = \eta(I - I_{th0} - I_{off}(T_0 + (VI - P_0)R_{th})) \tag{7}$$

该表达式包含了所有模型参数以及 VCSEL 的各个变量，根据该式就可以建

立参数估计的回归模型。

3.1.2 基于最小二乘回归的参数估计模型

根据 L-I 模型的关系式(7)，可以得到光功率和输入电压、电流的关系式，

$$f_1(P_0) = f(I,V) \tag{8}$$

用实测数据估计模型的参数，设实测数据的光功率、输入电流和输入电压分别为：

$$\begin{aligned} P_0 &= P_{0j} \quad j = 1,2,3,\cdots 1401. \\ I &= I_j \quad j = 1,2,3,\cdots 1401. \\ V &= V_j \quad j = 1,2,3,\cdots 1401. \end{aligned} \tag{9}$$

根据最小二乘准则的经验公式，

$$\sum_{i=1}^{m} \left| y_i - f(x_i) \right|^2 \tag{10}$$

可知实测数据的最小二乘估计需要满足误差 S' 最小。即

$$\min S' = \sum_{j=1}^{n} \left| (P_{0j}) - f(I_j, V_j) \right|^2 \tag{11}$$

其中 n=1401， S' 被称为损失函数。建立完最小二乘回归模型之后，在模型求解过程中使用 Adagrad 算法求得在 S' 最小情况下的各个模型参数。

3.2 基于 Adagrad 算法的 L-I 模型参数求解

对于最小二乘回归模型的求解，采用 Adagrad 算法对模型求解。Adagrad 在深度学习领域内是十分流行的算法，因为它能很快地实现优良的结果。经验性结果证明 Adagrad 算法在实践中性能优异，相对于其他种类的随机优化算法具有很大的优势。

3.2.1 Adagrad 算法原理

Adagrad 算法是一种基于梯度的优化算法，它能够对每个参数自适应寻求不同的学习速率，对稀疏特征，得到比较大的学习更新，对非稀疏特征，得到比较小的学习更新。

在 Adagrad 的更新规则中，对于学习率不设置固定值，每次迭代过程中，参数优化时使用不同的学习率 η，它会随着每次迭代而根据历史梯度的变化而变化。

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_t + \in}} \bullet g_{t,i} \tag{12}$$

其中， $G_t \in R^{d*d}$ 是一个对角矩阵，每个对角线位置 i 的值累加到 t 次迭代的对应参数 θ_i 梯度平方和。$\in$ 是平滑项，可以防止进行除零操作，一般取值 $1e-8$ 。

Adagrad 主要优势在于它能够为每个参数自适应不同的学习速率，而一般的人工都是设定为 0.01。

3.2.2 L-I 模型参数的求解

根据以上模型，并结合附件中给出的实测数据，可以利用 Adagrad 算法求解出最小二乘回归模型。

通过该算法，利用 Python 编程可得到模型的参数估计值。图 1 表示的是损失函数的值随迭代次数增加的变化情况。本次实验的初值利用附录所提供的参数。

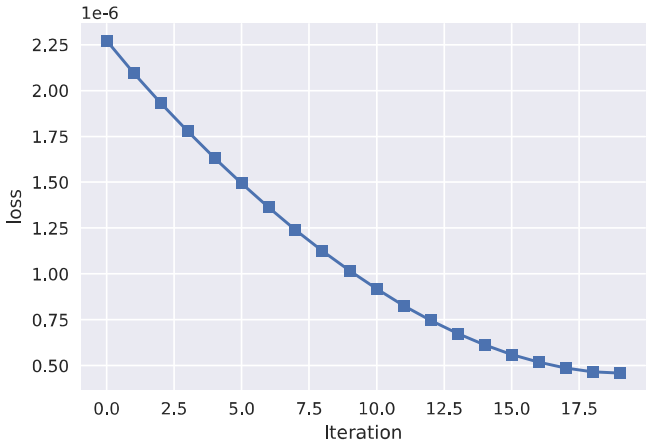

图 1 误差下降曲线

图 1 中横坐标表示迭代次数，纵坐标表示损失函数的值。从图 1 中可以看到，算法经过 14 步迭代后收敛，loss 值从 2.25E-6 下降到 0.5E-6 左右，此时在训练集上误差达到最小，参数估计效果最好。

计算得到的模型参数估计值如表 2 所示，

表 2 模型参数估计值

参数	估计值	附录所给参考初值
η	0.4999999837	0.5
I_{th0}	0.30001628E-3	0.3E-3
R_{th}	2.6E3	2.6E3
a_0	1.24601628E-3	1.246E-3
a_1	-2.54337081E-5	-2.545E-5
a_2	3.07098559E-7	2.908E-7
a_3	1.65566334E-8	-2.531E-10
a_4	1.63006796E-8	1.022E-12

从表 2 中可以看出，经过迭代，与题目附录表中提供的参考初值相比，几乎所有的参数都得到更新，其中，a_2，a_3，a_4 改变较大，可以看出，算法确实为模型寻找到了更优的参数。最优模型的 MSE 为 **4.9424E-06**，误差率为 **18.16%**。

3.3 不同温度下 L-I 曲线的绘制与应用实例分析

该部分首先绘制不同温度下的 L-I 曲线，然后根据得到的曲线图对问题 1 中的应用实例做出解答。

3.3.1 不同温度下 L-I 曲线的绘制

求解出模型参数后，将各个参数带入到推导出的各变量间关系表达式(7)，即：

$$P_0 = \eta(I - I_{th0} - I_{off}(T_0 + (VI - P_0)R_{th})$$

将数据带入模型进行计算，便可以画出不同温度下的 L-I 曲线进行对比分析。本次实验选取 $T_0 = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$，在此组温度下绘制 L-I 曲线，如图 2 所示。

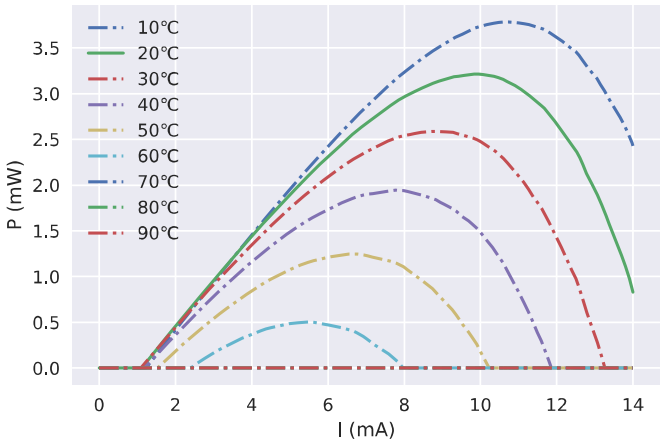

图 2 不同温度下的 L-I 曲线

图 2 的横坐标表示了输入电流 I，纵坐标表示了光功率 P，**题目中要求的 20 摄氏度下的 L-I 曲线在本图中用实线表示**，其余温度下的曲线用虚线表示。从图中可以看出，L-I 曲线都经历了一个先上升，后下降的过程。并且温度越低的曲线从最小开始上升，达到的峰值也最高，最上面的曲线为 10 摄氏度下的 L-I 曲线，第二条曲线为 20 摄氏度下的 L-I 曲线，以此类推。当温度达到一定程度后，激光器便不再工作。即在 70、80、90 摄氏度下 VCSEL 激光器均不存在光功率（此时曲线与横坐标平行），也即外部条件不足以使 VCSEL 激光器正常工作。

3.3.2 应用实例分析

当电信机房里面的激光器在直流输入时输出的平均光功率低于 2mW 时，用户的光猫无法正常检测到信号。那么，根据建立的 L-I 模型，只需要将光功率限制在 2mW 以上，并且方程存在实数解时，就可以确定此时的工作温度。

为了展示的更清晰，本文绘制了不同温度下的 L-I 曲线，并在其中做一条穿过纵坐标 2mW 并且与横坐标平行的直线，直线上方即为可行温度的区域。

由于肉眼观察到 2mW 直线穿过的区域基本上在 40 摄氏度 L-I 曲线的上方，为了更精确确定可行温度，特绘制了 35 到 43 摄氏度的 L-I 曲线，如图 3 所示。

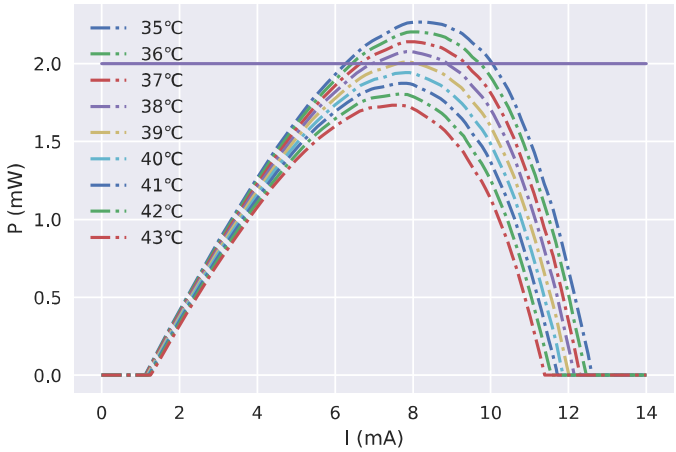

图 3 光功率大于 2mW 时的可行温度区域

图 3 的横坐标表示了输入电流 I，纵坐标表示了光功率 P。在图 3 中的直线上方即为可行区域，在图中能够看出，在 39 摄氏度附近时，电信机房里面的 VCSEL 激光器便不够保证用户正常的使用网络。

为了使模型的结果更为精确，通过模型求解得到当前光猫的最高工作温度为 39.3 摄氏度，与图中结果相符。

四、问题 2 模型的建立与求解

根据问题重述与分析中对问题 2 的分析，下面对问题 1 中得到的 L-I 模型进行相关研究。

4.1 L-I 模型精度与误差分析

该部分分析了 L-I 模型的精度，并对产生这种误差的原因做了探讨。本文采取 MSE 指标作为模型精度的评价标准，指标的计算公式如下：

$$MSE = \frac{1}{n}\sum_{t=1}^{n}\left(observed_t - predicted_t\right)^2 \tag{13}$$

4.1.1 L-I 模型精度分析

根据问题 1 得到的模型参数计算得到了该参数下的光功率值，与实测数据相比较，能够得到 20 摄氏度时该参数下 L-I 模型的精度值。下图 4 表示了光功率实际值与模型计算值的关系。

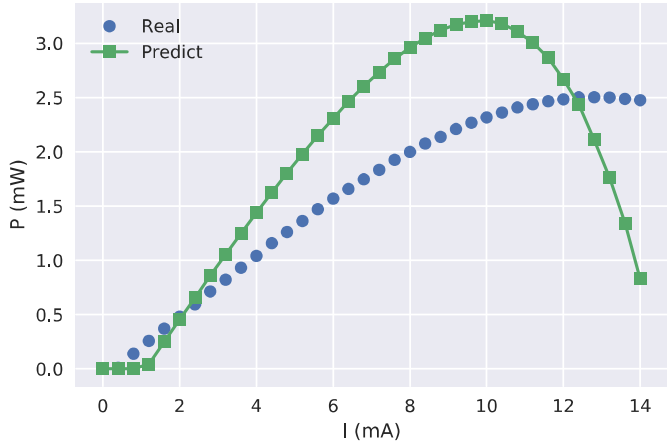

图 4 20 摄氏度下 L-I 模型的光功率误差曲线

图 4 的横坐标表示了输入电流 I，纵坐标表示了光功率 P。图中蓝色的圆点代表了光功率实测数据，绿线代表了新的模型参数下 L-I 模型计算得到的光功率值。特别需要说明的是，实线靠近横坐标的部分，即输入电流为 0-2mA 时，有一小段的光功率为零，这段零值是人为置零的结果，其真实值是负数（这也正是下一部分误差原因分析的重点）。

为了更好的展示模型的精确程度，本文以步长为 200 的方式等间隔采样了 8 个样本，计算其光功率预测值如表 3 所示。

表 3 L-I 模型光功率精度值

采样点	光功率实测数据	L-I 模型计算值	精度值
1	0.253088	0.033482	13.229%
2	0.538248	0.546481	98.471%
3	1.356633	1.972926	54.572%
4	1.830268	2.727623	50.971%
5	2.205671	3.170544	56.255%
6	2.437399	3.011809	76.434%
7	2.501925	1.782799	71.257%
8	2.477613	0.860503	34.731%

由表 3 可知，模型的最高精度值为 98.471%，最低精度值为 13.229%。精度范围变化较大，而且总体上来看，光功率的精度比较低，模型的参数还需要优化调整。通过对所有 1401 个数据进行计算得出，整个模型的 MSE 值为 **4.7886E-8**，误差率为 **2.18%**。相比于原始模型误差率下降了 **15.98%**。

4.1.2 L-I 模型误差原因分析

通过分析问题 1 中建立的模型，并对 L-I 模型参数的整个计算过程进行分析后，可能导致误差的原因有以下几个。

- **原因 1：实测值的零值干扰**

  由于实测数据中的起始部分存在零值，在原始模型中，不存在某一部分的值是恒为 0 的，可以预见，在计算整个模型的损失函数的时候，这一部分数据就成为了不可避免的误差，减小这部分误差，搜索到的最优模型会整体向这部分倾斜，这样估计的模型参数误差会较大。

  本文通过修正模型和引入样本权重来解决这一问题。

- **原因 2：过拟合**

  由于算法原因，有可能导致过拟合的现象出现，导致模型的参数不准确。在各变量间的关系表达式中，

$$P_0 = \eta(I - I_{th0} - I_{off}(T_0 + (VI - P_0)R_{th})) \tag{14}$$

$$I_{off}(T) = \sum_{n=0}^{\infty} a_n T^n \tag{15}$$

  由于 I_{off} 是一个多项式拟合函数，所以对最高项 n 的选择也是一个难题，选择较低的 n 可能欠拟合，模型不具备较好的表达能力；选择较大的 n 又可能导致模型过拟合。为了更好的分析此问题在本数据集上的情况，本文选取 n=[1, 2, 3, 4]进行试验并绘制拟合情况，如图 5 所示。

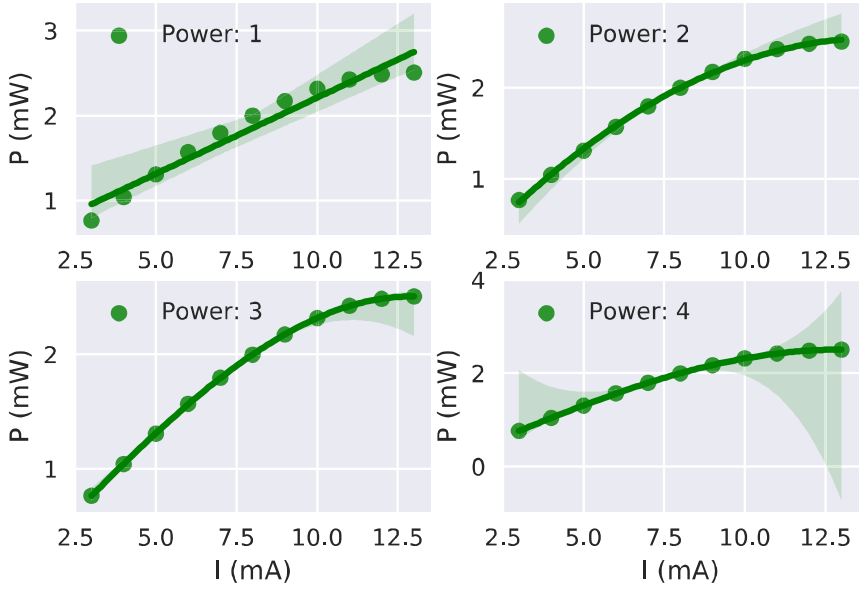

图 5 过拟合分析曲线

图 5 中横坐标是输入电流值，纵坐标是光功率值，实线代表了拟合函数，实线附近的阴影代表了拟合值与测试样本的误差，阴影越大，误差越大。

左上角的 1 图代表最高次幂为 1 的多项式时拟合的效果，可以很明显的看出，函数并没有很好的穿过样本，模型欠拟合；高次幂为 2、3 的多项式时拟合的效果相对较好，函数能完整的穿过训练样本，两端的误差也比较小；而最高次幂为 4 的多项式，虽然函数也能很好的穿过训练样本，但是在图像两端的误差却非常大，很明显出现了过拟合的现象。

所以，n 的过大可能导致过拟合，这是模型产生误差的一个重要的原因。为解决这一问题，我们引入正则项。

- **原因 3：陷入局部最优**

  由于问题的复杂性，算法有可能在优化过程中陷入局部最优值，从而导致拟合的效果并不好，进而导致模型参数的不准确。

4.2 L-I 模型的改进

根据上一节中对误差原因的探讨，下面将通过解决这些问题以达到模型改进的目的，进而得到更优的模型参数。由于 L-I 模型参数的调整范围很小，所以本文同时应用了以下三种方法对模型进行改进，以期收到良好的参数估计效果。

4.2.1 引入线性整流函数修正模型改善零值干扰

线性整流函数(Rectified Linear Unit, ReLU)[23]是人工神经网络中一种常用的激活函数，用于解决神经网络的梯度消失等问题。常见的线性整流修正函数的数学公式如下：

$$F(x)=\max(0,x) \tag{16}$$

可以看到，将线性整流函数与我们的模型相结合，可以预见到，模型中大于 0 的部分不会改变，而小于 0 的部分会被全部置为 0，不再影响模型最优的搜索，可以提高模型的精度。同时由于 ReLU 求导简单，在模型寻优时不会增加额外的负担，相对于其他修正函数，使用 ReLU 修正模型的计算成本大幅下降。

4.2.2 加入权重参数改善零值干扰

根据光功率样本值的数值都是非负数这一特点，可以在原模型的基础上对最小二乘估计式加入权重参数。

定义权重参数 φ 为：

$$\varphi=\varepsilon_w\theta+\lambda_w \tag{17}$$

式中，θ 为权重主参数，该参数由实测数据值的大小决定。ε 和 λ 都是调节参数，便于对模型进行细微调整，以获得更好的优化结果。下面推导权重主参数 θ 的公式。由于算法在优化的过程中考虑了零值的存在，就难免产生干扰，加入权重参数的目的就是尽量弱化零值的存在。

已知光功率实测数据值为 P_{0j}，记数据的最大值为 $P_{0\max}$。则权重主参数 θ 为

$$\theta=\frac{P_{0j}}{P_{0\max}} \tag{18}$$

上式能够将零值的权重降低，而加大其他实测值的权重。将权重参数加入到最小二乘估计式中可得，

$$\min S' = \sum_{j=1}^{n} \varphi_j \left| (P_{0j}) - f(I_j, V_j) \right|^2 \tag{19}$$

此时，零值以及较小值对于优化过程的影响就相对较小，再加上调节参数的存在能够对优化结果进行动态的调整，计算得到的模型参数能够得到优化。

4.2.3 加入正则项改善过拟合

随着训练过程的进行、复杂度增加，会出现训练出来的数据过拟合训练集，对训练集外的数据不工作的情况。

过拟合最终形成的拟合函数波动很大。因此过拟合的时候，拟合函数的系数往往非常大，在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。

而正则化是通过约束参数的范数使其不要太大，所以可以在一定程度上减少过拟合情况。因此采用 L2 正则化方法，防止过拟合现象的发生。

L2 正则化就是在代价函数后面加上正则化项，如下式所示：

$$\min S_0 = \min S' + \frac{\delta}{2k} \sum_{w} w^2 \tag{20}$$

其中，$\sum_{w} w^2$ 为所有参数平方的和，k 为训练集的样本大小，δ 是正则项系数，用于权衡正则项与 $\min S$ 项的比重。

L2 正则化项有让 w“变小”的效果，更小的权值 w，表示网络的复杂度更低，对数据的拟合效果很有帮助。

4.2.4 加入 Adam 算法改善优化效果

Adam 算法[24]即自适应时刻估计方法（Adaptive Moment Estimation），能计算每个参数的自适应学习率。

假定 $f(\theta)$ 为噪声目标函数：即关于参数 θ 可微的随机标量函数。我们对怎样减少该函数的期望值比较感兴趣，即对于不同参数 θ，f 的期望值 $E[f(\theta)]$。其中 $f1(\theta),\cdots,fT(\theta)$ 表示在随后时间步 $1,\cdots,T$ 上的随机函数值。这里的随机性来源于随机子样本（小批量）上的评估和固有的函数噪声。而表示 $ft(\theta)$ 为关于 θ 的梯度，即在实践步骤 t 下 ft 对 θ 的偏导数向量。

该算法更新梯度的指数移动均值 $m(t)$ 和平方梯度 $v(t)$，$m(t)$ 为梯度的第一时刻平均值，$v(t)$ 为梯度的第二时刻非中心方差值。

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{21}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{22}$$

其中，$\hat{m}_t$ 和 $\hat{v}_t$ 分别为梯度的第一个时刻平均值和第二个时刻方差。而参数 β_1 和 β_2 控制了这些移动均值指数衰减率。他们的范围是，

$$\beta_1 \in [0,1)$$
$$\beta_2 \in [0,1) \tag{23}$$

移动均值本身使用梯度的一阶矩（均值）和二阶原始矩（有偏方差）进行估计。然而因为这些移动均值初始化为 0 向量，所以矩估计值会偏差向 0，特别是在初始时间步中和衰减率非常小（即 β 接近于 1）的情况下是这样的。但初始化偏差很容易抵消，因此我们可以得到偏差修正的估计。

算法的效率可以通过改变计算顺序而得到提升，即使用参数更新的最终公式。

$$\alpha_t = \alpha \bullet \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \tag{24}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha_t}{\sqrt{\hat{v}} + \in} \hat{m}_t \tag{25}$$

式(14)、(15)为参数更新的最终公式。

4.2.5 加入 Adam 算法初始化偏差修正

根据算法原理部分的叙述，Adam 利用了初始化偏差修正项。本部分将由二阶矩估计推导出这一偏差修正项，一阶矩估计的推导完全是相似的。首先我们可以求得随机目标函数 f 的梯度，然后我们希望能使用平方梯度的指数移动均值和衰减率 β_2 来估计它的二阶原始矩。

令 $g1,\cdots,gT$ 为时间步序列上的梯度，其中每个梯度都服从一个潜在的梯度分布 $gt \sim p(gt)$。现在初始化指数移动均值 $v0=0$，而指数移动均值在时间步 t 的更新可表示为：其中 $gt \wedge 2$ 表示对应元素之间的乘积。同样我们可以将其改写为在前面所有时间步上只包含梯度和衰减率的函数，即消去 v：

$$v_t = (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \bullet g_i^2 \tag{26}$$

我们希望知道时间步 t 上指数移动均值的期望值 $E[v_t]$ 如何与真实的二阶矩相关联，所以可以对这两个量之间的偏差进行修正。下面同时对表达式(17)的左边和右边取期望，得到：

$$E[vt] = E[g_t^2] \bullet (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} + \zeta = E[g_t^2] \bullet (1-\beta_2) + \zeta \tag{27}$$

如果真实二阶矩 $E[g_t^2]$ 是静态的，那么 $\zeta = 0$。否则 ζ 可以保留一个很小的值，这是因为应该选择指数衰减率 β_1 以令指数移动均值分配很小的权重给梯度。所以初始化均值为零向量就造成了只留下了 $(1-\beta_t \wedge 2)$ 项。因此在算法中除以了 ζ 项以修正初始化偏差。

4.3 改进 L-I 模型的求解

根据上一节中对模型的修正，下面对问题 1 中的模型进行改进求解，相关求解代码请查看附录中程序部分。

改进模型之后的模型参数估计值如表 4 所示，

表 4 改进模型后的参数估计值

参数	改进后估计值	改进前估计值	附录所给参考初值
η	0.4999995	0.4999999837	0.5
I_{th0}	0.30051E-3	0.30001628E-3	0.3E-3
R_{th}	2.6E3	2.6E3	2.6E3
a_0	1.24650755E-3	1.24601628E-3	1.246E-3
a_1	-2.49432942E-5	-2.54337081E-5	-2.545E-5
a_2	7.96413545E-7	3.07098559E-7	2.908E-7
a_3	0	1.65566334E-8	-2.531E-10
a_4	0	1.63006796E-8	1.022E-12

从表 4 中可以看出，相较于原始模型，我们提出的改进模型估计出来的参数有较大变化，其中 a_3 、 a_4 的变化最为明显，由于采用了 L1 的正则项， a_3 、 a_4 的值在求解最优时被置为 0。

为了更好的展示求解最优模型的过程，本文绘制了损失函数随迭代次数增加的变化情况如图 6 所示。

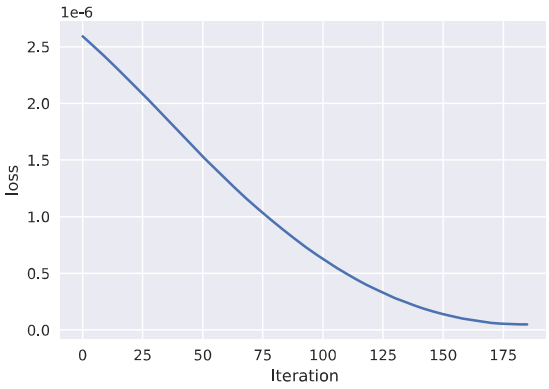

图 6 误差的均方根变化曲线

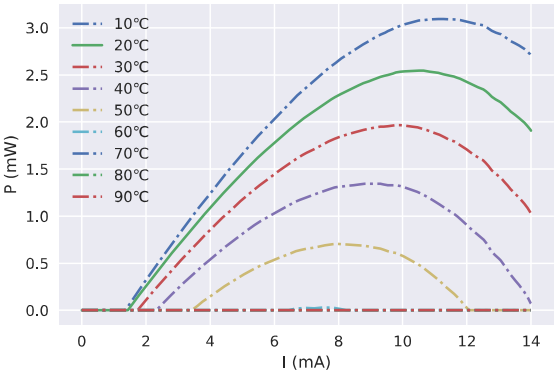

图 7 不同温度下的 L-I 曲线

图 6 的横坐标代表了迭代次数，纵坐标代表了损失函数的值，从图 6 中可以看到，损失函数值从开始的 2.5E-6 下降到 0 附近，在 175 步左右收敛到最优。

通过改进模型绘制出不同温度下的激光器L-I曲线如图8所示，和前文相似，本次实验选取的初始温度集合依然为 $T_0 = \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$。

图 7 中的横坐标是输入电流值，纵坐标是光功率值，相似的，20°C 下的 L-I 曲线用实线表示，其余温度下的 L-I 曲线用虚线表示。从图 7 中可以看出，同原始模型一样，非零的 L-I 曲线都经历了一个先升后降的过程，但相比原始模型，改进模型在同温度下 L-I 曲线的峰值更低，这也符合实测数据情况。在此模型下，模型是恒大于等于 0 的，会自动求解出光功率结果，不需要后续小于 0 的部分。

为了更好的比较不同温度下的结果，本文将所提出的改进模型得到的不同温度下的 L-I 曲线与原始模型的 L-I 曲线相比较，为清晰起见，不同温度下的 L-I 曲线分别画在不同的坐标轴下。由于在 60°C 之后模型的数值恒等于 0 了，所以此次试验只绘制了 $T_0 = \{10, 20, 30, 40, 50, 60\}$ 的对比图如图 8 所示。

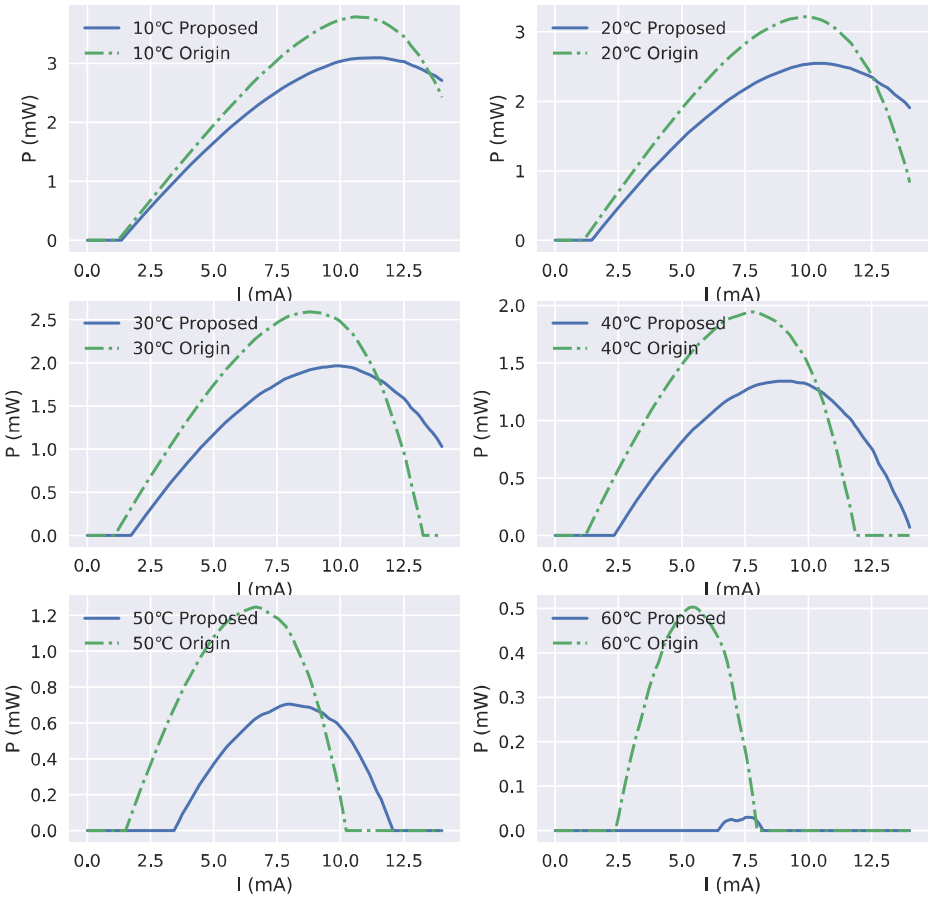

图 8 不同温度下的 L-I 曲线

图 8 中横坐标代表输入电流，纵坐标代表光功率，从图 8 可以看出，相对于原始模型，温度越低曲线的变化情况越明显。在 10℃ 时原始模型的曲线峰值接近 4，本文所提模型的曲线峰值超过 3，相差 25%左右；在 20℃ 时原始模型的曲线峰值在 3.2 左右，本文所提模型的曲线峰值超过 2.5，相差 28%左右；在 30℃ 时原始模型的曲线峰值超过 2.5，本文所提模型的曲线峰值接近 2，相差 20%左右；在 40℃ 时原始模型的曲线峰值接近 2，本文所提模型的曲线峰值在 1.4 左右，相差 30%左右；在 50℃ 时原始模型的曲线峰值接近 1.2，本文所提模型的曲线峰值超过 0.5，相差 42%左右；在 60℃ 时原始模型的曲线峰值接近 0.5，本文所提模型的曲线峰值接近于 0，只有轻微的凸起。

相对于原始模型，本文所提的改进模型光功率值开始上升时所对应的电流也更大，并随着温度的增大越发明显，在 10℃、20℃时两者非常接近，只是有轻微差别；而在 30℃、40℃ 时，两个模型差别比较明显；在 50℃、60℃ 时，两个模型差别非常明显，相差超过 100%。

对于改进模型，本文同样的分析了问题 1 中所提到光猫能否使用的问题。假定电信机房里的激光器在直流输入时输出平均光功率低于 2mW 时，用户的光猫无法检测到信号。那么，在电信机房里 VCSEL 激光器工作的环境温度最高不能超过多少度才能保证用户可以正常的使用网络。

在改进模型中，该问题的解无疑将会改变。图 9 绘制出了 2mW 左右的光功率下较为接近的 L-I 曲线，如下所示。

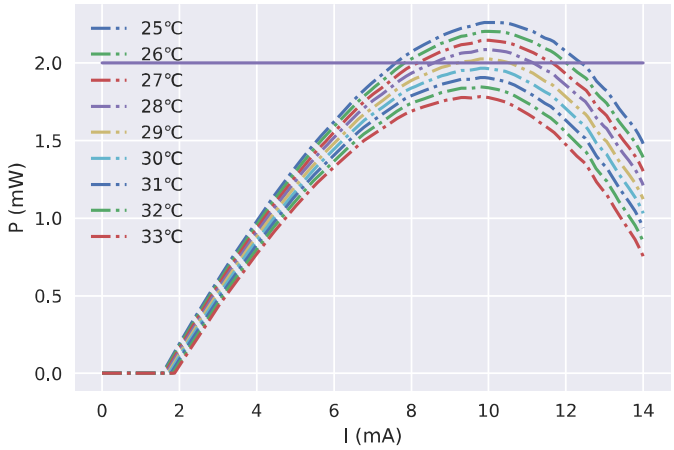

图 9 改进模型光功率大于 2mW 时的可行温度区域

图 9 中横坐标代表输入电流，纵坐标代表光功率，2mW 直线上方即为可行区域。通过观察图 9，可以发现在当温度不超过 29 摄氏度时，电信机房里面的 VCSEL 激光器能够保证用户正常的使用网络。

为了使模型的结果更为精确，通过模型求解得到当前光猫的最高工作温度为 29.2 摄氏度，于图中结果相符。

特别的，为了减少变量关系式中的变量个数，现将输入电流和输入电压的关系式导出，以达到简化模型的目的。

通过电流的多项式对电压进行拟合可计算出两者的关系式，如图 10 所示。

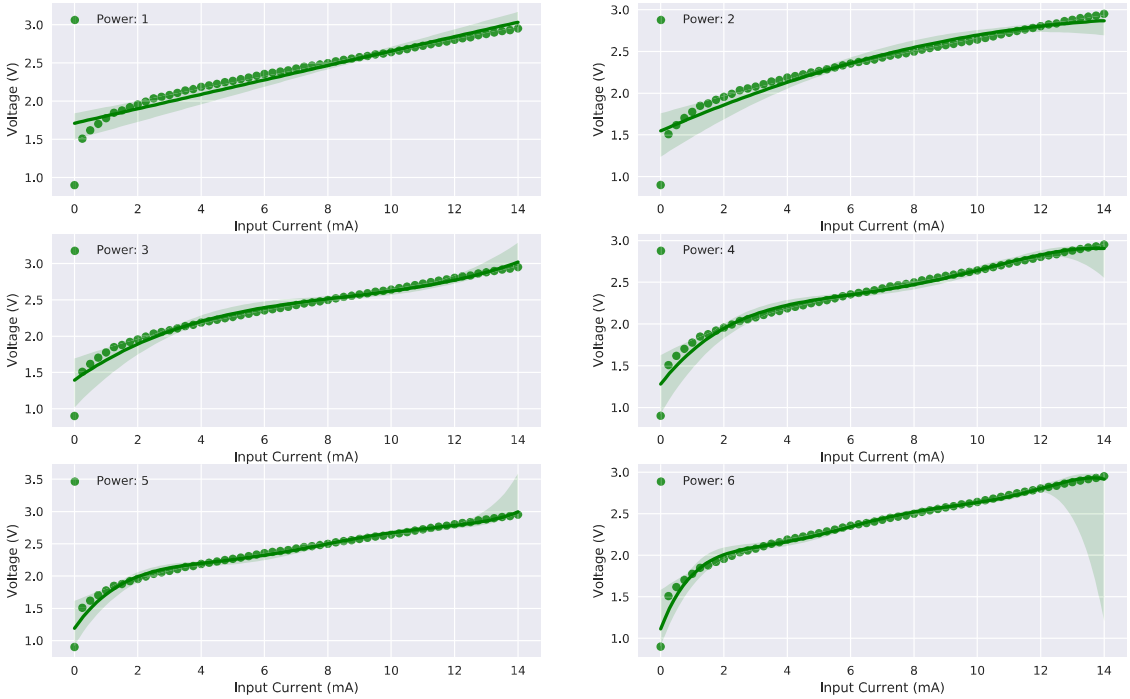

图 10 输入电流多项式拟合电压曲线图

图 10 中横坐标表示输入电流，纵坐标表示输入电压。从左上方到右下方分别展示了最高次幂为 1-6 的多项式拟合电压的效果以及误差情况，图中的淡绿色阴影部分面积即为拟合的误差大小。最高次幂为 1、2、3 的多项式函数拟合状况不好，而最高次幂为 6 的函数出现了过拟合状况，最高次幂为 4、5 的多项式拟合效果较好，既能有效的穿过所有样本点，又没有产生较大的测试误差。考虑到模型运算和求解的简便性，本文选取了 4 次多项式。

最终得到了输入电压和输入电流之间的关系，如下式：

$$U = a_0 + b_0 I + c_0 I^2 + d_0 I^3 + e_0 I^4 \tag{28}$$

关系式中的参数如下表所示。

表 5 改进前后模型光功率精度值

参数	a_0	b_0	c_0	d_0	e_0
数值	1.27903	4.82719E-1	-8.89868E-2	7.8071E-3	-2.37271E-4

在利用电压估计到电流之后，模型就能减少一个参数的输入，提升模型性能。

4.4 改进 L-I 模型的结果分析

根据改进 L-I 模型后的模型参数计算得到了该参数下的光功率值，与实测数据相比较，能够得到 20 摄氏度时该参数下 L-I 模型的精度值。下图 11 表示了光功率实际值与改进后模型计算值之间的差异。

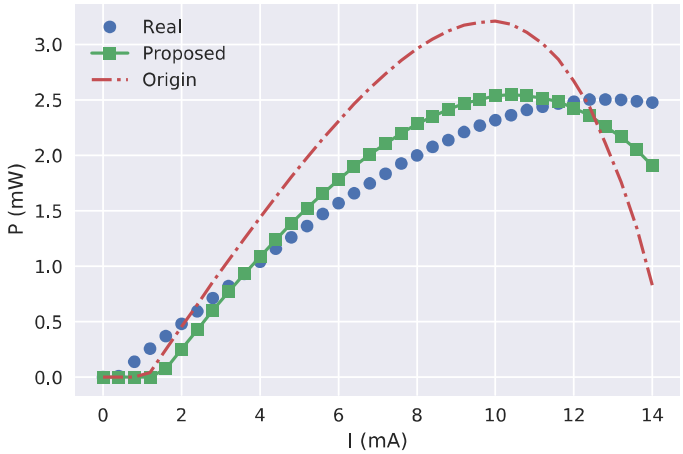

图 11 20 摄氏度时改进 L-I 模型的光功率误差曲线

图 11 中横坐标表示输入电流，纵坐标表示输入电压。图中的蓝色圆点是真实值，红色点画线表示的是未改进时模型的光功率计算值，而绿色的线表示的是改进之后模型下的光功率计算值，很明显的可以看出误差大大降低了。

由于插图并不能精确得到光功率的精度值，下面计算同样的 8 个等间隔采样点的光功率精度值。为了能够有所对比，这八个采样点和改进前精度分析中的采样点相同。如表 6 所示。

表 6 改进前后模型光功率精度值

采样点	实测数据	改进前计算值	精度值	改进后计算值	精度值
1	0.253088	0.033482	13.229%	0.125341	49.525%
2	0.538248	0.546481	98.471%	0.334032	62.059%
3	1.356633	1.972926	54.572%	1.518411	88.075%
4	1.830268	2.727623	50.971%	2.099691	85.301%
5	2.205671	3.170544	56.255%	2.463859	88.294%
6	2.437399	3.011809	76.434%	2.515362	96.801%
7	2.501925	1.782799	71.257%	2.172458	86.832%
8	2.477613	0.860503	34.731%	1.917004	77.373%

由表可知，改进后模型的整体精度值较高，从图 11 总体上来看，模型的参数优化较为成功。

五、问题 3 模型的建立与求解

根据问题重述与分析中对问题 3 的分析，下面对小信号带宽模型进行分析。

5.1 小信号幅频响应参数模型的建立

该部分根据附录 2 给出的一种基于速率方程的建模方法来完成模型的建立，其中繁琐的推导过程并不一一列写，只列出对模型求解较为关键的式子。

将偏置电流和注入激光器的外部驱动电流带入到激光器速率方程，可以得到：

$$\begin{cases} \dfrac{dN}{dt}=\dfrac{\eta_i\left(I-I_{th0}-I_{off}\left(T\right)\right)}{q}-\dfrac{N}{\tau_n}-\dfrac{G_0\left(N-N_0\right)S}{1+\varepsilon S} \\ \dfrac{dS}{dt}=-\dfrac{S}{\tau_p}+\dfrac{\beta N}{\tau_n}+\dfrac{G_0\left(N-N_0\right)S}{1+\varepsilon S} \end{cases} \tag{29}$$

其中，N 为载流子数，t 为时间，η_i 为注入效率，q 为电子电量，取值 $1.6\times10^{-19}C$，N_0 为透明载流子数，I 为注入的外部驱动电流，$I_{off}(T)$ 为与温度相关的偏置电流，τ_n 为载流子复合寿命，τ_p 为光子寿命，G_0 为增益系数，S 为光子数，β 为受激辐射耦合系数。

VCSEL 输出的光功率与光子数成正比，假定比例因子为 k，

$$P_0=kS \tag{30}$$

给 VCSEL 加载上小信号，

$$\begin{aligned} I\left(t\right)&=I_s+i\left(f\right)e^{j2\pi ft} \\ N\left(t\right)&=N_s+n\left(f\right)e^{j2\pi ft} \\ S\left(t\right)&=S_s+s\left(f\right)e^{j2\pi ft} \end{aligned} \tag{31}$$

其中，$i(f)$，$n(f)$ 和 $s(f)$ 均足够小。省略掉小信号响应数学表达式推导的过程，载流子浓度和光子数的数学表达式如下：

$$N_s=\frac{P_0/(k\tau_p)+G_0N_0P_0/(k+\varepsilon P_0)}{\beta/\tau_n+G_0P_0/(k+\varepsilon P_0)} \tag{32}$$

$$S_s=\frac{\eta_i\left(I-I_{th0}-I_{off}\left(T\right)\right)/q-N_s/\tau_n}{G_0\left(N_s-N_0\right)} \tag{33}$$

通常，在通信系统中信号响应有如下形式，

$$h(f)=\frac{X}{(i2\pi f)^2+(i2\pi f)Y+Z} \tag{34}$$

其中，

$$Y=\frac{1}{\tau_p}+\frac{1}{\tau_n}+\frac{G_0P_s}{k+\varepsilon P_s}-\frac{G_0\left(N_s-N_0\right)}{\left(1+\varepsilon P_s/k\right)^2} \tag{35}$$

$$Z=\frac{1}{\tau_p\tau_n}+\frac{G_0P_s}{\tau_p\left(k+\varepsilon P_s\right)}-\frac{\left(1-\beta\right)G_0\left(N_s-N_0\right)}{\tau_n\left(1+\varepsilon P_s/k\right)^2} \tag{36}$$

最后得到归一化小信号响应为

$$H\left(f\right)\equiv\frac{h\left(f\right)}{h\left(0\right)}=\frac{Z}{\left(j2\pi f\right)^2+\left(j2\pi f\right)Y+Z} \tag{37}$$

5.2 小信号幅频响应参数模型的求解

在模型求解中，采用莱文贝格－马夸特(Levenberg–Marquardt)算法进行求解。Levenberg–Marquardt 用以解决非线性最小二乘法问题，它第一次被 Kenneth Levenberg 提出[25]，后被 Donald Marquardt[26]修正。Levenberg–Marquardt 算法介于梯度下降法和高斯牛顿法（Gauss-Newton）中间，Levenberg–Marquardt 比 Gauss-Newton 更强大。这意味着，在许多情况下，它能够找到一个解决方案，即使它开始时距离最终的最优值很远。Levenberg–Marquardt 也可以看作是用信赖域方法改进的高斯–牛顿法。

Levenberg–Marquardt 法迭代方程为：

$$x^{s+1}=x^s+\Delta \tag{38}$$

其中，步长 Δ 的表达式为，

$$\Delta=-(J_f^TJ_f+\lambda E)^{-1}J_f^Tf \tag{39}$$

$$J_f=\begin{bmatrix}\frac{\partial f_0}{\partial x_0} & \cdots & \frac{\partial f_0}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_n}{\partial x_0} & \cdots & \frac{\partial f_n}{\partial x_n}\end{bmatrix} \tag{40}$$

其中，E 为单位矩阵。由公式可知，如果比例系数 $\lambda=0$，则为高斯-牛顿法，如果 λ 取值很大，则 Levenberg–Marquardt 接近梯度下降法。每迭代成功，则减小一次，这样在接近误差目标的时候，逐渐与高斯-牛顿法相似。

由于 Levenberg–Marquardt 算法利用了近似的二阶导数信息，它比梯度下降法快的多，可以较梯度下降法提高速度十几甚至上百倍。当 λ 足够大时，总可以保证 $(J_f^TJ_f+\lambda I)$ 是正定的，从而保证其可逆。

算法的每次迭代都对 λ 进行自适应调整。当接近解时，λ 逐渐减小，权值调整类似于高斯-牛顿法，利用类似于二阶导数的信息，可以快速收敛到最优解。当远离解时，λ 逐渐增大，权重调整又类似于梯度下降法，可以进行全局搜索。所以 Levenberg–Marquardt 算法同时具备了牛顿法和梯度法的优点。

利用提供的参数可能的初值作为迭代初值，得到了 Levenberg–Marquardt 算

法优化后的模型参数值，如表 7 所示。

表 7 模型参数估计值

参数	估计值	附录所给参考初值
η_i	0.290087811944204	0.7
β	-0.433751523591667	1E-5
τ_n	-0.012299727069767	9.6E-9
k	-3.60187228705102	1.5E-8
G_0	1.8E6	1.8E6
N_0	4.97E5	4.97E5
τ_P	6.99206216358124E-12	3.8E-12
ε	6.67052986037427E-6	4.7E-8

从表 7 中可以看出，经过迭代计算得到的模型参数值与附录表中提供的参考初值相比，参数相差较大，验证了函数确实在最优值附近区间中搜寻到了比初始值更优的值。

为了探究环境温度对相应曲线的影响，本文绘制出了在同一偏置电流下（偏置电流为 7.5mA），不同的环境温度下的带宽响应曲线。

由于在一些境温度下带宽响应曲线差别不大，为了清晰起见，分别绘制了 10 种温度下的带宽响应曲线和几个典型环境温度下的带宽响应曲线，如图 12、13 所示。

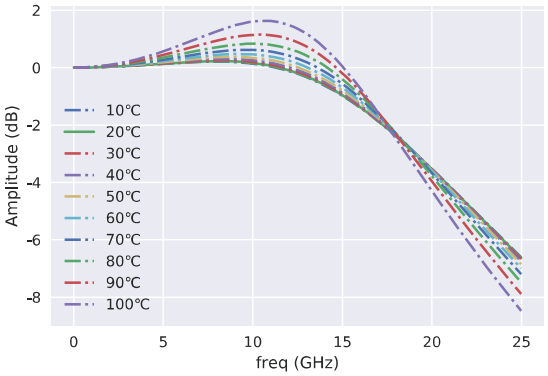

图 12 10 种温度下的带宽响应曲线

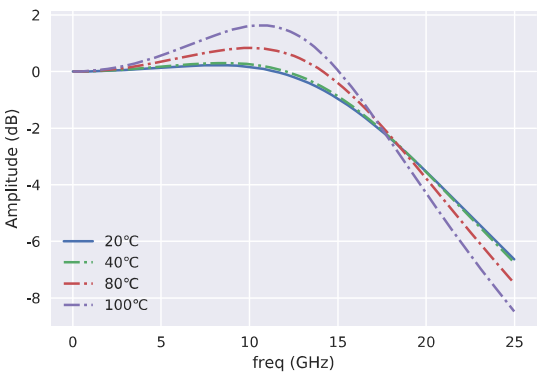

图 13 4 种温度下的带宽响应曲线

在图 12 和图 13 中横坐标表示频率，纵坐标表示不同频率对应的幅度，其中包含有 20 摄氏度下偏置电流为 7.5mA 时的仿真输出曲线。图 12 为 10 种不同温度下的带宽响应曲线，为了更好的体现 20 摄氏度的带宽响应曲线效果，又绘制了图 13 所示的 4 种不同温度带宽响应曲线。**两图中的实线代表的就是 20 摄氏度下 7.5mA 偏置电流时的带宽响应曲线。**

从图 13 中可以看出，在温度较低时，相应曲线比较平滑，从开始便缓慢下降，在温度较高时，相应曲线经历了一个先上升后下降的过程，并且在相同的温差下，温度越高曲线峰值越高，下降越迅速。

为了探究偏置电流对相应曲线的影响，本文绘制出了同一环境温度下（环境温度为 20 摄氏度），不同的偏置电流下的带宽响应曲线，其中包含有 20 摄氏度下偏置电流为 7.5mA 时的仿真输出曲线。

同样的，由于绘制了很多个偏置电流下的带宽响应曲线且部分曲线比较接近，为了清晰起见，分别绘制了 8 种偏置电流下的带宽响应曲线和几个典型偏置电流下的带宽响应曲线。如图 14、15 所示。

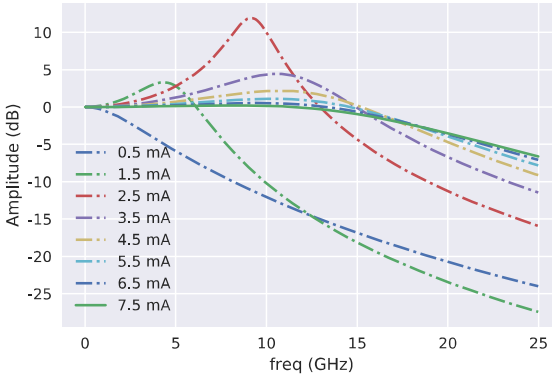

图 14 8 种电流下的带宽响应曲线

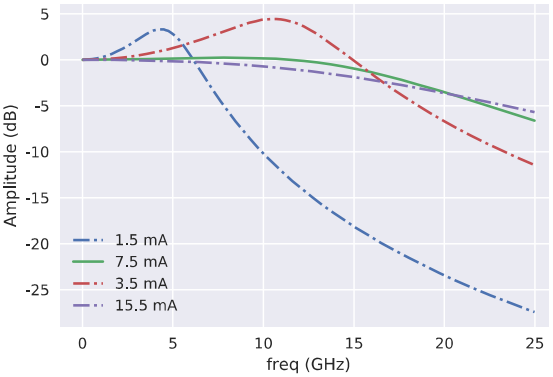

图 15 4 种电流下的带宽响应曲线

在图 14 和图 15 中，横坐标表示频率，纵坐标表示不同频率对应的幅度，图 14 为 8 种不同偏置电流下的带宽响应曲线，为了更好的体现 7.5mA 时的带宽响应曲线效果，又绘制了图 15 所示的 4 中不同偏置电流时的带宽响应曲线。**两图中的实线代表的就是 20 摄氏度下 7.5mA 偏置电流时的带宽响应曲线。**

和图 12、图 13 相似，不同电流下详细曲线的走势也大不相同，在低电流下，相应曲线经历了一个先上升后下降的过程，在高电流下，相应曲线缓慢下降。和不同温度下的相应曲线有所区别的是，非常低的电流下相应曲线从开始便下降迅速，随着电流增高，相应曲线的峰值也经历了一个先上升，后下降的过程，在偏置电流为 2.5 mA 左右达到最高。

5.3 激光器温度和偏置电流对器件带宽曲线的影响

为了更清晰的叙述温度和偏置电流对相应曲线的影响，本文总结如下。

5.3.1 激光器的温度对器件带宽的影响

根据图 12、13 得到的同一偏置电流时不同温度下的带宽响应曲线，可以做如下分析。

- **影响 1：温度能够改变带宽大小**
  由图可知，随着温度的不断升高，带宽响应曲线中器件的带宽范围就会越来越小，这对于追求高带宽的器件来说，高温严重影响了用户的上网速度，所以在实际应用中，应该尽量降低器件的温度，以达到更宽的带宽。
- **影响 2：温度能够改变带宽曲线的平坦度**
  由图可知，随着温度的升高，可以明显的看出器件的-3dB 带宽范围内的带宽曲线越来越陡峭，而-3dB 带宽范围外的带宽曲线同样随着温度的升高而变得越来越陡峭，而实际应用中的需要是，尽可能的使-3dB（或-10dB）带宽范围内的曲线平坦。

5.3.2 激光器的偏置电流对器件带宽的影响

根据图 14、15 得到的同一温度时不同偏置电流下的带宽响应曲线，可以做如下分析。

- **影响 1：偏置电流能够改变带宽大小**

由图可知，基本上（0.5 和 1.5mA 相反）随着偏置电流的不断升高，带宽响应曲线中器件的带宽范围会越来越大，这对于追求高带宽的器件来说，在实际应用中应该在合理范围内尽可能的增加器件的偏置电流。

- **影响 2：偏置电流能够改变带宽曲线的平坦度**

由图可知，激光器的偏置电流越低，器件的带宽曲线越不稳定或者说越陡峭。特别是较低偏置电流时，这一现象尤为明显。

5.4 固定温度和偏置电流时更宽带宽的实现

通过改变带宽模型的参数可以达到更宽带宽的目的，下面是改变某个参数值时，器件带宽响应曲线的变化情况，从中可以明显的看出带宽的范围变化。如图 16 所示。

由于不同参数的数量级差别很大，所以本文研究了最优参数值下参数改变-10%至 10%的情况，每 1%的间隔选为研究点。例如本文求解出的τ_n的最优参数值为-0. 0122997270697670，则本文在其他参数不变的情况下，选取了τ_n为{-0.00614986353488,-0.00737983624186,-0.00860980894884，-0.00983978165581, -0.0110697543628, -0.0122997270698, -0.0135296997767, -0.0147596724837, -0.0159896451907, -0.0172196178977 }这十个点进行试验。

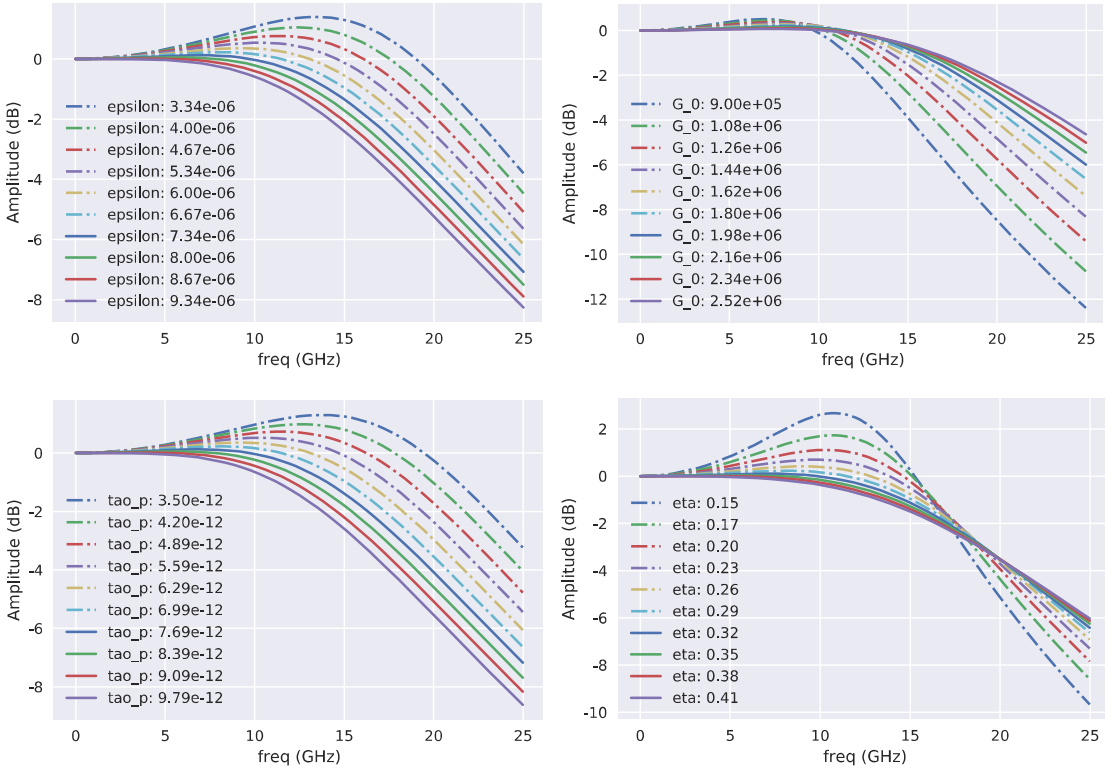

图 16 改变参数时带宽大小变化

图 16 横坐标表示频率，纵坐标表示不同频率对应的幅度，四个子图分别改变 4 个参数时，带宽响应曲线的变化情况，其中左上角到右下角依次是改变了ε、G_0、τ_p、η_i之后带宽响应曲线的变化情况。

所以，通过改变这些固定的参数能够使器件获得更宽的带宽。调整参数的规律如下表 8 所示。

表 8 参数变化时对带宽影响

参数	参数值变化情况	带宽效果
ε	越小	带宽越宽
G_0	越大	带宽越宽
τ_P	越小	带宽越宽
η_i	越大	带宽越宽

表 8 给出了各个参数的变化情况对器件带宽的影响，并且可以得到想要更宽带宽时，参数需要作出的调整。

5.5 变参数下带宽响应曲线幅度变化的研究

通过改变所有参数的取值，能够观察到哪些参数的变化会引起 3dB 范围内的部分频率处幅度高于 0 频位置的情况。得到这些变化规律之后就能够在实际应用中通过调整参数的办法来实现带宽曲线的平坦与否。参数变化下带宽曲线的变化情况如下图 17 所示。

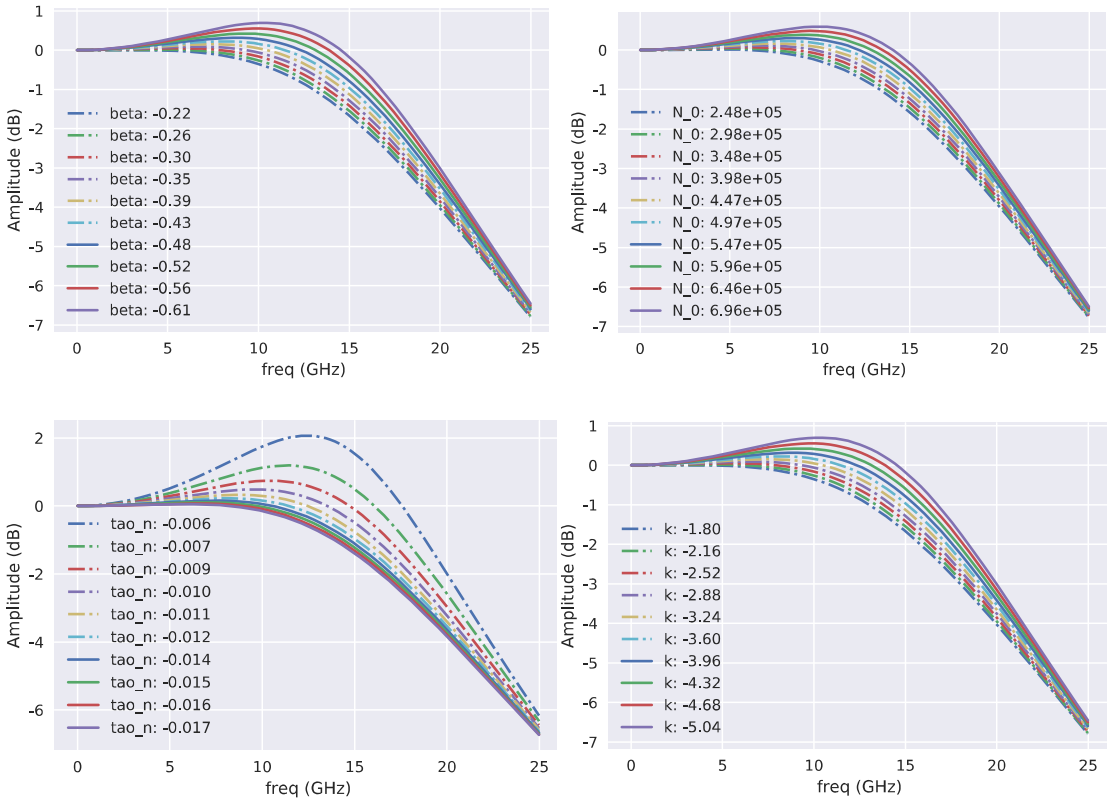

图 17 改变参数时曲线平坦度变化

图 17 横坐标表示频率，纵坐标表示不同频率对应的幅度，四个子图分别表示了改变 4 个参数时，带宽响应曲线的平坦度变化情况，其中左上角到右下角依次是改变了 β、N_0、τ_n、k 之后带宽响应曲线的变化情况。

所以，通过改变这些固定的参数能够使器件的带宽响应曲线的平坦度发生变化。调整参数的规律如下表 9 所示。

表 9 参数变化时对带宽响应曲线平坦度影响

参数	参数值变化情况	曲线效果
β	越大	3dB 内曲线更平坦
N_0	越小	3dB 内曲线更平坦
τ_n	越小	3dB 内曲线更平坦
k	越大	3dB 内曲线更平坦

特别需要说明的是，上面选取的四个参数在变化时均能够达到带宽大小基本相同，而曲线的平坦度不同。通过调整这几个参数能够达到在带宽大小基本不变的前提下，实现带宽响应曲线 3dB 范围内曲线的平坦。

而改变其他四个参数也能够改变 3dB 范围内曲线的平坦度，但是相应的带宽大小就改变了。有两个参数比较特殊，即参数变化方向上带宽宽度和平坦度都向着最优情况发展，如下表 10 所示。

表 10 参数变化时对平坦度和带宽影响

参数	参数值变化情况	曲线效果	带宽效果
G_0	越大	3dB 内曲线更平坦	带宽越宽
η_i	越大	3dB 内曲线更平坦	带宽越宽

上面两个参数才是最优的变参数对象，改变参数能够同时加大带宽还有 3dB 范围内曲线的平坦度。

六、问题 4 模型的建立与求解

模型过于复杂，不仅需要更大的计算量来支持模型，也更容易造成过拟合。为了使得模型运行速度更快，根据问题重述与分析中对问题 4 的分析，下面探讨更好的带宽模型建模方式。

6.1 改进带宽模型的建立

根据对模型的分析，发现 *Y, Z* 的计算是原模型中最为复杂的部分。通过适当简化问题 3 中的带宽模型可以实现提升模型运算速度的目的。经过实验，我们提出了快速小信号等效带宽模型(Faster Small Signal Equivalent Circuit Model, Faster-SSECM)[13]，下面分别讨论 4 种建模方式：

- Faster SSECM-tao(**改进模型** 1)

  通过改变组成信号响应 $h(f)$ 的 Y、Z 多项式，可以减少模型运算时间，进而就能够提高模型的运算速度。

  模型 1 是将 Y、Z 简化成如下形式：

$$\begin{aligned} Y &= \frac{G_0 P_s}{k+\varepsilon P_s} - \frac{G_0\left(N_s - N_0\right)}{\left(1+\varepsilon P_s / k\right)^2} \\ Z &= \frac{G_0 P_s}{\tau_p\left(k+\varepsilon P_s\right)} - \frac{(1-\beta) G_0\left(N_s - N_0\right)}{\tau_n\left(1+\varepsilon P_s / k\right)^2} \end{aligned} \tag{41}$$

  由于模型简化了是原模型中参数 τ 在 Y,Z 中的影响，在下文中，本文标记此种模型为 Faster-SSECM-tao。

- Faster SSECM-taoY(**改进模型** 2)

  模型 2 是将 Y、Z 简化成如下形式：

$$\begin{aligned} Y &= \frac{G_0 P_s}{k+\varepsilon P_s} \\ Z &= \frac{G_0 P_s}{\tau_p\left(k+\varepsilon P_s\right)} - \frac{(1-\beta) G_0\left(N_s - N_0\right)}{\tau_n\left(1+\varepsilon P_s / k\right)^2} \end{aligned} \tag{42}$$

  由于模型简化了是原模型中参数 τ 在 Y,Z 中的影响，并简化了 Y 模型中的低次项，在下文中，本文标记此种模型为 Faster-SSECM-taoY。

- Faster SSECM-taoZ(**改进模型** 3)

  模型 3 是将 Y、Z 简化成如下形式：

$$\begin{aligned} Y &= \frac{G_0 P_s}{k+\varepsilon P_s} - \frac{G_0\left(N_s - N_0\right)}{\left(1+\varepsilon P_s / k\right)^2} \\ Z &= \frac{G_0 P_s}{\tau_p\left(k+\varepsilon P_s\right)} \end{aligned} \tag{43}$$

  由于模型简化了是原模型中参数 τ 在 Y, Z 中的影响，并简化了 Z 模型中的低次项，在下文中，本文标记此种模型为 Faster-SSECM-taoZ。

- Faster SSECM-taoYZ(改进模型 4)

  模型 4 是将 Y、Z 简化成如下形式：

$$Y = \frac{G_0 P_s}{k + \varepsilon P_s}$$
$$Z = \frac{G_0 P_s}{\tau_p \left(k + \varepsilon P_s\right)} \tag{44}$$

  由于模型简化了是原模型中参数τ在 Y, Z 中的影响，也简化了 Y,Z 模型中的低次项，在下文中，本文标记此种模型为 Faster-SSECM-taoYZ。

通过建立以上 4 种模型，来测试模型的运算速度。上面的模型中仅仅提到了更改部分的模型公式，模型的其余部分和问题 3 中建立的模型一致。下面分别求解 4 种模型，并进行相关分析。

6.2 改进带宽模型的求解

6.2.1 改进模型后的运算速度分析

运用求解问题 3 时的最小二乘回归模型以及 Levenberg–Marquardt 算法对带宽模型的参数进行优化求解。下面是四种改进模型求解后得到的带宽模型参数表，如表 11 所示。

表 11 4 种改进带宽模型参数估计值

参数	改进模型 1	改进模型 2	改进模型 3	改进模型 4
η_i	0.2864060892	0.3360406465	0.2687379502	0.681930482
β	1.0000000E-5	-0.106172260	1.0000000E-5	1.0000000E-5
τ_n	-0.0013546011	-0.014305384	-0.003417724	-8.17801577E-8
k	0.0008556465	-15.706720481	0.008153232	-5.042866363E-8
G_0	1.8E6	1.8E6	1.8E6	1.8E6
N_0	4.97E5	4.97E5	4.97E5	4.97E5
τ_P	3.20770653E-12	1.159472451E-11	3.29953712E-12	1.055893691E-11
ε	4.408411687E-6	3.288013033E-6	5.650105289E-6	1.649581461E-7

经过 Levenberg–Marquardt 算法的迭代计算，不同改进模型下的模型参数估计值之间相差很大，4 种改进模型的效果需要详细的分析，下面给出了 4 种模型的运算速度分析表，如表 12 所示。

表 12 4 种改进带宽模型运算速度值

模型名称	误差（MSE）	精度保留值	运行时间	加速比率
原模型	0.0751	-	0.0025551	-
改进模型 1	0.3099	24.23%	0.0014949	41.49%
改进模型 2	**0.0751**	**100.00%**	0.001261	50.65%
改进模型 3	0.2155	34.85%	0.001143	55.27%
改进模型 4	0.9558	7.86%	**0．000746**	**70.80%**

经过改进后的 4 种模型中，运行时间最短、同时也是加速比率最大的是改进模型 4，但其缺点是精度较差，只有原模型精度的 7.86%。完全达到原模型精度的改进模型 2 误差值只有 0.0751，运行时间快，其加速比率为 50.65%。这两个模型可以

作为改进带宽模型的参考。

6.2.2 改进模型后的带宽分析

对模型进行改进后，带宽的变化情况是关注的重要指标。下面分析了改进之后的带宽响应曲线。

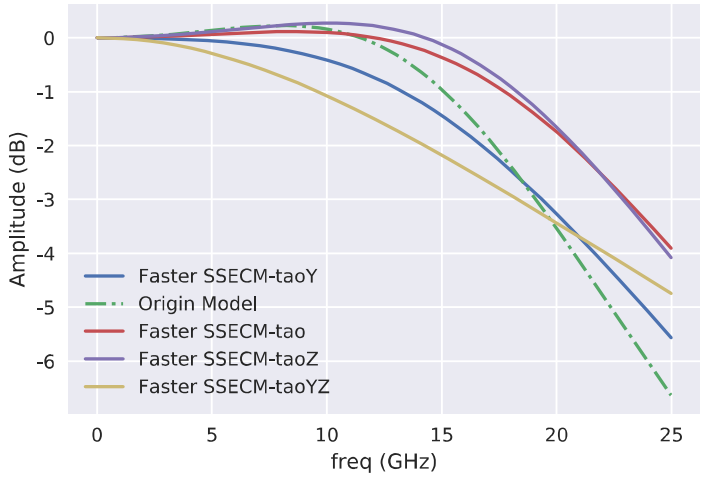

图 18 改进模型后的带宽响应曲线

图 18 横坐标表示频率，纵坐标表示不同频率对应的幅度。从图中可以看出，模型经过改进后，改进模型的带宽相比原模型都有了较大的变化，下面以表格的方式重点分析了改进模型的带宽变化情况（关注 3dB 带宽和 10dB 带宽），具体情况如表 13 所示。

表 13 4 种改进带宽模型带宽变化情况

模型	3dB 带宽（与原模型比较）	10dB 带宽（与原模型比较）
改进模型 1	变大	变大
改进模型 2	变大	变大
改进模型 3	变大	变大
改进模型 4	变小	变大

经过改进后的 4 种模型，仅改进模型 4 的带宽与原模型相比减小了。其余几种改进模型的带宽都要比原模型的大。

6.3 最优改进模型的选择与带宽响应曲线绘制

综合以上两小节对改进模型的运算速度和带宽的评估，下面给出两项指标都较为优秀的改进模型，希望能够为工程实践提供一定的参考。

在运算速度方面，改进模型 2 具有精度高、运算速度快的特点，而在带宽方面，无论是 3dB 带宽还是 10dB 带宽都要比原模型大，综上考虑，改进模型 2 的性能最佳。

利用改进模型 2 所构造的模型重新绘制不同温度以及不同偏置电流下的带宽响应曲线，下图为不同温度下以及不同偏置电流下的带宽响应曲线，为了能够清晰的展示曲线效果，每组的第一幅图展示了 8 种不同温度或偏置电流的带宽响应曲线，第二幅图展示了包含典型值的的四条曲线，如图 19、图 20、图 21、图 22 所示。

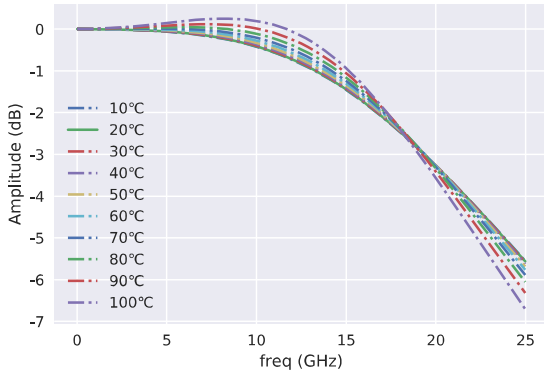

图 19 8 种温度下的带宽响应曲线

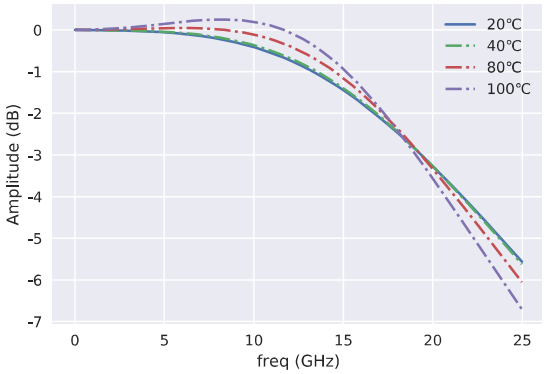

图 20 4 种温度下的带宽响应曲线

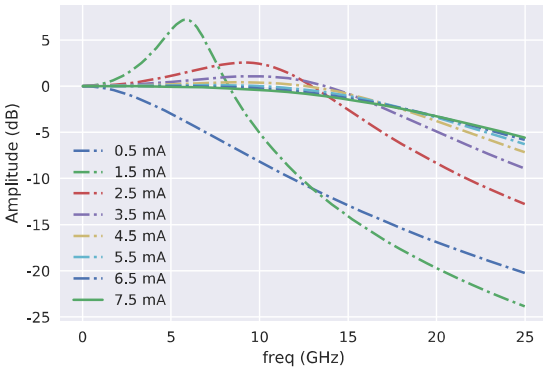

图 21 8 种偏置电流下的带宽响应曲线

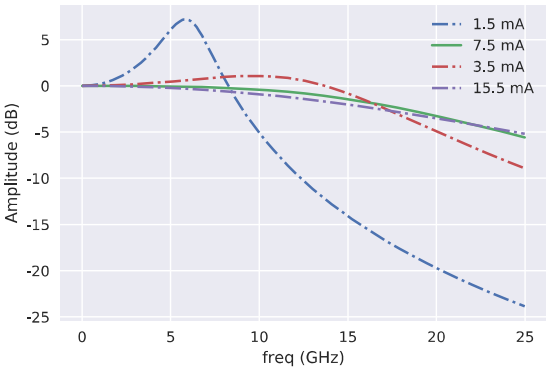

图 22 4 种偏置电流下的带宽响应曲线

以上四幅图中横坐标表示频率，纵坐标表示不同频率对应的幅度。以上四幅图为最优改进模型下绘制的不同温度、不同偏置电流下的带宽响应曲线，从图中可以明显看到温度以及偏置电流变化对器件带宽以及曲线平坦度的影响，这两个变化因素对于器件的影响基本上和原模型一样。

下面绘制了最优改进模型与原模型的带宽响应比较曲线，同样是分两种情况，一种情况是同一偏置电流下（7.5mA），不同温度时的带宽响应曲线比较情况，另一种情况是同一温度下（20 摄氏度），不同偏置电流的带宽响应曲线的比较情况。如图 23、图 24 所示。

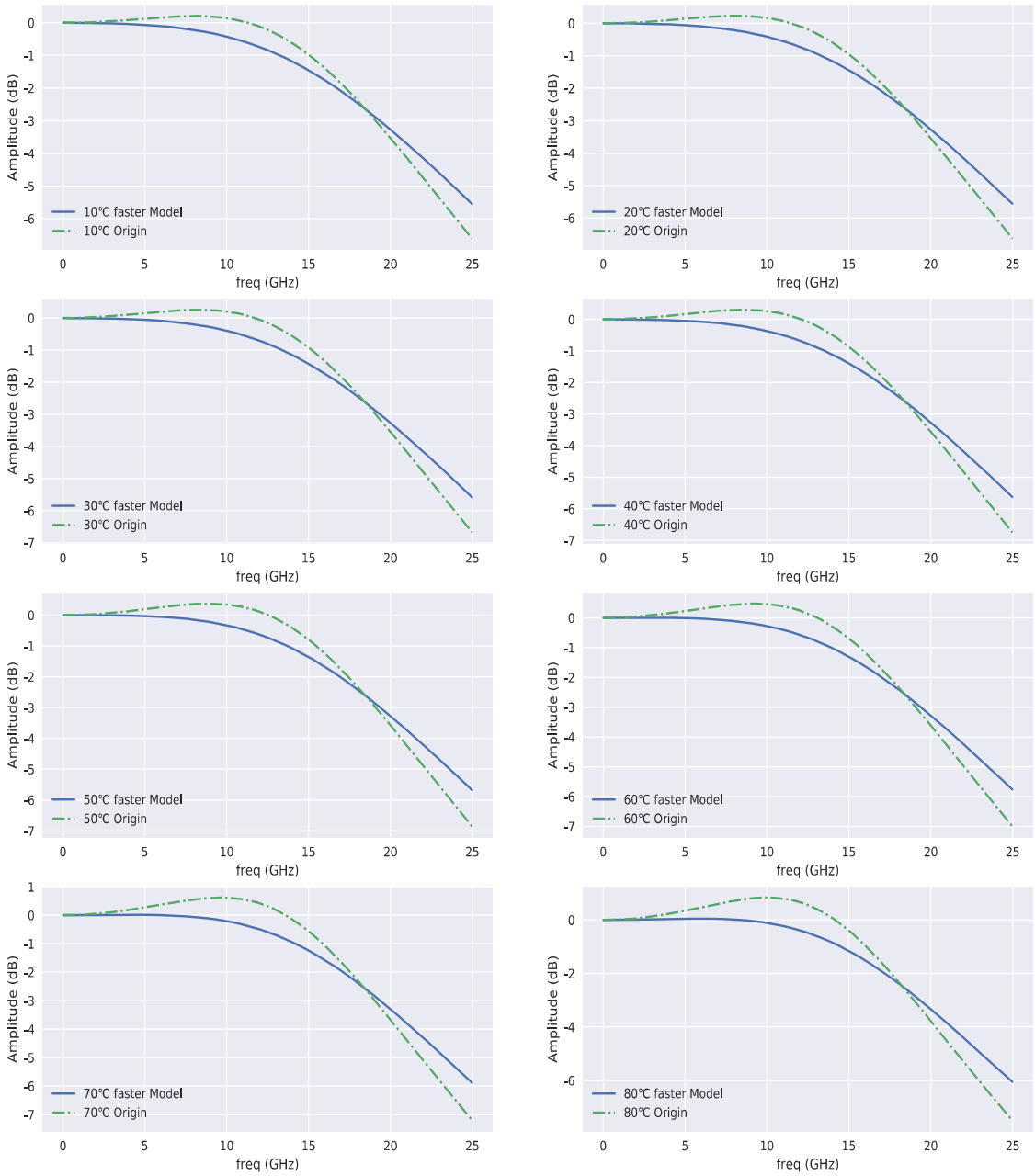

图 23 模型改进前后不同温度的带宽响应比较

图 23 横坐标表示频率，纵坐标表示不同频率对应的幅度。由图可以明显的看出改进后的模型的 10dB 带宽增加了，3dB 带宽差别较小。由于激光器通常采用 10dB 带宽，所以改进后的模型有应用价值。

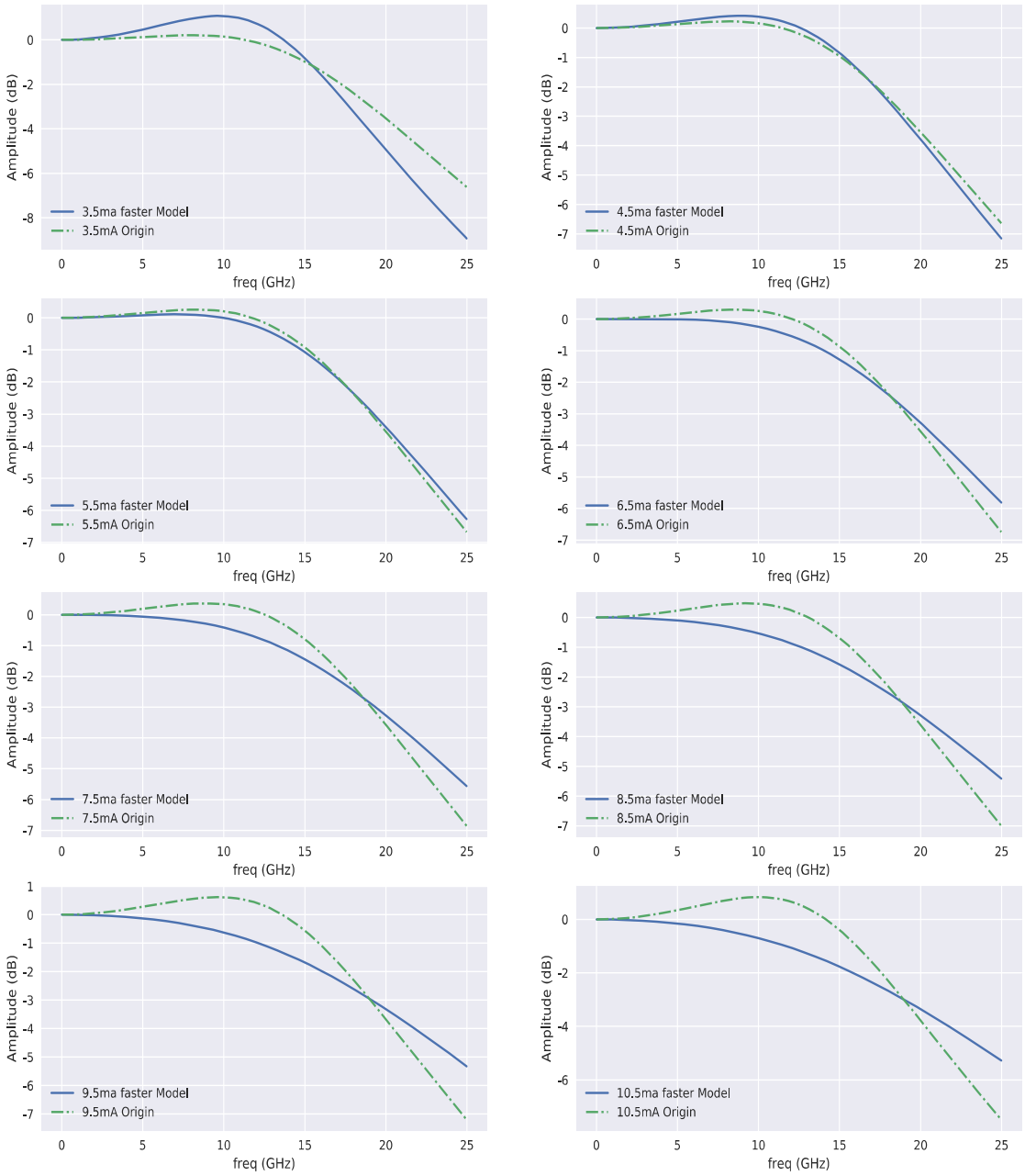

图 24 模型改进前后不同偏置电流下的带宽响应比较

图 24 横坐标表示频率，纵坐标表示不同频率对应的幅度。由图可知，在偏置电流低于 4.5mA 的情况下，改进后模型的带宽较小于原模型，而在偏置电流大于 4.4mA 的情况下，改进后模型的优点越来越明显，其带宽较大。

六、模型评价

6.1 模型的优点

根据模型建立与求解过程中对模型以及结果的分析，现总结模型的优点如下。

- **模型建立过程中设定了调整参数，便于对结果进行调整：**在问题 2 对 L-I 模型进行改进时，改进模型中加入了便于调整迭代结果的调整参数。这使得通过调整这些参数，可以达到使优化结果更加逼近真实值的效果，从而能够让模型得到的 L-I 模型参数更加准确，为绘制更准确的 L-I 曲线奠定数据基础。
- **模型引入先进算法（Adam 算法）以改善优化效果：**在问题 2 模型求解部分，引入 Adam 算法对模型进行了迭代优化。Adam 算法的一个重要优点就是它能够对初始化偏差进行修正，正在很大程度上避免了误差随迭代次数累积的情况。
- **模型充分考虑了带宽模型参数的敏感性：**在问题 3 中充分考虑了带宽模型的参数敏感性，通过对参数的细微调整获得了更优的结果，同时这也为带宽模型的改进提供了思路。

6.2 模型的缺点

根据模型建立与求解过程中对模型以及结果的分析，现总结模型的缺点如下。

- **模型对于初始值较为敏感：**在模型求解的过程中发现，模型在不同初始值下的迭代结果差异非常大，这是造成模型误差的主要原因。

参考文献

[1]Nakwaski W, et al, Self-consistent thermal-electrical modeling of proton-implanted top-surface emitting semiconductor lasers, OE/LASE'94. International Society for Optics and Photonics, 365-387, 1994.

[2]Michalzik R, Ebeling K J. Modeling and design of proton-implanted ultralow-threshold vertical-cavity laser diodes. IEEE journal of quantum electronics, 29(6): 1963-1974, 1993.

[3]Wipiejewski T, et al, Size-dependent output power saturation of vertical-cavity surface-emitting laser diodes, IEEE Photonics Technology Letters, 8(1): 10-12, 1996.

[4]Mena P V, et al, A simple rate-equation-based thermal VCSEL model, Journal of lightwave Technology, 17(5): 865-872, 1999.

[5]梁锋等, 垂直腔面发射激光器的温度模型, 半导体学报, 28(7): 1125-1129, 2007.

[6]Mena P V, et al, A comprehensive circuit-level model of vertical-cavity surface-emitting lasers, Journal of Lightwave Technology, 17(12): 2612-2632, 1999.

[7]Suzuki N, et al, High speed 1.1-μm-range InGaAs-based VCSELs, IEICE transactions on electronics, 92(7): 942-950, 2009.

[8]Duchi J, et al, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research, 12(Jul): 2121-2159, 2011.

[9]毛周. 1.55 μm VCSEL 高频响应特性的优化研究[D]. 西安电子科技大学, 2013.

[11]程俊强, 高速光纤通信系统中垂直腔面发射激光器的特性研究, 北京邮电大学, 2007.

[12]韩威, 1.55 μm 高速激光器材料结构设计及外延生长, 河北工业大学, 2006.

[13]毛陆虹等, 垂直腔面发射激光器的小信号电路模型和调制特性 (英文), 半导体学报, 1: 018, 2002.

[14]Grant M, et al, CVX: Matlab software for disciplined convex programming. 2008.

[15]Kelley M, et al, The Jupyter/IPython architecture: a unified view of computationa l research, from interactive exploration to communication and publication, AGU Fall Meeting Abstracts. 2014.

[16]Paszke A, et al, PyTorch, http://pytorch.org/docs/master/ , 20170918.

[17]McKinney W. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.", 2012.

[18]Walt S, et al, The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering, 13(2): 22-30, 2011.

[19]Jones E, et al, Oliphant T, Peterson P. {SciPy} : open source scientific tools for {P ython}, 2014.

[20]Hunter J D, Matplotlib: A 2D graphics environment, Computing In Science & En gineering, 9(3): 90-95, 2007.

[21]Michael Waskom, Seaborn tutorial, http://seaborn.pydata.org/tutorial.html , 20170 918.

[22]Zeiler M Dm ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.

[23]Krizhevsky A, et al, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, 2012: 1097-1105, 2012.
[24]Kingma D, et al, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[25]Levenberg K, A method for the solution of certain non-linear problems in least squares, Quarterly of applied mathematics, 2(2): 164-168, 1944.
[26]Marquardt D W, An algorithm for least-squares estimation of nonlinear parameters, Journal of the society for Industrial and Applied Mathematics, 11(2): 431-441, 1963.

附录

```
#Question  1
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *
sns.set_style()
myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False

L20C = pd.read_excel('L-I-20C.XLSX')
#S21_5 = pd.read_excel('S21_5.xlsx')
L20C.columns = ['I','P','U','Ta']
#x =L20C['I'].values
#y = L20C['P'].values
plt.plot(L20C['I'].values,L20C['P'].values)
I = torch.from_numpy(L20C['I'].values)*0.001
V = torch.from_numpy(L20C['U'].values)
P = torch.from_numpy(L20C['P'].values)*0.001
learning_rate =0.000000000002
len_a = 5
T0 = 20
etanp  = np.random.randn(1)
I_th0np = np.random.randn(1)
R_thnp = np.random.randn(1)
anp = np.random.randn(len_a)
#init
etanp = np.asarray([0.5])
I_th0np = np.asarray([0.3e-3])
R_thnp = np.asarray([2.6e3])
anp = np.asarray([01.246e-3,-2.545e-5,2.908e-7,2.531e-10,1.002e-12])
eta =Variable(torch.from_numpy(etanp), requires_grad=True)
I_th0 = Variable(torch.from_numpy(I_th0np), requires_grad=True)
R_th =Variable(torch.from_numpy(R_thnp), requires_grad=True,)
a = Variable(torch.from_numpy(anp), requires_grad=True)

IVP = Variable(torch.mul(I,V) - P)
```

```
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
diff = eta*(Variable(I)-I_th0 -I_off) -Variable(P)
loss = diff.pow(2).mean()
#loss = diff.abs().mean()
optimizer = torch.optim.Adam([eta,I_th0,R_th,a], lr = learning_rate)
#optimizer = torch.optim.SGD([eta,I_th0,R_th,a], lr = learning_rate)
#optimizer =torch.optim.Adagrad([eta,I_th0,R_th,a], lr = learning_rate)
losslist = []
for i in range(1000):
    losslist.append(loss.data.numpy()[0])
    loss.backward()
    optimizer.step()
    #print (loss.data.numpy()[0])
    IVP = Variable(torch.mul(I,V) - P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
    diff = eta*(Variable(I)-I_th0 -I_off) -Variable(P)
    loss = diff.pow(2).mean()
    #loss = diff.abs().mean()
    if (loss.data.numpy()[0] - losslist[-1])>0:
        print 'early break'
        break
#print(losslist[-1])
img =plt.figure()
plt.plot(losslist,'-s')
plt.xlabel('Iteration')
plt.ylabel('loss')
ax = plt.gca()
ax.yaxis.get_major_formatter().set_powerlimits((0,1))
img.savefig('pic/loss.pdf')
print 'eta:', eta.data.numpy()[0]
print 'I_th0:', I_th0.data.numpy()[0]
print 'R_th:', R_th.data.numpy()[0]
print 'a:', a.data.numpy()
def myfun(T0 =20):
    IVP = Variable(torch.mul(I,V) - P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
    predicrP = eta*(Variable(I)-I_th0 -I_off)
    data = predicrP.data.numpy()
    data[data<0] =0
    return data
#print len(predicrP.data.numpy())
```

```
img = plt.figure()
for i in range(9):
    num =i*10+10
    if i == 1:
        plt.plot(L20C['I'].values,myfun(num)*1000,'-',label =
str(num)+'℃'.decode('utf8'))
    else:
        plt.plot(L20C['I'].values,myfun(num)*1000,'-.',label =
str(num)+'℃'.decode('utf8'))
#myfun(10)
plt.legend()
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
#plt.plot(L20C['I'].values,[2]*1401,'-')
img.savefig('pic/Temperature.pdf')
#print len(predicrP.data.numpy())
img = plt.figure()
for i in range(9):
    num =i+35
    plt.plot(L20C['I'].values,myfun(num)*1000,'-.',label = str(num)+'℃
'.decode('utf8'))
#myfun(10)
plt.legend(loc = 2)
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.plot(L20C['I'].values,[2]*1401,'-')
img.savefig('pic/Temperature2mw.pdf')
img = plt.figure()
plt.plot(L20C['I'].values[::40],L20C['P'].values[::40],'o',label =
'Real')
plt.plot(L20C['I'].values[::40],(myfun(20)*1000)[::40],'-s',label =
'Predict')

plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.legend(loc = 2)
img.savefig('pic/RealvsPredict.pdf')
pd.DataFrame(P).to_csv('p.csv',header =None,index=None)
pd.DataFrame(myfun(20)*1000).to_csv('Q1_20.csv',header=None,index=Non
e)
df = []
```

```
for i in range(9):
    num =i*10+10
    df.append(myfun(num)*1000)
ddf =
pd.DataFrame(np.asarray(df).T).to_csv('Q1_all.csv',header=None,index=
None)

#Qurstion  2
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
L20C = pd.read_excel('L-I-20C.XLSX')
#S21_5 = pd.read_excel('S21_5.xlsx')
L20C.columns = ['I','P','U','Ta']
I = torch.from_numpy(L20C['I'].values)*0.001
V = torch.from_numpy(L20C['U'].values)
P = torch.from_numpy(L20C['P'].values)*0.001

learning_rate =0.00000002
len_a = 5
T0 = 20
lambd =0.0000000000
alpha = 0.00000000000
#d = 0.00001
#L=300
etanp  = np.random.randn(1)
I_th0np = np.random.randn(1)
R_thnp = np.random.randn(1)
anp = np.random.randn(len_a)
#d#= -0.0003
#init
etanp = np.asarray([0.5])
I_th0np = np.asarray([0.3e-3])
R_thnp = np.asarray([2.6e3])
```

```
anp = np.asarray([01.246e-3,-2.545e-5,2.908e-7,2.531e-10,1.002e-12])#
#anp = np.asarray([01.246e-3,-2.545e-5,2.908e-7])#,#,0.0])#
dnp = np.asarray([0.0000])
eta =Variable(torch.from_numpy(etanp), requires_grad=True)
I_th0 = Variable(torch.from_numpy(I_th0np), requires_grad=True)
R_th =Variable(torch.from_numpy(R_thnp), requires_grad=True)
d =Variable(torch.from_numpy(dnp), requires_grad=True)

a = Variable(torch.from_numpy(anp), requires_grad=True)
#sp = torch.nn.Softplus()
sp = torch.nn.ReLU()
IVP = torch.mul(Variable(I)+d,Variable(V)) - Variable(P)
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)#+a[3]*T.pow(3)+a[4]*T.pow(4)
diff = sp(eta*(Variable(I)+d-I_th0 -I_off)) -Variable(P)
#diff = torch.mul(diff,Variable(P)/Variable(P).max())
baseloss = diff.pow(2).mean()
L1regularizer  =  eta.abs().mean() +
I_th0.abs().mean()+R_th.abs().mean() +a.abs().mean()
L2regularizer = eta.pow(2).mean() +
I_th0.pow(2).mean()+R_th.pow(2).mean() +a.pow(2).mean()
loss = baseloss+lambd*L1regularizer + alpha*L2regularizer
#loss = diff.abs().mean()
optimizer = torch.optim.Adam([eta,I_th0,R_th,a,d], lr = learning_rate)
#optimizer = torch.optim.SGD([eta,I_th0,R_th,a], lr = learning_rate)
#optimizer =torch.optim.Adagrad([eta,I_th0,R_th,a], lr = learning_rate)
losslist = []
for i in range(5000):
    losslist.append(loss.data.numpy()[0])
    loss.backward()
    optimizer.step()
    #print (loss.data.numpy()[0])
    IVP = torch.mul(Variable(I)+d,Variable(V)) - Variable(P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)#+a[3]*T.pow(3)#+a[4]*T.pow(4)
    diff = sp(eta*(Variable(I)+d-I_th0 -I_off)) -Variable(P)
    #diff = torch.mul(diff,Variable(P)/Variable(P).max())
    baseloss = diff.pow(2).mean()
    L1regularizer  =  eta.abs().mean() +
I_th0.abs().mean()+R_th.abs().mean() +a.abs().mean()
    L2regularizer = eta.pow(2).mean() +
I_th0.pow(2).mean()+R_th.pow(2).mean() +a.pow(2).mean()
    loss = baseloss+lambd*L1regularizer + alpha*L2regularizer
    #loss = diff.abs().mean()
```

```
    if (loss.data.numpy()[0] - losslist[-1])>0:
        print 'early break'
        break
#print(losslist[-1])
print 'loss_min :'min(losslist)
img =plt.figure()
plt.plot(losslist,'-')
plt.xlabel('Iteration')
plt.ylabel('loss')
ax = plt.gca()
ax.yaxis.get_major_formatter().set_powerlimits((0,1))

def myfun(T0 =20):
    IVP = torch.mul(Variable(I),Variable(V)) - Variable(P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
    regularizer  =  a.abs().sum()
    predicrP = sp(eta*(Variable(I)-I_th0 -I_off))# +lambd*regularizer
    data = predicrP.data.numpy()
    #data[data<0] =0
    return data
plt.plot(L20C['I'].values,L20C['P'].values,'-.',label = 'Real')
plt.plot(L20C['I'].values,myfun(20)*1000,label = 'Predict')
```

#Question2

```
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
L20C = pd.read_excel('L-I-20C.XLSX')
#S21_5 = pd.read_excel('S21_5.xlsx')
L20C.columns = ['I','P','U','Ta']
I = torch.from_numpy(L20C['I'].values)*0.001
```

```
V = torch.from_numpy(L20C['U'].values)
#P = torch.from_numpy(L20C['P'].values)*0.001
setp = 25
img = plt.figure(figsize=(16,10))
plt.subplot(321)
sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=1,truncate=True,label = 'Power: 1',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)

#fig = plt.figure()
plt.subplot(322)
sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=2,truncate=True,label = 'Power: 2',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)

#fig = plt.figure()
plt.subplot(323)
sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=3,truncate=True,label = 'Power: 3',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)

#fig = plt.figure()
plt.subplot(324)
sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=4,truncate=True,label = 'Power: 4',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)

plt.subplot(325)

sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=5,truncate=True,label = 'Power: 5',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)
plt.subplot(326)
```

```
sns.regplot(L20C['I'].values[::setp],L20C['U'].values[::setp],ci=100,
order=6,truncate=True,label = 'Power: 6',color='g')
plt.xlabel('Input Current (A)')
plt.ylabel('Voltage (V)')
plt.legend(loc =2)
img.savefig('pic/I_V.pdf')

#Q2 plot
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
L20C = pd.read_excel('L-I-20C.XLSX')
#S21_5 = pd.read_excel('S21_5.xlsx')
I = torch.from_numpy(L20C['I'].values)*0.001
V = torch.from_numpy(L20C['U'].values)
P = torch.from_numpy(L20C['P'].values)*0.001
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
learning_rate =0.000000002
len_a = 5
T0 = 20
lambd =0.00000000000001
alpha = 0.00000000000000001
```

```
ew =0.99
lambdw =0.000001
#d = 0.00001
#L=300
etanp  = np.random.randn(1)
I_th0np = np.random.randn(1)
R_thnp = np.random.randn(1)
anp = np.random.randn(len_a)
#d#= -0.0003
#init
etanp = np.asarray([0.5])
I_th0np = np.asarray([0.3e-3])
R_thnp = np.asarray([2.6e3])
anp = np.asarray([01.246e-3,-2.545e-5,2.908e-7,2.531e-10,1.002e-12])#
#anp = np.asarray([01.246e-3,-2.545e-5,2.908e-7])#,#,0.0])#
dnp = np.asarray([0.0000])
eta =Variable(torch.from_numpy(etanp), requires_grad=True)
I_th0 = Variable(torch.from_numpy(I_th0np), requires_grad=True)
R_th =Variable(torch.from_numpy(R_thnp), requires_grad=True)
d =Variable(torch.from_numpy(dnp), requires_grad=True)

a = Variable(torch.from_numpy(anp), requires_grad=True)
#sp = torch.nn.Softplus()
sp = torch.nn.ReLU()
IVP = torch.mul(Variable(I)+d,Variable(V)) - Variable(P)
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)#+a[3]*T.pow(3)#+a[4]*T.pow(4)
diff = sp(eta*(Variable(I)+d-I_th0 -I_off)) -Variable(P)
#diff = ew*torch.mul(diff,Variable(P)/Variable(P).max()) + lambdw
baseloss = diff.pow(2).mean()
L1regularizer  =  eta.abs().mean() +
I_th0.abs().mean()+R_th.abs().mean() +a.abs().mean()
L2regularizer = eta.pow(2).mean() +
I_th0.pow(2).mean()+R_th.pow(2).mean() +a.pow(2).mean()
loss = baseloss+lambd*L1regularizer + alpha*L2regularizer
#loss = diff.abs().mean()
optimizer = torch.optim.Adam([eta,I_th0,R_th,a,d], lr = learning_rate)
#optimizer = torch.optim.SGD([eta,I_th0,R_th,a], lr = learning_rate)
#optimizer =torch.optim.Adagrad([eta,I_th0,R_th,a], lr = learning_rate)
losslist = []
for i in range(5000):
    losslist.append(loss.data.numpy()[0])
    loss.backward()
    optimizer.step()
```

```
    #print (loss.data.numpy()[0])
    IVP = torch.mul(Variable(I)+d,Variable(V)) - Variable(P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)#+a[3]*T.pow(3)#+a[4]*T.pow(4)
    diff = sp(eta*(Variable(I)+d-I_th0 -I_off)) -Variable(P)
    #diff = ew*torch.mul(diff,Variable(P)/Variable(P).max()) + lambdw
    baseloss = diff.pow(2).mean()
    L1regularizer  =  eta.abs().mean() +
I_th0.abs().mean()+R_th.abs().mean() +a.abs().mean()
    L2regularizer = eta.pow(2).mean() +
I_th0.pow(2).mean()+R_th.pow(2).mean() +a.pow(2).mean()
    loss = baseloss+lambd*L1regularizer + alpha*L2regularizer
    #loss = diff.abs().mean()
    if (loss.data.numpy()[0] - losslist[-1])>0:
        print 'early break'
        break
#print(losslist[-1])
print 'loss_min :',min(losslist)
print 'rmse:',(sp(eta*(Variable(I)+d-I_th0 -I_off))
-Variable(P)).pow(2).mean().data.numpy()
img = plt.figure()
plt.plot(losslist,'-')
plt.xlabel('Iteration')
plt.ylabel('loss')
ax = plt.gca()
ax.yaxis.get_major_formatter().set_powerlimits((0,1))
img.savefig('pic/lossQ2.pdf')
print 'eta:',eta.data.numpy()
print 'I_th0:',I_th0.data.numpy()
print 'R_th:',R_th.data.numpy()
print 'a:',a.data.numpy()
img = plt.figure()

plt.plot(L20C['I'].values[::40],L20C['P'].values[::40],'o',label =
'Real')
plt.plot(L20C['I'].values[::40],(myfun(20)*1000)[::40],'-s',label =
'Proposed')
plt.plot(L20C['I'].values[::40],pd.read_csv('Q1_20.csv',header=None).
values[::40],'-.',label = 'Origin')


plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.legend(loc = 2)
```

```
img.savefig('pic/RealvsPredictQ2.pdf')
def myfun(T0 =20):
    IVP = torch.mul(Variable(I),Variable(V)) - Variable(P)
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)#+a[3]*T.pow(3)+a[4]*T.pow(4)
    regularizer  =  a.abs().sum()
    predicrP = sp(eta*(Variable(I)-I_th0 -I_off))# +lambd*regularizer
    data = predicrP.data.numpy()
    #data[data<0] =0
    return data
#print len(predicrP.data.numpy())
img = plt.figure()
for i in range(9):
    num =i*10+10
    if i == 1:
        plt.plot(L20C['I'].values,myfun(num)*1000,'-',label =

str(num)+'℃'.decode('utf8'))

    else:
        plt.plot(L20C['I'].values,myfun(num)*1000,'-.',label =

str(num)+'℃'.decode('utf8'))

#myfun(10)
plt.legend()
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
#plt.plot(L20C['I'].values,[2]*1401,'-')
img.savefig('pic/TemperatureQ2.pdf')
#print len(predicrP.data.numpy())
img = plt.figure()
for i in range(9):
    num =i+25

    plt.plot(L20C['I'].values,myfun(num)*1000,'-.',label = str(num)+'℃

'.decode('utf8'))
#myfun(10)
plt.legend(loc = 2)
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.plot(L20C['I'].values,[2]*1401,'-')
img.savefig('pic/Temperature2mwQ2.pdf')
fig = plt.figure()

l=200
```

```
plt.plot(L20C['I'].values[:l:8],L20C['P'].values[:l:8],'o',label =
'Real Data')
plt.plot(L20C['I'].values[:l],predicrP.data.numpy()[:l]*1000,label =
'Predict Data')
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.legend()
fig.savefig('pic/zeroQuestion.eps')
#overfitting
#plt.plot(L20C['I'].values[:l:8],L20C['P'].values[:l:8],'o',label =
'Real Data')
fig = plt.figure()
plt.subplot(221)
sns.regplot(L20C['I'].values[200:1400:100],L20C['P'].values[300:1400:
100],ci=100,order=1,truncate=True,label = 'Power: 1',color='g')
plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.legend(loc =2)
plt.subplot(222)
sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400:
100],ci=100,order=2,truncate=True,label = 'Power: 2',color='g')
plt.legend(loc =2)

plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.subplot(223)
sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400:
100],ci=100,order=3,truncate=True,label = 'Power: 3',color='g')
plt.legend(loc =2)

plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
plt.subplot(224)
sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400:
100],ci=100,order=4,truncate=True,label = 'Power: 4',color='g')
plt.legend(loc =2)

plt.xlabel('I (mA)')
plt.ylabel('P (mW)')
fig.savefig('pic/overfitting.pdf')

fig = plt.figure(figsize=(9,9))
df = pd.read_csv('Q1_all.csv',header=None)
for i in  range(6):
```

```
    plt.subplot(321+i)

    num = i*10+10

    plt.plot(L20C['I'].values,myfun(num)*1000,'-',label = str(num)+'℃
Proposed'.decode('utf8'))

    plt.plot(L20C['I'].values,df[i].values,'-.',label = str(num)+'℃
Origin'.decode('utf8'))

#sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400
:100],ci=100,order=1,truncate=True,label = 'Power: 1',color='g')
    plt.xlabel('I (mA)')
    plt.ylabel('P (mW)')
    plt.legend(loc =2)

fig.savefig('pic/temVs.pdf')
pd.DataFrame(myfun(20)*1000).to_csv('Q2_20.csv',header=None,index
=None)


#Question 3
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False

#L20C = pd.read_excel('L-I-20C.XLSX')
S21_5 = pd.read_excel('S21_5.xlsx')
S21_5.columns = ['f','S21','I_b','Ta']
#parameter init
I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))
```

```
V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
eta_inp = np.asarray([0.290087811944204])
betanp = np.asarray([-.433751523591667])
tao_nnp = np.asarray([-.0122997270697670])
knp = np.asarray([-3.60187228705102])
G_0np = np.asarray([1.8e6])

N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([6.99206216358124e-12])
epsilonnp = np.asarray([6.67052986037427e-06])



#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
```

```
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))

#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)


N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
P_s = k*S_s
Y =1./tao_p +1./tao_n +G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 +
epsilon*P_s/k)/(1 + epsilon*P_s/k)
Z = 1./tao_p/tao_n +G_0*P_s/tao_p/(k + epsilon*P_s) -
k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1 + epsilon*P_s/k)/(1 + epsilon*P_s/k)
#H_pred1 = Z/(-(2*np.pi*f)*(2*np.pi*f)-(2*np.pi*f)*Y+Z)
YY = Y.data.numpy()
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
y = 20*np.log(np.abs(H_pred))/np.log(10)
pd.DataFrame(y).to_csv('Q4all.csv',header=None,index=None)
plt.plot(S21_5['f'].values,S21_5['S21'].values,'-.',label = 'Real')
plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')

#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.xlabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')

plt.legend()

def ComputeAmplitude(T0 =20,
                     I = 7.5,
                     eta_inp = np.asarray([0.290087811944204]),
```

```
                    betanp = np.asarray([-.433751523591667]),
                    tao_nnp = np.asarray([-.0122997270697670]),
                    knp = np.asarray([-3.60187228705102]),
                    G_0np = np.asarray([1.8e6]),
                    N_0np = np.asarray([4.97e5]),
                    tao_pnp = np.asarray([6.99206216358124e-12]),
                    epsilonnp = np.asarray([6.67052986037427e-06])
                   ):
    #parameter init
    I = Variable(torch.from_numpy(np.asarray([I])))*1e-3
    f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
    H =  Variable(torch.from_numpy(S21_5['S21'].values))

    V = Variable(torch.from_numpy(np.asarray([2.464597008])))
    q = 1.6e-19
    #0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06
    '''
    eta_inp = np.random.randn(1)*.7
    betanp = np.random.randn(1)*1e-5
    tao_nnp = np.random.randn(1)*9.6e-9
    knp = np.random.randn(1)*1.5e-8
    G_0np = np.random.randn(1)*1.8e6
    N_0np = np.random.randn(1)*4.97e5
    tao_pnp = np.random.randn(1)*3.8e-12
    epsilonnp = np.random.randn(1)*4.7e-8
    '''




    #
    eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
    beta =Variable(torch.from_numpy(betanp), requires_grad=True)
    tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
    k = Variable(torch.from_numpy(knp), requires_grad=True)
    G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
    N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
    tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
    epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

    #Q1 parameter
    Pnp = np.asarray([1.904172485])*1e-3
```

```
    etanp = np.asarray([0.499999999959])
    I_th0np = np.asarray([0.000300000041534])
    R_thnp = np.asarray([2600.0])
    #anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
    anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

    eta =Variable(torch.from_numpy(etanp))
    I_th0 = Variable(torch.from_numpy(I_th0np))
    R_th =Variable(torch.from_numpy(R_thnp))
    a = Variable(torch.from_numpy(anp))
    P =Variable(torch.from_numpy(Pnp))

    #S21 = torch.from_numpy(S21_5['S21'].values)
    IVP = torch.mul(I,V) - P
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)


    N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k
+ epsilon*P))
    S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
    P_s = k*S_s
    Y =1./tao_p +1./tao_n +G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)
    Z = 1./tao_p/tao_n +G_0*P_s/tao_p/(k + epsilon*P_s) -
k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1 + epsilon*P_s/k)/(1 + epsilon*P_s/k)
    #H_pred1 = Z/(-(2*np.pi*f)*(2*np.pi*f)-(2*np.pi*f)*Y+Z)
    YY = Y.data.numpy()
    ZZ = Z.data.numpy()
    ff =f.data.numpy()
    j=np.asarray([1j])
    H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
    y = 20*np.log(np.abs(H_pred))/np.log(10)
    return y
    img = plt.figure()
tem = [20,40,80,100]
for num in tem:
    #num =  i*20+10
    if num==20:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 = num),'-',label
=str(num)+'℃'.decode('utf8'))
```

```
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 =
num),'-.',label =str(num)+'℃'.decode('utf8'))
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/temdB4.pdf')
img = plt.figure()
#tem = [20,40,80,100]
for i in range(10):
    num =  i*10+10
    if num==20:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 = num),'-',label
=str(num)+'℃'.decode('utf8'))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 =
num),'-.',label =str(num)+'℃'.decode('utf8'))
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend()
img.savefig('pic/temdBall.pdf')
img = plt.figure()

for i in range(8):
    num =  i+0.5
    if i==7:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-',label
=str(num)+ ' mA')
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-.',label
=str(num)+ ' mA')
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
```

```
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/dianliuAll.pdf')
img = plt.figure()
tem = [1.5,7.5,3.5,15.5]
for num in tem:
    #num =  i*10+10
    if num==7.5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-',label
=str(num)+ ' mA')
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-.',label
=str(num)+' mA')
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend()
img.savefig('pic/dianliu4.pdf')
img = plt.figure()
for i in range(10):
    num = .290087811944204+0.290087811944204*0.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(eta_inp =
np.asarray([num])),'-',label = 'eta: %.2f'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(eta_inp =
np.asarray([num])),'-.',label = 'eta: %.2f'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/eta.pdf')
img = plt.figure()
flag = -0.433751523591667
for i in range(10):
    num = flag+flag*0.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(betanp =
np.asarray([num])),'-',label = 'beta: %.2f'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(betanp =
np.asarray([num])),'-.',label = 'beta: %.2f'%(num))
```

```
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/beta.pdf')
img = plt.figure()
flag = -0.0122997270697670
for i in range(10):
    num = flag+flag*0.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(tao_nnp =
np.asarray([num])),'-',label = 'tao_n: %.3f'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(tao_nnp =
np.asarray([num])),'-.',label = 'tao_n: %.3f'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/tao_n.pdf')
img = plt.figure()
flag = -3.60187228705102
for i in range(10):
    num = flag+flag*0.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(knp =
np.asarray([num])),'-',label = 'k: %.2f'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(knp =
np.asarray([num])),'-.',label = 'k: %.2f'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/knp.pdf')
img = plt.figure()
flag = 1.8e6
for i in range(10):
    num = flag+flag*0.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(G_0np =
np.asarray([num])),'-',label = 'G_0: %1.e'%(num))
    else:
```

```
        plt.plot(S21_5['f'].values,ComputeAmplitude(G_0np =
np.asarray([num])),'-.',label = 'G_0: %1.e'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/G_0.pdf')
img = plt.figure()
flag = 4.97e5
for i in range(10):
    num = flag+flag*.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(N_0np =
np.asarray([num])),'-',label = 'N_0: %1.e'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(N_0np =
np.asarray([num])),'-.',label = 'N_0: %1.e'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/N_0.pdf')
img = plt.figure()
flag = 6.99206216358124e-12
for i in range(10):
    num = flag+flag*.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(tao_pnp =
np.asarray([num])),'-',label = 'tao_p: %.2e'%(num))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(tao_pnp =
np.asarray([num])),'-.',label = 'tao_p: %.2e'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/tao_p.pdf')
img = plt.figure()
flag = 6.67052986037427e-06
for i in range(10):
    num = flag+flag*.1*(i-5)
    if i>5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(epsilonnp =
np.asarray([num])),'-',label = 'epsilon: %.2e'%(num))
```

```
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(epsilonnp =
np.asarray([num])),'-.',label = 'epsilon: %.e'%(num))

plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/epsilon.pdf')
```

#Question 4

```
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
import time as t
#L20C = pd.read_excel('L-I-20C.XLSX')
S21_5 = pd.read_excel('S21_5.xlsx')
S21_5.columns = ['f','S21','I_b','Ta']
now  = t.time()
# quick Model
#parameter init
I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))

V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
```

```
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
eta_inp = np.asarray([0.3360406466471866])
betanp = np.asarray([-.1061722591892961])
tao_nnp = np.asarray([-15.7067204809799])
knp = np.asarray([-3.60187228705102])
G_0np = np.asarray([1.8e6])

N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([1.15947245110257e-11])
epsilonnp = np.asarray([3.28801303308812e-06])



#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))
```

```
#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)


N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
P_s = k*S_s
Y =G_0*P_s/(k + epsilon*P_s)
Z = G_0*P_s/tao_p/(k + epsilon*P_s) - k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)
#H_pred1 = Z/(-(2*np.pi*f)*(2*np.pi*f)-(2*np.pi*f)*Y+Z)
YY = Y.data.numpy()
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
yqucik = 20*np.log(np.abs(H_pred))/np.log(10)
print t.time()-now
# quick Model
#parameter init
now  = t.time()

I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))

V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
```

```
eta_inp = np.asarray([0.286406089244119])
betanp = np.asarray([1.00000000000000e-05])
tao_nnp = np.asarray([-0.00135460108423014])
knp = np.asarray([.000855646516271248])
G_0np = np.asarray([1.8e6])

N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([3.20770652975277e-12])
epsilonnp = np.asarray([4.40841168670671e-06])



#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))

#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
```

```
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
P_s = k*S_s
Y =G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 + epsilon*P_s/k)/(1 +
epsilon*P_s/k)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s) - k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)

YY = Y.data.numpy()
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
y1 = 20*np.log(np.abs(H_pred))/np.log(10)
print t.time()-now
# quick Model
#parameter init
now  = t.time()

I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))

V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
eta_inp = np.asarray([0.286406089244119])
betanp = np.asarray([1.00000000000000e-05])
```

```
tao_nnp = np.asarray([-0.00135460108423014])
knp = np.asarray([.000855646516271248])
G_0np = np.asarray([1.8e6])

N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([3.20770652975277e-12])
epsilonnp = np.asarray([4.40841168670671e-06])



#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))

#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)



IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
```

```
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
P_s = k*S_s
Y = G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 + epsilon*P_s/k)/(1 +
epsilon*P_s/k)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s) - k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)

YY = Y.data.numpy()
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
yor = 20*np.log(np.abs(H_pred))/np.log(10)
print t.time()-now
# quick Model
#parameter init
I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))

V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
eta_inp = np.asarray([0.268737950199567])
betanp = np.asarray([1.00000000000000e-05])
tao_nnp = np.asarray([-0.00341772353672945])
knp = np.asarray([.00815323238705132])
G_0np = np.asarray([1.8e6])
```

```
N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([3.29953711914025e-12])
epsilonnp = np.asarray([5.65010528859981e-06])



#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))

#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)



IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
```

```
P_s = k*S_s
Y =G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 + epsilon*P_s/k)/(1 +
epsilon*P_s/k)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s)

YY = Y.data.numpy()
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
y2 = 20*np.log(np.abs(H_pred))/np.log(10)
print t.time()-now
# quick Model
#parameter init
I = Variable(torch.from_numpy(np.asarray([7.5])))*1e-3
T0 = 20
f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
H =  Variable(torch.from_numpy(S21_5['S21'].values))

V = Variable(torch.from_numpy(np.asarray([2.464597008])))
q = 1.6e-19
#0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

eta_inp = np.random.randn(1)*.7
betanp = np.random.randn(1)*1e-5
tao_nnp = np.random.randn(1)*9.6e-9
knp = np.random.randn(1)*1.5e-8
G_0np = np.random.randn(1)*1.8e6
N_0np = np.random.randn(1)*4.97e5
tao_pnp = np.random.randn(1)*3.8e-12
epsilonnp = np.random.randn(1)*4.7e-8
eta_inp = np.asarray([0.6819304820021858])
betanp = np.asarray([1.00000000000000e-05])
tao_nnp = np.asarray([-8.17801576979332e-08])
knp = np.asarray([-5.04286636253122e-08])
G_0np = np.asarray([1.8e6])

N_0np = np.asarray([4.97e5])
tao_pnp = np.asarray([1.05589369120407e-11])
epsilonnp = np.asarray([1.64958146147851e-07])
```

```
#
eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
beta =Variable(torch.from_numpy(betanp), requires_grad=True)
tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
k = Variable(torch.from_numpy(knp), requires_grad=True)
G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

#Q1 parameter
Pnp = np.asarray([1.904172485])*1e-3
etanp = np.asarray([0.499999999959])
I_th0np = np.asarray([0.000300000041534])
R_thnp = np.asarray([2600.0])
#anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

eta =Variable(torch.from_numpy(etanp))
I_th0 = Variable(torch.from_numpy(I_th0np))
R_th =Variable(torch.from_numpy(R_thnp))
a = Variable(torch.from_numpy(anp))
P =Variable(torch.from_numpy(Pnp))

#S21 = torch.from_numpy(S21_5['S21'].values)
IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)


IVP = torch.mul(I,V) - P
T = T0+IVP*R_th
I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k +
epsilon*P))
S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
P_s = k*S_s
Y =G_0*P_s/(k + epsilon*P_s)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s)

YY = Y.data.numpy()
```

```
ZZ = Z.data.numpy()
ff =f.data.numpy()
j=np.asarray([1j])
H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
y3 = 20*np.log(np.abs(H_pred))/np.log(10)
print t.time()-now
img = plt.figure()
#plt.plot(S21_5['f'].values,S21_5['S21'].values,'-.',label = 'Real')
plt.plot(S21_5['f'].values,y,label = 'Faster Model-taoY')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
plt.plot(S21_5['f'].values,pd.read_csv('Q4all.csv',header=None).value
s,'-.',label = 'Oringin Model')
plt.plot(S21_5['f'].values,y1,label = 'Faster Model-tao')
plt.plot(S21_5['f'].values,y2,label = 'Faster Model-taoZ')
plt.plot(S21_5['f'].values,y3,label = 'Faster Model-taoYZ')

#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')

plt.legend()
img.savefig('pic/faster.pdf')
now = t.time()
Y =1./tao_p +1./tao_n +G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 +
epsilon*P_s/k)/(1 + epsilon*P_s/k)
Z = 1./tao_p/tao_n +G_0*P_s/tao_p/(k + epsilon*P_s) -
k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1 + epsilon*P_s/k)/(1 + epsilon*P_s/k)
print t.time()-now
now = t.time()
Y = G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 + epsilon*P_s/k)/(1 +
epsilon*P_s/k)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s) - k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)
print t.time()-now
now = t.time()
Y = G_0*P_s/(k + epsilon*P_s)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s) - k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1
+ epsilon*P_s/k)/(1 + epsilon*P_s/k)
print t.time()-now
now = t.time()
Y = G_0*P_s/(k + epsilon*P_s) - G_0*(N_s-N_0)/(1 + epsilon*P_s/k)/(1 +
epsilon*P_s/k)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s)
print t.time()-now
```

```
now = t.time()
Y = G_0*P_s/(k + epsilon*P_s)
Z =  G_0*P_s/tao_p/(k + epsilon*P_s)
print t.time()-now

#Q4 plot

import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torch.autograd import Variable
from numpy.random import randint
from sklearn import metrics
%matplotlib inline
from matplotlib.font_manager import *

myfont = FontProperties(fname='SansSerif.ttf')
matplotlib.rcParams['axes.unicode_minus']=False
import time as t
#L20C = pd.read_excel('L-I-20C.XLSX')
S21_5 = pd.read_excel('S21_5.xlsx')
S21_5.columns = ['f','S21','I_b','Ta']
def ComputeAmplitude(T0 =20,
                     I = 7.5):# quick Model
    #parameter init
    I = Variable(torch.from_numpy(np.asarray([I])))*1e-3
    f =  Variable(torch.from_numpy(S21_5['f'].values))*1e9
    H =  Variable(torch.from_numpy(S21_5['S21'].values))

    V = Variable(torch.from_numpy(np.asarray([2.464597008])))
    q = 1.6e-19
    #0.225831488087848  -0.970065761537841  1.69074235247716
-0.222239791688644  1800000 597000  7.46603808781914e-12
4.12031396456752e-06

    eta_inp = np.random.randn(1)*.7
    betanp = np.random.randn(1)*1e-5
    tao_nnp = np.random.randn(1)*9.6e-9
    knp = np.random.randn(1)*1.5e-8
    G_0np = np.random.randn(1)*1.8e6
    N_0np = np.random.randn(1)*4.97e5
    tao_pnp = np.random.randn(1)*3.8e-12
```

```
    epsilonnp = np.random.randn(1)*4.7e-8
    eta_inp = np.asarray([0.3360406464718666])
    betanp = np.asarray([-.1061722591892961])
    tao_nnp = np.asarray([-15.706720480979903])
    knp = np.asarray([-3.601872287051024])
    G_0np = np.asarray([1.8e6])

    N_0np = np.asarray([4.97e5])
    tao_pnp = np.asarray([1.1594724511025704e-11])
    epsilonnp = np.asarray([3.288013033088124e-06])



    #
    eta_i = Variable(torch.from_numpy(eta_inp), requires_grad=True)
    beta =Variable(torch.from_numpy(betanp), requires_grad=True)
    tao_n =Variable(torch.from_numpy(tao_nnp), requires_grad=True,)
    k = Variable(torch.from_numpy(knp), requires_grad=True)
    G_0 = Variable(torch.from_numpy(G_0np), requires_grad=True)
    N_0 = Variable(torch.from_numpy(N_0np), requires_grad=True)
    tao_p = Variable(torch.from_numpy(tao_pnp), requires_grad=True)
    epsilon = Variable(torch.from_numpy(epsilonnp), requires_grad=True)

    #Q1 parameter
    Pnp = np.asarray([1.904172485])*1e-3
    etanp = np.asarray([0.499999999959])
    I_th0np = np.asarray([0.000300000041534])
    R_thnp = np.asarray([2600.0])
    #anp = np.asarray([1.24600004e-03, -2.54499585e-05, 2.90841508e-07,
2.94589863e-10, 4.24732955e-11])
    anp = np.asarray([1.246e-03, -2.545e-05, 2.908e-07, -2.531e-10,
1.022e-12])

    eta =Variable(torch.from_numpy(etanp))
    I_th0 = Variable(torch.from_numpy(I_th0np))
    R_th =Variable(torch.from_numpy(R_thnp))
    a = Variable(torch.from_numpy(anp))
    P =Variable(torch.from_numpy(Pnp))

    #S21 = torch.from_numpy(S21_5['S21'].values)
    IVP = torch.mul(I,V) - P
    T = T0+IVP*R_th
    I_off = a[0]+a[1]*T+a[2]*T.pow(2)+a[3]*T.pow(3)+a[4]*T.pow(4)
```

```
    N_s = (P/(k*tao_p) +G_0*N_0*P/(k + epsilon*P))/(beta/tao_n+G_0*P/(k
+ epsilon*P))
    S_s = (eta_i*(I- I_th0 -I_off)/q -N_s/tao_n)/G_0/(N_s-N_0)
    P_s = k*S_s
    Y =G_0*P_s/(k + epsilon*P_s)
    Z = G_0*P_s/tao_p/(k + epsilon*P_s) -
k*(1-beta)*G_0*(N_s-N_0)/tao_n/(1 + epsilon*P_s/k)/(1 + epsilon*P_s/k)
    #H_pred1 = Z/(-(2*np.pi*f)*(2*np.pi*f)-(2*np.pi*f)*Y+Z)
    YY = Y.data.numpy()
    ZZ = Z.data.numpy()
    ff =f.data.numpy()
    j=np.asarray([1j])
    H_pred =ZZ/((j*2*np.pi*ff)*(j*2*np.pi*ff)+(j*2*np.pi*ff)*YY+ZZ)
    yqucik = 20*np.log(np.abs(H_pred))/np.log(10)
    return yqucik
img = plt.figure()
#tem = [20,40,80,100]
for i in range(10):
    num =  i*10+10
    if num==20:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 = num),'-',label
=str(num)+'℃'.decode('utf8'))
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 =
num),'-.',label =str(num)+'℃'.decode('utf8'))
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend()
img.savefig('pic/fasterTemAll.pdf')
img = plt.figure()
tem = [20,40,80,100]
for num in tem:
    #num =  i*10+10
    if num==20:
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 = num),'-',label
=str(num)+'℃'.decode('utf8'))
    else:
```

```
        plt.plot(S21_5['f'].values,ComputeAmplitude(T0 =
num),'-.',label =str(num)+'℃'.decode('utf8'))
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend()
img.savefig('pic/fasterTem4.pdf')
img = plt.figure()

for i in range(8):
    num =  i+0.5
    if i==7:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-',label
=str(num)+ ' mA')
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-.',label
=str(num)+ ' mA')
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend(loc =3)
img.savefig('pic/fasterDianliuAll.pdf')
img = plt.figure()
tem = [1.5,7.5,3.5,15.5]
for num in tem:
    #num =  i*10+10
    if num==7.5:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-',label
=str(num)+ ' mA')
    else:
        plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-.',label
=str(num)+' mA')
#plt.plot(S21_5['f'].values,y,label = 'Predict')
#plt.plot(S21_5['f'].values[::100],y[::100],'')
#plt.plot(S21_5['f'].values,H_pred1.data.numpy(),label = 'Predict2')
plt.ylabel('Amplitude (dB)')
plt.xlabel('freq (GHz)')
plt.legend()
img.savefig('pic/fasterDianliu4.pdf')
```

```
q3df =pd.read_csv('q3Tem.csv',header =None)
fig = plt.figure(figsize=(16,16))
df = pd.read_csv('Q1_all.csv',header=None)
for i in  range(8):
    plt.subplot(421+i)

    num = i*10+10
    plt.plot(S21_5['f'].values,ComputeAmplitude(T0 = num),'-',label =
str(num)+'℃ faster Model'.decode('utf8'))

    plt.plot(S21_5['f'].values,q3df[i].values,'-.',label = str(num)+'℃
Origin'.decode('utf8'))

#sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400
:100],ci=100,order=1,truncate=True,label = 'Power: 1',color='g')
    plt.ylabel('Amplitude (dB)')
    plt.xlabel('freq (GHz)')
    plt.legend(loc =3)

fig.savefig('pic/fasterTemVS.pdf')
fig = plt.figure(figsize=(16,16))
df = pd.read_csv('Q1_all.csv',header=None)
for i in  range(8):
    plt.subplot(421+i)

    num = i + 3.5
    plt.plot(S21_5['f'].values,ComputeAmplitude(I = num),'-',label =
str(num)+'ma faster Model'.decode('utf8'))
    plt.plot(S21_5['f'].values,q3df[i].values,'-.',label=str(num)+'mA
Origin'.decode('utf8'))

#sns.regplot(L20C['I'].values[300:1400:100],L20C['P'].values[300:1400
:100],ci=100,order=1,truncate=True,label = 'Power: 1',color='g')
    plt.ylabel('Amplitude (dB)')
    plt.xlabel('freq (GHz)')
    plt.legend(loc =3)

fig.savefig('pic/fasterIVS.pdf')
```