Table of Contents

Introduction 1.1

第一章 工具 1.2

- 第一节 Gitbook的安装与使用 1.2.1
- 第二节 1.2.2

第二章 代码 1.3

- 第一节 C++ 1.3.1
- 第二节 C# 1.3.2

第三章 日志 1.4

- 第一节 任务 1.4.1
- 第二节 问题 1.4.2
- 第三节 需要做的事情 1.4.3

第四章 XXX 1.5

- 第一节 比特币代码分析 1.5.1
  - 第一段 VS2017下调试Bitcoincode 1.5.1.1
- 第二节 密码学笔记 1.5.2
- 第三节 比特币下的区块链 1.5.3

第五章 算法 1.6

- 第一节 插值 1.6.1

结束 1.7

My Awesome Book

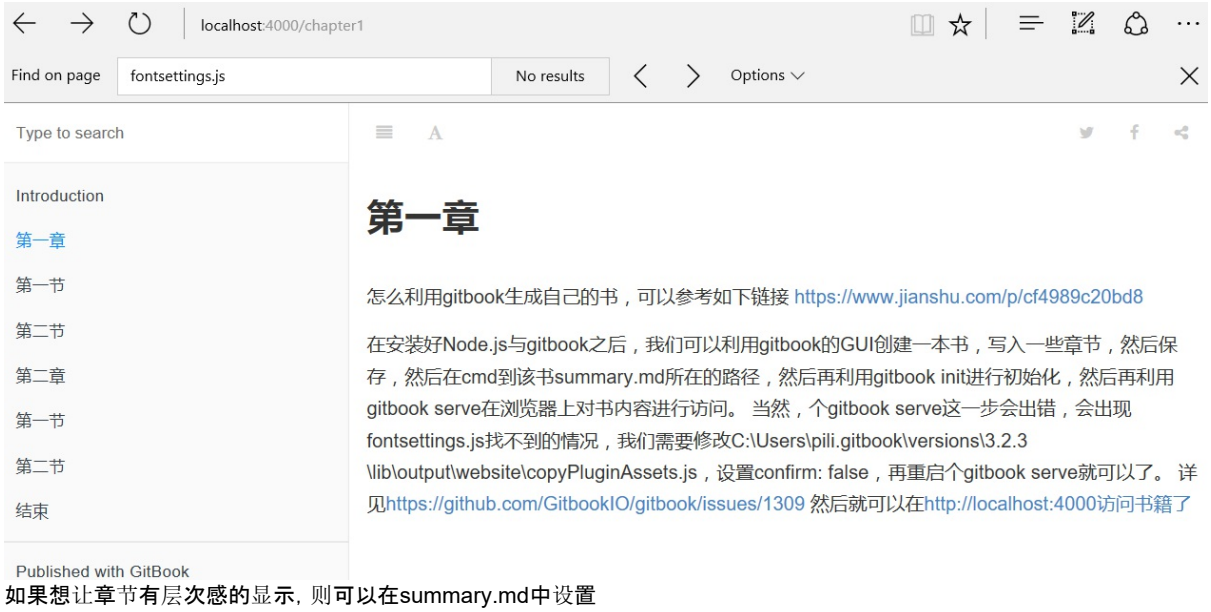
日知录是为效法顾炎武的《日知录》而做，里面会记录自己在学习与工作中学到的东西，该书的目的是为了打造一本属于自己的百科全书，也是自己思想体系的体现。最终里面的每一章可能代表一个方向与学科，比如C++，C#最终会成为一章，经济，中国史会成为一章。如果经济学里面记录的内容不多，就编成一章，如果内容多了，则会编成多章，比如宏观经济学，微观经济学，计量经济学。这样，最终一门大的学科，比如，经济学，就编成了一卷。

第一章:工具

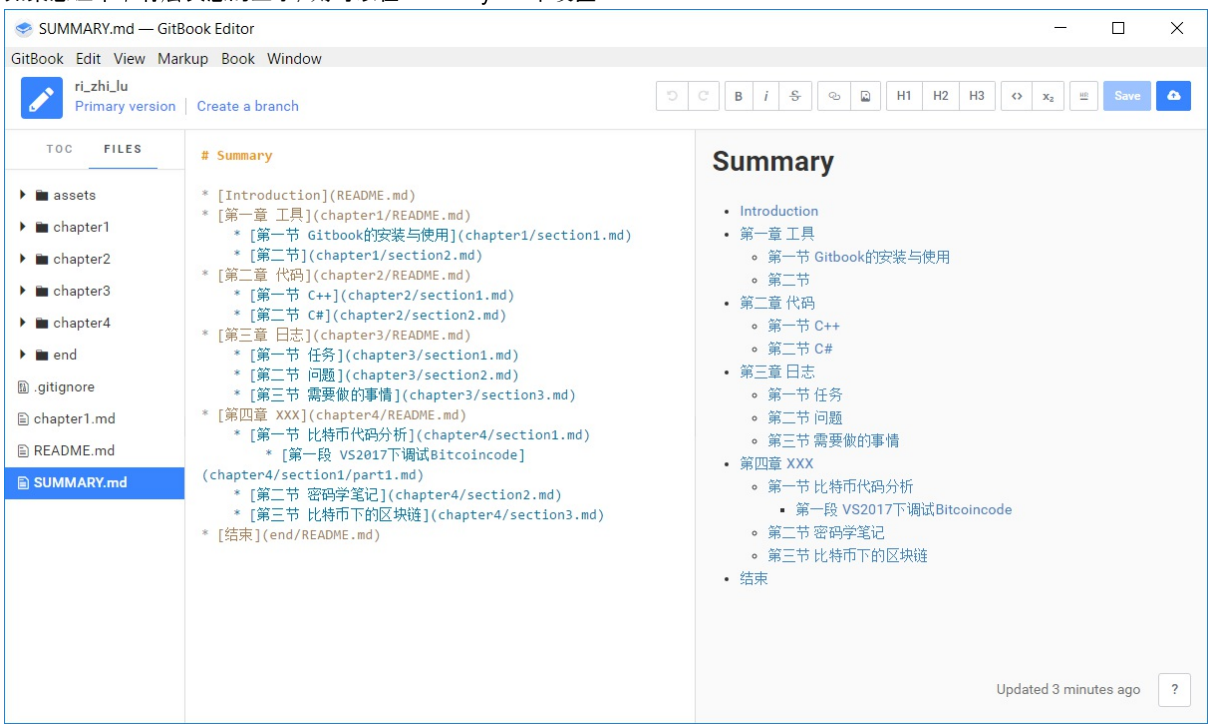
第一节 Gitbook的安装与使用

Gitbook是Github旗下的产品，它提供书籍的编写与管理的功能，一个核心特征就是，它管理书也像管理代码一样，可以对数据进行fork,建立新的branch,使得书也可以进行版本迭代。也可以与多人合作，来编辑，更新书籍，适合单人创作或者多人共同创作。怎么利用gitbook生成自己的书，可以参考如下链接 https://www.jianshu.com/p/cf4989c20bd8
https://www.cnblogs.com/Lam7/p/6109872.html

在安装好Node.js与gitbook之后，我们可以利用gitbook的GUI创建一本书，写入一些章节，然后保存，然后在cmd到该书summary.md所在的路径，然后再利用gitbook init进行初始化，然后再利用gitbook serve在浏览器上对书内容进行访问。当然，个gitbook serve这一步会出错，会出现fontsettings.js找不到的情况，我们需要修改C:\Users\pili.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js，设置confirm: false，再重启个gitbook serve就可以了。详见https://github.com/GitbookIO/gitbook/issues/1309 然后就可以在 http://localhost:4000 访问书籍了



如果想让章节有层次感的显示，则可以在summary.md中设置



第二节 编辑数学公式

可以参考https://598753468.gitbooks.io/tex/content/fei_xian_xing_fu_hao.html
数学公式的编辑类似于Latex.

需要安装mathjax

```
npm install mathjax
安装特定版本的npm install mathjax@2.6.1
```

首先在书籍project的最顶成新建一个book.json,内容如下

```
{
    "gitbook": "3.2.3",
    "plugins": ["mathjax"],
    "links": {
        "sidebar": {
            "Contact us / Support": "https://www.gitbook.com/contact"
        }
    },
    "pluginsConfig": {
    "mathjax":{
            "forceSVG": true
        }
        }
}
```

然后再用gitbook install命令安装mathjax

gitbook pdf ./ mybook.pdf

http://ldehai.com/blog/2016/11/30/write-with-gitbook/

no such file fontsettings.js

第二章:日志

```
int main(int argc, char* argv[])
{
    SetupEnvironment();

    // Connect bitcoind signal handlers
    noui_connect();

    return (AppInit(argc, argv) ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

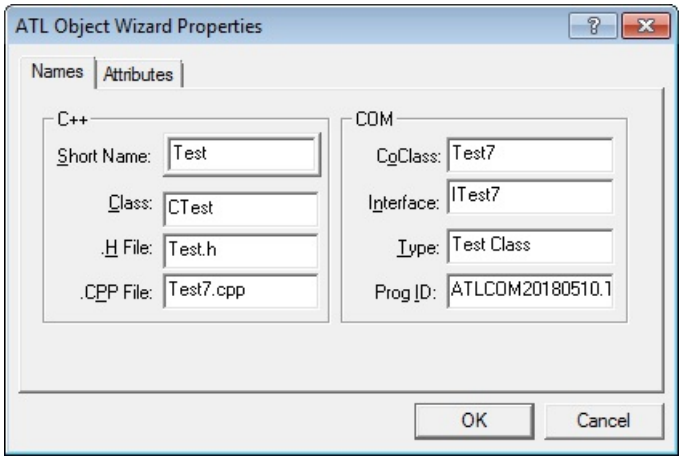
第一节 C++

接口：即抽象类，就是包含至少一个纯虚函数的类

```
一个类里面实现多种接口Iinterface, IinterfaceB, IinterfaceC
    IE84TPTimeOutUIAck : public IUnknown
    {
    public:
        virtual HRESULT STDMETHODCALLTYPE ResponseTpTimeOutUIAck(
            /* [in] */ short nPortNo,
            /* [in] */ short nTpNo) = 0;

    };
```

New ATL object 时，选择simple object，然后属性设置如下：则可以实现一个类中有多组接口函数。



C++通过ATL来实现接口的继承。

```
// Cr3halObj
class ATL_NO_VTABLE Cr3halObj :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<Cr3halObj, &CLSID_r3halObj>,
    public IConnectionPointContainerImpl<Cr3halObj>,
    public Ir3halObj,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IHALEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IMMIEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE90Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE116Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IFFUEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IWaferProtrusionEvent>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_ISignalTowerDevice>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84HOAVBLEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84TPTimeOutUIAck>, //TP3 connection point
    public IR3HALDiag
{

DECLARE_EMUCLASSFACTORY_SINGLETON(Cr3halObj)
```

COM组件接口继承的实现 https://blog.csdn.net/dingbaosheng/article/details/624504

TL学习笔记03

第二节 C#基本知识

```
Linq 语句
q = from sl in m_cassette.slot_descs
     where sl.slot_seq == i
     orderby sl.recipe_seq ascending
     select sl;
```

```
C#委托
Action<int, string, string> dr = this.OnFinishedLoadingWaferInternal;
    this.Invoke(dr, new object[] { slot, waferID, lotID });

http://www.cnblogs.com/akwwl/p/3232679.html
Delegate至少0个参数，至多32个参数，可以无返回值，也可以指定返回值类型
Func可以接受0个至16个传入参数，必须具有返回值
Action可以接受0个至16个传入参数，无返回值
Predicate只能接受一个传入参数，返回值为bool类型
```

```
 Mutex的使用
 // Summary:
        //     Initializes a new instance of the System.Threading.Mutex class with a Boolean
        //     value that indicates whether the calling thread should have initial ownership
        //     of the mutex, a string that is the name of the mutex, and a Boolean value that,
        //     when the method returns, indicates whether the calling thread was granted initial
        //     ownership of the mutex.
        //
        // Parameters:
        //   initiallyOwned:
        //     true to give the calling thread initial ownership of the named system mutex if
        //     the named system mutex is created as a result of this call; otherwise, false.
        //
        //   name:
        //     The name of the System.Threading.Mutex. If the value is null, the System.Threading.Mutex
        //     is unnamed.
        //
        //   createdNew:
        //     When this method returns, contains a Boolean that is true if a local mutex was
        //     created (that is, if name is null or an empty string) or if the specified named
        //     system mutex was created; false if the specified named system mutex already existed.
        //     This parameter is passed uninitialized.
        //
        // Exceptions:
        //   T:System.UnauthorizedAccessException:
        //     The named mutex exists and has access control security, but the user does not
        //     have System.Security.AccessControl.MutexRights.FullControl.
        //
        //   T:System.IO.IOException:
        //     A Win32 error occurred.
        //
        //   T:System.Threading.WaitHandleCannotBeOpenedException:
        //     The named mutex cannot be created, perhaps because a wait handle of a different
        //     type has the same name.
        //
        //   T:System.ArgumentException:
        //     name is longer than 260 characters.
        [ReliabilityContract(Consistency.WillNotCorruptState, Cer.MayFail)]
```

```
        [SecurityCritical]
        public Mutex(bool initiallyOwned, string name, out bool createdNew);
        var mutex = new Mutex(false, RS7.IF.CommonFunction.CommonString.MutexName, out createdNew);
```

```
as 的用法
IModuleCommonInterface iMagnet = MagnetSystem.GetInstance as IModuleCommonInterface;
is : 检查一个对象是否兼容于其他指定的类型,并返回一个Bool值,永远不会抛出异常
object o = new object();
        if (o is Label)
         {
             Label lb = (Label)o;
             Response.Write("类型转换成功");
         }
         else
         {
             Response.Write("类型转换失败");
         }
as:与强制类型转换是一样的,但是永远不会抛出异常,即如果转换不成功,会返回null
object o = new object();
         Label lb = o as Label;
         if (lb == null)
         {
             Response.Write("类型转换失败");
         }
         else
         {
             Response.Write("类型转换成功");
         }
```

```
Event
public static event EventHandler<NexusWaferLocationEventArgs> WaferLocationEvent
        {
            add { m_waferLocationEvent += value; }
            remove { m_waferLocationEvent -= value; }
        }
```

```
UserControl是C#在做Windows窗体应用程序时，经常用到的一个控件。他和Form一样，是一个展示型的控件。
本文介绍下Usercontrol在开发中常用的一些属性和方法。
```

```
C# 用Linq的方式实现对Xml文件的基本操作(创建xml文件、增删改查xml文件节点信息)
https://www.cnblogs.com/mingmingruyuedlut/archive/2011/01/27/1946239.html
https://blog.csdn.net/songyi160/article/details/50824274
```

```
接口使用interface 关键字进行定义，可由方法、属性、事件、索引器或这四种成员类型的任意组合构成。
接口的特性：
1.接口类似于抽象基类，不能直接实例化接口；接口中的方法都是抽象方法，实现接口的任何非抽象类型都必须实现接口的所有成员：
当显式实现该接口的成员时，实现的成员不能通过类实例访问，只能通过接口实例访问。
当隐式实现该接口的成员时，实现的成员可以通过类实例访问，也可以通过接口实例访问，但是实现的成员必须是公有的。
2.接口不能包含常量、字段、运算符、实例构造函数、析构函数或类型、不能包含静态成员。
3.接口成员是自动公开的，且不能包含任何访问修饰符。
4.接口自身可从多个接口继承，类和结构可继承多个接口，但接口不能继承类。
```

```
Lamda 表达式
To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator
```

```
=>,
and you put the expression or statement block on the other side. For example, the lambda expression x => x *
x
specifies a parameter that’s named x and returns the value of x squared. You can assign this expression to a
delegate type, as the following example shows:
delegate int del(int i);
static void Main(string[] args)
{
    del myDelegate = x => x * x;
    int j = myDelegate(5); //j = 25
}
public bool FloatTopLayer
{
    get => ForwardParameters.GetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, true);
    set => ForwardParameters.SetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, value)
;
}
```

在泛型类和泛型方法中产生的一个问题是，在预先未知以下情况时，如何将默认值分配给参数化类型 T：
T 是引用类型还是值类型。
如果 T 为值类型，则它是数值还是结构。
给定参数化类型 T 的一个变量 t，只有当 T 为引用类型时，语句 t = null 才有效；只有当 T 为数值类型而不是结构时，
语句 t = 0 才能正常使用。 解决方案是使用 default 关键字，此关键字对于引用类型会返回 null，对于数值类型会返回零。
对于结构，此关键字将返回初始化为零或 null 的每个结构成员，具体取决于这些结构是值类型还是引用类型。
对于可以为 null 的值类型，默认返回 System.Nullable(Of T)，它像任何结构一样初始化

```
public class GenericList<T> {
    private class Node {
        public Node next;
        public T Data;
    }
    private Node head;
    public T GetFirst() {
        T temp = default(T);
        if(head != null) {
            temp = head.data;
        }
        return temp
    }
}
```

为控件添加权限
1.让需要设置权限的控件继承一个接口IUserManagement，这个接口就只有一个方法OnUpdateUserChange，
这个方法里面处理不同用户的权限。

```
public void OnUpdateUserChange(UserLevel userLevel)
        {
            if (userLevel == UserLevel.USER_ADMIN)
            {
                btAdd.Enabled = true;
                btEdit.Enabled = true;
                btDelete.Enabled = true;
            }
            else if (userLevel == UserLevel.USER_ENGINEER)
            {
                btAdd.Enabled = true;
                btEdit.Enabled = true;
                btDelete.Enabled = true;
            }
            else
            {
```

```
                btAdd.Enabled = false;
                btEdit.Enabled = false;
                btDelete.Enabled = false;
            }
        }

private List<IUserManagement> m_lsUser = new List<IUserManagement>();
//User Management 把不同控件添加到m_lsUser
m_iuserManageContainer.AddModule(m_treeView);
m_iuserManageContainer.AddModule(m_sysConfigEditorCtrl);
m_iuserManageContainer.AddModule(m_probeLibraryMainDlg);
m_iuserManageContainer.AddModule(m_defaultRecipeCtrl);
m_iuserManageContainer.AddModule(m_userManagementDlg);
m_iuserManageContainer.AddModule(m_rawDataExplorer);
m_iuserManageContainer.AddModule(this);
m_iuserManageContainer.OnUpdateUserChange();

void AddModule(IUserManagement user){m_lsUser.Add(user);}
然后调用每个控件自己的OnUpdateUserChange函数
public void OnUpdateUserChange()
{
    foreach (var item in m_lsUser)
    {
        item.OnUpdateUserChange(USERLEVEL);
    }
}
```

```
 C#中，使用ServiceController类控制windows服务，使用之前要先添加引用：System.ServiceProcess，
 然后在命名空间中引用：using System.ServiceProcess。
private void StartService()
{
    this._controller = new ServiceController("ServicesName");
    this._controller.Start();
    this._controller.WaitForStatus(ServiceControllerStatus.Running);
    this._controller.Close();
}
```

```
C#委托和事件(Delegate、Event、EventHandler、EventArgs)
https://www.cnblogs.com/Scl891004X/p/6142917.html
```

```
事件
class Events : IDrawingObject
{
    event EventHandler PreDrawEvent;

    event EventHandler IDrawingObject.OnDraw
    {
        add
        {
            lock (PreDrawEvent)
            {
                PreDrawEvent += value;
            }
        }
        remove
        {
            lock (PreDrawEvent)
            {
```

```
                PreDrawEvent -= value;
            }
        }
    }

}
```

value是c#中的“属性”
例如c#某个类中有一个成员变量(字段)，为了安全性，外部如果要访问它，必须通过“属性”来访问：
private int _id;//这是一个成员变量，private表示是私有的，外部不可访问

```
public int ID
{
     set { _id = value; } //value就是外部为“属性”ID所赋的值

     get { return _id; } //当外部访问“属性”ID时，返回id的值

}
```

上下文关键字 value 用在普通属性声明的 set 访问器中。此关键字类似于方法的输入参数

DllImport是System.Runtime.InteropServices命名空间下的一个属性类，其功能是提供从非托管DLL导出的函数的必要调用信息。
DllImport属性应用于方法，要求最少要提供包含入口点的dll的名称。
DllImport的定义如下：

```
[AttributeUsage(AttributeTargets.Method)]
     public class DllImportAttribute: System.Attribute
     {
      public DllImportAttribute(string dllName) {…} //定位参数为dllName
      public CallingConvention CallingConvention; //入口点调用约定
      public CharSet CharSet;                                    //入口点采用的字符接
      public string EntryPoint;                                   //入口点名称
      public bool ExactSpelling;                                 //是否必须与指示的入口点拼写完全一致，默认false
      public bool PreserveSig;                                    //方法的签名是被保留还是被转换
      public bool SetLastError;                                   //FindLastError方法的返回值保存在这里
      public string Value { get {…} }
     }
[DllImport("mpi.dll", EntryPoint = "setMFCmode", CallingConvention = CallingConvention.Cdecl)]     internal s
tatic extern void setMFCmode();
[DllImport("mpi.dll", EntryPoint = "ver", CallingConvention = CallingConvention.Cdecl)]            internal s
tatic extern StringBuilder ver(int print);
[DllImport("mpi.dll", EntryPoint = "openDCE", CallingConvention = CallingConvention.Cdecl)]        internal s
tatic extern IntPtr openDCE(string pwdin, string cnfname);
```

第三章 日志

第一节 任务

1. 20180508
    i. Debug Miura 以对miura整体架构有个了解，知道哪些是硬件，哪些是与硬件无关
    ii. 中段目的是抽离出一个与Miura机器无关的通用平台
    iii. 最终目的是把RS塞进这个通用平台，也就是把RS建立在这个通用的平台之上
    iv. Boost 库中有uBLAS——Boost 线性代数基础程序库，因此，C++涉及矩阵运算的可以调用这个库。
        i. 自己加工下，但是要考虑跨平台的问题，因此不能封装成dll
2. 20180509
    i. 通过Miura提炼出通用的模板（类似于基类），那么通用的模板应该包含哪些东西？这是核心的问题。

第二节 问题项目:

基础线性代数库

1. 2D插值
    i. 基于拟合公式，而拟合公式就是1D插值
    ii. 第一个，实现克里金插值
2. 版本的迭代问题，每次也不求代码很完善，一次次测试与迭代，在过程中追求完美。
    i. 每个版本要说明改进的地方，为啥要改
    ii. https://github.com/lbbc1117/Nacho
    iii. 矩阵求逆需要记下一些notes或者文档。
        i. 这些只是一些练习，真实的可能会调用一些现成的库。目的是练练C++，也是为以后工作备用
    iv. SVD 求解论文
        i. http://people.duke.edu/~hpgavin/SystemID/References/Golub+Reinsch-NM-1970.pdf
    v. **Armadillo** C++ library for linear algebra scientific computing http://arma.sourceforge.net/
        i. https://github.com/conradsnicta/armadillo-code/
        ii. https://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries
    vi. 适合自己的才是最好的，
        i. 做一个自己实用的线性代数库，尽可能能供多的人去继承，去开发。
        ii. 要有自己的开发文档。
    vii. eigen：https://github.com/Li-Simon/eigen-git-mirror
        i. 功能比**Armadillo** 强大

第三节 需要做的事情

20180508

- [ ] 怎么把自己用gitbook创作的书同步到网上，这样就可以方便查看了，最好能设置只能个人看。
- [ ] C# Lamda表达式还是需要多看一些，自己写一些应用，用于Linq。
- [ ] Gitbook上怎么索引一个程序，这样就不要占用太多空间了。
- [ ] 自己写一个常见的优化算法的C++程序,涉及最常用的一些算法，主要是拟牛顿法这些，还有一些常见的机器学习算法，可以直接在工业中应用的高质量代码
    - [ ] 要保证库的基础性，不能依赖其它的库，但是也要保证库代码量尽量小

20180509

- [ ] 任何时候，都需要对自己的决策(无论是投资还是工作选择)持怀疑态度，反身思考，以确证自己决策的合理性
- [ ] 怎么通过密码学保证你写的东西的时间是没有被篡改的？
- [ ] 把密码学笔记传到Gitbook上，通过图片的形式，有时间的话，要转成电子文字版而不是图像版
- [ ] 35988250+Li-Simon@users.noreply.github.com Li-Simon

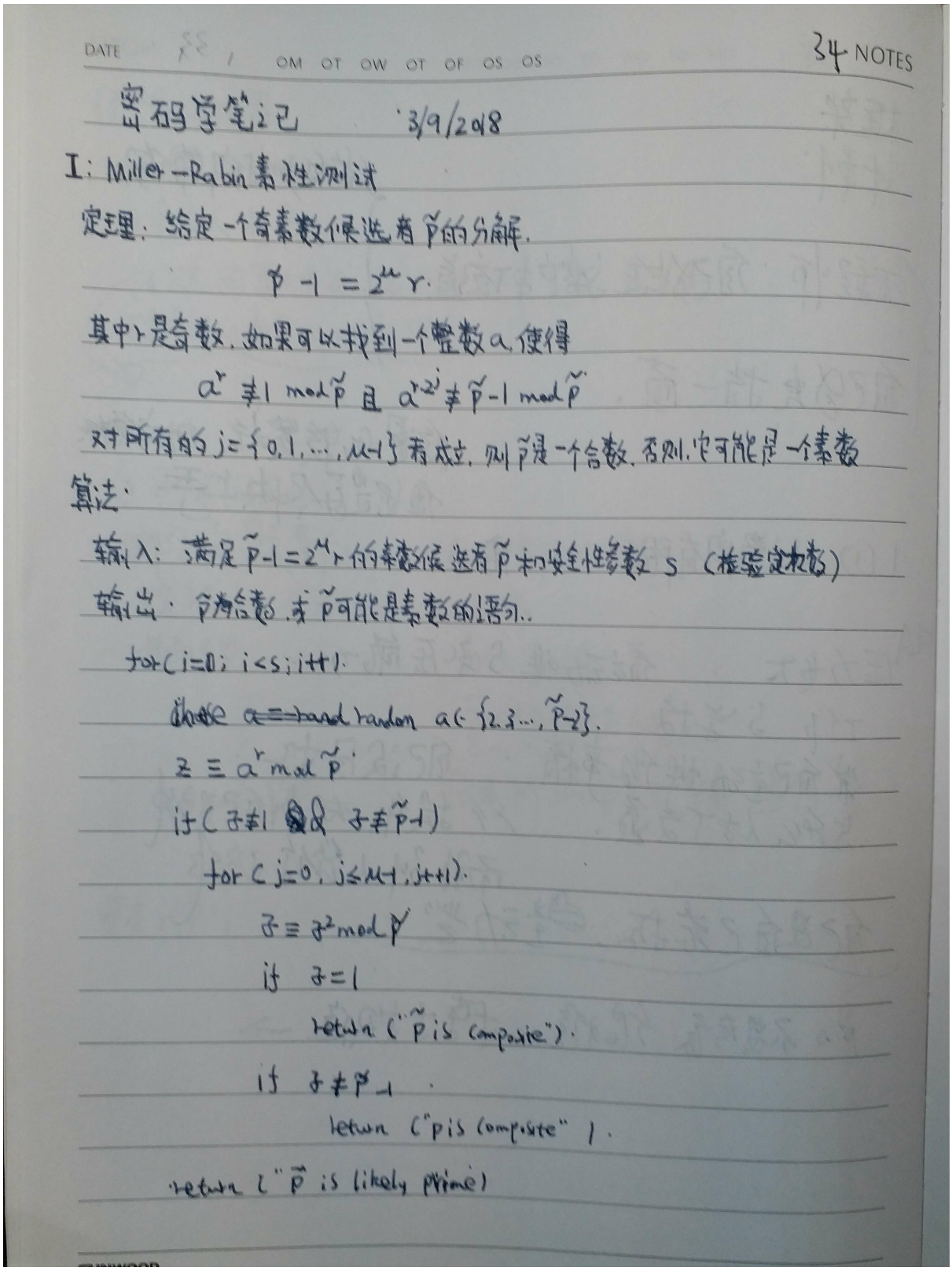
20180510

- [ ] Mathjax不能用
- [ ] 有空就思考用blockchain来做一些事情，知识的最终目的还是要转化成产品，变成有价值的东西。学习能力比知识储备更重要，只学不用会花费大量时间，因此更强调在应用的过程中学习。
- [ ] 一个类里面实现多组接口

第四章 XXX

第一节 比特币代码分析

第二节 密码学笔记



DATE 55 / OM OT OW OT OF OS OS 34 NOTES

密码学笔记 3/9/2018

I: Miller-Rabin 素性测试

定理：给定一个奇素数候选者 $\tilde{p}$ 的分解：

$$\tilde{p}-1=2^{u}r$$

其中 r 是奇数，如果可以找到一个整数 a，使得

$$a^{r}\not\equiv 1 \bmod \tilde{p} \text{ 且 } a^{r2^{j}}\not\equiv \tilde{p}-1 \bmod \tilde{p}$$

对所有的 $j=\{0,1,\dots,u-1\}$ 都成立，则 $\tilde{p}$ 是一个合数，否则，它可能是一个素数

算法：

输入：满足 $\tilde{p}-1=2^{u}r$ 的素数候选者 $\tilde{p}$ 和安全性参数 s（检验重数）

输出：合数结论，或 $\tilde{p}$ 可能是素数的语句。

```
for(i=0; i<s; i++)
    choose a==rand random a∈{2,3,...,p̃-2}.
    z ≡ a^r mod p̃.
    if(z≠1 && z≠p̃-1)
        for(j=0, j≤u-1, j++)
            z ≡ z^2 mod p̃
            if z=1
                return("p̃ is composite").
            if z≠p̃-1
                return("p is composite").
return("p̃ is likely prime")
```

第三节 比特币下的区块链

第五章:算法

第一节 **Kriging**插值

https://xg1990.com/blog/archives/222
空间插值问题，就是在已知空间上若干离散点 $$(x_i,y_i)$$的某一属性(如气温，海拔)的观测值$$z_i = z(x_i,y_i)$$的条件下，估计空间上任意一点$$(x,y)$$的属性值的问题。
直观来讲，根据地理学第一定律，

All attribute values on a geographic surface are related to each other, but closer values are more strongly related than are more distant ones.
大意就是，地理属性有空间相关性，相近的事物会更相似。由此人们发明了反距离插值，对于空间上任意一点$$(x,y)$$ 的属性$$z = z (x , y)$$ ，
定义反距离插值公式估计量 $$\hat{z} = \sum^{n}_{i=0}{\frac{1}{d^\alpha}z_i}$$
其中$$α$$通常取1或者2。

即，用空间上所有已知点的数据加权求和来估计未知点的值，权重取决于距离的倒数（或者倒数的平方）。

那么，距离近的点，权重就大；距离远的点，权重就小。反距离插值可以有效的基于地理学第一定律估计属性值空间分布，但仍然存在很多问题：

$$α$$的值不确定 用倒数函数来描述空间关联程度不够准确

因此更加准确的克里金插值方法被提出来了

克里金插值公式$$\hat{z*o*} = *sum^{*n*}*{i=0}{\lambda_iz_i}$$
其中$$\hat{z_0}$$是点$$(x_o,y_o)$$处的估计值，即：$$z_o=z(x_o,y_o)$$
假设条件：

1. 无偏约束条件 $$E(\hat{z_0} - z_0)$$
2. 优化目标/代价函数 $$J = Var((\hat{z_0} - z_0))$$取极小值
3. 半方差函数$$r*{ij}$$*与空间距离*$$d*{ij}$$存在关联,并且这个关联可以通过这这两组数拟合出来，因此可以用距离$$d*{ij}$$*来求得*$$r*{ij}$$

半方差函数$$r*{ij} = *sigma^2 - C*{ij}$$; 等价于$$r*{ij} = *frac*{1}{2}E[(z_i-z_j)^2]$$*
求得*$$r*{ij}$$之后，我们就可以求得$$\lambda_i$$

$$\begin{bmatrix}r*{11}&r*{12}&\cdots&r*{1n}&1\
r*{21}&r*{22}&*cdots&r*{2n}&1\\cdots&\cdots&\cdots&\cdots&\cdots\r*{n1}&r*{n2}&\cdots&r*{nn}&1\1&1&*cdots&1&0\end*{bmatrix}\begin*{bmatrix} \lambda_1\ \lambda_2\\cdots\\lambda_n\0\end*{bmatrix}=\begin*{bmatrix} r*{1o}\
r*{2o}\\cdots\r*{no}\1\end*{bmatrix}$$

结束