目录

GEP-0009 ZMATH 地震数据处理模型

杨昆仑

版本：6b97decb067dfe5fb64f3540bea7fab135b9d9df

日期：2025-05-01

可编程处理模块 ZMATH 在 GeoEast 系统中引入 Lua 脚本语言[1] 引擎和脚本处理能力。该模块与 GeoEast 系统的关系可以分为四部分进行理解：执行机制 (数据流模型)、数据模型 (地震道表示)、映射关系和信号处理函数扩展。本文档基于 ZMATH 模块编写，部分示例脚本片段使用了 GeoEastv3.5 中定义的道头名字，简要介绍使用脚本语言进行地震数据操作的基础。需要注意，示例中部分道头名字需要更改为 GeoEastv4.x 相应的道头名字后才支持在 v4.x 版本中执行。

ZMATH 模块使用的地震数据流和数据模型比目前系统更加简单灵活。上述概念除在 ZMATH 模块中使用外，也可以作为理解所有其它地震处理模块和接口的抽象地震数据表达和操作模型。本文假设用户已经熟悉 GeoEast 并有一定的 Lua 脚本语言基础。如果从未了解 Lua 语言，可以首先参考后面的应用实例。

1 执行机制

GeoEast 作业可以看作一个流水线，地震数据就像流水线上被加工的零件，而作业中的每个处理模块就像流水线上的一个加工步骤。ZMATH 以道集为单位执行，即对每个道集运行一次如图-1所示的作业参数卡 <ensemble run> 中的指令。

为了实现更多的功能，在作业开始时，处理任何道集之前，ZMATH 模块运行一次图-1所示参数卡 <flow begin> 中的指令。在作业结束前，处理所有道集之后，运行一次参数卡 <flow ending> 中的指令。为了限制 ZMATH 灵活性可能带来的不便，用户必须在参数卡 <headers update> 中明确给出需要更新的道头名字，并在 <seismic update> 中声明是否更新数据域，否则在执行指令过程中更改的变量不会映射回地震数据流中。具体映射关系在后面详述。

用户指令 (脚本) 的执行机制如图-2所示。ZMATH 模块依次“看到”数据流中的每个道集，且仅能“看到”当前处理的一个道集，并执行道集处理指令，完成对地震数据的处理。

1.1 语法限制

因为 GeoEast 作业编辑器的限制，在作业卡中编辑脚本时**不能使用 * 逗号 ***！但很多语法结构必须使用逗号，为了绕过这个奇葩限制，用户**必须使用竖杠“|”代替**。例如

```
amplitude, phase = vector1:fft()
for i=1, 100 do
    print(string.format("Hello, number %d", i))
end
```

[1] 该脚本语言引擎核心使用 MIT 自由授权协议的第三方库，可在商业软件中使用

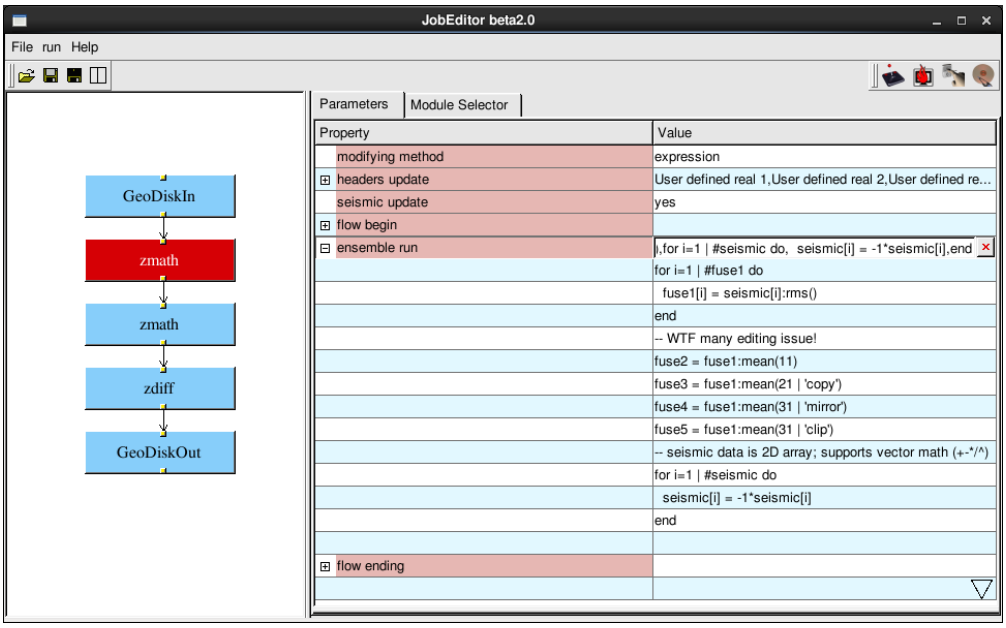

图 1: ZMATH 模块用户界面

在 `modifying method=expression` 模式下，作业卡中输入上述脚本指令时，会出现语法错误。上述原始指令只能使用于外部脚本文件中，在作业卡中必须输入下面的形式

```
amplitude | phase = vector1:fft()                    -- 多个返回值
for i=1 | 100 do                                     -- for 循环
    print(string.format("Hello| number %d" | i))      -- 参数列表
end
```

ZMATH 模块在 `expression` 模式下内部会将竖杠“|”替换为逗号，从而绕过 GeoEast 作业编辑器的这个限制。当然最好的解决方案是取消程序的奇葩逗号分行方式。

1.2 道集预选

注意：该功能为临时添加，后续可能会删除。如果能够使用 `TrcSelect` 或者 `TrcSplit` 模块所提供的功能是更优雅的解决方案。ZMATH 在作业运行过程中需要花费较多 CPU 时间用于脚本引擎和 GeoEast 执行系统之间交换数据，即使只需对大数据中的少量进行处理，这种数据交换依然无法避免。

2 数据模型

为方便扩展，ZMATH 的数据模型不区分道头和地震数据采样点，而是将它们统称为有名字的 (Named) 数据域 (Field)。每个地震道 (TRACE) 都包含完全相同的数据域，地震道可以包含**任意多个任意命名的任意长度和类型的域**，道集 (GATHER) 可以**包含任意个道，且每个道集的道数不必相同**。该数据模型可描述 SEGY 和 SU 文件以及任意灵活的道头。数据模型的表现形式如图-3所示。注意：此处数据模型为用户 (算法模块开发人员和处理员) 视角的数据模型，数据如何在磁盘或内存中存储是系统开发层面的问题。

为限制灵活性，域设置应根据需要由相应企业或行业标准约定，处理员根据项目类型选择遵守。例如，处理 OBN 和 VSP 需要不同种类的数据域 (道头)，对此分别制定处理标准约定使用什么样的道头名字，数据

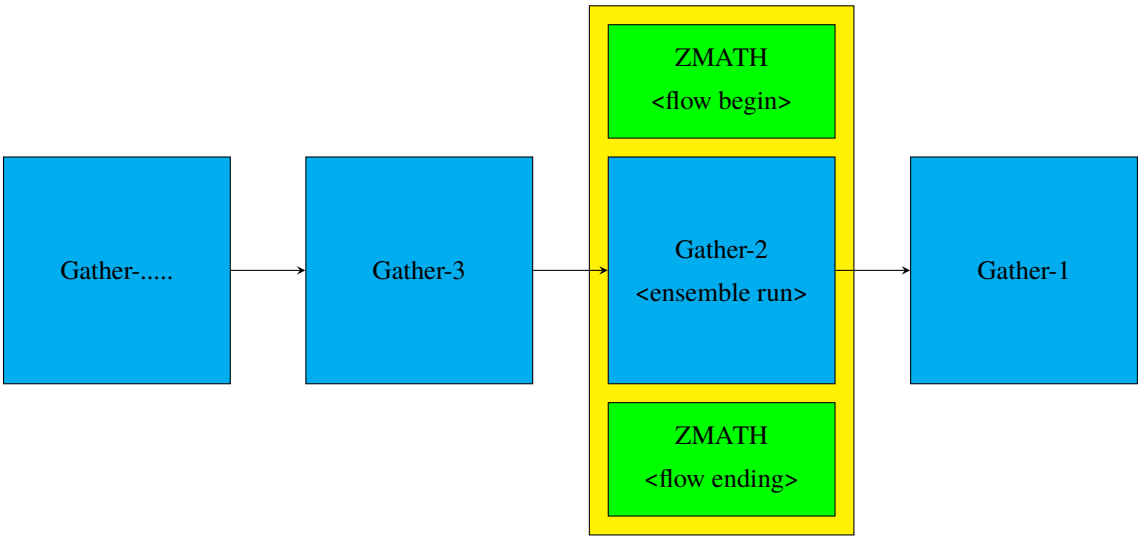

图 2: ZMATH 数据流和执行顺序

模型可承载这些类型数据。例如，在自适应减时，同一道头可能包含一个输入数据和多个模型数据；或者多分量处理时，同一道头下不仅包含 P 波数据，还有 Vz、Vx 和 Vy 数据。

2.1 域 (Field)

域是包含地震道信息的容器。域一般包含名字、类型和长度等信息。域的类型可以是整型、浮点、长整型或双精度浮点。长度大于 1 的域可以具有采样间隔等属性信息。域名应符合一般编程语言变量的定义，即名字以字母开头的字符串，仅包含字母、数字和下划线，且不与常用的编程语言关键字冲突。考虑到 Lua 和 Python 在地震资料处理中的潜在应用，域名选择应特别注意避免与这两种语言的关键字冲突，且约定禁止使用大写字母。ZMATH 中域名全部用小写表示。

2.2 道头

道头映射为长度 (Length) 为 1 的域，对于单道数据来说，道头映射为普通变量 (另一工具 ZLIST 采用此单道模型)；对于 ZMATH 的道集数据来说，道头映射为向量 (一维数组)，其中第 i 个元素对应道集中的第 i 道该域的值。如图-3所示，偏移距 offset 映射成名字为 offset，类型为向量的变量。途中高亮显示的 offset[3] 表示当前正在处理道集中第 3 道的偏移距。

2.3 数据

而地震数据映射为长度大于 1 的域。对于 ZMATH 道集，地震采样映射为矩阵 (二维数组)，其中第一个索引表示第几道，第二个索引表示第几个采样点。由于 GeoEast 并没有域的概念，每道仅允许一组地震数据采样，数据没有名字，在本文的数据模型中 ZMATH 默认用变量名 seismic 访问。如图-3所示，高亮显示的 seismic[4][9] 表示第 4 道的第 9 个采样点数据。

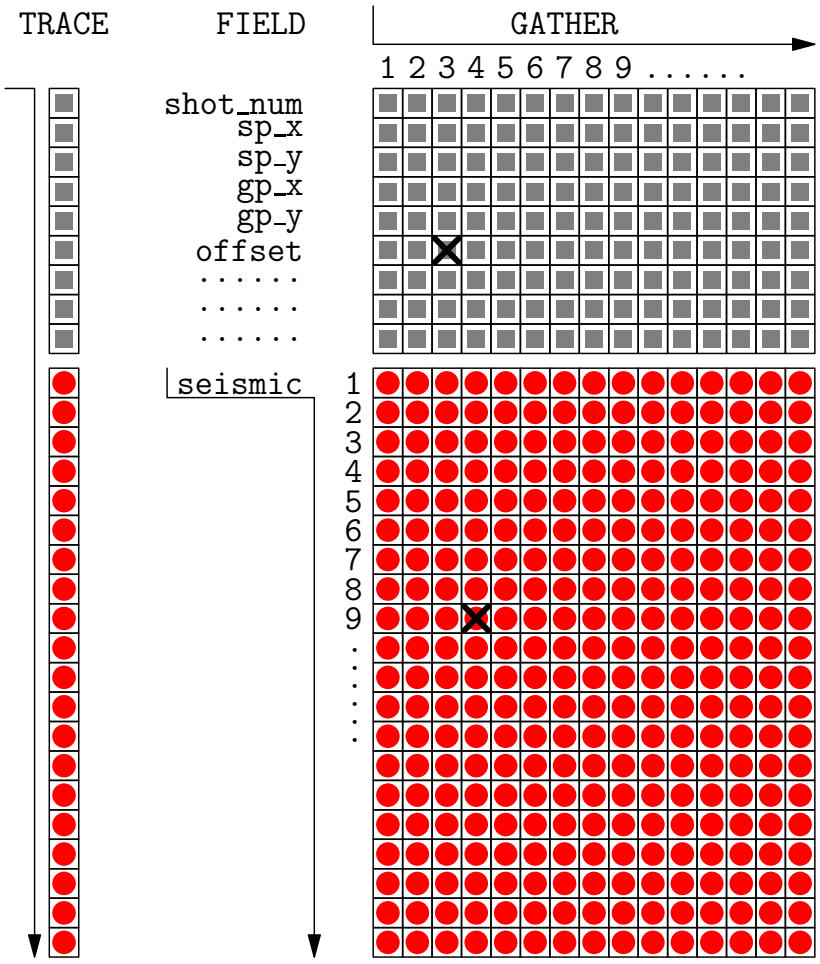

图 3: 数据模型示意图

高亮显示数据点对应 `offset[3]` 和 `seismic[4][9]` 位置

2.4 示例

在道集处理模式下，单道处理时对道集中每一道进行循环即可。在图-3所示的数据模型中，定义了如表-1所示的数据域：

本文仅定义了模型的机制 (Mechanism)，具体存储方式由策略 (Policy) 或企业标准 (Standard) 规定，从而实现机制和策略的分离和解偶。即在**无需改变底层代码**的情况下满足不同需求。对于上述数据模型，ZMATH 像操作道头一样通过名字操作地震数据，并且可以随意混合操作道头和地震数据。

3 映射关系

在 ZMATH 模块中，所有地震处理需要的信息和功能都映射到 Lua 脚本引擎中，以变量、方法和函数的形式供用户使用。脚本语言内置的有一般变量 (标量) 和表 (数组和类数组结构)，为了与文中地震数据模型相对应，并容纳面向对象的信号处理方法需要，我们额外定义了扩展向量和矩阵类型。按照 Lua 语言的习惯约定，向量和矩阵的索引都是从 1 开始。与普通类型不同，ZMATH 中的这些扩展类型可以让用户调用相应的方法实现对应的信号处理功能。

表 1: Field of the Trace

域名 (Field Name)	格式 (Format)	长度 (Length)	采样间隔	文档
shot_num	int	1	0	炮点编号
sp_x	double	1	0	炮点 x 坐标
sp_y	double	1	0	炮点 y 坐标
gp_x	double	1	0	检波点 x 坐标
gp_y	double	1	0	检波点 y 坐标
offset	float	1	0	偏移距
.....	.....			.....
seismic	float	>1	4ms	P 波采样点

3.1 向量

向量是一维数组。假如道集包含 `n` 个地震道，且存在名字为 `offset` 的域，则它将被映射为同名的向量。向量可以使用 Lua 语言的数组长度 `#` 运算符，即表达式 `n==#offset`(相等判断) 为真。向量可以作为一个整体使用，进行加减乘除等运算；也可以和数组一样通过下标访问其任意元素。例如单独通过赋值更新 `offset[3]`，数据位置如示意图-3中高亮显示，改变第三道的偏移距。向量作为扩展数据类型，具有部分类似面向对象的编程语言特性，拥有一些信号处理的“方法”。例如 `offset:mean(11)` 为对该向量进行 11 点均值滤波。详细的信号处理扩展函数可以参考其它章节。

3.2 矩阵

矩阵是二维数组，其每一列都是一个向量。假如每个地震道包含 `m` 个采样点，道集所有采样点映射为名字是 `seismic` 的二维数据。最后一个数据点为 `seismic[n][m]`，其中表示地震道的索引为高维度，表示采样点的索引为低维度。对于数组长度运算符 `#`，存在如下关系：`n==#seismic`；当 `i∈[1,n]` 时，`m==#seismic[i]`。例如 `siesmic[3][9]` 表示第 3 道中第 9 个采样点，数据位置如图-3所示。其中 `seismic[i]` 可作为向量使用，表示第 `i` 道的所有数据，同时拥有我们所扩展的向量的所有信号处理“方法”。作为矩阵，也拥有自己的信号处理“方法”，例如转秩生成新的二维数组：

```
1 Tseis = seismic:transpose()
```

此时 `Tseis[i]` 表示道集中第 `i` 个采样点组成的时间切片，它也是一个向量，拥有向量的所有方法。在本例中 `m==#Tseis`，即一共有 `m` 个时间切片。当 `i∈[1,m]` 时，表达式 `n==#Tseis[i]` 为真，即每个时间切片的长度为 `n`，即当前道集的地震道数。

进入模块时，ZMATH 预先将地震道集映射为脚本语言可以访问的向量和矩阵。退出模块时，模块从脚本语言环境获取映射的向量和矩阵值，更新地震道集。例如简单扩散补偿示例：

```
1 Tseis = seismic:transpose()      --转秩为时间切片访问
2 for i=1, #Tseis do               --循环时间方向
3     Tseis[i] = i*Tseis[i]        --按时间切片索引补偿，标量向量乘
4 end
5 seismic = Tseis:transpose()      --转秩返回初始映射关系
```

示例演示了数据映射关系，具体的运算方式和信号处理函数此处不再详细描述。

3.3 示例

道集处理中，道头对应的域映射为向量 (横向行)；以 offset[3] 为例，表示第 3 道的偏移距，offset 整体表示道集中该域对应的行组成的向量；数据对应的域映射为矩阵，以 seismic[3][9] 为例，表示第 3 道的第 9 个采样点，而 seismic[3] 作为一个向量表示图-3中第 3 列的所有采样点。

该数据模型能够支撑存储和表达各种类型的地震数据，通过制定适当的标准进行约束，也可以表达速度等相关数据。也能够较为容易的映射为 Lua、Python 以及 C/C++ 的数据结构和编程接口。同时简单并容易理解，能够简化处理系统的开发和使用环境。

4 扩展功能

除了 Lua 标准库 (例如 math、os 和 string) 包含的函数以外，GeoEast 的 ZMATH 模块还定义了一系列方便地震信号处理的扩展。其中包括一些常量和函数，但大部分扩展函数的作用对象为 ZMATH 扩展的向量和矩阵，以面向对象的语法格式使用，形式为 object:func()。在后续的例子中作用于向量对象用 vector:func() 表示，作用于矩阵对象用 matrix:func() 表示。

4.1 常量

常量是 ZMATH 预先设置的普通变量，通过该变量用户可以获得模块的运行环境信息，用户可以但在任何情况下都**不应该**更改这些预置变量。预置常量均为**大写**字母组成。

4.1.1 SI

采样间隔 SI 是一个预定义变量，Sample Interval，按照 GeoEast 约定单位是微秒，如果采样间隔是 4ms 则 SI=4000。用户不应试图改变该变量，改变预定义常量造成的后果不可预知。需要注意，除非特别声明，大部分函数和变量都使用国际单位制，即千克、米和秒。

```
1 print("SI="..SI)     -- 在log中打印当前作业的Sample Interval
```

注意：如果有多个长度大于 1 的数据域 (目前 GeoEast 不支持)，每个数据域可能有不同的采样率，需要定义额外的接口形式让用户分别访问其采样率，暂时不考虑。

4.1.2 NS

地震数据样点个数 NS 是一个预定义变量，Number of Samples，在 <ensemble run> 环境中可以使用 #seismic[1] 来获取样点数。但在 <flow begin> 和 <ensemble run> 中不存在表示地震数据的 seismic 变量，用户可以使用 NS 来在上述两个环境中获取样点数。

```
1 assert(NS==#seismic[1])      --<ensemble run>环境中成立
```

4.1.3 GROUP

从 1 开始的道集序号，GeoEast 作业可以认为是由道集组成的数据流，ZMATH 按顺序依次处理数据流中的每个道集，每处理一个道集 GROUP 的值增加 1。如果想在 log 中打印道集的序列号：

```
1 print(string.format("Processing group %8d, ntrace=%8d", GROUP, #seismic))
```

上述脚本打印道集序列号及该道集所包含的道数。在作业结束时，GROUP 变量依然可以访问，其值为所有流经 ZMATH 模块的道集计数。

4.1.4 TRACE

从 1 开始的道序号，因为 ZMATH 是道集处理模块，TRACE 为道集中第一道在整个数据流中的序列号，如果想在 log 中打印道集中每一道的序列号：

```
for i=1, #seismic do
    index = TRACE+i-1
    print(string.format("Processing trace %8d", index))
end
```

每处理完一个道集，TRACE 值会自动增加。在作业结束时，该变量的值为该模块处理的总道数。注意：使用 #seismic 获得道数且后续无其它使用需求时，比 #offset 或任意其它道头名字获取道数的运行开销要大很多。因为 ZMATH 会在用 #seismic 计算道集长度时不得不导入导出地震数据。

4.2 函数

大部分扩展功能通过向量和矩阵的方法函数实现，部分功能和数据模型没有特别的联系，因此单独定义了一些函数以方便地震信号处理。

4.2.1 vector

默认创建一个长度和 seismic 一样的向量，也可以接收一个参数，创建指定长度的向量。与 Lua 中普通的表或数组不同，向量创建后长度不能改变，也不能使用 Lua 内置的 table 函数操作向量。但是量可以使用 ZMATH 扩展的向量信号处理方法函数。

```
v1 = vector()       -- 创建和作业中地震道长度相同的向量并初始化为零
v2 = vector(n)      -- 创建长度为n(整数)的向量并初始化为零
v4 = vector({3,2,1}) % TODO, new added function!!!
v3 = {}             -- 创建空的普通表，不能调用向量的扩展方法
```

创建元素为零的向量有很多种方法，下面实例中用到了之后定义的一些向量方法：

```
v1 = vector()                       -- 长度和地震采样相同
v2 = vector(#seismic[1])            -- 长度和地震采样相同
v3 = seismic[1]:zero()              -- 长度和地震采样相同
v4 = seismic[1]*0                   -- 长度和地震采样相同
v5 = seismic[1]-seismic[1]          -- 长度和地震采样相同
v6 = vector(1000)                   -- 长度为1000
v7 = seismic[1]:copy(1000):zero()   -- 长度为1000
```

其中第 1 至 5 行脚本具有完全相同的作用。第 6 和 7 行具有完全相同的作用，其中第 7 行从道集第一道地震数据向量中截取前 1000 个采样点 (假设长度足够) 生成新向量并清零、复制返回；最终效果都是 vector 成为长度 1000，值全部为零的向量。

4.2.2 matrix

默认创建一个维度与 seismic 一样的矩阵，也可以接受一到两个参数，分别表示创建矩阵的列数和行数。如果只有一个参数，默认行数即地震数据采样点数，用法如下：

```
amatrix = matrix()      -- 创建和作业中道集维度相同的矩阵并初始化为零
amatrix = matrix(m)     -- 创建与包含m道的道集维度相同的矩阵
amatrix = matrix(m, n)  -- 创建包含m道且每道n个样点的矩阵
```

创建元素为零的矩阵有很多种方法，下面实例中用到了之后定义的一些矩阵方法：

```
matrix1 = matrix()
matrix1 = seismic:zero()
matrix1 = matrix(#seismic)
matrix1 = matrix(#seismic, #seismic[1])
matrix2 = matrix(80, 1000)
matrix2 = seismic:copy(80, 1000):zero()
```

受限于数据流，ZMATH 暂时不能改变地震数据的长度和采样间隔。用户新建的向量和矩阵如果维度与输入不同，则只能用作处理中的临时变量，不能直接用于将结果传递给下一个模块。

4.2.3 tosample

继承自 SEGY，GeoEast 采样间隔以微秒为单位。为避免该约定和 ZMATH 使用的国际单位制间的转换冲突，定义了转换函数 tosample。只有一个参数时，函数使用地震道时间采样间隔转换；用户给出第二个参数时，使用该参数为采样间隔，所有参数采用国际单位制。使用方法如下：

```
index = tosample(time_in_seconds)           --返回值为整数，向下取整
index = tosample(time_in_seconds, SI*1E-6)  --使用国际单位制
index = tosample(distance_in_meter, dx_in_meter)
```

使用国际单位制是因为该函数可能用于时间采样的序列 (向量)，也可能用于空间采样的序列。在接收两个参数时，可以等效认为该函数定义如下：

```
function tosample(t, dt)
    return 1+math.floor(t/dt)   --向下取整，参见Lua语言math函数库
end
```

该函数可用于协助修改地震数据采样，或者将采样写入道头中。例如下面的例子将 1120 毫秒处的地震数据采样点存入道头 f_user1 中，然后将该采样点清零

```
stime = 1.12                            -- 时间1120 ms
index = tosample(stime)                 -- 对应采样点索引
for i=1, #f_user1 do
    f_user1[i] = seismic[i][index]      -- 采样存入f_user1道头
    seismic[i][index] = 0               -- 该时刻数据清零
end
```

4.2.4 fsample

和 tosample 类似，该函数返回浮点数索引。浮点数索引可以用于 sampler 和 sampler2d 函数，将数据作为一个连续函数的采样，获取重构的连续函数在任意位置的采样值。

```
slice = fsample(time_in_seconds)            --返回值为浮点数
slice = fsample(time_in_seconds, SI*1E-6)   --使用国际单位制
slice = fsample(distance_in_meter, dx_in_meter)
```

在接收两个参数时，fsample 等效的函数定义如下：

```
function fsample(t, dt)
    return 1+t/dt               --Lua索引从1开始，因此+1
end
```

4.2.5 list

格式化并打印到 LIST 内，类似于 Lua 语言 print(string.format(args)) 函数。不同之处在于 GeoEast 中使用 print 打印输出到 LOG 文件，而扩展函数 list 打印输出到 LIST 文件。

```
list(">>>Processing group %6d", GROUP)
for i=1, #seismic do
    index = TRACE+i-1
    list("Processing trace %8d", index)
end
```

4.2.6 linefit

最小二乘线性拟合。函数接受一个或两个向量参数，其长度必须大于等于 2。如果只提供一个参数，则使用从 1 开始的索引作为拟合参数 x。

```
a, b = linefit(y)       -- y=a*i+b, 返回a,b
a, b = linefit(y, x)    -- y=a*x+b
```

如果需要计算拟合均方根误差，可以用如下方法

```
a, b = linefit(y, x)    -- y=a*x+b
e    = y-(a*x+b)        -- e 是拟合误差向量
rms  = e:rms()          -- 均方根误差
max  = e:abs():max()    -- 最大误差
```

该函数可以用来反推观测系统，例如拟合 cmp_x 与 cmp 道头可以获得 cmp 网格尺寸。拟合大地坐标和局部坐标系中坐标之间的关系，可以反推坐标旋转和平移量（5.2），在分析和验证数据时提供便利。

4.2.7 TODO-geomfit

```
x0, y0, angle = geomfit(gx, gy, lx, ly)
-- m is 2x2 旋转矩阵
```

拟合旋转矩阵和平移量。注：因为存在 x0,y0，无法直接最小二乘拟合旋转矩阵，可以选择数据中任意点为参考点，使用相对参考点的坐标（本地-参考点本地坐标），（全局-参考点全局坐标）从而从方程中先排除 (x0,y0)，计算出旋转矩阵，然后再求解 (x0,y0)。

4.2.8 TODO-spline

增加 spline 拟合函数!

```
lparam = spline_fit(ytab, xtab)
y = spline_eval(lparam, x)    -- x>=xtab.min() and x<=xtab.max()
```

4.2.9 match

已实现 (用在交叉鬼波脚本，文档需完善)。返回两个向量或矩阵之间的匹配滤波函数，及匹配后的结果。

```
filter, mathed_a = match(a, b, flen)
filter, mathed_a = match(a, b, flen, white_noise)
--estimate least square filter of length flen to match a to b
-- filter, matched_a = a:match(b [, flen])
-- if ignore, flen=1, in number of points, round up to odd!
-- white noise level, 0.01 for 1%
```

匹配滤波可能不稳定，尤其是在信号频带比目标窄时，需要求解时加入白噪稳定输出的滤波器。(将给叶总的测试结果图件加入此处文档)。

注意：如果 a 和 b 是 matrix，做特别处理！或者作为方法使用，`filter, matched_b=b:match(a)` 作为仅接受向量的函数，结合`:copy()` 方法，使用更灵活！考虑增加 reshape(resize) 函数！使用 resize!

4.2.10 gsplit

依据某个整型道头将其它道头或数据分组，例如使用 `i_user1`，将 `i_user1==0` 的道和 `i_user1~=0` 的道分成两个虚拟道集。如下将两个道头和数据分组

```
shotnum0, shotnum1  = gsplit(i_user1, source_no)
offset0,  offset1   = gsplit(i_user1, offset)
seismic0, seismic1  = gsplit(i_user1, seismic)
assert(#offset0+#offset1==#offset)  -- 长度关系
```

其中 `shotnum0`、`offset0` 和 `seismic0` 一一对应于原道集中 `i_user1==0` 的道；另一组则对应于原道集中 `i_user1~=0` 的道。当然该函数也可以用来对 `i_user1` 自身进行分组，或者使用任何其它道头进行分组。分组后的名字可以为任意变量名，例子中的名字选取为便于记忆，非强制要求。

```
i_user1_0,  i_user1_1 = gsplit(i_user1, i_user1)
```

这样分组的意义是什么呢? GeoEast 不支持 CGG Geovation2 中多个 `TRACE-vector` 的概念，通过分组可以近似模拟该概念。例如，在进行 PZ 求和时，假设已经进行振幅和相位匹配矫正，如果 P 对应的地震数据 `i_user1` 标记为 0，Z 对应的地震数据 `i_user1` 标记为 1，然后将它们合并入同一个道集内，并按照一一对应的道集排序方式，确保一个道集内有等量的 P 和 Z 道，然后进行分离、处理然后合并

```
pseis, zseis = gsplit(i_user1, seismic)
for i=1, #pseis do
    up = (pseis[i]+zseis[i])/2.0
    dw = (pseis[i]-zseis[i])/2.0
    pseis[i] = up
    zseis[i] = dw
end
seismic = gmerge(i_user1, pseis, zseis)    --proposal
```

经过上述指令处理后选取 i_user1 值为 0 的部分对应上行波 up，值为 1 的部分对应下行波 dw。上述分选合并仅需针对需要处理的道头和数据，不关心的道头无需特别处理即维持原来的对应关系。

如果 i_user1 中包含 0,1,2 三种数值，需要进行三分组，可以使用下面的办法

```
function gsplit3(mark, head)
    head0, headx = gsplit(mark, head)
    mark0, markx = gsplit(mark, mark)
    markx = markx-1
    head1, head2 = gsplit(markx, headx)
    return head0, head1, head2      -- 注意三分组的合并需要特别处理!
end
```

4.2.11 gmerge

依据某个整型道头将原来使用 gsplit 分组的道头或数据合并。使用方法如下

```
shotnum0, shotnum1  = gsplit(i_user1, source_no)
seismic0, seismic1  = gsplit(i_user1, seismic)
-- 任意处理，但不要改变向量长度!
source_no = gmerge(i_user1, shotnum0, shotnum1)
seismic   = gmerge(i_user1, seismic0, seismic1)
```

针对上述定义的三分组 gsplit3 的合并函数如下

```
function gmerge3(mark, head0, head1, head2)
    mark0, markx = gsplit(mark, mark)
    markx = markx-1
    headx = gmerge(markx, head1, head2)
    head  = gmerge(mark, head0, headx)
    return head
end
```

4.3 向量

向量即一维数组，同时具有一系列数组操作和信号处理相关的方法。道集的道头被映射为向量。道集中每一道地震数据也被映射为向量，经过转秩，道集中地震数据的时间或深度切片也可以被映射为向量。向量支持 $+-\times\div$ 和指数运算符。向量的长度使用 # 操作符获取。例如 #offset, #sp_x, #source_no 等，均表示对应道头向量的长度，即道集中的道数。

4.3.1 zero

该函数复制指定向量 (不改变原向量) 并将其全部或部分清零后返回，它可以接受 0、1 或者 2 个参数，分别对应清除全部，头部和区间 (包含) 范围。注意该方法返回的向量长度始终与原向量相同。使用方法如下：

```
vector2 = vector1:zero()          -- 复制vector1并将所有元素清零返回，vector1自己不变
vector2 = vector1:zero(z)         -- 复制vector1并将从1(包含)到z(包含)元素清零返回
vector2 = vector1:zero(a,z)       -- 复制vector1并将从a(包含)到z(包含)元素清零返回
```

如果希望将 vector1 清零，需要将新向量赋值给 vector1，如下面第 1-3 行所示；如果希望切除 1 秒内的所有地震数据且无渐变过度，可以像下面第 4-7 行所示：

```
f_user1 = f_user1:zero()              -- 道头f_user1清零
i_user1 = i_user1:zero(#i_user1)      -- 道头i_user1清零
i_user2 = i_user2:zero(1, #i_user2)   -- 道头i_user2清零
imute = tosample(1.0)-1               -- 小于1000毫秒的最大采样点
for i=1, #seismic do
    seismic[i] = seismic[i]:zero(imute) -- 地震数据切除至1000毫秒
end
```

当然也可以使用之前的函数创建一个新的相同长度零向量并赋值或者向量运算：

```
f_user1 = vector(#f_user1)        -- 道头f_user1清零
i_user1 = i_user1*0               -- 道头i_user1清零
i_user2 = i_user2-i_user2         -- 道头i_user2清零
```

4.3.2 copy

该函数复制指定向量并返回，它可以接受 0、1 或者 2 个参数，分别对应复制全部、头部和区间 (包含) 范围。其使用方法如下面所示

```
vector2 = vector1:copy()          -- 复制vector1所有元素返回，vector1自己不变
vector2 = vector1:copy(z)         -- 复制vector1从1(包含)到z(包含)部分并返回
vector2 = vector1:copy(a,z)       -- 复制vector1从a(包含)到z(包含)部分并返回
```

需要注意向量:zero() 方法总是返回同样长度的向量，并将其中指定元素清零，参数将被内部限定在向量长度范围内；而:copy() 方法返回指定范围元素组成的新向量，其长度随参数变化，且参数范围无限定。该函数用于获取指定范围的子集；如果参数给定范围超出给定向量的首尾，则可用于获取**超集**，超出原向量的部分用零元素填充。例如

```
vector2 = vector1:copy(#vector1+5)          -- 尾部补5个零
vector3 = vector1:copy(1-7, #vector1+7)     -- 首尾各补7个零
```

无参数使用方法如下面第 1 行所示。因模块中的向量扩展支持 $+-\times\div$ 和指数运算，其中第 2、第 3、第 4 和第 5 行脚本可以达到完全相同的效果。

```
smart = vector1:copy()
smart = vector1*1
smart = vector1+0
smart = vector1/1
smart = vector1^1
silly = vector1           --引用
```

需要注意第 6 行与其它不同，直接赋值导致 `silly` “引用” `vector1`，即它们实际上指向同一个向量，此时如果改变 `silly` 中元素的值，`vector1` 也会跟着变化，直到这种 “引用” 关系被破坏，例如：

```
print(vector1[1])         --初始值
silly = vector1           --引用
silly[1] = -1*silly[1]
print(vector1[1])         --将变为初始值的负数
silly = anythingelse      --对vector1的引用终止
```

4.3.3 conv

4.3.4 corr

4.3.5 sort

该函数对指定向量进行排序

```
v2 = v1:sort()
```

4.3.6 abs

该函数复制指定向量 (不改变原向量) 并将其全部或部分元素取绝对值返回，它可以接受 0、1 或者 2 个参数，分别对应取全部，头部和区间范围。该方法与:zero() 类似，总是返回同样长度的向量。使用方法如下

```
vector2 = vector1:abs()              -- 复制vector1所有元素并返回其绝对值
vector3 = vector1:abs(z)             -- 复制vector1所有元素并将从1到z(包含)元素取绝对值
vector3 = vector1:abs(a,z)           -- 复制vector1所有元素并将从a到z(包含)元素取绝对值
```

它在效果上等价于下面的函数

```
function vector_abs_func(vector1)
    local vector2 = vector(#vector1)    -- 创建一个等长向量
    for i=1, #vector1 do
        vector2[i] = math.abs(vector1[i])
    end
    return vector2                      -- 返回
end
```

如果实现了向量比较操作符号重载和 where 方法，向量的 abs 方法等效于

```
vector2 = vector1:where(vector1>=0, -1*vector1)
```

4.3.7 agc

TODO: add test job!

4.3.8 rms

该函数计算指定向量中所有元素或部分元素的均方根值，使用方法如下

```
value1 = vector1:rms()               --计算向量所有元素的均方根
value2 = vector1:rms(z)              --计算向量从1到z(包含)元素的均方根
value3 = vector1:rms(a,z)            --计算向量从a到z(包含)元素的均方根
```

如果希望计算向量指定范围元素的均方根值，也可以结合前面:copy() 方法，用指定范围元素产生一个临时向量，然后计算其均方根。例如:

```
value4 = vector1:copy(z):rms()       --计算向量从1到z(包含)元素的均方根
value5 = vector1:copy(a,z):rms()     --计算向量从a(包含)到z(包含)的均方根
```

上面 value4 与之前 value2，value5 与之前 value3 计算方法结果完全相同，但因为串联两个方法计算效率略低。如果希望计算每一道地震数据的 rms 并存储在 f_user1 中，可以像下面这样:

```
for i=1, #seismic do
    f_user1[i] = seismic[i]:rms()
end
```

单个地震道，垂直的时间 (t) 采样方向，所有采样构成一个向量。道集的每个道头，水平的空间 (x) 采样方向，所有采样也构成一个向量，如图-3所示，同样可以调用 rms 函数，例如假设单道的均方根存储在道头 f_user1 中，可以像下面计算并打印每个道集的均方根：

```
gather_rms = f_user1:rms()            --gather_rms为临时变量，并非道头
list("Group=%4d RMS=%e\n", GROUP, gather_rms)
```

4.3.9 min

该函数返回向量的最小值和该最小值首次出现的索引位置。它可以接收 0 到 2 个参数，并且可以返回 1 到 2 个参数，使用方法如下所示

```
value, index = vector1:min()        --返回vector1中的最小值和首次出现的索引位置
value, index = vector1:min(a)       --返回vector1从1到a(包含)中的最小值和索引
value, index = vector1:min(a,b)     --返回vector1从a(包含)到b(包含)中的最小值和索引
```

实际使用中可以忽略第二个返回值。下列示例代码中第 1、2 和 3 行效果相同。

```
value = vector1:min()
value = vector1:min(#vector)
value = vector1:min(1, #vector)
value, index = vector1:min()
value = vector1:min(100)            --vector[1]->vector[100]中的最小值
value = vector1:min(9, 300)         --vector[9]->vector[300]中的最小值
```

4.3.10 max

该函数返回向量的最大值和该最大值首次出现的索引位置。其参数和用法与 min 方法完全相同。

4.3.11 meanvalue

该函数返回向量的均值。

```
value = vector1:meanvalue()         --返回vector1的均值
value = vector1:meanvalue(z)        --返回vector1从1到z(包含)的均值
value = vector1:meanvalue(a,b)      --返回vector1从a(包含)到z(包含)的均值
```

4.3.12 mean

该函数对向量进行均值滤波，它可以接受 1 或者 2 个参数，分别为滤波器点数和边界处理方法，滤波结果作为新的向量返回。使用方法如下

```
new_vector = vector:mean(npoints)
new_vector = vector:mean(npoints[, 'copy'|'mirror'|'clip'])
```

滤波器点数如不是奇数，则将被向上越近至奇数。滤波窗口以输出采样点为中心对称。如果第 2 个参数 (字符串) 为 copy，在边界附近参滤波窗口超出边界时，边界外采样点取边界上采样点值。如果参数为 mirror，则边界外的采样取边界内镜像点的值。如果参数为 clip，则滤波窗口滑动至边界时，受边界限制，不能超出边界，此时滤波窗口不再以输出采样为中心对称点。

4.3.13 median

该函数对向量进行中值滤波，它的使用方法及参数的意义和 mean 完全一样

```
new_vector = vector:median(npoints)
new_vector = vector:median(npoints[, 'copy'|'mirror'|'clip'])
```

中值滤波对窗口内所有采样点进行排序，取排序后中间点的值作为该窗口输出。与均值滤波相比，中值滤波能够更好的去除数据中的奇异点。

4.3.14 alpha_trim

该函数对向量进行 alpha_trim 滤波，它结合了均值和中值滤波的特点，在对滤波窗口内的采样点进行排序后，排除首尾两端比例为 alpha 的采样点后，对剩余的采样点取平均。使用方法如下

```
new_vector = vector:alpha_trim(npoints, alpha)      -- alpha in [0, 1]
new_vector = vector:alpha_trim(npoints, alpha [, 'copy'|'mirror'|'clip'])
```

当 alpha=0 时，窗口内所有采样点参与平均，该函数与均值滤波相同，当 alpha=1 时，窗口内除中心点外所有采样点均被排除，该函数与中值滤波相同。

4.3.15 top_trim

该函数与 alpha_trim 滤波类似，在对滤波窗口内的采样点进行排序后，排除**大值端**比例为 alpha 的采样点后，对剩余的采样点取平均。使用方法如下

```
new_vector = vector:top_trim(npoints, alpha)        -- alpha in [0, 1]
new_vector = vector:top_trim(npoints, alpha [, 'copy'|'mirror'|'clip'])
```

当 alpha=0 时，窗口内所有采样点参与平均，该函数与均值滤波相同，当 alpha=1 时，窗口内除最小值外所有采样点均被排除。

4.3.16 bottom_trim

该函数与 top_trim 类似，在对滤波窗口内的采样点进行排序后，排除**小值端**比例为 alpha 的采样点后，对剩余的采样点取平均。使用方法如下

```
new_vector = vector:bottom_trim(npoints, alpha)     -- alpha in [0, 1]
new_vector = vector:bottom_trim(npoints, alpha [, 'copy'|'mirror'|'clip'])
```

当 alpha=0 时，窗口内所有采样点参与平均，该函数与均值滤波相同，当 alpha=1 时，窗口内除最大值外所有采样点均被排除。

4.3.17 ricker

该函数在向量上指定位置叠加用户自定义频率和振幅的 Ricker 子波。如果忽略第 4 个参数，默认使用地震数据的时间采样间隔。如果指定采样间隔，则 t0 和 si 必须使用国际单位制。使用方法如下

```
vector1 = vector1:ricker(t0, pf, amp [, si])
-- t0  : 叠加时间(s)或位置(m), Ricker子波的零点位置
-- pf  : 峰值频率(Peak Frequency)
-- amp : 峰值振幅(Amplitude)
-- si  : 向量采样间隔(Sample Interval), 国际单位制
```

每调用一次该函数，叠加一个子波。如果需要加入多个 Ricker 子波，反复调用该函数即可。如果需要在道头上叠加该子波，需要将频率和波数，时间采样率和空间采样率进行对应。如果在每一道 300ms 和 500ms 分别放置一个峰值频率为 15Hz 和 20Hz，振幅为 3 和 1 的 Ricker 子波，可以使用下面的示例

```
for i=1,#seismic do
    seismic[i] = seismic[i]:ricker(0.3, 15, 3)  -- 0.3秒, 15Hz, 振幅3
    seismic[i] = seismic[i]:ricker(0.5, 20, 1)  -- 0.1秒, 20Hz, 振幅1
end
```

4.3.18 ormsby

该函数在向量上指定位置叠加用户自定义频率和振幅的 Ormsby 子波。如果忽略 amp 振幅参数，默认值为 1；如果指定采样间隔，则 t0 和 si 必须使用国际单位制。使用方法如下

```
vector1 = vector1:ormsby(t0, f1, f2, f3, f4[, amp, si])
-- t0  : 叠加时间(s)或位置(m), Ricker子波的零点位置
-- amp : 峰值振幅(Amplitude), 默认为1.0
-- si  : 向量采样间隔(Sample Interval), 国际单位制
```

模块使用的由频率点 f_1, f_2, f_3, f_4(单位 Hz) 定义的 Ormsby 子波表达式如下：

$$A(t) = \frac{\pi f_1^2}{f_2 - f_1} sinc^2(f_1 t) - \frac{\pi f_2^2}{f_2 - f_1} sinc^2(f_2 t) - \frac{\pi f_3^2}{f_4 - f_3} sinc^2(f_3 t) + \frac{\pi f_4^2}{f_4 - f_3} sinc^2(f_4 t) \tag{1}$$

实例可参考 Ricker 子波。

4.3.19 ofilter

该函数使用零相位 Ormsby 子波作为带通滤波器，返回滤波后的向量。其中 $f_2 \to f_3$ 为通带，小于 f_1 和大于 f_4 为阻带，频率参数从小至大，单位为 Hz。使用方法如下

```
vector1 = vector1:ofilter(f1, f2, f3, f4)
-- f2-f3  : 通带范围
```

如果需要分别获取通带和阻带信号，可以使用如下方式

```
pass = vector1:ofilter(f1, f2, f3, f4)
stop = vector1-pass      --阻带信号
```

为了提升计算效率，模块同时提供了对应的矩阵方法。下面两种滤波方式等效，但使用矩阵方法计算效率更高，且模块可能会尝试利用多线程进行并行，进一步缩短运行时间

```
for i=1, #seismic do
    seismic[i] = seismic[i]:ofilter(f1, f2, f3, f4)
end
```

```
seismic = seismic:ofilter(f1, f2, f3, f4)    --矩阵方法
```

注. 详见矩阵:ofilter(4.4.11) 方法。

4.3.20 fft

快速傅里叶 (Fourier) 变换。将向量变换到频率域，通常变换的结果在复数域，为方便地震资料处理，此处将复数表示为振幅和相位组成的向量。Fourier 变换函数接受一个可选参数，表示正变换的长度。如果变换长度小于向量，则多余采样点被忽略；如果变换长度大于向量，则不足采样点补零；默认变换长度为**向量长度**。变换函数的使用方法如下

```
amplitude, phase = vector1:fft()              --按大于或等于向量长度的最小偶数变换
amplitude, phase = vector1:fft(n)             --按长度n进行Fourier变换
amplitude, phase = vector1:copy(n):fft()      --等同于上一行
amplitude, phase = vector1:fft(#vector1)      --按向量长度进行变换
```

傅里叶变换计算等效于公式-2，公式中 x_n 为长度 N 的实数序列，X_k 为长度 N 的复数序列。

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi}{N}kn} \tag{2}$$

实数傅里叶变换关于零点共轭对称，即 $X_i = \bar{X}_{N-i}$。如果向量的长度为 nr；只需 $[\frac{nr}{2}]+1$ 个复频率表示从 0 到奈奎斯特 (Nyquist)，与常用 `fftw` 和 `mkl` 等软件库一致。

```
nr = #vector1                                 --变换长度
amplitude, phase = vector1:fft()              --变换至频率域
nc = #amplitude                               --频率域长度
assert(nc==math.floor(nr/2)+1 and nc==#phase)    --长度关系
```

其中振幅和相位表达的复数与实部和虚部表达的对应关系如下

$$R + iI = Ae^{i\phi} \tag{3}$$

由于傅里叶变换的周期性，在进行相移，卷积等操作后，结果的长度可能超过输入向量的长度，在没有补零时，圆卷积的结果超过输入向量原始长度的部分会从尾部折叠到开始位置，形成信号混叠。线性卷积需要通过补零的方式避免结果发生混叠。圆卷积和线性卷积算法示意如下：

```
amp1, pha1 = vector1:fft(#vector1)      --amp1,pha1 长度为[#vector1/2]+1
amp2, pha2 = amp1*amp1, pha1+pha1       --振幅相乘、相位相加，卷积定理
vector2 = amp2:ift(pha2, #vector1)      --圆卷积结果，vector3 长度为#vector1
--------------------------------------------------------------------------
amp3, pha3 = vector1:fft(2*#vector1-1)                  --变换长度2*#vector1-1
amp4, pha4 = amp3*amp3, pha3+pha3                       --卷积定理
vector4 = amp4:ift(pha4, 2*#vector1-1):copy(#vector1)   --线性卷积结果，长度#vector1
```

为了提升计算效率，模块同时提供了对应的矩阵方法。例如下面道集中每一道都需要进行变换时

```
m, ns = #seismic, #seismic[1]
nfreq = math.floor(ns/2)+1
amp, pha = matrix(m, nf), matrix(m, nf)
```

```
for i=1, #seismic do
    amp[i], pha[i] = seismic[i]:fft(ns)
end
```

下面的方法可以获得相同效果，且模块有机会利用并行提升计算效率，参考矩阵 `fft1d`(4.4.4) 方法

```
amp, pha = seismic:fft1d(#seismic[1])    --矩阵方法
```

4.3.21 ift

快速傅里叶反变换。将振幅向量反变换回时间或空间域。第一个参数为等长向量表示振幅对应的相位；第二个参数为可选，表示变换长度；使用方法如下

```
vector = amplitude:ift(phase)        -- 按2*#amplitude-2长度进行反变换
vector = amplitude:ift(phase, n)     -- n一般取fft正变换长度
```

其中第二个参数如果与正变换保持一致，则正反变换算子互逆。经过正反变换后向量的差异仅为数值计算误差。在下面例子中，如果 `n`≥`#vector1`，则两个向量的值相同 [2]

```
amplitude, phase = vector1:fft(n)           --按长度n进行Fourier变换
vector2 = amplitude:ift(phase, n)           --按长度n反变换
assert(#vector2==n)                         --注意vector2的长度!
```

如果希望保障正反变换后长度保持一致，可以使用下面两种方法。注意因其变换长度不同，对应的物理意义稍有区别，根据实际需要选择

```
amplitude, phase = vector1:fft(#vector1)         -- 按向量长度正变换
vector2 = amplitude:ift(phase, #vector1)         -- 按向量长度反变换
-------------------------------------------------------------------
amplitude, phase = vector1:fft(n)                   -- 按长度n正变换
vector3 = amplitude:ift(phase, n):copy(#vector1)    -- 按长度n反变换并对齐
```

参考矩阵 `ift1d`(4.4.5) 方法获得更高的计算效率。

4.3.22 cfft

复数域傅里叶变换。地震数据全部为实数，复数一般仅在计算过程中出现，且只能记录为实部和虚部，或者振幅与相位。实数 Fourier 变换后在频率域为共轭对称，`v:fft`(4.3.20) 省略了庸余部分后长度约为变换前一半；`v:cfft()` 按变换公式全部保留，变换前后长度相同。

```
amplitude, phase = vector1:cfft()              -- vector1 为振幅
amplitude, phase = vector1:cfft(phase1)        -- phase1 为相位
```

示例中第一行 `vector1` 表示变换前振幅，即 $Ae^{i\phi}$ 中的 A 部分；因未提供相位谱参数，默认为零相位，即 $\phi = 0$；此时振幅与实部等价。第二行使用相位变量作为变换参数。

[2] 此处相同为数学物理意义上相同，数值上由于计算过程中浮点数的表示精度限制，可能造成少许不同，因此应在条件判断中应避免使用浮点数，例如 `if vector1[1]==vector2[1] then ...` 可能得到不合预期的结果

4.3.23 cift

复数域傅里叶反变换。如果变换前的数据构成共轭对称序列，则反变换后虚部为零，即 $\phi = 0$。如果对 Fourier 和傅里叶变换的计算方式存在疑问，可以参考 FFTW 或者 MKL 文档。

```
amp2, pha2 = amplitude:cift()                    -- vector1 为振幅谱
amp2, pha2 = amplitude:cift(phase)               -- phase1 为相位谱
```

4.3.24 cabs_arg

使用欧拉公式 $e^{i\phi} = cos(\phi) + isin(\phi)$，将复数从实部虚部表达方式转为振幅和相位。

```
amplitude, phase = real1:cabs_arg()              -- real1 为实部
amplitude, phase = real1:cabs_arg(imag1)         -- imag1 为虚部
```

4.3.25 creal_imag

将复数从振幅相位表达方式转换为实部和虚部表达方式。

```
real1, imag1 = amplitude:creal_imag()            -- real1 为实部
real1, imag1 = amplitude:creal_imag(phase)       -- imag1 为虚部
```

4.3.26 shift

该函数对向量进行平移。如果向量为时间采样序列，则 shift 为时移，如果为空间采样，则该函数进行空间平移。移动量可以是整数或者浮点数。如果是时移，平移量的单位是秒，采样间隔默认与地震信号相同。如果是空间平移，单位是米，必须提供空间采样间隔。使用方法如下

```
vector2 = vector1:shift(t)               -- t为平移时间，正负t平移方向相反
vector2 = vector1:shift(t, si)           -- si采样间隔，单位@秒@
vector2 = vector1:shift(x, dx)           -- x,dx 空间单位@米@
```

常用的对地震信号进行时移仅需使用一个参数即可。当信号平移超过向量尾部时会折叠返回到向量头部，这是通过相移进行平移时的特性，该特性在后面自相关计算时会有其应用。如果希望避免折叠，可以在 shift 操作前补零，例如

```
n = #vector1
vector2 = vector1:copy(2*n):shift(t):copy(n)     -- 补零延长、时移再截断
assert(#vector2==n)                              -- 长度一致
```

这里使用国际单位制定义平移量，因为向量既可以是时间方向的采样，也可能是空间方向的采样。如果是空间平移，参数为移动距离和空间采样率。在后面拖缆检波器移动矫正 (Receiver Motion Correction) 示例中将详细演示空间平移的使用方法。

4.3.27 ishift

该函数对向量进行平移。平移量将四舍五入至最近整数采样点，不对数据进行任何数学运算，其好处是速度更快且完全可逆。使用方法如下：

```
vector2 = vector1:ishift(t)              -- t为平移时间，正负t平移方向相反
vector2 = vector1:ishift(t, si)          -- si采样间隔，单位@秒@
vector2 = vector1:ishift(n, 1)           -- 平移n个采样点
```

用于整数点时移，上下扩边。前面介绍的 shift(4.3.26) 方法使用频率域相移法或时间域插值滤波器（视具体实现方式而定），在增加并去除相同时移后存在少量数值计算误差；而 ishift 为整数采样点的移动复制，不存在任何数学运算。在适用的场合应优先使用 ishift 方法。

4.3.28 inst

该函数计算向量的瞬时振幅和瞬时相位，瞬时振幅表示信号的能量包络。使用方法如下

```
amplitude, phase = vector1:inst()
vector2 = vector(#vector1)
vector3 = vector(#vector1)
for i=1, #amplitude do
    vector2[i] = amplitude[i]*math.cos(phase[i])    -- 原始信号
    vector3[i] = amplitude[i]*math.sin(phase[i])    -- Hilbert变换
end
```

对于给定信号 $s(t)$，其希尔伯特变换记为 $\hat{s}(t)$，则可以得到其分析信号

$$s_A(t) = s(t) + j\hat{s}(t) \tag{4}$$

$$s_A(t) = A(t)e^{j\psi(t)} \tag{5}$$

其中 $A(t)$ 和 $\psi(t)$ 分别对应瞬时振幅和瞬时相位，即 `inst()` 方法的两个返回值。如果需要获取信号的瞬时频率，可以由下面的公式计算

$$\nu(t) = \frac{1}{2\pi}\frac{d\psi}{dt} \tag{6}$$

根据欧拉公式，可以得到变换关系，也可以用来计算信号的希尔伯特变换

$$s(t) = A(t)\cos(\psi(t)) \tag{7}$$

$$\hat{s}(t) = A(t)\sin(\psi(t)) \tag{8}$$

4.3.29 sampler-proposal

提议，似乎已经实现，有待进一步验证。该函数将向量视一个连续函数的采样，重构该函数并获取任意点的采样。采样可以取任意点，内部使用较高精度插值方法实现。

```
avalue = seismic[1]:sampler(fsample(0.1354))    -- 135.4ms采样值
vector = seismic[1]:sampler(indices)            -- 数组向量
```

4.3.30 vmap

提议，用于 NMO 及类似算法。后续加入文档。

```
avalue = seismic[1]:vmap(map)    -- 135.4ms采样值
```

表 2: 比较函数

lt	<	Less Than	小于
le	<=	Less than or Equal	小于等于
gt	>	Greater Than	大于
ge	>=	Greater than or Equal	大于等于
eq	==	EQual	等于
ne	~=	Not Equal	不等于

4.3.31 比较函数

比较函数用于比较向量和标量或者和另一个向量，并返回比较结果，支持的比较函数如表-2所示。为了与 GeoEast 道头兼容，我们选择使用 0 表示 false，使用 1 表示 true，在条件判断中需要注意。

下面的实例演示了“小于”方法和它的等效代码。

```
vector3 = vector1:lt(vector2)
function less_than(vector1, vector2)
    assert(#vector1==#vector2)
    vector3 = vector1:zero()     -- 清零
    for i=1, #vector1 do
        if vector1[i]<vector2[i] then
            vector3[i]=1
        end
    end
    return vector3
end
```

4.3.32 数学运算

和比较运算一样，列表

4.3.33 where

参考 numpy.where 函数，需要配合 lt,gt,le,ge,eq,ne 等向量比较运算符的重载。能用，但功能有待进一步完善。

4.4 矩阵

矩阵即二维数组，同时具有一系列二维数组操作和信号处理相关的方法。道集中的地震数据被整体映射为矩阵 seismic。矩阵第一维的长度 #seismic 为道集的道数。其中 seismic[i] 表示第 i 道数据组成的向量，每一道的采样点数相同，可以用 #seismic[i] 来获取。因为道集的道数不确定，但至少有一道，通常使用 #seismic[1] 来获取采样点个数。

4.4.1 zero

该函数复制指定矩阵并将其全部或部分清零后返回，它可以接受 0、2 或者 4 个参数，分别对应清除全部，头部和区间 (包含) 范围。使用方法如下：

```
matrix2 = matrix1:zero()                -- 全部清零
matrix2 = matrix1:zero(x1,t1)           -- 元素[1,1]->[x1,t1]部分清零
matrix2 = matrix1:zero(x1,t1,x2,t2)     -- 元素[x1,t1]->[x2,t2]部分清零
```

如果希望将地震数据所有道的前 100 个和最后 100 个采样全部切除[3]：

```
ntraces = #seismic                      -- 道集内道数
nsample = #seismic[1]                   -- 第一道采样数
seismic = seismic:zero(ntraces,100)     -- 从第一道至最后一道前100个采样
seismic = seismic:zero(1,nsample-99,ntraces,nsample)    -- 最后100个采样
```

当然也可以用下面的循环每一道和每一个采样点的方式完成，执行速度稍慢一些：

```
for i=1, ntraces do
    for j=1, 100 do                 -- 前100个采样，假设道长度足够，数组不越界
        seismic[i][j] = 0
    end
    for j=nsample-99,nsample do     -- 后100个采样
        seismic[i][j] = 0
    end
end
```

4.4.2 copy

该函数复制指定矩阵并返回，它可以接受 0、1、2 或者 4 个参数，分别对应复制全部、头部和区间 (包含) 范围。使用方法如下：

```
matrix2 = matrix1:copy()                -- 全部复制(m,n)
matrix2 = matrix1:copy(x2)              -- 复制[1,1]->[x2,n]部分
matrix2 = matrix1:copy(x2,t2)           -- 复制[1,1]->[x2,t2]部分
matrix2 = matrix1:copy(x1,t1,x2,t2)     -- 复制[x1,t1]->[x2,t2]部分
```

矩阵的:zero() 方法总是返回同样长度的矩阵，将其中指定元素清零，参数将被限定在合理范围内。而:copy() 方法返回指定范围元素组成的新矩阵。该函数用于获取指定范围的子集；如果参数给定范围超出给定向量的首尾，则可用于获取**超集**，超出部分用零元素填充。例如

```
m, n    = #matrix1, #matrix1[1]         -- 矩阵尺寸
matrix2 = matrix1:copy(m+5, n+5)        -- 尾部补5个零
matrix3 = matrix1:copy(1-7,1-7,m+7,n+7) -- 四周镶边7个零
```

矩阵暂不支持 $+ - \times \div$ 运算。与向量一样，需要注意复制和引用的不同！

```
smart = matrix1:copy()
silly = matrix1          --引用，改变silly的元素，matrix1跟着改变
```

[3]假设采样个数满足切除要求

4.4.3 transpose

该函数对矩阵进行转秩操作并返回。使用方法如下：

```
matrix2 = matrix1:transpose()
```

道集处理中，seismic 的每一列对应一道，同时也可以作为向量对象，使用向量的所有方法函数，这时函数作用于时间方向的向量。转秩之后的矩阵，每一列对应原来的时间或深度切片，这时可以用向量的方法函数进行切片处理，最后再转秩回到输出对应的形态。具体可参照后面的 mean，median 例子。

4.4.4 fft1d

加入例子，参考 (4.3.20)。

4.4.5 ift1d

加入例子，参考 (4.3.21)。

4.4.6 fft2d

该函数计算矩阵的二维傅里叶变换。变换结果为复频率域，为了方便处理，将其表示为振幅和向量组成的矩阵。具体可以参考向量的傅里叶方法。使用方法如下，

```
amplitude, phase = matrix1:fft2d()          -- 按默认尺寸进行FFT变换
amplitude, phase = matrix1:fft2d(m, n)      -- 按(m,n)进行FFT变换
assert(#amplitude==m and #amplitude[1]==math.floor(n/2)+1)
```

二维傅里叶变换等效于两个一维傅里叶变换，低维与向量:fft() 方法一样仍然是实数到复数变换，长度关系类似，即默认变换长度为**大于或等于向量长度的最小偶数**。高维为复数到复数变换，前后长度不变，默认在输出时转换为振幅和相位。上述使用方法与下面的例子完全等价，用户可以根据习惯选择

```
amplitude, phase = matrix1:fft2d()                      -- 按默认尺寸进行FFT变换
m = #matrix1
n = #matrix1[1]+#matrix1[1]%2                           -- 大于等于的最小偶数
amplitude, phase = matrix1:fft2d(m, n)                  -- 与前面第1行等价
amplitude, phase = matrix1:copy(m, n):fft2d()           -- 与前面第2行等价
```

使用中需要注意，傅里叶变换的时间零点为第一个时间采样点，空间零点为第一道 (第一个空间采样点)。用户需按上述要求自行分选排序。

4.4.7 ift2d

该函数计算矩阵的二维傅里叶反变换。使用方法如下，

```
matrix1 = amplitude:ift2d(phase)          -- 按默认尺寸进行FFT反变换
assert(#matrix1==#amplitude and #matrix1=2*(#amplitude-1))
matrix2 = amplitude:ift2d(phase, m, n)    -- 按(m,n)进行FFT反变换
assert(#matrix2==m and #matrix2[1]==n)
```

4.4.8 cfft2d

已实现，增加文档。

```
amplitude2, phase2 = matrix1:cfft2d()
amplitude3, phase3 = matrix1:cfft2d(phase1)
```

4.4.9 cift2d

已实现，增加文档。

```
amplitude2, phase2 = matrix1:cift2d()
amplitude3, phase3 = matrix1:cift2d(phase1)
```

4.4.10 ishift

4.4.11 ofilter

参考向量 `ofilter(4.3.19)`

4.4.12 obliquity

在 `head_lib.lua` 里实现的简单 obliquity 矫正。

```
-- dx   : trace interval
-- vref : reference (water) velocity
-- smax : maximum scale
seismic = obliquity(seismic, dx, vref, smax)
```

注意：该方法仅用于验证和演示 FK 域矫正方法。规划中倾斜矫正将可以与鬼波压制算子合并在稀疏 Taup 域实现，这样可以避免存在空间假频时出射角度变换映射错误问题，且矫正时可以更好的控制噪音。

4.4.13 extrap

简单二维波场延拓。注意：该方法仅用于培训和演示。如需要该功能请求证其它 `GeoEast` 模块。

4.4.14 sampler2d-proposal

4.4.15 mean-deprecated

因为该方法已经不建议使用，不再详细说明。该方法如果作用于 `seismic`，等效于横向 (时间或深度切片) 平滑。替代方案如下

```
Tseis = seismic:transpose()
for i=1, #Tseis do
    Tseis[i] = Tseis[i]:mean(7)     -- 7点均值滤波，详见vector:mean()
end
seismic = Tseis:transpose()
```

矩阵的每一列均可以作为向量使用。对于道集中的地震采样，每一列向量代表一道，此时向量的方法作用于时间或深度方向。转秩`:transpose()` 可以方便的操作时间或者深度切片向量。

4.4.16 median-deprecated

因为该方法已经不建议使用，不再详细说明。该方法如果作用于 seismic，等效于横向 (时间或深度切片) 平滑。替代方案如下

```
Tseis = seismic:transpose()
for i=1, #Tseis do
    Tseis[i] = Tseis[i]:median(7)       -- 7点中值滤波，详见vector:median()
end
seismic = Tseis:transpose()
```

替代方案近列举了最简单的参数形式，所有向量的方法，各种参数组合都可以使用。

4.5 滤波器-proposal

增加滤波器函数库包?signal.match(), signal.filter(), signal.butter(), signal.low_pass(), signal.high_pass()。看怎么规划这个东西。叫做 filter 扩展，还是 signal 扩展?

5 应用实例

5.1 观测系统

如果采用向量运算的形式计算观测系统，只能使用加减乘除幂和前面定义的可以接受向量参数的扩展函数和方法。基本观测系统信息如下

```
dy, dx   = 37.5, 12.5           -- 观测系统网格大小37.5X12.5
cmp_x    = (sp_x+gp_x)/2.0      -- 中心点x
cmp_y    = (sp_y+gp_y)/2.0      -- 中心点y
cmp_line = cmp_y/dy
cmp      = cmp_x/dx
offsetx  = gp_x-sp_x            -- 非GeoEast道头，临时变量，向量
offsety  = gp_y-sp_y            -- 非GeoEast道头，临时变量，向量
offset   = (offsetx^2+offsety^2)^0.5
```

向量是 GeoEast 自定义的扩展，因此采用向量运算形式不能使用 Lua 语言的 math 函数库。如果采用标量的形式计算，则可以自由使用，更加灵活

```
dy, dx   = 37.5, 12.5           -- 观测系统网格大小37.5X12.5
for i=1, #cmp_x do
    cmp_x[i]    = (sp_x[i]+gp_x[i])/2.0             -- 中心点x
    cmp_y[i]    = (sp_y[i]+gp_y[i])/2.0             -- 中心点y
    cmp_line[i] = cmp_y[i]/dy
    cmp[i]      = cmp_x[i]/dx
    offsetx     = gp_x[i]-sp_x[i]                   -- 非GeoEast道头，普通临时变量
    offsety     = gp_y[i]-sp_y[i]                   -- 非GeoEast道头，普通临时变量
    offset[i]   = (offsetx^2+offsety^2)^0.5
    offset[i]   = math.sqrt(offsetx^2+offsety^2)    -- 与上一行等效
end
```

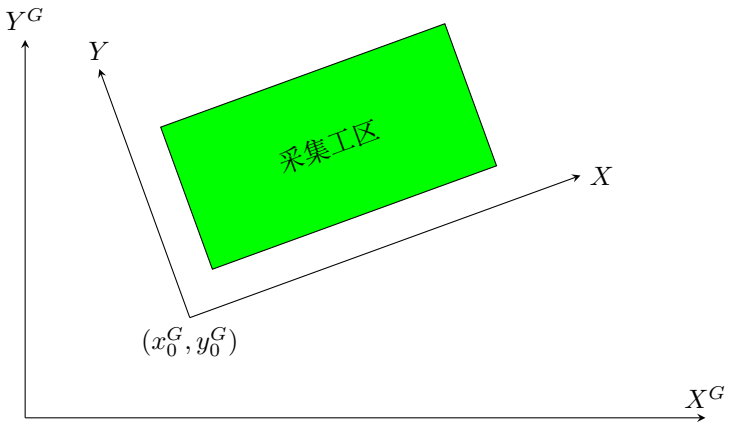

图 4: 本地坐标和全局坐标映射关系

5.2 坐标旋转

如果项目工区在大地坐标 (以下称全局坐标 (X^G, Y^G)) 系统下有一定旋转角，建立一个与采集方向一致的局部坐标系 (x, y) 将简化后续处理。如果假设本地坐标系与全局坐标的逆时针夹角为 θ，坐标原点在全局坐标系中的位置为 (x_0^G, y_0^G)，如图-4所示。坐标变换关系为：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x^G - x_0^G \\ y^G - y_0^G \end{bmatrix} \tag{9}$$

如果假设全局坐标存储在道头 `sp_x` 和 `sp_y`，可用下面的例子对其进行平移和旋转，转化为本地坐标并存储在道头 `sp_x_r` 和 `sp_y_r` 中：

```
x0, y0      = 154334.5, 744123.3        -- 本地坐标原点
theta       = 26.326/180.0*math.pi      -- 角度转弧度
costheta    = math.cos(theta)           -- 临时变量
sintheta    = math.sin(theta)           -- 临时变量
for i=1, #sp_x do
    sp_x_r[i] =  costheta*(sp_x[i]-x0)+sintheta*(sp_y[i]-y0)
    sp_y_r[i] = -sintheta*(sp_x[i]-x0)+costheta*(sp_y[i]-y0)
end
```

对于道头中的其它坐标，逐一变换即可。

5.3 共偏移矩分组

对于共偏移距成像算法，需要对输入数据按偏移距分组。有些宽方位角数据则需要 COV(Common Offset Vector) 或者 COT(Common Offset Tile) 分组，或者按照方位角和偏移距进行分组规则化和偏移

```
step = 200                              -- Offset 分组间隔参数
for i=1, #sp_x do
    dx      = gp_x[i]-sp_x[i]           -- 临时变量, X 偏移距
    dy      = gp_y[i]-sp_y[i]           -- 临时变量, Y 偏移距
    foffset = math.sqrt(dx*dx+dy*dy)    -- 临时变量, 浮点偏移距
    offset_bin[i] = math.floor(foffset/step)        -- 输出至道头 offset_bin
end
```

例子中偏移距范围 $[0, 200)$ 分为第 1 组，范围 $[200, 400)$ 分为第 2 组，以此类推。如果需要 COV 分组，只需分别对两个方向按分组步长进行取整计算即可。如果需要对方位角进行分组，可以用下面的方式计算方位角，然后按照分组步长进行取整运算即可

```
for i=1, #sp_x do
    dx      = gp_x[i]-sp_x[i]           -- 临时变量, X 偏移距
    dy      = gp_y[i]-sp_y[i]           -- 临时变量, Y 偏移距
    if dx==0 then
        azimuth[i] = math.pi/2.0        -- 输出至道头
    else
        azimuth[i] = math.atan(dy/dx)   -- 输出至道头
    end
end
```

该方法计算的方位角范围为 $(-\frac{\pi}{2}, \frac{\pi}{2}]$，如需要角度单位或者范围为 $[0, 2\pi)$，也可很容易实现。

5.4 推测观测系统

使用 `linefit`

5.5 冲击响应估计

如果一个模块可以等效为某种滤波器，我们可以通过将地震数据重置为一个脉冲作为输入，去探测该模块的冲击响应，从而分析其工作原理。但很多模块的工作都是非线性的，即其冲击响应可能随输入数据的振幅分布特征而变化。为应对这种，随机植入脉冲，然后叠加获得冲击响应函数。

5.6 邻道相关

与相邻的下一道进行相关，并将相关零点时移至 500 毫秒采样处。

5.7 地震频谱

本例将地震道从时间域数据变换为频率域振幅谱数据。脚本首先 (L1) 从道集第一道地震数据获取采样点个数；然后 (L2) 从道集第 1 道至最后一道进行循环；(L3) 按照长度 $2 * ns - 1$ 进行傅里叶变换，将振幅谱向量存储在地震数据名下，并忽略相位信息。注意，变换长度 $2 * ns - 1$ 保证傅里叶变换返回的谱向量长度为 ns，与地震数据长度一致，否则将出错。

```
ns = #seismic[1]
for i=1, #seismic do
    seismic[i] = seismic[i]:fft(2*ns-1)
end
```

变换后的振幅谱第 1 个采样点表示 0 频率，最后一个采样点表示奈奎斯特频率，而奈奎斯特频率的具体值由地震数据的采样间隔决定。实例很容易改为输出相位谱。

5.8 船速估计

在拖缆采集中，抽取**相同气枪**阵列的 1 个共 Channel 道集，并按炮序排列。假设放炮时间以秒为单位单调增长记录在道头 `time` 中，则振源船的移动速度可以计算如下

```
for i=1, #sp_x-1 do                          -- 循环第一道至倒数第二道
    dt       = time[i+1]-time[i]             -- 本次和下次放炮间隔
    dx       = sp_x[i+1]-sp_x[i]
    dy       = sp_y[i+1]-sp_y[i]
    dist     = math.sqrt(dx*dx+dy*dy)        -- 气枪移动距离，假设等于船移动距离
    vel      = dist/dt                       -- 移动速度 m/s
    f_user1[i] = vel                         -- 记录到 f_user1 道头中
end
f_user1[#sp_x] = f_user1[#sp_x-1]            -- 假设最后一炮速度未变化
f_user1        = f_user1:median(7)           -- 中值滤波
```

注意：例中如果道集仅有一道数据时将出错，特例并未处理；记录放炮时间的道头可能周期性重置，需要结合日期正确计算时差；非连续采集 (故障中断) 的线束该简化算法无法正常估计。

5.9 检波器移动矫正

`matrix:transpose()` 和 `vector:shift()`，但是需要注意两侧外插一些数据避免数据错误。[4]

5.10 最小相位

`vector:inst()` 进行 Hilbert 变换。

5.11 频谱搬运

`vector:inst()` 清华大学陆文凯 2003 专利。

5.12 动矫正演示

`vector:sampler()` 方法演示。f

5.13 动矫正拉伸

拉伸量计算。

5.14 伪二维平层模型

`vector:ricker()`

[4] `GeoEast` 相关专用模块正在研发，此处仅作为算法原理演示使用。

5.15 伪二维 SRME

5.16 伪二维 MWD

6 临时-ormsby

通带从 f_3 到 f_4 的 Ormsby 子波定义如下：

$$A(t) = \frac{\pi f_1^2}{f_2 - f_1} sinc^2(f_1 t) - \frac{\pi f_2^2}{f_2 - f_1} sinc^2(f_2 t) - \frac{\pi f_3^2}{f_4 - f_3} sinc^2(f_3 t) + \frac{\pi f_4^2}{f_4 - f_3} sinc^2(f_4 t) \tag{10}$$

如果 Fourier 变换定义为下列形式：

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int f(x) e^{-i\omega x} dx \tag{11}$$

则

$$sinc^2(ax) \Longrightarrow \frac{1}{\sqrt{2\pi a^2}} tri(\frac{\omega}{2\pi a}) \tag{12}$$

$$\hat{A(\omega)} = \frac{\pi f_1^2}{f_2 - f_1} \frac{1}{\sqrt{2\pi f_1^2}} tri(\frac{\omega}{2\pi f_1}) - \cdots \tag{13}$$

$$\hat{A(\omega)} = \sqrt{\frac{\pi}{2}} \frac{f_1}{f_2 - f_1} tri(\frac{\omega}{2\pi f_1}) - \cdots \tag{14}$$

7 临时-butterworth

传输函数

$$G(\omega) = |H(j\omega)| = \frac{G_0}{\sqrt{1 + (\frac{j\omega}{j\omega_c})^{2n}}} \tag{15}$$

8 错误信息

`ZMATH` 中常见错误信息解释。特别是部分由 `luaL_checktype()` 给出的错误信息！