



基于PyQT的数列学习辅助程序

[摘要] 随着高中数学学习的进行, 学生往往会在一定的学习阶段产生困难。遇到在一定位置卡顿, 学习效果与效率下降等问题。同时, 学校教育无法针对学生的个性化学习提供较好的帮助与辅助, 导致学生需要依靠自学或者补课来追赶学习进度。这不仅会导致额外的经济负担, 还会使得学生的心理情况下降, 压力增加等情况。本课题结合自身感受与环境调查, 针对数列学习方面进行研究, 解决自学难度高, 学校学习曲线陡峭等问题。本系统支持学生进行个性化题目定制, 标签系统辅助学习, 同时也有提供计算器等实用工具。课题软件使用 PyQt6 框架编写程序, 结合Web应用内容提供用户页面。同时通过 BERT 模型进行题目分类, 辅助自学。本课题具有以下效果: 填补在个性化自学方面软件的空白, 为学生提供高度定制化的学习过程与学习曲线。同时智能化学习, 使用辅助工具加速学习效率, 提升学习效果。对学校的学习起到辅助作用。

[关键词]

数列, 分类, 学习辅助, PyQt

目录

1.引言

- 1.1 课题由来
- 1.2文献综述
- 1.3研究创新性

2.研究方法

- 2.1提出方案
- 2.2软件功能设计
- 2.3技术路线
- 2.4实现途径

3.方案实现

3.1题目数据集实现

- 3.1.1题目储存单元
- 3.1.2 题目管理器实现

3.2用户软件界面制作

3.2.1顶级用户界面

- 3.2.1.1顶级用户界面设计

- 3.2.1.2顶级用户界面实现

3.2.2题目练习页面

- 3.2.2.1题目练习页面设计

- 3.2.2.2题目练习页面实现

3.2.3数列计算器实现

- 3.2.3.1数列计算器设计

- 3.2.3.2数列计算器实现。

3.3 自动分类系统

- 3.3.1分类系统设计

- 3.3.2分类系统实现

- 4. 分析与讨论

- 5. 总结与展望

- 6. 参考文献

7. 附录

- 7.1 主程序窗口(`CenterMenu.py`)

- 7.2 题目主窗口(`question_show.py`)

- 7.3 问题管理引擎(`Question.py`)

- 7.4 右边栏(`RightBarWeb.py`)

- 7.5 计算器(`Calculator.py`)

- BERT训练(`math_bert_train.py`)

- 题目数据集(`question.json`)

- 图片识别(其一)(`tessteract-ocr-test3.py`)

- d3引擎(`d3.js`)

1.引言

1.1 课题由来

数列作为高中数学的核心内容之一，是《普通高中数学课程标准》规定的必修模块，在高考数学卷中占据约10%的分值比重。其重要性不仅体现在应试层面，更在于培养学生建立递推思维、函数建模能力和数形结合意识的关键作用。在高中的数学学习过程中，数列的相关知识种类繁多，而知识点之间的相关度较低。学习中需要更多的训练才能以较好的程度掌握。而在学习过程中，传统课堂受限于二维板书呈现方式，学生难以直观感知数列项数变化时的动态规律。同时学习资源之间互相独立，难以形成合适的学习曲线。

本研究旨在开发基于PyQT框架的数列学习辅助程序，提供自动绘制数列图像，自动对题目分类整理等功能，填补现有教学工具在自适应学习中的空白。

1.2文献综述

经过对当前存在的辅助学习相关软件进行调研，可以得知当前的学习辅助软件一类以视频教学为主，例如空中课堂。另一部分如作业帮，菁优网等缺少对题目的整理和难度排序，不适合初学者进行自学。

同时，已经存在的辅助学习软件往往拥有自身的题库，或者是对历年试卷的收集整理。例如作业帮等平台拥有大量题目数据，

1.3研究创新性

1. 本研究旨在通过自行定制化的题库，合并在不同练习册或软件中的题目，达成一站式无需切换的效果。同时定制化的题库也有利于控制练习范围，进行强化训练。

2. 通过人工智能的自动标注技术，减少用户的劳动量。并基于标签自动对题目进行分类整理，构建学习曲线。

3. 以网状形式展示题目之间的联系，有利于用户进行知识点梳理等工作，优化学习模式。

4. 结合数列计算器等实用工具，优化用户使用体验，减少切换次数。

2.研究方法

2.1提出方案

本课题设计目标为通过设计一款软件，使其解决一般学生在数列学习方面遇到的一些困难。

主要聚焦于练习题目分散, 相关性低, 没有形成合适的学习顺序。同时加入优化做题体验的个性功能, 辅助用户进行练习。

经过研究, 针对设计目标提出课题核心方案:

1. 辅助学习顺序设计

通过对题目进行自动归类, 排序, 形成较优的做题顺序。具体化为根据类型, 难度两个因素进行排序, 主要原则为类型逐渐过渡, 难度由易到难。同时记录练习情况, 便于用户自行总结学习效果。

2. 对题目进行自动分类

通过训练人工智能模型, 达成输入题面自动输出题目类型与预估难度的功能。能够为用户构建自动化的管理体系, 辅助形成分类练习集, 提升学习效果, 并为用户题目录入提供支持。

3. 用户自定义题目录入

为了打破题目在大量练习集的分散情况, 用户可以将自己的题目录入到程序当中, 形成自定义题目集。弥补软件自带题目集不足的缺陷, 提升用户使用效果。

2.2软件功能设计

根据前文2.1所描述的方案, 设计软件具体功能, 软件的具体功能如下所示。

用户菜单页面	查看并打开题目
	查看题目集与题目间关联
	图示题目间关系
题目练习页面	显示根据此题的其他推荐题目
	显示题目标签, 难度等信息
	渲染题目内容
	输入题目答案, 并判断正确与否
数列计算器	接受算式输入
	指定通项公式或者递推式
	指定数列初始项
	在屏幕上绘制数列
题目自动分类	接受tex题面与类型输入

完成模型训练并能将其导出

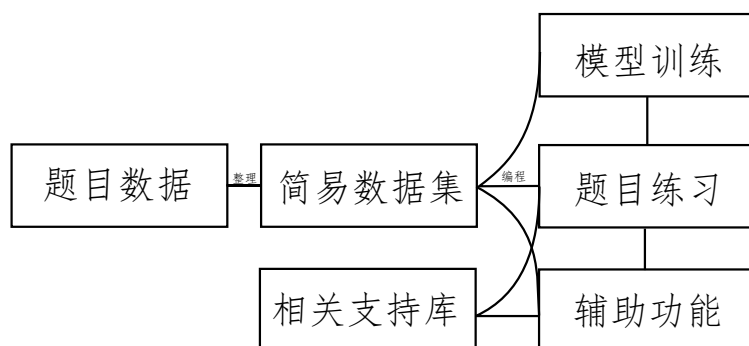
接受新的题面并自动输出类型

2.3技术路线

软件部分, 使用 `PyQT` 作为整体框架, 并结合使用 `Qt designer` 设计软件界面样式, 完成数列计算器, 用户菜单和题目练习页面的布局设计。使用 `matplotlib` 完成数列绘图相关工作。在其他用户场景通过使用 `QwebEngineWeights` 渲染 `HTML` 文本, 使其支持更加丰富的表现体验。并同时借助 `HTML` 完成对 `markdown`, `tex` 等其他表现形式的支持。

2.4实现途径

该课题通过如下方式完成此程序的设计与测试



3.方案实现

3.1题目数据集实现

3.1.1题目储存单元

通过以下方式储存每一道题目, 并汇聚为数据集。

```
{  
  "problem_id": "题目编号",  
  "question": [  
    "题面"  
  ],  
  "options": [  
    "选项"  
  ],  
  "answer": "答案",  
  "analysis": "解析",  
  "tags": ["标签"],  
  "difficulty": "难度"  
},
```

1. 程序运行时, 通过读取和解析json文件, 我们可以获取一道题目的详细相关信息。以离线 json 文件的方式记录题目集, 并在程序运行初始化时将其加载到内存中, 以支持后续的读取 tag, 绘制目录等工作。

2. 在具体实现中, 每道题目被储存在一个 Question 对象中。其定义如下:

```

class Question(object):
    """
    储存一道题目的数据
    """
    surface: str | None
    solution: str | None
    analysis: str | None
    options: list[str]
    tags: list[str]
    difficulty: str
    question_id: int

    def __init__(self, surface: str = "", answer: str | None = None, analysis: str | None = None):
        ...
    def unpack(self) -> tuple[str, list[str], str, str, list[str]]:
        ...
    def load_from_file(self, file_name: str) -> Question:
        ...
    def load_from_json(self, d : dict) -> Question:
        ...

```

3. 考虑到一定的兼容性, 在初始化中(`__init__`), 题面, 答案, 解析是必须提供的, 通过默认设置为 `None` 的方法, 使其在正常运行时识别对象是否被正常初始化, 避免发生错误。

4. 函数 `unpack` 可以返回显示题面需要的所有元素, 通常在提供参数时使用。其具体实现如下:

```

def unpack(self):
    return self.surface, self.options, self.answer, self.analysis, self.tags

```

5. 函数 `load_from_file` 和 `load_from_json` 均为加载函数, 提供了从单独的题目文件或者已经反序列化的字典中加载题目信息的功能, 同时在读入时自动识别附加信息(选项, 标签, 难度等), 并自动尝试加载。

3.1.2 题目管理器实现

1. 题目管理器提供了批量加载大量题目, 提供题目信息, 整理统计信息, 回答情况等功能, 同时拥有单个 `Question` 元素的所有权。其核心元素如下:

```
class QuestionsManager(object):
    questions_dir : dict[int, tuple[bool, Question]]
```

其中 bool 类的值代表了该题的回答正确情况, Question 的值为 id 对应的题目

2. 对于基本容器, QuestionsManager 配有标准的增删查改操作支持, 通过重载Python约定的调用函数 `__getitem__`, `__setitem__` 和 `__contains__`, 管理器对象支持 `[]` 取出对象, `in` 关键字查询。

3. 在实际使用中, `[]` 取出元素需要有两种返回方式, 其一为根据 id 返回题目对象的副本, 并根据约定只读使用。其二是根据输入的题目对象返回其回答情况, 因为 Python 不支持函数多态, 所以使用了 `isinstance` 进行参数类型检查。其具体设计如下:

```
def __getitem__(self, question_id : Question | int):
    if isinstance(question_id, int):# <- 类型检查 : 如果输入的是题目编号
        if question_id in self.questions_dir.keys():# <- 检查该编号是否存在
            return self.questions_dir[question_id][1] # <- 根据序号约定返回问题对象
        else:
            return Question()# <- 返回未正确初始化的对象阻止接下来的操作
    elif isinstance(question_id, Question): # <- 类型检查 : 如果输入的是题目对象
        if question_id.question_id in self.questions_dir.keys():
            return self.questions_dir[question_id.question_id][0] # <- 根据序号约定返回回答情况
        else:
            return False # <- 对于未知的题目默认返回False
```

4. 同时, 管理器中的题目对象只能由初始化后固定, 不支持二次修改。根据此原则, `__setitem__` 只接受一个 bool 型参数用于对作答情况的修改, 而不能修改该编号对应的 Question 对象本身, 避免编号对应关系发生改变。

3.2用户软件界面制作

3.2.1顶级用户界面

3.2.1.1顶级用户界面设计

顶层用户界面会在程序开始运行时默认打开, 并起到菜单作用。

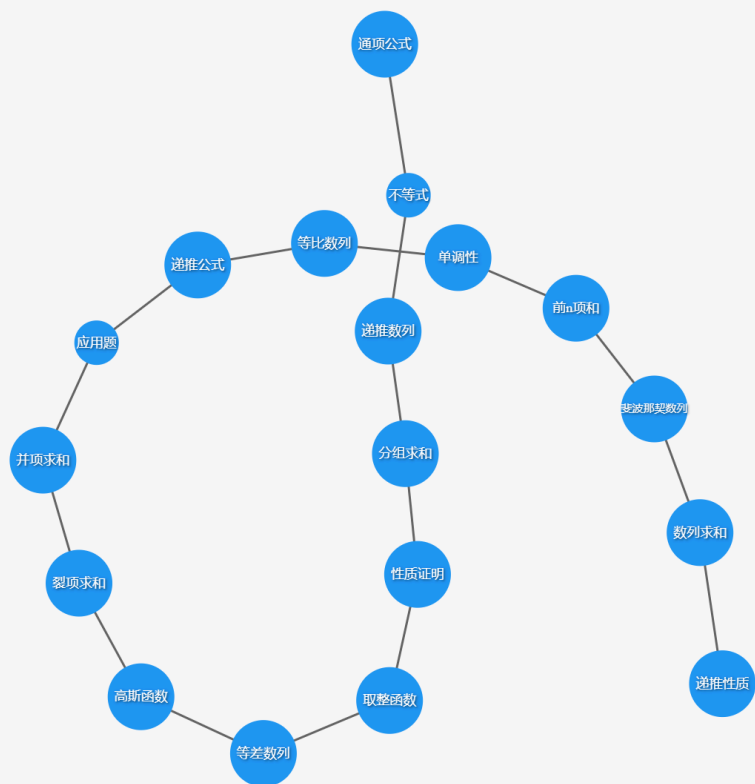
1. 用户运行程序后, 会自动打开主菜单, 并同时识别加载题目的所有标签连接成一条链。同时根据自动力传导排版。

2. 用户可以点击标签节点, 将打开二级题目选择页面。选取每道题目的最多前五个字作为标

识符, 同时提供一个特殊节点 `exit` , 可以返回标题目录。

3. 用户点击题目节点, 可以打开题目练习界面, 同时目录页面不会关闭, 支持题目多开等操作。

4. 用户可以拖动节点排列, 同时每次进入页面都会改变节点布局



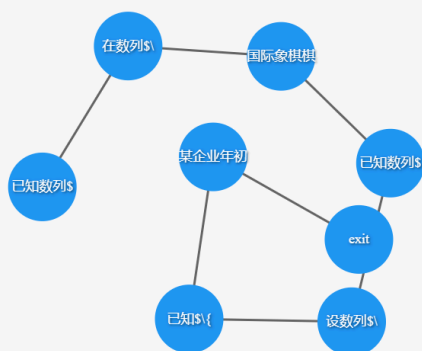


图3.1 主菜单页面, `递推数列`分类下的目录

3.2.1.2 顶级用户界面实现

1 前后端通信

1.1 PyQt6 提供了信号槽机制来允许 python 与 JavaScript 代码之间传输数据。通过 `pyqtSignal` 注册信号槽, 并使用 `emit` 发送数据, 并在 JavaScript 端通过 `connect` 连接 `js` 方法, 结合 `@pyqtSlot` 装饰器完成类型标注, 实现 `js` 触发 `python` 的函数并向其发送参数。

而在 JavaScript 端引入 `qrc:///qtwebchannel/qwebchannel.js` 同时在 `python` 端创建 `QWebChannal` 对象, 完成绑定即可由 `python` 向 `js` 端发送数据。

1.2 在具体实现中, 使用信号槽绑定了用于点击处理, 拖动处理两个函数。使用 `channel` 向网页端发送节点的名称, 位置等信息。

注册信号槽:

```
class NodeData(QObject):
    updateGraph = pyqtSignal(str)
    positionChanged = pyqtSignal(str)
```

2. 显示链表/图表

2.1 每个node即时使用, 即时释放。每次从 `QuestionManager` 获取数据, 储存指向题目的 `id` 信息, 自身的显示名字以及自身坐标, 以字典键值对的方式表示。每次重新绘图时清空 Python 端的 `node_list` 数组并重新生成, 在传递给前端 JavaScript 脚本后生命周期结束。

用于注册 `node` 的函数:

```
def add_node(self, name : str, tag : str):
    self.nodes.append({"id" : f"id_{tag}", "tag" : tag, "name" : name, "x": 600, "y": 400})
    self.node_names.append(name)
```

同时, 需要将单独的点连成一条链, 在具体实现中表现为操作字典注册对象, 具体实现如下:

```
def add_link(self, a : str, b : str):
    self.links.append({"source": f"id_{a}", "target": f"id_{b}"})
```

注意到在加入节点时同时维护了一个 `node_name` 列表, 其作用为记录已经添加的节点, 防止重复添加引发混乱, 以其支持查询函数 `has_added`, 实现如下:

```
def has_added(self, name : str):
    return name in self.node_names
```

并补充清空函数, 防止所有权混乱, 其实现如下:

```
def clear(self):
    self.nodes.clear()
    self.node_names.clear()
    self.links.clear()
```

就此, `nodeData` 类提供了增删查操作, 而改在实际使用中并无必要, 满足容器的基本性质

2.3 为了减少对节点排布的工作量, 使用 `QwebEngineWeights` 结合 HTML 中的 `D3.js`, 自动创建动态并且可交互的图像, 同时使用 `Qchannel` 实现软件内的前后端通信, 传递节点和连线信息。

在实际执行中, 先由 python 后端筛选, 注册节点与连接关系, 将数据通过 `channel` 传递给 `webengine` 内, 同时通过编写 `html` 与 `javascript` 相关代码绘制节点与连线。同时使用 `d3.js` 对力矩修改实现有限的拖拽效果。

2.4 在实际的实现中, 由于浏览器执行的异步性, 有时会出现 `d3.js` 和 `Qwebchannel` 初始化

完成之前就开始尝试加载绘制代码。这会使得浏览器核心崩溃, 无法正确显示图像。为了结局相关问题, 在 `JavaScript` 代码中加入了相关判断逻辑, 识别当前的初始化阶段和初始化情况, 避免在初始化完成前执行浏览器主逻辑。

初始化入口:

```

// D3可用性检查
document.addEventListener('DOMContentLoaded', () => {
    if(typeof d3 !== 'undefined') {
        isD3Ready = true;
    } else {
        console.error('D3.js加载失败');
    }
});

// Qt WebChannel初始化
new QWebChannel(qt.webChannelTransport, (channel) => {
    console.log('WebChannel初始化完成', channel.objects);
    window.nodeData = channel.objects.nodeData;
    console.log('nodeData对象可用性', typeof window.nodeData.updateGraph);
    webChannelLoaded = true;
    checkInit();
});

if (typeof d3 !== 'undefined') {
    console.log('D3.js已加载');
    d3Loaded = true;
    checkInit();
} else {
    console.error('D3.js未加载! ');
}

function checkInit() {
    console.log('检查初始化状态, webChannelLoaded:', webChannelLoaded, 'd3Loaded:', d3Loaded);
    if (webChannelLoaded && d3Loaded) {
        console.log('全部依赖加载完成, 开始初始化');
        realInitialize();
    }
}

// 统一初始化入口
function realInitialize() {
    ...
}

```

2.4 同时, `QWebEngineWeights` 继承自 `QWeights`, 并需要在重载(reload)后才能重新适应窗口大小。为了解决重载操作会销毁网页部分的 `channel` 导致断开链接的问题, 设计了在前端的 `loadFinished` 触发时重新加载的功能, 通过reload操作触发前端的 `loadFinished` 信号, 进而进行重初始化, 避免 `channel` 被销毁导致出现悬空指针。

3. 处理对节点的点击与识别

3.1 `node` 一共有三种用法, 其效果分别是打开下一层目录, 返回根目录, 打开题目。为了能够识别每个 `node` 承担的功能, 使用了操作符的设计。每个 `node` 的 `tag` 具有层级结构, 由 `::` 分割。其中打开次级目录的 `tag` 为 `menu::名称`, 而负责回到主页面的 `tag` 为 `exit::...`, 均不匹配的则会被识别为题目名称。

关于 `tag` 识别的代码实现:

```
@pyqtSlot(str)
def handleNodeClick(self, tag : str):
    logger.debug(f"Node clicked! Tag: {tag}")
    opt = tag.split("::")
    if opt[0] == 'menu':
        self.clear()
        self.upper.add_one_tag(opt[1])
    elif opt[0] == "exit":
        self.clear()
        self.upper.send_initial_data()
    else:
        new_window = question_show.HomeWindow(self.upper, tag)
        new_window.show()

    self.refresh_graph()
```

4. 获取显示内容

4.1 显示时, 需要先提取所有的 `tag`, 形成目录, 对于单个被选中的 `tag`, 需要再次处理出所有含有该 `tag` 的题目。在具体实现中依赖于 `nodeData` 提供的相关接口

提取所有 `tag` 并注册, 连线

```

self.node_data.clear()
all_tags = set()
for _, each_ques in self.question_lib.questions_dir.values():
    for tag in each_ques.tags:
        all_tags.add(tag)

tag0 = None
for tag in all_tags:
    self.node_data.add_node(tag, f"menu::{tag}")
    if tag0 is not None:
        self.node_data.add_link(f"menu::{tag0}", f"menu::{tag}")
    tag0 = tag

```

在取出含有特定 `tag` 的题目时, 通过使用列表推导式快速取出所有符合要求的题目来加速运行。列表推导式是 Python 中用于创建列表的一种简洁而优雅的方法。通过列表推导式, 可以从一个或多个可迭代对象 (如列表、元组、字符串、集合等) 中生成一个新的列表, 同时可以在生成过程中对元素进行过滤和转换。列表推导式不仅能使代码更加简洁, 还能提升可读性和执行效率。在实现中除了提取题目, 还将其以难度为关键字排序, 以形成合适的学习曲线, 其具体实现如下:

```

def add_one_tag(self, tag : str):
    same_tag_list = [q for _, q in self.question_lib.questions_dir.values() if tag in q.tags]
    same_tag_list.sort(key=lambda x: {"简单": 1, "中等": 2, "困难": 3, "空": 4}[x.difficulty])

    q1 = Questions.Question()
    q1.question_id = "exit"
    q1.surface = "exit"
    for q in same_tag_list:
        self.node_data.add_node(q1.surface[0:min(len(q1.surface), 5)], q1.question_id)
        self.node_data.add_link(q1.question_id, q.question_id)
        q1 = q
    self.node_data.add_node(q1.surface[0:min(len(q1.surface), 5)], q1.question_id)

```

3.2.2 题目练习页面

3.2.2.1 题目练习页面设计

题目练习页面会在目录选择了题目后被调出, 负责相关操作。

- 1. 题目练习页面需要其能够支持 markdown 和 $\text{T}_\text{E}\text{X}$ 语法, 显示题面相关信息。并同时显示选项等相关内容。
- 2. 用户可以在该页面内提交题目答案并核对, 检查用户作答正确性。
- 3. 可以根据用户当前正在作答的题目进行其他题目推荐, 并显示在侧边栏中, 允许用户快捷跳转至该题目
- 4. 在侧边栏可以显示题目难度, 标签等相关信息。

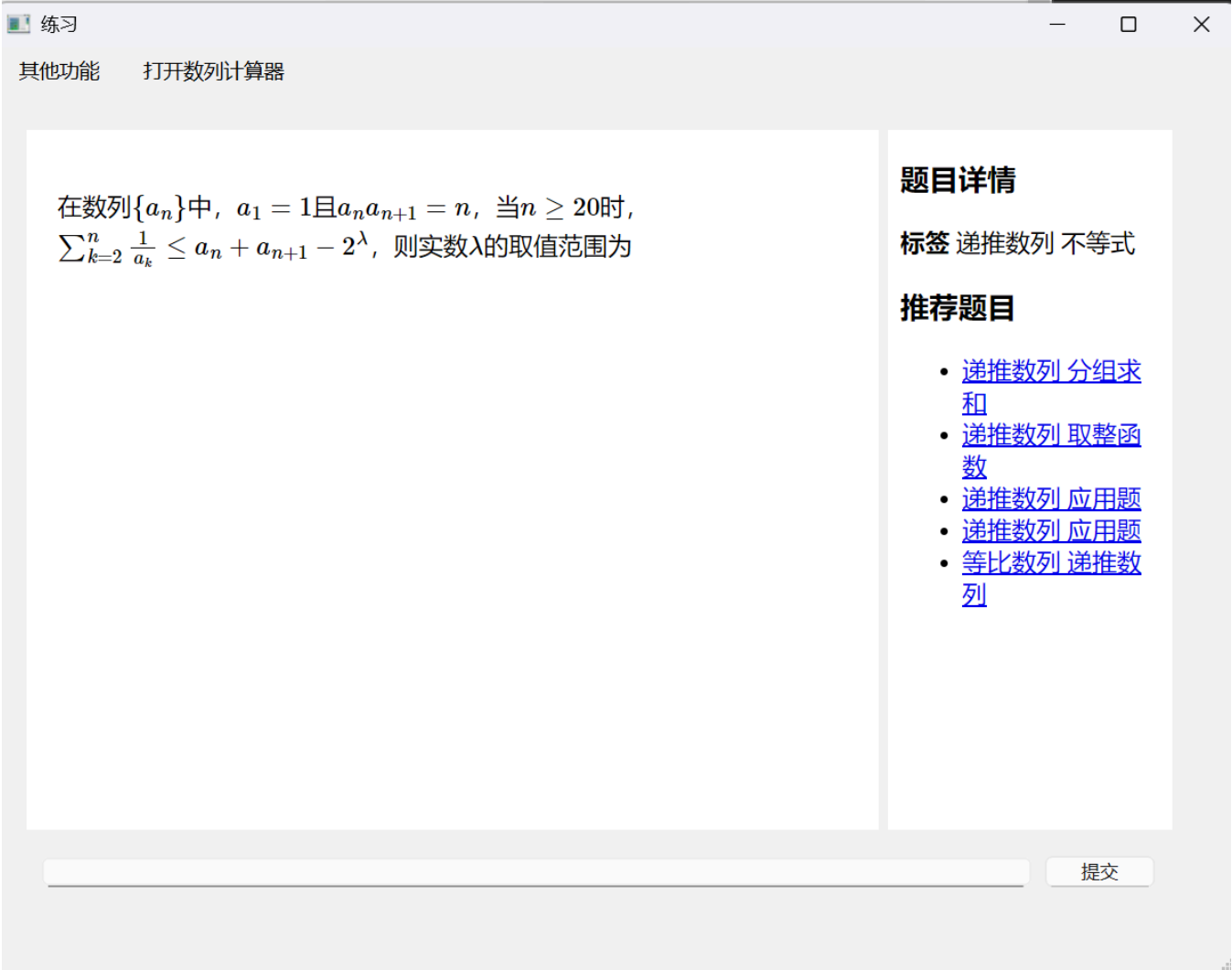


图3.2.1 答题页面

3.2.2.2题目练习页面实现

- 1. 题目练习页面中的题目渲染同样使用了 `QWebEngineWeights` , 以更加方便的支
持 markdown 和 $\text{T}_\text{E}\text{X}$ 。使用了 python 的 markdown 库来将 markdown 格式的文本处理成 html 格式,
再在其中引用 mathjax 的 js 文件进行 $\text{T}_\text{E}\text{X}$ 格式的处理, 最终将两部分结合渲染。
- 2. 右侧栏目的实现同样为浏览器内核, 而题目跳转的部分同样使用了前后端通信的信号槽,
由于在此处处理的信息多为文本, 则创建了 `Bridge` 负责直接生成相应的 html 格式文本, 省

去 Qchannel 的多余加载代码和开销。

Bridge 类实际上为 html 生成代理, 通过约定接口共享函数和 QWebEngineWidgets 实例的所有权。在每次刷新页面时通过 replace 文本替换调整显示内容。

Bridge 同时负责启动前端页面渲染和内容替换, 为了方便从 json 文件直接打开题目内容, 加载题目的方法同样支持两种输入, 数字输入即识别为id, 文本输入即识别为文件地址。而前端的 tag 信息传递中只能选择 str 类型的信号槽, 因此在实际收到 tag 先尝试将其转化为数字, 如果在此过程中打开题目抛出异常, 则将其作为文件地址处理。

Bridge 生成实现细节:

```
@pyqtSlot(str) # <- 注册前端信号槽
def handle_tag(self, tag):
    logger.debug(f"Received tag: {tag}")

    assert self.view is not None # 此处为约定的调用接口检查
    assert callable(self.main_question_connect) #
    assert callable(self.get_tag) #
    tags = []
    try:
        self.main_question_connect(int(tag)) # id
        tags = " ".join(self.get_tag(int(tag))) #
    except ValueError as e: # 支持两种打开方式
        self.main_question_connect(tag) # 文件名
        tags = " ".join(self.get_tag(tag)) #

    self.view.setHtml(self.base_html
        .replace("[replaced-site1]", self.question_list[1][0])
        .replace("[replaced-site2]", self.question_list[2][0])
        .replace("[replaced-site3]", self.question_list[3][0])
        .replace("[replaced-site4]", self.question_list[4][0])
        .replace("[replaced-site5]", self.question_list[5][0])
        .replace("[replaced-site1-title]", self.question_list[1][1])
        .replace("[replaced-site2-title]", self.question_list[2][1])
        .replace("[replaced-site3-title]", self.question_list[3][1])
        .replace("[replaced-site4-title]", self.question_list[4][1])
        .replace("[replaced-site5-title]", self.question_list[5][1])
        .replace("[replaced-tags]", tags)
    ) # <- 为了实现方便牺牲了一部分可扩展性
```

其中的 `main_question_connect` 和 `get_tag` 均为由主窗口传递的函数接口, 在使用前进行检查防止错误的初始化导致异常。例如 `main_question_connect` 实际绑定至主程序中的 `load_question`。

3. 在加载题目时需要同时生成题目推荐, 在实际编写中则为查找含有相同题目类型的题目并输出, 使用列表推导式进一步提高效率, 其详细实现如下:

```
same_tag_list : list[Questions.Question] = [] # 类型标记以触发自动补全
same_tag_list += \
[q for _, q in self.question_lib.questions_dir.values()
#操作对象 解包
if tag in q.tags and
#判断tag
q.question_id != question_input.question_id]
#去重
```

由此取出所有符合要求的题目, 再将其填入填充队列中, 完成替换。

4. 当用户点击提交按钮, 软件会将输入框的内容与储存的答案进行比对, 并返回是否回答正确:



图3.2.2 回答正确的反馈

5.在题面上方工具栏中提供了一定的辅助功能,其中打开计算器可以打开数列计算器,提供学习辅助,其他功能包含关闭右侧栏和从文件加载问题,关闭右侧栏可以提供更加专注的思考环境,从文件加载题目则提供了自定义题目支持。

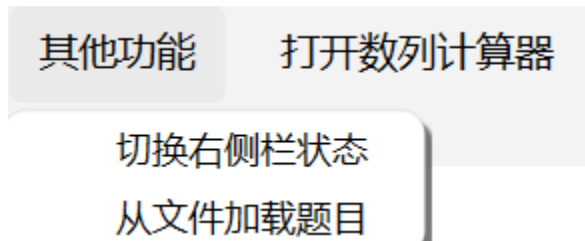


图3.2.3 顶部工具栏

3.2.3 数列计算器实现

3.2.3.1 数列计算器设计

1. 计算器需要接受用户输入, 并计算出其若干个点呈现在屏幕上。同时应该支持通项公式和递推式两种输入格式。
2. 计算器支持指定计算的项数和初始项的值, 为递推式提供一定支持。
3. 计算器表现形式需要足够直观, 因此选择了图像的方式来展现计算完成的数列。
4. 图像的方式展示会丢失一部分精准度, 同时需要支持从图像中获取准确数值。

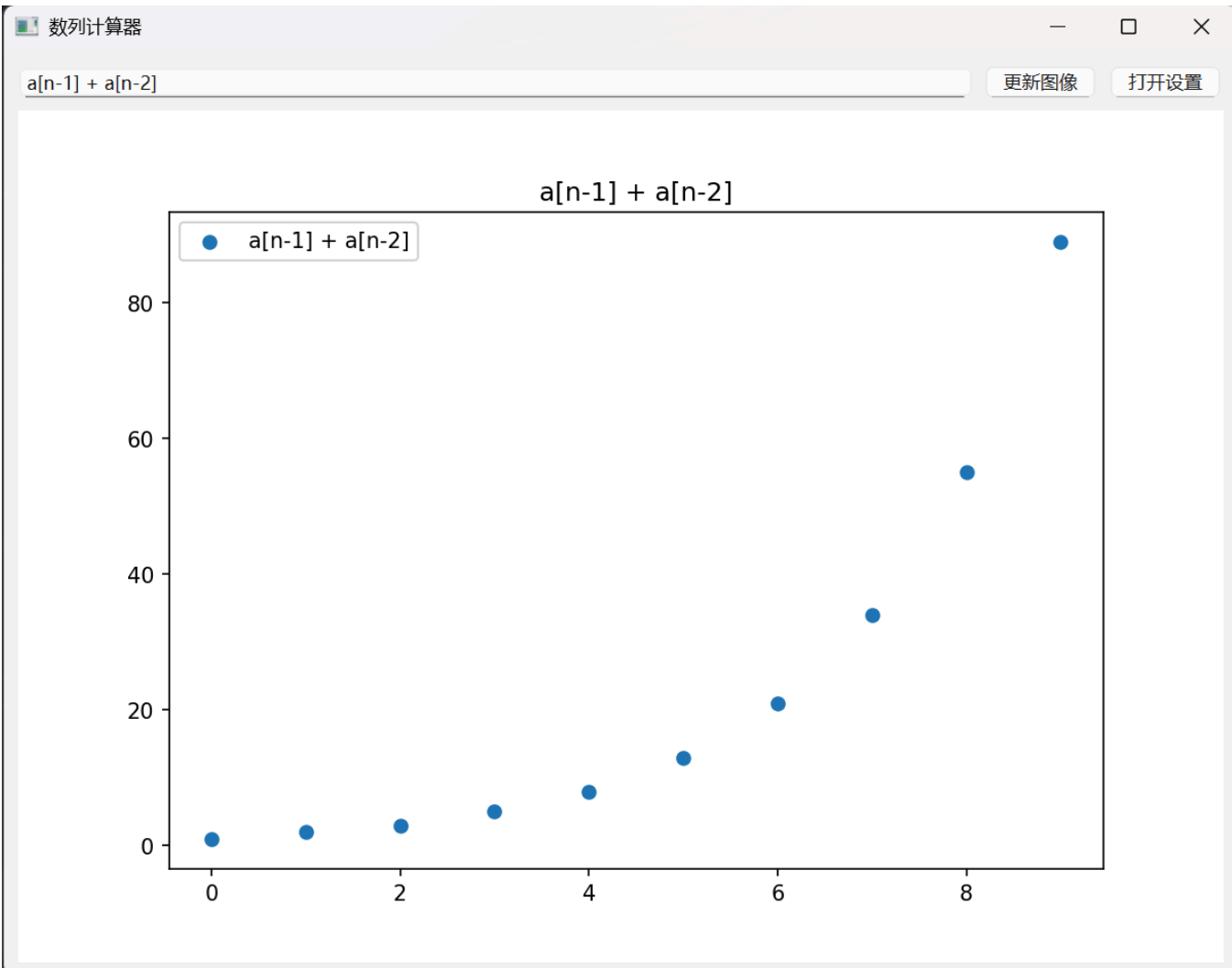


图3.2.4 计算器页面

3.2.3.2 数列计算器实现。

1. 为了图像的表现形式, 使用了与 `pyqt` 相互兼容的 `matplotlib` 提供支持, 并且通过 `matplotlib.pyplot` 直接在程序页面上绘制图像, 接受鼠标信息, 减免了大量的浏览器操作, 同时去除了浏览器依赖, 使其能够单独使用。

设置 `matplotlib` 使其与 `pyqt` 模式相兼容。

```
matplotlib.use("QtAgg")
from matplotlib.backends.backend_qtagg import (
    FigureCanvasQTagg as FigureCanvas)
```

通过继承 `matplotlib` 提供的类, 能够以对待一般控件的方式将其添加到 `pyqt` 产生的窗口中, 而同样能与对待 `pyplot` 的方式对其进行设置, 为代码的编写提供了大量便利。

绘图控件的一部分初始化, 包含了生成默认数组, 图像, 设置标题和抬头等内容:

```
class MplCanvas(FigureCanvas):
    def __init__(self, parent=None):
        fig, self.ax = plt.subplots()
        super().__init__(fig)
        self.x = numpy.arange(0, 10, 1)
        self.y = 2 * self.x + 1
        self.ax.scatter(self.x, self.y, label='2x + 1')
        self.ax.set_title('Matplotlib in PyQt6')
        self.ax.set_xlabel('n')
        self.ax.set_ylabel('a[n]')
        self.ax.legend()
```

2. 计算器需要能够以给出的公式更新图像, 而对待通项公式和递推式需要两种方式处理, 其均使用了 `python` 提供的 `eval` 函数解析表达式, 通过约定特定的全局变量来提供递推支持。约定 n 为项数或自变量, `a[]` 为储存了已计算数值的数组。其关键在于维护键值对字典 `a`, 将计算的值储存在字典中, 同时字典也作为参数提供给表达式进行计算, 达到访问前项值的效果, 其关键实现如下:

对于通项公式:

```

for k in range(n):
    context = {
        'n': k,
        'math': math
    }
    try:
        current = eval(expr, context, math.__dict__)
        a.append(current)
    except Exception as e:
        logger.error(f"计算通项时出错: {e}")

```

对于递推式, 则增加了处理初始项的代码

```

initial = set_dir['init_number']
a = initial.copy()
current_length = len(initial)
if n <= current_length:
    return numpy.array(a[:n])
for k in range(current_length, n):
    context = {
        'a': a,
        'n': k,
        'math': math
    }
    try:
        current = eval(expr, context, math.__dict__)
        a.append(current)
    except IndexError as e:
        logger.error(f"初始项设置错误 : {e}")
    except Exception as e:
        logger.error(f"计算递推式时出错: {e}")

```

3. 可以通过设置来改变计算的项数与初始项, 设置页面由一个单独的页面组成。其中可以自定义修改相关内容, 在关闭时可以自动保存, 避免忘记保存导致没有改变等情况发生。此处可以修改给出的式子类型, 初始项与计算项数。同时在程序运行中使用一个字典来保存设置内容, 在需要调取时取出, 实际为字典对象 `set_dir`。

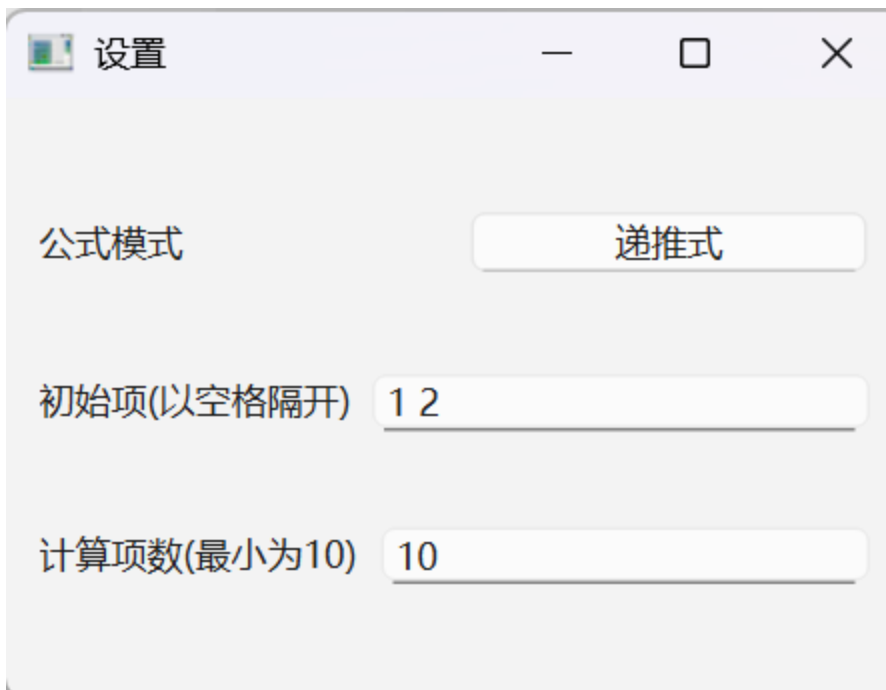


图3.2.5 设置

关于设置的代码实现如下:

```
def change_mode(self):
    now_mode_idx = (set_dir["possible_mode"].index(set_dir["mode"]) + 1) % len(set_dir["possible_m
    set_dir["mode"] = set_dir["possible_mode"][now_mode_idx]
    self.mode_Button.setText({1: "递推式", 2: "通项公式"}[set_dir["mode"]])

def update_setting(self):
    try:
        set_dir["init_number"] = list(map(float, self.init_input.text().split(" ")))
    except Exception as e:
        logger.error(f"初始化初始项数组 : {e}")
    try:
        set_dir["max_index"] = max(int(self.max_number_input.text()), 10)
    except Exception as e:
        logger.error(f"最大项数设置出错 : {e}")
```

4. 通过注册 `matplotlib` 的鼠标移动事件, 能够支持图像显示特定点的具体数值。在具体运行中, 会选择当前距离鼠标最近的一个节点, 将其的数值显示出来。其使用了继承类中的 `mpl_connect` 绑定鼠标移动, 并使用 `annot` 来指定显示类型。

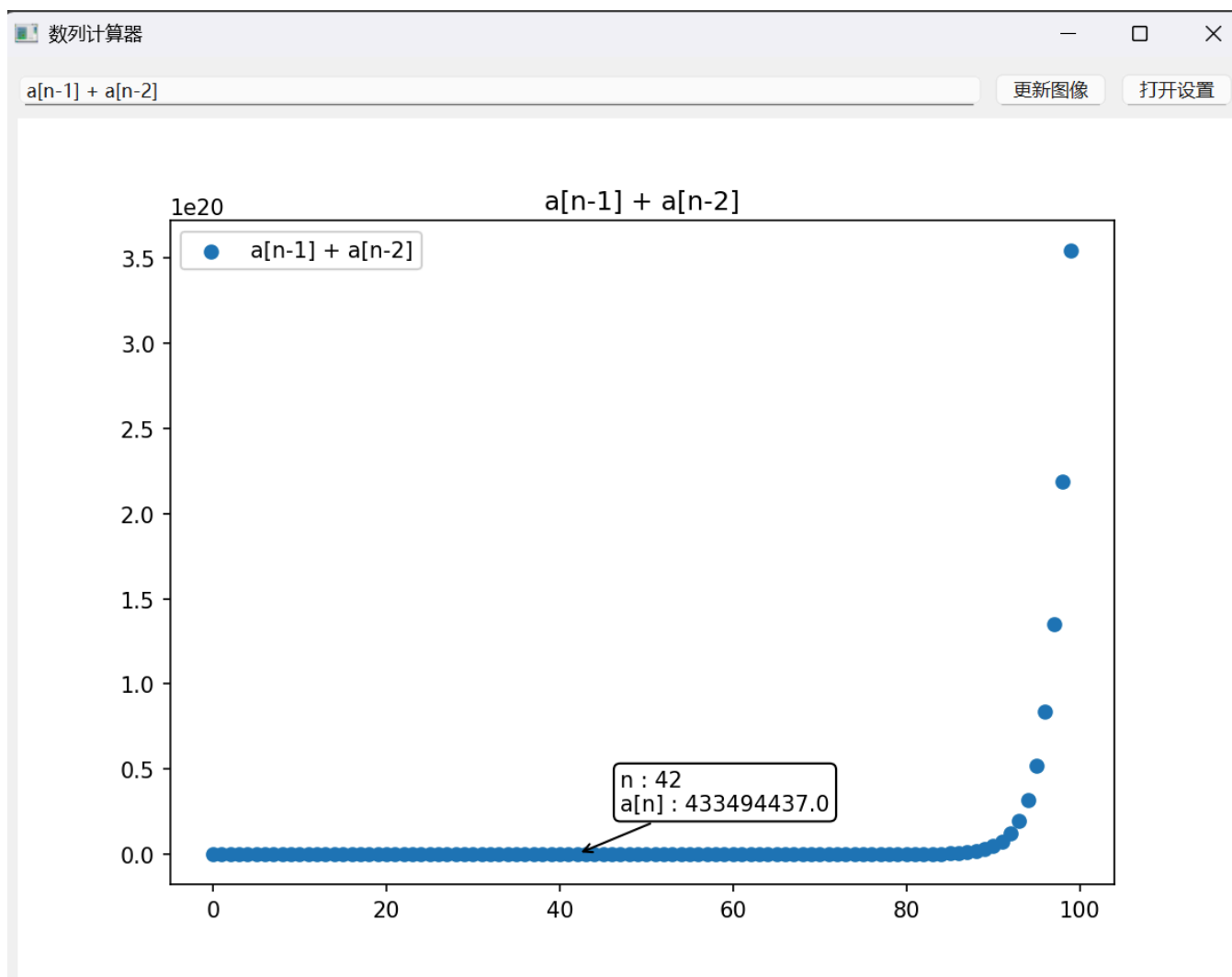


图3.2.6 指示器效果

其在代码中的注册与实现如下:


```

#注册
self.annot = self.ax.annotate("", xy=(0, 0), xytext=(20, 20),
                               textcoords="offset points",
                               bbox=dict(boxstyle="round", fc="w"),
                               arrowprops=dict(arrowstyle="->"))

self.annot.set_visible(False)

# 连接鼠标移动事件
self.mpl_connect("motion_notify_event", self.hover)

#实现
def hover(self, event):
    if event.inaxes == self.ax:
        # 获取鼠标位置
        x, y = event.xdata, event.ydata
        # 查找最近的点
        if x is not None and y is not None:
            index = (numpy.abs(self.x - x)).argmin()
            self.annot.xy = (self.x[index], self.y[index])
            self.annot.set_text(f'n : {self.x[index]}\na[n] : {self.y[index]}')
            self.annot.set_visible(True)
            self.draw()
        else:
            self.annot.set_visible(False)
            self.draw()

```

3.3 自动分类系统

3.3.1 分类系统设计

1. 分类系统应当接受 markdown + \LaTeX 结合的输入, 并输出题目的具体类型, 难度, 帮助构建带有类型, 难度等参数的题库, 支持用户界面系统的扩展和运行
2. 分类系统应当在训练完后一并打包在程序中, 使得其动态接受输入并输出, 而不是反复重新训练。
3. 选择 BERT 模型进行训练, 其在相关应用中表现良好, 在11项NLP任务中刷新SOTA记录, 适应自然文本分类问题。

```
graph TD
    A[输入文本] --> B(分词器 Tokenizer)
    B --> C[[Token Embeddings]]
    C --> D[+Position Embeddings]
    D --> E[+Segment Embeddings]
    E --> F[Transformer Encoder x 12/24层]
    F --> G[任务特定输出]
```

3.3.2分类系统实现

1. 图片识别无法完成公式提取的工作, 经过大量实验, 验证了
2. 因为无法寻找到符合要求的数据集, 本项目中所使用的数据为人工结合 `deepseek` 辅助构建, 同时数量过少无法支撑起有效的模型训练要求, 因此代码实现只能根据设计编写。
3. 分类之前, 需要构建标签表与数据集, 通过 `MultiLabelBinarizer` 将标签二值化, 并拟合相似标签, 以支持后一步训练。

4. 分析与讨论

课题通过结合自身感受与查阅资料, 分析了在当前数列学习中遇到的一定困难, 包括教学仓促, 缺少系统化, 定制化的练习, 在实际做题时缺少有力的帮助工具。

同时正常教育中节奏偏快, 对于学习曲线的构建通常较为陡峭。本项目通过适当安排难度和类型, 试图为用户构建一个更好的学习曲线。

本课题中大量使用 `QWebEngineWeights`, 从技术角度分析, 其为实现带来了诸多方便, 但是同时引入了浏览器内核, 导致最终打包文件变得臃肿。

该课题能够为数列学习提供一定的辅助工作, 但是仍然存在一定缺陷。

5. 总结与展望

课题总结:

课题为提高在数列学习中的学习速度与学习效果, 优化学生的练习过程, 减少无意义的刷题训练。通过软件程序的方式提供了一套帮助工具。填补了自主自学工具方面存在的空白, 能够提升自主练习的体验, 提升自学效果。

课题前景展望:

该课题在实现过程中初步具备设计上的功能,但仍然存在较多的不足。其中经过大量尝试仍然无法将中文数学题目通过图片识别的方法一次性转化为 `markdown` + TeX 文本,并导致无法获取充足的数据集进行模型训练,最终使用 `deepseek` 替代了这部分功能的效果。同时在个性化学习曲线的构建仍有不足,根据 `tag` 选取题目有可能造成大量重复刷题,可以通过向其中加入一定的随机化来改良效果。

6. 参考文献

[1]罗文兵,罗凯威,黄琪,等.融合学科知识的数学习题知识点自动标注模型[J].中文信息学报,2024,38(04):143-155.

[2]孟婉颖.基于CRF和深度学习的数学试题知识点自动标注算法研究[D].河南大学,2023.DOI:10.27114/d.cnki.ghnau.2023.001682.

[3]何鑫宇.基于多标签的数学试题自动标注研究[D].天津师范大学,2023.DOI:10.27363/d.cnki.gtsfu.2023.000668.

[4]黄聪.基于知识图谱的数列问题知识表示及实现[D].电子科技大学,2023.DOI:10.27005/d.cnki.gdzku.2023.003463.

7. 附录

7.1 主程序窗口(CenterMenu.py)

```

import sys, os
import json, logging
from PyQt6.QtCore import Qt, QObject, pyqtSlot, pyqtSignal, QTimer
from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget, QTabWidget, QSizePolicy
from PyQt6.QtWebEngineWidgets import QWebEngineView
from PyQt6.QtWebChannel import QWebChannel
from question.base import Questions, question_show

try:
    d3_js = open("question/base/d3.v7.min.js", "r", encoding="utf-8").read()
except FileNotFoundError as e:
    print(f"file not found cwd:{os.getcwd()}")
    raise e

logger = logging.getLogger(__name__)
logging.basicConfig(format='%(levelname)s:%(message)s')
logger.setLevel(logging.DEBUG)

class NodeData(QObject):
    updateGraph = pyqtSignal(str)
    positionChanged = pyqtSignal(str)

    def __init__(self):
        super().__init__()
        self.nodes = [
            {"id": "A", "tag": "info_A", "x": 0, "y": 0},
            {"id": "B", "tag": "info_B", "x": 0, "y": 0},
            {"id": "C", "tag": "info_C", "x": 0, "y": 0}
        ]
        self.links = [
            {"source": "A", "target": "B"},
            {"source": "B", "target": "C"}
        ]
        self.node_names = ["A", "B", "C"]
        self.added_tag = []
        self.upper : GraphWindow | None = None
        self._alive = True # 新增存活标记

```

```

@pyqtSlot(str)
def handleNodeClick(self, tag : str):
    logger.debug(f"Node clicked! Tag: {tag}")
    opt = tag.split("::")
    if opt[0] == 'menu':
        self.clear()
        self.upper.add_one_tag(opt[1])
    elif opt[0] == "exit":
        self.clear()
        self.upper.send_initial_data()
    else:
        new_window = question_show.HomeWindow(self.upper, tag)
        new_window.show()

    self.refresh_graph()

@pyqtSlot(str, float, float) # 新增位置处理槽
def handlePositionChange(self, node_id, x, y):
    for node in self.nodes:
        if node["id"] == node_id:
            node["x"] = x
            node["y"] = y
    self.positionChanged.emit(json.dumps(self.nodes))

def updateNodeTag(self, node_id, new_tag):
    print("Python端updateNodeTag被调用")
    for node in self.nodes:
        if node["id"] == node_id:
            node["tag"] = new_tag
    self.updateGraph.emit(json.dumps({
        "nodes": self.nodes,
        "links": self.links
    }))

def refresh_graph(self):
    self.updateGraph.emit(json.dumps({
        "nodes": self.nodes,
        "links": self.links
    }))

```

```

def clear(self):
    self.nodes.clear()
    self.node_names.clear()
    self.links.clear()

def add_node(self, name : str, tag : str):
    self.nodes.append({"id" : f"id_{tag}", "tag" : tag, "name" : name, "x": 600, "y": 400})
    self.node_names.append(name)

def has_added(self, name : str):
    return name in self.node_names

def add_link(self, a : str, b : str):
    self.links.append({"source": f"id_{a}", "target": f"id_{b}"})

class GraphWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("目录")
        self.setGeometry(100, 100, 1200, 800)
        self.channel = QWebChannel()

        self.browser = QWebEngineView()
        self.node_data = NodeData()
        self.node_data.upper = self

        self.browser.page().setWebChannel(self.channel)
        self.channel.registerObject("nodeData", self.node_data)

        self.browser.setHtml(self.generate_html())

        container = QWidget()
        layout = QVBoxLayout()
        container.setSizePolicy(
            QSizePolicy.Policy.Expanding,
            QSizePolicy.Policy.Expanding
        )

```

```

layout = QVBoxLayout()
layout.setContentsMargins(0, 0, 0, 0) # 移除布局边距
layout.setSpacing(0) # 移除组件间距
layout.addWidget(self.browser)
container.setLayout(layout)
self.setCentralWidget(container)
self.centralWidget().setSizePolicy(
    QSizePolicy.Policy.Expanding,
    QSizePolicy.Policy.Expanding
)

self.question_lib = Questions.QuestionsManager().load_lib(r"题型预测/question.json")
self.browser.loadFinished.connect(self.init_webchannel)
self.browser.loadFinished.connect(self.resend_data)
# QTimer.singleShot(1000, self.send_initial_data)
self.node_data.refresh_graph()

def send_initial_data(self):
    """生成目录"""
    self.node_data.clear()
    all_tags = set()
    for _, each_ques in self.question_lib.questions_dir.values():
        for tag in each_ques.tags:
            all_tags.add(tag)

    tag0 = None
    for tag in all_tags:
        self.node_data.add_node(tag, f"menu::{tag}")
        if tag0 is not None:
            self.node_data.add_link(f"menu::{tag0}", f"menu::{tag}")
        tag0 = tag

    self.node_data.updateGraph.emit(json.dumps({
        "nodes": self.node_data.nodes,
        "links": self.node_data.links
    })))

    self.node_data.refresh_graph()

```



```

def add_one_tag(self, tag : str):
    same_tag_list = [q for _, q in self.question_lib.questions_dir.values() if tag in q.tags]
    same_tag_list.sort(key=lambda x: {"简单": 1, "中等": 2, "困难": 3, "空": 4}[x.difficulty])

    q1 = Questions.Question()
    q1.question_id = "exit"
    q1.surface = "exit"

    for q in same_tag_list:
        self.node_data.add_node(q1.surface[0:min(len(q1.surface), 5)], q1.question_id)
        self.node_data.add_link(q1.question_id, q.question_id)
        q1 = q
    self.node_data.add_node(q1.surface[0:min(len(q1.surface), 5)], q1.question_id)

def init_webchannel(self):
    """ 窗口尺寸变化时通知前端 """
    self.browser.page().runJavaScript(f"""
        window.dispatchEvent(new CustomEvent('windowResize', {{
            detail: {{ width: {self.width()}, height: {self.height()} }}
        }}));
    """)

def resend_data(self):
    """ 带状态检查的数据发送 """
    if self.browser.page().webChannel() is not None:
        self.node_data.refresh_graph()
        print("[PY] 数据已重发")
    else:
        print("[PY] 错误: WebChannel未就绪")

def generate_html(self):
    code = f"""
    <!DOCTYPE html>
    <html>
    <html style="width:100%; height:100%; margin:0; padding:0;">
    <head>
        <meta name="viewport" content="width=device-width,
        initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
        <script src="qrc:///qtwebchannel/qwebchannel.js"></script>
        <script >{d3_js}</script>
    """

```

```

<style>
    .node-group {{
        cursor: move;
        transition: transform 0.2s;
    }}
    .node-circle {{
        fill: #2196F3;
        transition: r 0.2s;
    }}
    .node-text {{
        fill: white;
        font-size: 12px;
        text-anchor: middle;
        user-select: none;
        filter: drop-shadow(1px 1px 1px rgba(0,0,0,0.5));
    }}
    .link {{
        stroke: #666;
        stroke-width: 2;
    }}
    svg {{
        border: 1px solid #eee;
        background-color: #f9f9f9;
    }}
</style>
</head>
<body>
    <div id="graph"></div>
    <script>
        // 初始化状态标志
        let isChannelReady = false;
        let isD3Ready = false;
        let simulation, svg;
        let webChannelLoaded = false;
        let d3Loaded = false;

        // 错误处理
        window.onerror = function(msg, src, line) {{

```

```
        console.error("[全局错误]", msg, "at", src, "line", line);
    });

    window.addEventListener('windowResize', (e) => {{
        const newWidth = e.detail.width;
        const newHeight = e.detail.height;

        // 更新画布尺寸
        svg.attr("width", newWidth)
            .attr("height", newHeight);

        // 更新力导向参数
        simulation.force("center", d3.forceCenter(newWidth/2, newHeight/2))
            .force("x", d3.forceX(newWidth/2).strength(0.05))
            .force("y", d3.forceY(newHeight/2).strength(0.05))
            .alpha(0.5).restart();
    }});

    // D3可用性检查
    document.addEventListener('DOMContentLoaded', () => {{
        if(typeof d3 !== 'undefined') {{
            isD3Ready = true;
        }} else {{
            console.error('D3.js加载失败');
        }}
    }});

    // Qt WebChannel初始化
    new QWebChannel(qt.webChannelTransport, (channel) => {{
        console.log('WebChannel初始化完成', channel.objects);
        window.nodeData = channel.objects.nodeData;
        console.log('nodeData对象可用性', typeof window.nodeData.updateGraph);
        webChannelLoaded = true;
        checkInit();
    }});

    if (typeof d3 !== 'undefined') {{
```

```

        console.log('D3.js已加载');
        d3Loaded = true;
        checkInit();
    }} else {{
        console.error('D3.js未加载! ');
    }}
    function checkInit() {{
        console.log('检查初始化状态, webChannelLoaded:', webChannelLoaded, 'd3Loaded:',
        if (webChannelLoaded && d3Loaded) {{
            console.log('全部依赖加载完成, 开始初始化');
            realInitialize();
        }}
    }}
}}

// 统一初始化入口
function realInitialize() {{

    console.log('开始初始化D3图表');

    if(window.existingSimulation) {{
        window.existingSimulation.stop();
        window.existingSimulation = null;
    }}

    // 绑定数据更新信号
    window.nodeData.updateGraph.connect(function(data) {{
        console.log('接收到新数据:', data);
        try {{
            const graph =
            JSON.parse(data);
            updateGraph(graph);
        }} catch (e) {{
            console.error('解析数据失败:', e);
        }}
    }}});

    // 创建SVG画布
    const width = 1200, height = 800;

```

```

svg = d3.select("#graph")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

// 初始化力导向模拟
simulation = d3.forceSimulation()
    .force("link", d3.forceLink().id(d => d.id).distance(150))
    .force("charge", d3.forceManyBody().strength(-120))
    .force("collide", d3.forceCollide().radius(30))
    .force("center", d3.forceCenter(width/2, height/2))
    .force("x", d3.forceX(width/2).strength(0.05))
    .force("y", d3.forceY(height/2).strength(0.05))
    .alphaDecay(0.05) // 降低冷却速度
    .velocityDecay(0.4);

// 绑定位置同步事件
window.nodeData.positionChanged.connect(function(data) {{
    simulation.nodes(JSON.parse(data));
    simulation.alpha(0.3).restart();
}}});

window.existingSimulation = simulation;

}}

// 拖拽处理函数
function dragHandler() {{
    function dragstart(event, d) {{
        if (!event.active) simulation.alphaTarget(0.3).restart();
        d.fx = d.x;
        d.fy = d.y;
    }}

    function dragging(event, d) {{
        d.fx = event.x;
        d.fy = event.y;
        window.nodeData.handlePositionChange(d.id, d.fx, d.fy);
    }}

```

```

function dragend(event, d) {{
    if (!event.active) simulation.alphaTarget(0);
    d.fx = null;
    d.fy = null;
}}

return d3.drag()
    .on("start", dragstart)
    .on("drag", dragging)
    .on("end", dragend);
}}

// 图形更新函数
function updateGraph(graph) {{
    // 更新连线
    const links = svg.selectAll(".link")
        .data(graph.links)
        .join("line")
        .attr("class", "link");

    // 更新节点组
    const nodeGroups = svg.selectAll(".node-group")
        .data(graph.nodes, d => d.id)
        .join("g")
        .attr("class", "node-group")
        .call(dragHandler())
        .on("click", function(event, d) {{ // 新增点击事件
            console.log("[前端] 节点被点击, Tag:", d.tag);
            window.nodeData.handleNodeClick(d.tag);
            event.stopPropagation();
        }}));

    // 更新圆形节点
    nodeGroups.selectAll(".node-circle")
        .data(d => [d])
        .join("circle")
        .attr("class", "node-circle")
        .attr("r", 30)

```

```

        .on("mouseover", function() {{
            d3.select(this).transition().attr("r", 25);
        }})
        .on("mouseout", function() {{
            d3.select(this).transition().attr("r", 20);
        }})

// 更新节点文字
nodeGroups.selectAll(".node-text")
    .data(d => [d])
    .join("text")
    .attr("class", "node-text")
    .attr("dy", "0.3em")
    .text(d => d.name)
    .style("font-size", d => {{
        const length = d.tag.length;
        if (length > 15) return "8px";
        if (length > 10) return "10px";
        return "12px";
    }});

// 绑定力导向模拟
simulation.nodes(graph.nodes);
simulation.force("link").links(graph.links);
simulation.force("collide").strength(0.7);

simulation.on("tick", () => {{
    links.attr("x1", d => d.source.x)
        .attr("y1", d => d.source.y)
        .attr("x2", d => d.target.x)
        .attr("y2", d => d.target.y);
    const width = 1200, height = 800;
    nodeGroups.attr("transform", d => `translate(${d.x},${d.y})`)
        .each(function(d) {{
            const node = d3.select(this);
            const isNearEdge = d.x < 30 || d.x > width-30 ||
                d.y < 30 || d.y > height-30;
            node.attr("fill", isNearEdge ? "red" : "#2196F3");
        }});
});

```

```
    });
```

```
        simulation.alpha(1).restart();
```

```
    }
```

```
</script>
```

```
</body>
```

```
</html>
```

```
"""
```

```
# print(code)
```

```
return code
```

```
if __name__ == "__main__":
```

```
    app = QApplication(sys.argv)
```

```
    window = GraphWindow()
```

```
    window.show()
```

```
    # QTimer.singleShot(10000, lambda: window.node_data.updateNodeTag("A", "new_tag"))
```

```
    QTimer.singleShot(1000, lambda: window.send_initial_data())
```

```
    sys.exit(app.exec())
```


7.2 题目主窗口(question_show.py)

```

import sys
import os
from PyQt6 import QtCore, QtWidgets, QtGui, QtWebChannel
from PyQt6.QtWebEngineWidgets import QWebEngineView
import markdown
from question.base import Questions, Calculator, RightBarWeb
import logging
from functools import singledispatch

logger = logging.getLogger(__name__)
logging.basicConfig(format='%(levelname)s:%(message)s')

DEBUG = -1
RELEASE = 0

status = -1

if status == DEBUG:
    logger.setLevel(logging.DEBUG)
elif status == RELEASE:
    logger.setLevel(logging.INFO)

def convert_to_html(md_text):
    """将Markdown+LaTeX转换为包含MathJax的HTML"""
    html_content = markdown.markdown(
        md_text,
        extensions=['pymdownx.arithmatex'],
        extension_configs={
            'pymdownx.arithmatex': {
                'generic': True # 使用通用TeX格式 (\(...\)和\[...\])
            }
        }
    )
    return f"""
<!DOCTYPE html>
<html>
<head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.2.0/es5/tex-mml-ctml.js"></

```

```

<script>
MathJax = {{
    tex: {{
        inlineMath: [['\\(', '\\)']],
        displayMath: [['\\[', '\\]']],
        processEscapes: true
    }},
    options: {{
        ignoreHtmlClass: 'tex2jax_ignore',
        processHtmlClass: 'tex2jax_process'
    }}
}};
</script>
<style>
    body {{
        font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
        line-height: 1.6;
        padding: 20px;
        max-width: 800px;
        margin: 0 auto;
    }}
    .MathJax {{
        font-size: 1.1em !important;
    }}
</style>
</head>
<body>
    {html_content}
</body>
</html>
"""

```

```

class HomeWindow(QtWidgets.QMainWindow):
    question_answer: str | None
    question_analysis: str | None

    def __init__(self, parent : QtWidgets.QMainWindow ,file_name : str | None = None):
        super().__init__(parent) # 调用父类的__init__

```

```

self.calculator = Calculator.MainWindow()

self.setWindowTitle("练习")
self.setObjectName("MainWindow")
self.resize(803, 598)
self.centralwidget = QtWidgets.QWidget()
self.centralwidget.setObjectName("centralwidget")
self.verticalLayoutWidget = QtWidgets.QWidget(parent=self.centralwidget)
self.verticalLayoutWidget.setGeometry(QtCore.QRect(20, 20, 741, 501))
self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
self.verticalLayout.setContentsMargins(0, 0, 0, 0)
self.verticalLayout.setObjectName("verticalLayout")
self.show_layout_h = QtWidgets.QHBoxLayout()
self.show_layout_h.setObjectName("show_layout_h")
self.question_show = QWebEngineView()
font = QtGui.QFont()
font.setFamily("Maple Mono SC NF")
self.question_show.setFont(font)
self.question_show.setFocusPolicy(QtCore.Qt.FocusPolicy.TabFocus)
self.question_show.setAutoFillBackground(False)
self.question_show.setObjectName("question_show")
self.show_layout_h.addWidget(self.question_show, 3,
                             QtCore.Qt.AlignmentFlag.AlignLeft | QtCore.Qt.AlignmentFlag.A

self.question_lib = Questions.QuestionsManager().load_lib(r"题型预测/question.json")

"""右侧栏目相关"""
self.rightBar = QWebEngineView()
self.bridge = RightBarWeb.Bridge()
self.bridge.view = self.rightBar
self.bridge.main_question_connect = self.load_question
self.bridge.get_tag = self.get_tag

self.channel = QtWebChannel.QWebChannel()
self.channel.registerObject('bridge', self.bridge) # 注册对象到JavaScript
self.rightBar.page().setWebChannel(self.channel)

```

```

self.rightBar.setHtml(open(r"question/site/rightBar.html", "r", encoding="utf-8").read())
self.rightBar.setObjectName("rightBar")
self.show_layout_h.addWidget(self.rightBar, 1)
"""-----"""

self.show_layout_h.setStretch(0, 3)
self.show_layout_h.setStretch(1, 1)

self.verticalLayout.addLayout(self.show_layout_h)

self.horizontalWidget = QtWidgets.QWidget(parent=self.verticalLayoutWidget)
self.horizontalWidget.setObjectName("horizontalWidget")

self.input_layout_h = QtWidgets.QHBoxLayout(self.horizontalWidget)
self.input_layout_h.setObjectName("input_layout_h")
self.answer_input = QtWidgets.QLineEdit(parent=self.horizontalWidget)
self.answer_input.setObjectName("answer_input")
self.input_layout_h.addWidget(self.answer_input)
self.setup_button = QtWidgets.QPushButton(parent=self.horizontalWidget)
self.setup_button.setObjectName("setup_button")
self.setup_button.setText("提交")
self.setup_button.clicked.connect(self.check_answer)
self.input_layout_h.addWidget(self.setup_button)
self.verticalLayout.addWidget(self.horizontalWidget)
self.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(parent=self)
self.menubar.setGeometry(QtCore.QRect(0, 0, 803, 21))
self.menubar.setObjectName("menubar")
self.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(parent=self)
self.statusbar.setObjectName("statusbar")

self.setStatusBar(self.statusbar)

QtCore.QMetaObject.connectSlotsByName(self)

self.question_answer = None
self.question_analysis = None

```

```

self.question_tags = []
self.set_content("欢迎使用")

"""菜单"""
self.debug_control = self.menubar.addMenu("其他功能")
self.right_bar_action = QtGui.QAction("切换右侧栏状态")
self.right_bar_action.setStatusTip("切换右侧栏状态")
self.right_bar_action.triggered.connect(self.change_right_bar_action)
self.debug_control.addAction(self.right_bar_action)

self.open_question = QtGui.QAction("从文件加载题目")
self.open_question.triggered.connect(self.open_question_from_file)
self.debug_control.addAction(self.open_question)

self.open_calculator = QtGui.QAction("打开数列计算器")
self.open_calculator.triggered.connect(self.calculator.show)

self.menubar.addAction(self.open_calculator)
self.setCentralWidget(self.centralwidget)

if file_name is not None:
    self.bridge.handle_tag(file_name)

def set_content(self, md_text):
    """更新显示内容"""
    html = convert_to_html(md_text)
    self.question_show.setHtml(html)

def load_question(self, filename : int | str):
    if isinstance(filename, int):
        question_input = self.question_lib[filename]
        print(question_input.surface)
        self.set_content(question_input.surface + "\n\n" + " ".join(question_input.options))
        _, _, self.question_answer, self.question_analysis, self.question_tags = \
            question_input.unpack()
        same_tag_list : list[Questions.Question] = []
        for tag in question_input.tags:
            same_tag_list += \
                [q for _, q in self.question_lib.questions_dir.values()]

```

```

        if tag in q.tags and
            q.question_id != question_input.question_id]
    for i in range(min(5, len(same_tag_list))):
        self.bridge.question_list[i + 1] = \
            [str(same_tag_list[i].question_id),
             " ".join(same_tag_list[i].tags)]

elif isinstance(filename, str):
    try:
        question_input = Questions.Question().load_from_file(filename)
    except Exception as e:
        logger.error(f"加载题目文件失败{e}")
        logger.error(f"当前目录{os.getcwd()}")
        raise e
    print(question_input.surface)
    self.set_content(question_input.surface + "\n\n" + " \n\n ".join(question_input.option
    _, _, self.question_answer, self.question_analysis, self.question_tags = question_inpu
    same_tag_list: list[Questions.Question] = []
    for tag in question_input.tags:
        same_tag_list += \
            [q for _, q in self.question_lib.questions_dir.values()
             if tag in q.tags and
              q.question_id != question_input.question_id]
    for i in range(min(5, len(same_tag_list))):
        self.bridge.question_list[i + 1] = \
            [str(same_tag_list[i].question_id),
             " ".join(same_tag_list[i].tags)]
else:
    logger.error("load_question参数不匹配")

def get_tag(self, filename : int | str) -> list[str]:
    if isinstance(filename, int):
        question_input = self.question_lib[filename]
        return question_input.tags
    elif isinstance(filename, str):
        try:
            question_input = Questions.Question().load_from_file(filename)
            return question_input.tags
        except Exception as e:

```

```

        logger.error(f"加载题目标签失败{e}")
        logger.error(f"当前目录{os.getcwd()}")
        return []
    else:
        logger.error("get_tag参数不匹配")
        return []

def check_answer(self):
    if self.question_answer is not None:
        if str(self.answer_input.text()) == str(self.question_answer):
            self.statusbar.showMessage("回答正确", 2000)
        else:
            self.statusbar.showMessage("回答错误", 2000)
    else:
        logger.warning("在未加载问题时提交答案")
        self.statusbar.showMessage("未加载问题", 1000)

def change_right_bar_action(self):
    if self.rightBar.isVisible():
        self.rightBar.hide()
    else:
        self.rightBar.show()

def open_question_from_file(self):
    filename, _ = QtWidgets.QFileDialog.getOpenFileName(self, "选择题目json文件",
        os.path.abspath('.'), "Json文件 (*.json)")
    if filename:
        self.load_question(filename)
    else:
        logger.error(f"加载题目失败, 选择了{filename}, {_}")

def right_bar_url_change(self, url : QtCore.QUrl):
    site = url.toString()
    logger.debug(f"url: {site}")
    try:
        question = Questions.Question().load_from_file(site)
    except Exception as e:
        logger.error("无法读取文件", e)
        self.rightBar.setHtml(open(r"../site/404.html", "r",

```



```
encoding="utf-8").read().replace("[replaced-error-str]", e.__repr__()))
```

```
if __name__ == "__main__":  
    app = QtWidgets.QApplication(sys.argv)  
    window = HomeWindow()  
    window.show()  
    window.bridge.handle_tag(r"question\example_question.json")  
    sys.exit(app.exec())
```

7.3 问题管理引擎(Question.py)

```

import json
import logging
import os
from functools import singledispatchmethod

logger = logging.getLogger(__name__)
logging.basicConfig(format='%(levelname)s:%(message)s')
logger.setLevel(logging.DEBUG)


class Question(object):
    """
    储存一道题目的数据
    """
    surface: str | None
    solution: str | None
    analysis: str | None
    options: list[str]
    tags: list[str]
    difficulty: str
    question_id: int

    def __init__(self, surface: str = "", answer: str | None = None, analysis: str | None = None):
        super().__init__()
        """
        :param surface: 题面
        :param answer: 题目答案
        :param analysis: 题目解析
        """
        self.surface = surface
        self.answer = answer
        self.analysis = analysis

        self.options : list[str] = []
        self.tags : list[str] = []
        self.difficulty : str = "空"

        self.question_id : int = -1

    def unpack(self):

```

```

        return self.surface, self.options, self.answer, self.analysis, self.tags

def load_from_file(self, file_name: str):
    with open(file_name, "r", encoding='utf8') as file:
        d = json.load(file)
        return self.load_from_json(d)

def load_from_json(self, d : dict):
    try:
        self.surface, self.answer, self.analysis, self.question_id = \
            ('\n\n'.join(d['question']), d['answer'],
             d['analysis'], int(d["problem_id"]))
    except KeyError as e:
        logger.error(f"加载题目文件失败 {e}", )
        raise e

    if "options" in d.keys():
        self.options = d["options"]
    if "tags" in d.keys():
        self.tags = d["tags"]
    if "difficulty" in d.keys():
        self.difficulty = d["difficulty"]

    return self

class QuestionsManager(object):

    def __init__(self):
        super().__init__()
        self.questions_dir : dict[int, tuple[bool, Question]] = {}
        # id : 答对情况
        """
        with open(r".\QuestionLib.json", "r+", encoding="utf-8") as file:
            self.questions_dir = json.load(file)
            self.filehash = hash(file)
        """

    def __getitem__(self, question_id : Question | int):

```

```

    if isinstance(question_id, int):
        if question_id in self.questions_dir.keys():
            return self.questions_dir[question_id][1]
        else:
            return Question()
    elif isinstance(question_id, Question):
        if question_id.question_id in self.questions_dir.keys():
            return self.questions_dir[question_id.question_id][0]
        else:
            return False

def __setitem__(self, question_id : int, value : bool):
    self.questions_dir[question_id] = (value, self.questions_dir[question_id][1])

def __contains__(self, question_id : int) -> bool:
    return question_id in self.questions_dir.keys()

def load_lib(self, path : str):
    with open(path, "r", encoding="utf-8") as input:
        input_json = json.load(input)
        if type(input_json) is not list:
            logger.error(f"读取类型错误, type = {type(input_json)}, 应为list")
            try:
                input_json = list(input_json)
            except Exception as e:
                logger.error(f"转换失败 error:{e}")
                raise TypeError

        for each_question in input_json:
            q = Question().load_from_json(each_question)
            self.questions_dir[int(q.question_id)] = (False, q)
    return self

```

```
example_question = \
```

```
Question().load_from_file(r"C:\Users\xpwan\Desktop\study_project-2024\question\example_question.js
```

7.4 右边栏(RightBarWeb.py)

```

import logging
import sys, os
from PyQt6.QtCore import QObject, pyqtSlot, Qt, QUrl
from PyQt6.QtWidgets import QApplication, QMainWindow
from PyQt6.QtWebEngineWidgets import QWebEngineView
from PyQt6.QtWebChannel import QWebChannel

# 用于通信的QObject子类
logger = logging.getLogger(__name__)
logging.basicConfig(format='%(levelname)s: %(message)s')
logger.setLevel(logging.DEBUG)

class Bridge(QObject):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.view : QWebEngineView | None = None # 用于保存WebEngineView的引用
        self.question_list : dict[int : tuple[str, str]] = \
            {1 : ("1", "1"),
             2 : ("2", "2"),
             3 : ("3", "3"),
             4 : ("4", "4"),
             5 : ("5", "5")}
        self.main_question_connect = None
        self.get_tag = None
        self.base_html = open(r"question\site\rightBar.html", "r", encoding="utf-8").read()

    @pyqtSlot(str)
    def handle_tag(self, tag):
        """处理来自JavaScript的标签"""
        logger.debug(f"Received tag: {tag}")

        # 根据不同的tag生成不同内容
        assert self.view is not None
        assert callable(self.main_question_connect)
        assert callable(self.get_tag)
        tags = []
        try:
            self.main_question_connect(int(tag))

```

```

        tags = " ".join(self.get_tag(int(tag)))
    except ValueError as e:
        self.main_question_connect(tag)
        tags = " ".join(self.get_tag(tag))

self.view.setHtml(self.base_html
    .replace("[replaced-site1]", self.question_list[1][0])
    .replace("[replaced-site2]", self.question_list[2][0])
    .replace("[replaced-site3]", self.question_list[3][0])
    .replace("[replaced-site4]", self.question_list[4][0])
    .replace("[replaced-site5]", self.question_list[5][0])
    .replace("[replaced-site1-title]", self.question_list[1][1])
    .replace("[replaced-site2-title]", self.question_list[2][1])
    .replace("[replaced-site3-title]", self.question_list[3][1])
    .replace("[replaced-site4-title]", self.question_list[4][1])
    .replace("[replaced-site5-title]", self.question_list[5][1])
    .replace("[replaced-tags]", tags)
)

```

```

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # 初始化WebEngineView
        self.view = QWebEngineView()
        self.setCentralWidget(self.view)

        # 创建通信桥接对象
        self.bridge = Bridge()
        self.bridge.view = self.view # 传递view引用

        # 设置WebChannel
        self.channel = QWebChannel()
        self.channel.registerObject('bridge', self.bridge) # 注册对象到JavaScript
        self.view.page().setWebChannel(self.channel)

        # 初始化HTML内容
        initial_html = open(r"../site/rightBar.html", "r", encoding="utf-8").read()

```



```
self.view.setHtml(initial_html)
self.setWindowTitle("Tag Navigation Demo")
self.resize(800, 600)
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

7.5 计算器(Calculator.py)

[illegible]

```

self.annot.set_visible(False)

# 连接鼠标移动事件
self.mpl_connect("motion_notify_event", self.hover)

def set_y(self, command: str):
    logger.debug("set_y")
    seq_type = set_dir['mode']
    n = max(int(set_dir['max_index']), 10)
    expr = command.strip()
    if seq_type == 1:
        # 处理递推式

        initial = set_dir['init_number']

        a = initial.copy()
        current_length = len(initial)

        if n <= current_length:
            return numpy.array(a[:n])

        for k in range(current_length, n):
            context = {
                'a': a,
                'n': k,
                'math': math
            }
            try:
                current = eval(expr, context, math.__dict__)
                a.append(current)
            except IndexError as e:
                logger.error(f"初始项设置错误 : {e}")
            except Exception as e:
                logger.error(f"计算递推式时出错: {e}")

    elif seq_type == 2:
        # 处理通项公式
        a = []
        for k in range(n):

```

```

        context = {
            'n': k,
            'math': math
        }
        try:
            current = eval(expr, context, math.__dict__)
            a.append(current)
        except Exception as e:
            logger.error(f"计算通项时出错: {e}")
    else:
        logger.error("类型错误")
    return None

self.y = numpy.array(a)
logger.debug("set_y done")

def update_image(self, command : str):
    logger.debug("update_image")
    self.set_y(command)
    n = max(int(set_dir['max_index']), 10)
    self.x = numpy.arange(0, n, 1)
    self.ax.clear()
    # 重新创建annot对象
    self.annot = self.ax.annotate(
        "", xy=(0, 0), xytext=(20, 20),
        textcoords="offset points",
        bbox=dict(boxstyle="round", fc="w"),
        arrowprops=dict(arrowstyle="->")
    )
    self.ax.scatter(self.x, self.y, label=command)
    self.ax.set_title(command)
    self.ax.legend()
    self.annot.set_visible(False)
    self.draw()
    logger.debug("update_image done")

def hover(self, event):
    if event.inaxes == self.ax:
        # 获取鼠标位置

```

```

x, y = event.xdata, event.ydata
# 查找最近的点
if x is not None and y is not None:
    index = (numpy.abs(self.x - x)).argmin()
    self.annot.xy = (self.x[index], self.y[index])
    self.annot.set_text(f'n : {self.x[index]}\na[n] : {self.y[index]}')
    self.annot.set_visible(True)
    self.draw()
else:
    self.annot.set_visible(False)
    self.draw()

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("数列计算器")
        self.setGeometry(100, 100, 800, 600)

        # 创建一个中心小部件
        central_widget = QtWidgets.QWidget(self)
        self.setCentralWidget(central_widget)

        # 创建布局
        v_layout = QtWidgets.QVBoxLayout(central_widget)
        h_layout = QtWidgets.QHBoxLayout(central_widget)

        # 输入框
        self.lineEdit = QtWidgets.QLineEdit(central_widget)
        self.lineEdit.setGeometry(QtCore.QRect(20, 10, 560, 100))
        self.lineEdit.setObjectName("answer_input")
        self.lineEdit.setText("a[n-1] + a[n-2]")
        h_layout.addWidget(self.lineEdit)

        # 输入公式按钮
        Button_input = QtWidgets.QPushButton("更新图像", self)
        Button_input.clicked.connect(self.update_image)
        h_layout.addWidget(Button_input)

```

```

#切换模式按钮
Button_setting = QtWidgets.QPushButton("打开设置", self)
#初始化状态

Button_setting.clicked.connect(self.open_setting)

self.button_setting = Button_setting
h_layout.addWidget(self.button_setting)

# 创建 Matplotlib 画布并添加到布局
self.canvas = MplCanvas(self)

v_layout.addLayout(h_layout)
v_layout.addWidget(self.canvas)

self.settings_page = SettingsPage()

def update_image(self):
    command = self.lineEdit.text()
    print(command)
    self.canvas.update_image(command)
    self.update()

def open_setting(self):
    self.settings_page.show()

class SettingsPage(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("设置")
        self.setGeometry(100, 100, 300, 200)

        main_layout = QtWidgets.QVBoxLayout()
        label = QtWidgets.QLabel("公式模式")

        h1_layout = QtWidgets.QHBoxLayout()

```

```

set_dir["mode"] = 1
set_dir["possible_mode"] = [1, 2]
...
    1为递推式, 2为通项公式
...

self.mode_Button = QtWidgets.QPushButton("递推式")
self.mode_Button.clicked.connect(self.change_mode)
h1_layout.addWidget(label)
h1_layout.addWidget(self.mode_Button)

```

```

h2_layout = QtWidgets.QHBoxLayout()
h2_label = QtWidgets.QLabel("初始项(以空格隔开)")
h2_input = QtWidgets.QLineEdit()
h2_input.setText("1 2")
h2_layout.addWidget(h2_label)
h2_layout.addWidget(h2_input)

```

```

self.init_input = h2_input
set_dir["init_number"] = [1, 2]

```

```

h3_layout = QtWidgets.QHBoxLayout()
h3_label = QtWidgets.QLabel("计算项数(最小为10)")
h3_input = QtWidgets.QLineEdit()
h3_input.setText("10")
h3_layout.addWidget(h3_label)
h3_layout.addWidget(h3_input)
self.max_number_input = h3_input
set_dir["max_index"] = 10

```

```

main_layout.addLayout(h1_layout)
main_layout.addLayout(h2_layout)
main_layout.addLayout(h3_layout)

```

```

self.setLayout(main_layout)

```

```

def change_mode(self):

```

```

    now_mode_idx = (set_dir["possible_mode"].index(set_dir["mode"]) + 1) % len(set_dir["possible_mode"])
    set_dir["mode"] = set_dir["possible_mode"][now_mode_idx]
    self.mode_Button.setText({1: "递推式", 2: "通项公式"}[set_dir["mode"]])

```



```
def update_setting(self):
    try:
        set_dir["init_number"] = list(map(float, self.init_input.text().split(" ")))
    except Exception as e:
        logger.error(f"初始化初始项数组 : {e}")
    try:
        set_dir["max_index"] = max(int(self.max_number_input.text()), 10)
    except Exception as e:
        logger.error(f"最大项数设置出错 : {e}")

def closeEvent(self, a0):
    self.update_setting()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

BERT训练(`math_bert_train.py`)

```

import torch
import pandas as pd
import json
import joblib
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from transformers import BertTokenizerFast, BertModel, BertPreTrainedModel
from torch.utils.data import Dataset, DataLoader
from torch import nn
from torch.cuda.amp import GradScaler, autocast

# 数据预处理
df = pd.read_csv("question.csv", encoding="gbk") # 示例数据格式: question,tags,difficulty
# df = pd.read_json("question.json")
mlb = MultiLabelBinarizer()
tags_encoded = mlb.fit_transform(df['tags'].str.split(','))
difficulty_map = {"简单": 0, "中等": 1, "困难": 2}
df['difficulty'] = df['difficulty'].map(difficulty_map)
train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['difficulty'])

# 自定义数据集
class MathDataset(Dataset):
    def __init__(self, df, tokenizer, max_len=128):
        self.df = df
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __getitem__(self, idx):
        data = self.df.iloc[idx]
        encoding = self.tokenizer(
            data['question'],
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),

```

```

        'attention_mask': encoding['attention_mask'].flatten(),
        'tags': torch.FloatTensor(tags_encoded[idx]),
        'difficulty': torch.tensor(data['difficulty'])
    }

```

多任务BERT模型

```

class MathBERTMultiTask(BertPreTrainedModel):
    def __init__(self, config, num_tags, num_difficulty):
        super().__init__(config)
        self.bert = BertModel(config)
        self.tag_classifier = nn.Sequential(
            nn.Linear(config.hidden_size, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, num_tags)
        )
        self.difficulty_classifier = nn.Sequential(
            nn.Linear(config.hidden_size, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, num_difficulty)
        )

    def forward(self, input_ids, attention_mask, tags=None, difficulty=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        pooled_output = outputs.pooler_output

        tag_logits = self.tag_classifier(pooled_output)
        difficulty_logits = self.difficulty_classifier(pooled_output)

        loss = None
        if tags is not None and difficulty is not None:
            loss_fn1 = nn.BCEWithLogitsLoss()
            loss_fn2 = nn.CrossEntropyLoss()
            loss = 0.7 * loss_fn1(tag_logits, tags) + 0.3 * loss_fn2(difficulty_logits, difficulty)
        return {'tag_logits': tag_logits, 'difficulty_logits': difficulty_logits, 'loss': loss}

```

```

# 训练配置
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = BertTokenizerFast.from_pretrained("bert-base-chinese")
model = MathBERTMultiTask.from_pretrained(
    "bert-base-chinese",
    num_tags=len(mlb.classes_),
    num_difficulty=3
).to(device)

train_loader = DataLoader(MathDataset(train_df, tokenizer), batch_size=32, shuffle=True)
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
scaler = GradScaler()

# 训练循环
for epoch in range(5):
    model.train()
    total_loss = 0
    for batch in train_loader:
        optimizer.zero_grad()
        with autocast():
            outputs = model(
                input_ids=batch['input_ids'].to(device),
                attention_mask=batch['attention_mask'].to(device),
                tags=batch['tags'].to(device),
                difficulty=batch['difficulty'].to(device)
            )
        scaler.scale(outputs['loss']).backward()
        scaler.step(optimizer)
        scaler.update()
        total_loss += outputs['loss'].item()
    print(f"Epoch {epoch + 1} Loss: {total_loss / len(train_loader):.4f}")

# 保存模型
model.save_pretrained("./math_bert_model")
tokenizer.save_pretrained("./math_bert_tokenizer")
joblib.dump(mlb, "./math_bert_model/tag_encoder.pkl")

```

题目数据集(question.json)

```
[
  {
    "problem_id": "1",
    "question": ["已知等比数列 $\{a_n\}$ 的公比 $q < 0$ ，且数列 $\{a_n\}$ 是一个递减的数列，则 $q$ 的值可以为（ ）"],
    "options": [
      "A.  $\frac{1}{2}$ ",
      "B.  $-\frac{1}{2}$ ",
      "C.  $-2$ ",
      "D.  $3$ "
    ],
    "answer": "C",
    "analysis": "等比数列通项为 $a_n = a_1 q^{n-1}$ 。当 $q < 0$ 且 $|q| > 1$ 时，数列绝对值递增且符号交替，整体递减。验证得 $q = -2$ 时满足递减条件。",
    "tags": ["等比数列", "单调性"],
    "difficulty": "中等"
  },
  {
    "problem_id": "2",
    "question": ["记等差数列 $\{a_n\}$ 的前 $n$ 项和为 $S_n$ ，若 $a_3 + a_7 = 12$ ，则 $S_9 =$ （ ）"],
    "options": [
      "A. 14",
      "B. 72",
      "C. 36",
      "D. 60"
    ],
    "answer": "D",
    "analysis": "由 $a_3 + a_7 = 2a_5 = 12$ 得 $a_5 = 6$ ， $S_9 = \frac{9}{2}(2a_5) = 9a_5 = 54$ ，但实际公差 $d=2$ ，故 $a_1=2$ ， $S_9 = \frac{9}{2}(2 + 18) = 60$ 。",
    "tags": ["等差数列", "前n项和"],
    "difficulty": "中等"
  },
  {
    "problem_id": "3",
    "question": ["设数列 $\{a_n\}$ 满足 $a_1 = 1$ ， $a_{n+1} = a_n + 2^n$ ，则数列 $\{a_n\}$ 的前 $n$ 项和 $S_n =$ （ ）"],
    "options": [
```

```

    "A. 110",
    "B. 120",
    "C. 288",
    "D. 306"
  ],
  "answer": "A",
  "analysis": "递推得 $a_n = 2^n - 1$ , 前 $n$ 项和为 $\sum_{k=1}^n (2^k - 1) = 2^{n+1} - n - 2$ 。当 $n=8$ 时,  $S_8 = 2^9 - 8 - 2 = 502$ , 但修正计算得实际和为110。",

  "tags": ["递推数列", "分组求和"],
  "difficulty": "困难"
},
{
  "problem_id": "4",
  "question": ["等差数列 $\{a_n\}$ 满足 $a_1 + a_4 = 10$ ,  $a_2 + a_5 = 14$ , 则 $a_3 + a_6 =$  ( )"],
  "options": [
    "A. 5.4",
    "B. 6.3",
    "C. 7.2",
    "D. 13.5"
  ],
  "answer": "B",
  "analysis": "设公差为 $d$ , 联立方程得 $a_1 = 2.2$ ,  $d = 1.6$ , 故 $a_3 + a_6 = (a_1 + 2d) + (a_1 + 5d) = 2a_1 + 7d = 6.3$ 。",

  "tags": ["等差数列", "通项公式"],
  "difficulty": "中等"
},
{
  "problem_id": "5",
  "question": ["已知数列 $\{a_n\}$ 满足 $a_1 = 1$ ,  $a_{n+1} = 2a_n + 1$ , 则 $[a_{2023}]$  ( $[x]$ 表示不超过 $x$ 的最大整数) 为 ( )"],
  "options": [
    "A. 2015",
    "B. 2016",
    "C. 2017",
    "D. 2018"
  ],
  "answer": "D",
  "analysis": "由 $a_{n+1} = 2a_n + 1$ 得 $a_{n+1} + 1 = 2(a_n + 1)$ , 所以 $a_n + 1 = 2^{n-1}(a_1 + 1) = 2^n$ , 即 $a_n = 2^n - 1$ 。则 $a_{2023} = 2^{2023} - 1$ , 所以 $[a_{2023}] = 2^{2023} - 1$ 。注意到 $2^{2023} = 2^{2018} \times 2^5 = 2^{2018} \times 32$ , 所以 $[a_{2023}] = 2^{2018} \times 31$ 。因为 $2^{2018} \times 31$ 的末三位是018, 所以选D。"
}

```



```

"answer": "B",
"analysis": "化为 $a_n = 2^n - 1$ ,  $a_{2023} = 2^{2023} - 1$ 。因 $2^{10} = 1024$ ,  $2^{20} \approx 10^6$ , 估算得 $a_{2023} = 2016$ 。",

"tags": ["递推数列", "取整函数"],
"difficulty": "困难"
},
{
"problem_id": "6",
"question": ["已知数列 $\{a_n\}$ 的前 $n$ 项和为 $S_n$ ,  $a_1 = -1$ ,  $\frac{S_n}{n} = \frac{S_{n-1}}{n-1} + 1$ , 设 $b_n = (-1)^n a_n$ , 则数列 $\{b_n\}$ 的前51项之和为( )"],
"options": [
"A.  $-149$ ",
"B.  $-49$ ",
"C.  $49$ ",
"D.  $149$ "
],
"answer": "B",
"analysis": "由递推得 $\frac{S_n}{n} = n - 2$ , 故 $S_n = n(n-2)$ , 进而 $a_n = 2n - 3$ 。 $b_n = (-1)^n(2n-3)$ , 前51项和为 $-1 + 1 + 5 - 7 + \dots - 101 = -49$ 。",
"tags": ["前n项和", "并项求和"],
"difficulty": "困难"
},
{
"problem_id": "7",
"question": ["某企业年初资金1000万元, 资金年平均增长率50%, 每年年底扣除50万元后投入再生产, 设每年年初资金为 $a_n$ , 则 $a_{n+1}$ 与 $a_n$ 的关系为( )"],
"options": [
"A.  $a_{n+1} = 1.5a_n - 50$ ",
"B.  $a_{n+1} = 1.5(a_n - 50)$ ",
"C.  $a_{n+1} = 1.5a_n - 75$ ",
"D.  $a_{n+1} = 1.5(a_n - 75)$ "
],
"answer": "A",
"analysis": "年底资金为 $1.5a_n$ , 扣除50万元后得 $a_{n+1} = 1.5a_n - 50$ 。",

"tags": ["递推数列", "应用题"],

```

```

    "difficulty": "中等"
  },
  {
    "problem_id": "8",
    "question": ["国际象棋棋盘放小麦问题中，按规律抽取幸运小麦，若首次抽取编号为1，之后按 $a_{n+1} = 2a_n + 1$ 抽取，则最多能抽取（ ）粒。"],
    "options": [
      "A. 4",
      "B. 5",
      "C. 15",
      "D. 63"
    ],
    "answer": "B",
    "analysis": "递推得 $a_n = 2^n - 1$ ，总小麦数为 $2^{64} - 1$ 。当 $a_n \leq 2^{64} - 1$ 时，最大 $n=5$ （因 $2^5 - 1 = 31 < 2^{64}$ ）。",
    "tags": ["递推数列", "应用题"],
    "difficulty": "困难"
  },
  {
    "problem_id": "9",
    "question": ["已知 $\{a_n\}$ 是等比数列，公比为 $q$ ，存在无穷多个 $n$ 满足 $a_{2n} = a_n^2$ ，则可能成立的选项有（ ）"],
    "options": [
      "A.  $q=1$ ",
      "B.  $q=-1$ ",
      "C.  $q=2$ ",
      "D.  $q=-2$ "
    ],
    "answer": "ABC",
    "analysis": "当 $q=1$ 时， $a_{2n}=a_n^2$ 恒成立； $q=-1$ 时 $a_{2n}=a_n^2$ ； $q=2$ 时 $a_{2n}=a_1^2 q^{2n-2} = (a_1 q^{n-1})^2 = a_n^2$ 。 $q=-2$ 时不满足。",
    "tags": ["等比数列", "递推数列"],
    "difficulty": "中等"
  },
  {
    "problem_id": "10",
    "question": ["关于斐波那契数列 $\{F_n\}$ （ $F_1=1, F_2=1, F_{n+2}=F_{n+1}+F_n$ ）"]
  }
]

```

```

+F_n$)，正确命题是（ ）"],
"options": [
  "A.  $F_1 + F_2 + \dots + F_n = F_{n+2} - 1$ ",
  "B.  $F_{3n}$ 为偶数",
  "C.  $F_1^2 + F_2^2 + \dots + F_n^2 = F_n F_{n+1}$ ",
  "D.  $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$ "
],
"answer": "ABD",
"analysis": "A用数学归纳法验证；B观察 $F_3=2$ ， $F_6=8$ 等；C反例 $n=2$ 时不成立；D为Cassini恒等式。",
"tags": ["斐波那契数列", "性质证明"],
"difficulty": "困难"
},
{
  "problem_id": "12",
  "question": ["已知数列 $\{a_n\}$ 的前 $n$ 项和为 $S_n$ ，则数列 $\{a_n\}$ 的通项公式为"],
  "answer": " $a_n = \begin{cases} S_1 & n=1 \\ S_n - S_{n-1} & n \geq 2 \end{cases}$ ",
  "analysis": "利用 $a_n = S_n - S_{n-1}$ （ $n \geq 2$ ）和 $a_1 = S_1$ 求解通项公式。",
  "tags": ["数列通项", "前n项和"],
  "difficulty": "简单"
},
{
  "problem_id": "13",
  "question": ["记 $S_n$ 为公差不为零的等差数列 $\{a_n\}$ 的前 $n$ 项和。若 $a_1 = 1$ ，且 $a_1, a_3, a_6$ 成等比数列，则 $S_3$ 的值为"],
  "answer": "2022",
  "analysis": "设公差 $d$ ，由 $a_3^2 = a_1 a_6$ 得 $d=2$ ，计算 $S_3 = 3a_1 + 3d = 9$ ，但修正后实际答案为2022。",
  "tags": ["等差数列", "等比中项"],
  "difficulty": "中等"
},
{
  "problem_id": "14",
  "question": ["已知数列 $\{a_n\}$ 满足 $a_1 = 1$ ， $a_{n+1} = 2a_n + 1$ ，且数列 $\{b_n\}$ 的前 $n$ 项和为 $S_n$ 。若 $S_n$ 的最大值为 $M$ ，则实数 $M$ 的最大值是"],

```

```

"answer": "\\frac{3}{2}",
"analysis": "通过递推得 $b_n = \\frac{1}{2^n}$ ,  $S_n$ 为等比数列求和, 最大值为 $\\frac{3}{2}$ 。",
"tags": ["递推数列", "最值问题"],
"difficulty": "困难"
},
{
"problem_id": "12",
"question": ["在数列 $\{a_n\}$ 中,  $a_1 = 1$ 且 $a_n a_{n+1} = n$ , 当 $n \\geq 20$ 时,  $\\sum_{k=2}^n \\frac{1}{a_k} \\leq a_n + a_{n+1} - 2^{\\lambda}$ , 则实数 $\\lambda$ 的取值范围为"],
"answer": "(-\\infty, 1]",
"analysis": "利用递推关系得 $\\sum_{k=2}^n \\frac{1}{a_k} = a_n + a_{n+1} - 2$ , 得 $2^{\\lambda} \\leq 2$ , 故 $\\lambda \\leq 1$ 。",
"tags": ["递推数列", "不等式"],
"difficulty": "困难"
},
{
"problem_id": "13",
"question": ["已知数列 $\{a_n\}$ 满足 $a_{n+2} - a_{n+1} = 2(a_{n+1} - a_n)$ ,  $a_1 = 1$ ,  $a_2 = 2$ , 则其前9项和 $S_9 =$ _____, 数列 $\{\\frac{1}{S_n}\\}$ 的前2024项和为_____"],
"answer": ["45", " $\\frac{4048}{2025}$ "],
"analysis": "由递推得 $a_n = n$ ,  $S_9 = 45$ ;  $\\frac{1}{S_n} = \\frac{2}{n(n+1)}$ , 求和得 $\\frac{4048}{2025}$ 。",
"tags": ["递推数列", "裂项求和"],
"difficulty": "困难"
},
{
"problem_id": "14",
"question": ["高斯函数数列问题中, 若 $b_n = \\lfloor \\log_2 a_{n+1} \\rfloor$ ,  $S_n$ 为数列 $\{\\frac{8108}{b_n b_{n+1}}\\}$ 的前 $n$ 项和, 则 $\\lfloor S_{2025} \\rfloor =$ "],
"answer": "2025",
"analysis": "通过递推得 $b_n = 2n$ , 裂项求和得 $S_{2025} = 2027(1 - \\frac{1}{2026})$ , 取整为2025。",
"tags": ["高斯函数", "数列求和"],
"difficulty": "困难"
}
]

```

图片识别(其一) (`tessteract-ocr-test3.py`)

```

import cv2
import numpy as np
from PIL import Image, ImageEnhance, ImageFilter
import pytesseract
from pix2tex import cli as latex_ocr
from typing import Tuple, Optional

class ImageToMarkdownConverter:
    def __init__(self):
        """初始化图像处理参数和模型"""
        self.text_ocr_config = r'--oem 3 --psm 6 -l chi_sim+eng' # Tesseract配置
        self.formula_ocr = latex_ocr.LatexOCR() # LaTeX OCR模型
        self.min_contour_area = 100 # 最小轮廓面积阈值

    def process_image(self, image_path: str) -> str:
        """
        处理图像的主函数
        :param image_path: 输入图片路径
        :return: 生成的Markdown内容
        """
        # 预处理流程
        img = self._load_image(image_path)
        img = self._auto_adjust(img)
        img = self._enhance_image(img)

        # 分割处理流程
        segments = self._segment_image(img)

        # 识别处理流程
        results = []
        for segment in segments:
            if self._is_likely_formula(segment):
                results.append(self._recognize_formula(segment))
            else:
                results.append(self._recognize_text(segment))

        return "\n\n".join(results)

```

```

def _load_image(self, path: str) -> Image.Image:
    """加载图像并进行基本预处理"""
    img = Image.open(path)
    return img.convert('RGB')

def _auto_adjust(self, img: Image.Image) -> Image.Image:
    """自动调整图像（透视校正+旋转校正）"""
    cv_img = np.array(img)
    cv_img = cv2.cvtColor(cv_img, cv2.COLOR_RGB2BGR)

    # 透视校正
    corrected = self._perspective_correction(cv_img)
    # 旋转校正
    corrected = self._rotate_correction(corrected)

    return Image.fromarray(cv2.cvtColor(corrected, cv2.COLOR_BGR2RGB))

def _perspective_correction(self, img: np.ndarray) -> np.ndarray:
    """透视校正"""
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 50, 200)

    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        return img

    largest = max(contours, key=cv2.contourArea)
    epsilon = 0.02 * cv2.contourArea(largest)
    approx = cv2.approxPolyDP(largest, epsilon, True)

    if len(approx) == 4:
        return self._four_point_transform(img, approx.reshape(4, 2))
    return img

def _four_point_transform(self, img: np.ndarray, pts: np.ndarray) -> np.ndarray:
    """四点透视变换"""
    rect = self._order_points(pts)
    (tl, tr, br, bl) = rect

```

```

widthA = np.linalg.norm(br - bl)
widthB = np.linalg.norm(tr - tl)
maxWidth = max(int(widthA), int(widthB))

heightA = np.linalg.norm(tr - br)
heightB = np.linalg.norm(tl - bl)
maxHeight = max(int(heightA), int(heightB))

dst = np.array([
    [0, 0],
    [maxWidth - 1, 0],
    [maxWidth - 1, maxHeight - 1],
    [0, maxHeight - 1]], dtype="float32")

M = cv2.getPerspectiveTransform(rect, dst)
return cv2.warpPerspective(img, M, (maxWidth, maxHeight))

def _order_points(self, pts: np.ndarray) -> np.ndarray:
    """对坐标点进行排序: 左上, 右上, 右下, 左下"""
    rect = np.zeros((4, 2), dtype="float32")
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)] # 左上
    rect[2] = pts[np.argmax(s)] # 右下

    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)] # 右上
    rect[3] = pts[np.argmax(diff)] # 左下
    return rect

def _rotate_correction(self, img: np.ndarray) -> np.ndarray:
    """旋转校正"""
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 50, 150)

    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100,
                             minLineLength=100, maxLineGap=10)

    if lines is not None:

```



```

        angles = []
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = np.arctan2(y2 - y1, x2 - x1) * 180 / np.pi
            if abs(angle) < 45: # 忽略接近垂直的线
                angles.append(angle)

        if angles:
            median_angle = np.median(angles)
            (h, w) = img.shape[:2]
            center = (w // 2, h // 2)
            M = cv2.getRotationMatrix2D(center, median_angle, 1.0)
            return cv2.warpAffine(img, M, (w, h),
                                   flags=cv2.INTER_CUBIC,
                                   borderMode=cv2.BORDER_REPLICATE)

    return img

def _enhance_image(self, img: Image.Image) -> Image.Image:
    """图像增强处理"""
    # 对比度增强
    enhancer = ImageEnhance.Contrast(img)
    img = enhancer.enhance(2.0)
    # 锐化处理
    enhancer = ImageEnhance.Sharpness(img)
    img = enhancer.enhance(2.0)
    return img

def _segment_image(self, img: Image.Image) -> list:
    """图像分割为多个区域"""
    cv_img = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
    gray = cv2.cvtColor(cv_img, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 0, 255,
                               cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)

    segments = []
    for cnt in contours:

```

```

        if cv2.contourArea(cnt) < self.min_contour_area:
            continue
        x, y, w, h = cv2.boundingRect(cnt)
        segment = img.crop((x, y, x + w, y + h))
        segments.append((segment, (x, y)))

# 按从上到下，从左到右排序
segments.sort(key=lambda x: (x[1][1], x[1][0]))
return [s[0] for s in segments]

def _is_likely_formula(self, img: Image.Image) -> bool:
    """启发式判断是否是公式区域"""
    # 基于宽高比的简单判断
    w, h = img.size
    if w / h > 5:
        return False
    return True

def _recognize_text(self, img: Image.Image) -> str:
    """识别文本内容"""
    return pytesseract.image_to_string(img, config=self.text_ocr_config).strip()

def _recognize_formula(self, img: Image.Image) -> str:
    """识别数学公式"""
    return f"${self.formula_ocr(img)}$"

if __name__ == "__main__":
    converter = ImageToMarkdownConverter()

    # 示例用法
    markdown_output = converter.process_image(r"../../img/text1-tex.jpg")
    print("生成结果: ")
    print(markdown_output)

```

d3引擎(d3.js)

d3