关于MIDASr包使用今天的一些学习笔记：

论文(R包）Mixed Frequency Data Sampling Regression Models-The R Package midasr.pdf中 **page 14** 的一个例子：

is willing to generate a low-frequency response variable y in the MIDAS with two higher-frequency series x and z where the impact parameters satisfy the exponential Almon lag polynomials of different orders as follows:

$$y_t = 2 + 0.1t + \sum_{j=0}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=0}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_t, \qquad (12)$$
$$x_{\tau_1} \sim \text{n.i.d.}(0,1), \quad z_{\tau_2} \sim \text{n.i.d.}(0,1), \quad \varepsilon_t \sim \text{n.i.d.}(0,1),$$

where $(x_{\tau_1}, z_{\tau_2}, \varepsilon_t)$ are independent for any $(\tau_1, \tau_2, t) \in \mathbb{Z}^3$, and

$$\beta_j^{(i)} = \gamma_0^{(i)} \frac{\exp\left(\sum_{s=1}^{q_i-1} \gamma_s^{(i)} j^s\right)}{\sum_{j=0}^{d_i-1} \exp\left(\sum_{s=1}^{q_i-1} \gamma_s^{(i)} j^s\right)}, \quad i = 1, 2,$$

where $d_1 = k_1 + 1 = 8$ is a multiple of the frequency ratio $m_1 = 4$, whereas $d_2 = k_2 + 1 = 17$ is not a multiple of $m_2 = 12$. Here $q_1 = 2$, $q_2 = 3$ with parameterizations

$$\boldsymbol{\gamma}_1 = (1, -0.5)^\top,$$
$$\boldsymbol{\gamma}_2 = (2, 0.5, -0.1)^\top,$$

```
set.seed(1001) # 生成随机数种子
n <- 250 # 定义n = 250，代表低频时间周期次数
trend <- 1:n # 定义trend为一个从1到250的向量，代表低频时间序列
x <- rnorm(4 * n) # x为与低频数据trend倍差为 4 的高频向量之一，使用rnorm随机生成
z <- rnorm(12 * n) # z为与低频数据trend倍差为 12 的另一个高频数据向量，使用rnorm随机生成
fn_x <- nealmon(p = c(1,-0.5), d = 8) # p为Almon滞后参数，d为系数个数
fn_z <- nealmon(p = c(2,0.5,-0.1), d = 17) # 同上
y <- 2+0.1*trend+mls(x,0:7,4)%*% fun_x+mls(z,0:16,12)%*% fn_z+rnorm(n)
```

- **nealmon** 函数

> 根据给出的参数和所需系数个数，计算归一化指数Almon滞后系数：返回系数向量
>
> 给出不受函数限制的MIDAS回归：

$$y_t = \sum_{h=0}^{d} \theta_h x_{tm-h} + z_t\beta + u_t$$

> 归一化指数Almon滞后参数以以下方式限制系数θ_h:

$$\theta_h = \delta \frac{exp(\lambda_1(h+1)+\ldots+\lambda_r(h+r)^r)}{\sum_{s=0}^{d} exp(\lambda_1(s+1)+\ldots+\lambda_r(h+1)^r)}$$

> 参数δ是向量p的第一个元素
> 其余参数的个数决定多项式的次数

这里的`fn_x <- nealmon(p = c(1,-0.5), d = 8)` 中
d1 = k1+1 = 8(k1是高频变量x的个数)
m1=4（倍差）

d2=k2+1=17（k2是高频变量z的个数）
m2=12 （倍差）

> p = c(1,-0.5)的估计在后面

MIDAS回归的一些规范示例（3.3）

假设现在我们(只有)对y、x和z的观察，它们被存储为向量、矩阵、时间序列或R中的列表对象，我们的意图是估计一个如式12所示的MIDAS回归模型:
(a)不限制参数(如U-MIDAS)并使用OLS;
(b)参数上的指数Almon滞后多项式约束(nealmon函数)，并使用NLS。
在(a)的情况下，OLS估计直接使用(构建多变量线性回归模型):

```
eq_u <- lm(y ~ trend + mls(x,k=0:7,m = 4) + mls(z,k=0:16,m=12))
```

或者相当于

```
eq_u <- midas_r(y ~ trend + mls(x, 0:7, 4) + mls(z, 0:16, 12), + start = NULL)
```

> 因变量取在~符号的左边，右边由解释变量组成。
> start:优化的初始值。必须是带有命名元素的列表
> data:包含混合频率数据的命名列表

`midas_r`从全局R环境中获取变量。可以显式地传递数据: `#?`

```
eq_u <- midas_r(y ~ trend + mls(x, 0:7, 4) + mls(z, 0:16, 12), + start = NULL, data
= list(y=y, trend=trend, x=x, z=z))
```

相同频率的变量可以存在于相同的data.frame中:

```
eq_u <- midas_r(y ~ trend + mls(x, 0:7, 4) + mls(z, 0:16, 12), start = NULL, data =
list(data.frame(y = y, trend = trend), x = x, z = z))
```

在这里，不需要为data.frame在列表中命名。
下面的R代码使用函数midas_r估计受约束情况(b)，并报告参数的NLS估计和相关的汇总统计信息。

```
eq_r <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z, 0:16, 12, + nealmon),
start = list(x = c(1, -0.5), z = c(2, 0.5, -0.1)))

summary(eq_r)
```

看一下参数

```
Formula y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z, 0:16, 12, nealmon)
Parameters
```

midas_r函数的语法类似于标准的R函数nls(**非线性回归建模函数**)。
模型通过熟悉的formula接口指定。包含的滞后项和使用的函数约束可以针对每个变量，并在midas_r使用的mls、fmls或dml函数中指定。有必要为每个系数受限的变量提供一个起始值列表，因为它隐式地定义了每个系列使用的约束函数的参数数量。

与nls函数的主要区别是有更多的数值优化算法的选择:函数midas_r的编写方式在理论上可以使用任何R优化函数。选择是通过function参数控制的。目前可以使用函数optim和nls，它们存在于标准R安装和包optimx中的函数optimx中(Nash和Varadhan 2011;Nash 2014)。上述函数的附加参数可以在对midas_r的调用中直接指定。例如，如果我们想使用Nelder和Mead的优化算法，这是函数optim中的默认选项，我们使用以下代码：

```
eq_r2 <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z, 0:16, 12, nealmon),
start = list(x = c(1, -0.5), + z = c(2, 0.5, -0.1)), Ofunction = "optim", method =
"Nelder-Mead")
```

如果我们想对函数nls中实现的部分线性最小二乘模型使用Golub-Pereyra算法，我们使用以下代码:

```
eq_r2 <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z, 0:16, 12, nealmon),
start = list(x = c(1, -0.5), + z = c(2, 0.5, -0.1)), Ofunction = "nls", method =
"plinear")
```

使用前一种算法的最终解作为起始值，可以用不同的算法重新估计NLS问题。
例如，众所周知，nls中的默认算法对起始值非常敏感 `#?` 。所以我们首先可以使用标准的Nelder-Mead算法找到"更可行的"起始值，然后使用nls得到最终结果:

```
eq_r2 <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon)+mls(z, 0:16, 12, nealmon),
start = list(x = c(1, -0.5), + z = c(2, 0.5, -0.1)), Ofunction = "optim", method =
"Nelder-Mead")

eq_r2 <- update(eq_r2, Ofunction = "nls")
```

所使用的优化函数的输出可以通过检查midas_r输出的元素opt来找到:

```
eq_r2 <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon)+mls(z, 0:16, 12, nealmon),
start = list(x = c(1, -0.5),z = c(2, 0.5, -0.1)), Ofunction = "optim", method =
"Nelder-Mead")

eq_r2$opt
```

这里我们观察到，Nelder-Mead算法对成本函数进行了502次评估。
R中的优化函数通过一个数值常数报告优化算法的收敛状态，0表示收敛成功。
这个数值常量被报告为midas_r输出的元素收敛.

```
eq_r2$convergence
[1] 1
```

在这种情况下，收敛并不成功。optim函数的帮助页面显示收敛码1表示达到迭代极限。为了提高收敛性，可以使用用户自定义的梯度函数。要使用它们，就必须定义约束的梯度函数。例如，对于nealmon限制，梯度函数的定义如下:(略)

> 如果最后结论不收敛，再来考虑这里的情况

限制的充分性检验(3.4)

给定midas_r估计的MIDAS回归模型，在相当标准的假设下(参见Kvedaras and Zemlys 2012和Kvedaras and Zemlys 2013)，可以使用包中的函数hAh_test和hAhr_test来检验函数限制的经验充分性。在平稳级数$\{y_t\}$的情况下，它们可以直接应用，而当$\{y_t\}$与解释变量协整时，在测试之前需要应用一个特殊的转换(例如，Bilinskas和Zemlys 2013)。
只要进程的错误是独立且相同分布的，就可以使用hAh_test
而hAhr_test则使用该测试的一个HAC-robust版本。
当残差中没有观察到显著的HAC时，我们建议使用hAh_test，这样就可以在小样本中有更精确的测试大小。在与解释变量协整的集成系列$\{y_t\}$的情况下，还有一些其他的替代方法(参见Bilinskas, Kvedaras和Zemlys 2013)。

这里报告了**检验统计量的值、自由度(方程3中参数绑定约束的数量)和零假设的经验意义，即函数约束是充分的。**

可以看出，这种规范实际上对应于基本的DGP，在通常的显著性水平上不能被拒绝，然而，例如，将变量z的函数约束的参数数量减少到只有两个而不是三个，使用这两种检验版本都非常强烈地拒绝:

当实证充分性在一些模型的适当显著性水平上不能被拒绝时，我们可以进一步依赖信息标准来选择最佳候选模型。

重点：模型的选择（3.5）

假设我们想要研究几个函数约束中的哪一个——例如，归一化(“nealmon”)或非归一化(“almonp”)指数Almon滞后多项式，或具有2或3阶多项式，等等——更适合于y对x和z的MIDAS回归模型(每个变量可能不同)。
由于滞后的最佳最大数量可能因 函数约束 和/或 变量/频率而不同，所以我们首先使用midasr函数expand_weights_lag定义与**每个解释变量对应的潜在模型集**，如下所示：

```
set_x <- expand_weights_lags(weights = c("nealmon", "almonp"), from = 0, to = c(5,
10), m = 1, start = list(nealmon = c(1, 1), almonp = c(1, 0, 0)))

set_z <- expand_weights_lags(c("nealmon", "nealmon"), 0, c(10, 20), 1, start =
list(nealmon = c(1, -1), nealmon = c(1, -1, 0)))
```

在这里，对于每个变量，向量(或列表),**weights**定义了要考虑的潜在约束，列表起始值**start**给出了适当的起始值，隐式定义了每个函数的参数数量

潜在的滞后项的结构由下列高频滞后范围给出:
from `[from; m * min(to)]` to `[from; m * max(to)]`