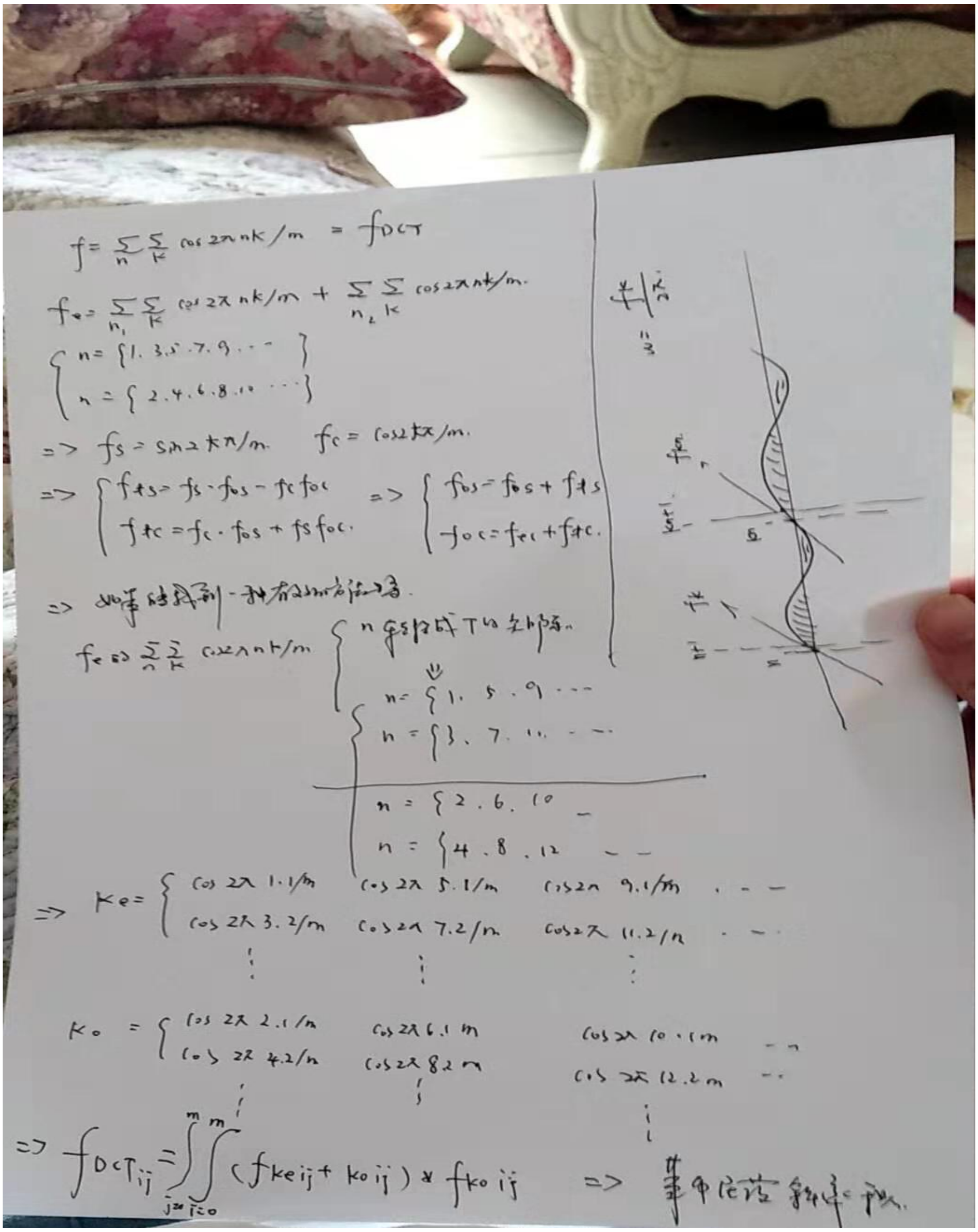
一种有效替代 FFT 的算法思想和它的论证过程手稿

《罗瑶光- 快速傅里叶- 莱布尼茨 2 分指数卷积》推导图片：



$$f = \sum_n \sum_k \cos 2\pi nk/m = f_{DCT}$$

$$f_e = \sum_{n_1} \sum_k \cos 2\pi nk/m + \sum_{n_2} \sum_k \cos 2\pi nk/m$$

$$\begin{cases} n = \{1, 3, 5, 7, 9, \cdots\} \\ n = \{2, 4, 6, 8, 10, \cdots\} \end{cases}$$

$$\Rightarrow f_s = \sin 2k\pi/m, \quad f_c = \cos 2k\pi/m$$

$$\Rightarrow \begin{cases} f_{\pm s} = f_s \cdot f_{os} - f_c f_{oc} \\ f_{\pm c} = f_c \cdot f_{os} + f_s f_{oc} \end{cases} \Rightarrow \begin{cases} f_{os} = f_{os} + f_{\pm s} \\ f_{oc} = f_{ec} + f_{\pm c} \end{cases}$$

$$f_e = \sum_n \sum_k \cos 2\pi nk/m \quad \begin{cases} n = \{1, 5, 9, \cdots\} \\ n = \{3, 7, 11, \cdots\} \end{cases} \quad \begin{cases} n = \{2, 6, 10, \cdots\} \\ n = \{4, 8, 12, \cdots\} \end{cases}$$

$$\Rightarrow K_e = \begin{cases} \cos 2\pi\, 1.1/m \quad \cos 2\pi\, 5.1/m \quad \cos 2\pi\, 9.1/m \quad \cdots \\ \cos 2\pi\, 3.2/m \quad \cos 2\pi\, 7.2/m \quad \cos 2\pi\, 11.2/m \quad \cdots \end{cases}$$

$$K_o = \begin{cases} \cos 2\pi\, 2.1/m \quad \cos 2\pi\, 6.1/m \quad \cos 2\pi\, 10.1/m \quad \cdots \\ \cos 2\pi\, 4.2/m \quad \cos 2\pi\, 8.2/m \quad \cos 2\pi\, 12.2/m \quad \cdots \end{cases}$$

$$\Rightarrow f_{DCT_{ij}} = \int_{j=0}^{m}\int_{i=0}^{m} (f_{keij} + k_{oij}) * f_{koij}$$

这个公式，仅仅感谢莱布尼茨。

历史思考：

2014 年有相当长的时间，进行傅里叶的卷积内核算子优化，我发现了一个大问题就是周期频率中的高阶变换的准确率问题。为此，我做了可逆运算快速傅里叶变换，发现非 2 的指数列数据存在缺陷，需要微负阶分数阶复杂计算。我思考了下，这个过程不应该这么痛苦。寻找一种有效的时序变换一直是我的一个主题。

2016-2017 年 我买了本盲分离的时序波动 书籍，仔细的看了遍，发现傅里叶在很多时候可以用离散余弦来进行替代，减少算子，特别是在那些没有强调变换精度的场合。我发现了一些猫腻。如果因为精度确定了速度，那么应该还有更为迅速的方式来进行时序变换。我一直在思考这个问题。

今年，我的时间比较充裕，于是我详细的阅读了莱布尼茨的分解手稿，另外感谢同济大学的 高等数学 绿色封面的教材指导书。我在常微分和求导的过程中为了关注 2 次求导的可逆性，我灵感来了，莱布尼茨的化简可以做临近斜率比分析，这个分析完美的解决了 快速傅里叶 不能做非 2 的指数数据准确处理的问题。而且快速替代了离散余弦变换的计算力。通过 算能优化公式 N= S(AON)/s(AON), 这个算法速度爆炸。

思考后的整理：

我命名为 LBNFT 全名为 罗瑶光-莱布尼茨-斜率-快速-变换 1 代 算法

算法目的：

1 弥补快速傅里叶算法的非 2 指数频率域准确性。

2 提高时序频率变换的算能。

3 寻找一种极速算法替代（分数阶同时有是高阶）的可逆变换准确性。

使用领域： 极速通信。

整理后化简得 **Java** 代码编程：

未优化版本代码：

```
public double[] swapDifferential(double[] input) {
      double[] output= new double[(int)size];
      for(int i= 0; i< size; i++) {
         for(int j= 1; j< size; j+= (i+ 1)) {
            double k= input[j]/input[j- 1];
            output[j]+= k;
         }
      }
      return output;
   }
```

变换后的应用验证

其可逆 变换和高阶变换的准确性，我采用了 4，8，16 位数据变换的采样验证， 结果是正确的，如下：

input：
input[0]=1;
input[1]=-30;
input[2]=-1000;
input[3]=30;
input[4]=1;
input[5]=30;
input[6]=1000; --------------------起始
input[7]=30;
input[8]=1;
input[9]=-30;
input[10]=-1000;
input[11]=30;
input[12]=1;
input[13]=30;
input[14]=1000; ----------------------结束 测试数据中间 7 位
input[15]=30;

输出：
0.0
-Infinity
-0.06944444444444445
-0.0035999999999999995
-2.2222222222222223
4050.0 -----------------------------起始
1.4814814814814816
0.007199999999999999
1.1111111111111112
-7200.0
-2.5
-0.0048
-1.1111111111111112
16200.0 --------------------------结束 上面有 7 位 验证成功
0.7407407407407408
0.007199999999999999

-Infinity 是因为 j 从 1 开始， 第一位没有进行有效变换。以后我会优化。

感谢和论证：

1 高顺， 以前和他 PK Leecode 排位，他有用向量机的题目为难过我一次。自从那时候开始我比较全面的思考过 邻接向量机和它的现实世界中的应用。

2 基督大学的我的离散数学 V. G 老师，给了我邻接矩阵计算的启蒙基础。

3 通过图片推导公式，我发现一个现象， 这个 cos2pi 是多余的，一旦用 w 替换，只需要用斜率比来叠加就能满足了。

4 快速傅里叶变换广泛的应用于工程、科学和数学领域。这里的基本思想在 1965 年才得到普及，但早在 1805 年就已推导出来。 1994 年美国数学家吉尔伯特·斯特朗把 FFT 描述为“我们一生中最重要的数值算法”，它还被 IEEE 科学与工程计算期刊列入 20 世纪十大算法。

罗瑶光

20190915