509系统工具箱编程规范

1. 整体
2. 排版
   1. 程序块要采用缩进风格编写，缩进的空格数为4个

说明：对于由开发工具自动生成的代码可以有不一致。

* 1. 相对独立的程序块之间、变量说明之后必须加空行

示例：如下例子不符合规范。

if len1>=len2

yout = yout1(len1-len2+1:end);

else

yout = fliplr(yout1(1:len1+len2-1));

end

y=yout./sqrt(sum(abs(input1.^2))\*sum(abs(input2.^2))); %Nomalize the output signal.

if nargin==2 %Determine the number of input parameters.

elseif varargin{1}

plot(abs(y))

end

应如下书写

if len1>=len2

yout = yout1(len1-len2+1:end);

else

yout = fliplr(yout1(1:len1+len2-1));

end

**此处应有空行**

y=yout./sqrt(sum(abs(input1.^2))\*sum(abs(input2.^2))); %Nomalize the output signal.

if nargin==2 %Determine the number of input parameters.

elseif varargin{1}

plot(abs(y))

end

* 1. 较长的语句要分成多行书写

长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

* 1. 若函数或过程中的参数较长，则要进行适当的划分
  2. 不允许把多个短语句写在一行中，即一行只写一条语句

示例：

如下例子不符合：

fs = 96000; b = 6000; fc = 12000;

应写作：

fs = 96000;

b = 6000;

fc = 12000;

* 1. 对齐只使用空格键，不使用TAB键

说明：以免用不同的编辑器阅读程序时，因TAB键所设置的空格数目不同而造成程序布局不整齐，不要使用BC 作为编辑器合版本，因为BC 会自动将8个空格变为一个TAB 键，因此使用BC合入的版本大多会将缩进变乱。

* 1. 函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格

例如：

for i = 1:1:m

…

for j = 1:1:n

…

end

end

1. 注释
   1. **一般情况下，源程序有效注释量必须在30％以上**

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不 能太少，注释语言必须准确、易懂、简洁。

* 1. 说明性文件注释：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明

function [y] = iqmod(x,fs,b,fc)

% IQMOD: An IQ modulator,this function can modulate the baseband signal to

% the passband.

**函数头中说明函数名称大写，help**

**时会加粗，更加直观。**

% y = IQMOD(x,fs,b,fc)

% INPUT：

% 'x' ：The sequence need to be modulated.

% 'fs'：System sampling rate. /Hz

% 'b' ：The band of the signal. /Hz

% 'fc'：The carrier frequency. /Hz

% OUTPUT：

% 'y' ：The signal modulated to passband.

% Call:None.

% Refrences:None.

% Author: Lilin

% E-mail: [lilin@hrbeu.edu.cn](mailto:lilin@hrbeu.edu.cn)

xUpsample = rectpulse(x,fs/b); %Upsampling the signal,Chip length = fs/b

t = 0:1/fs:(length(xUpsample)-1)/fs;

**函数语句中注释用% … 隔开**

% yCos=cos(2\*pi\*fc\*t);

% ySin=sin(2\*pi\*fc\*t);

% ynI = real(xUpsample);

% ynQ = imag(xUpsample);

% y = ynI.\*yCos-ynQ.\*ySin; %

y = real(xUpsample.\*exp(1i\*2\*pi\*fc\*t)); % Equivalent to these five lines before

y = y./max(abs(y)); % Nomalized the signal

* 1. 边写代码边注释

修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要及时删除。

* 1. 注释的内容要清楚、明了，含义准确，防止注释二义性

说明：错误的注释不但无益反而有害。

* 1. 避免在注释中使用缩写，特别是非常用缩写

说明：在使用缩写时或之前，应对缩写进行必要的说明。

* 1. 注释应与其描述的代码相近

对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置（空格隔开），不可放在下面，如放于上方则需与其上面的代码用空行隔开。

例如：

…

%% IQ Modulation

iqmodSignal = iqmod(modSignalIfftCp.',fs,b,fc).';

spectrogram(iqmodSignal,256,250,256,fs,'yaxis') %Plot time-frequency curve

主函数中代码块注释用 %% 分隔，与上方代码空行隔开；单句注释写在上方或右侧，用% … 分隔，函数头用% 隔开。

以下为错误示范:

…

%% IQ Modulation

**没有空行**

iqmodSignal = iqmod(modSignalIfftCp.',fs,b,fc).';

spectrogram(iqmodSignal,256,250,256,fs,'yaxis') % Plot time-frequency curve

或者

%% IQ Modulation

**不应有空行**

iqmodSignal = iqmod(modSignalIfftCp.',fs,b,fc).';

spectrogram(iqmodSignal,256,250,256,fs,'yaxis') %Plot time-frequency curve

或者

%% IQ Modulation

iqmodSignal = iqmod(modSignalIfftCp.',fs,b,fc).';

spectrogram(iqmodSignal,256,250,256,fs,'yaxis')

**不应放在后面**

%Plot time-frequency curve

* 1. **对于所有有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方**
  2. 避免在一行代码或表达式的中间插入注释

说明：除非必要，不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

* 1. 通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的

说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

* 1. 在代码的功能、意图层次上进行注释，提供有用、额外的信息

说明：注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读 者理解代码，防止没必要的重复注释信息。

示例：如下注释意义不大。

% if receiveFlag is TRUE

if receiveFlag

而如下的注释则给出了额外有用的信息。

% if mtp receive a message from links

if receiveFlag

* 1. 在程序块的结束行右方加注释标记，以表明某程序块的结束

说明：当代码段较长，特别是多重嵌套时，这样做可以使代码更清晰，更便于阅读。示例：参见如下例子。

if …

// program code

while index < MAX\_INDEX

**指明是哪一条while语句的结束**

// program code

end % end of while index < MAX\_INDEX

**指明是哪一条if语句的结束**

end % end of if ...

* 1. 注释格式尽量统一

建议某个功能程序块使用“%% ……”单句程序上方和右方使用“% …… ”。

注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用英文，熟悉代码的同时也可以熟悉专业术语。

1. 标识命名符
   1. 标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的 缩写，避免使人产生误解

说明：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成 缩写；一些单词有大家公认的缩写。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为tmp;

flag 可缩写为flg;

**此处大家补充**

statistic 可缩写为 stat ;

increment 可缩写为 inc ;

message 可缩写为 msg ;

* 1. 命名中若使用特殊约定或缩写，则要有注释说明

说明：应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进 行必要的注释说明。

* 1. 自己特有的命名风格，要自始至终保持一致，不可来回变化

说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。（即 命名规则中没有规定到的地方才可有个人命名风格）。

* 1. 对于变量命名，禁止取单个字符（如i、j、k...），建议除了要有具体含义外，还能 表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的

说明：变量，尤其是局部变量，如果用单个字符表示，很容易敲错（如 i 写成 j），而编译时又检查不出来，有可能为了这个小小的错误而花费大量的查错时间。

采用骆驼命名法：正如它的名称所表示的那样，是指混合使用大小写字母来构成变量和函数的名字，第一个单字首字母采用[小写字母](https://baike.baidu.com/item/%E5%A4%A7%E5%86%99%E5%AD%97%E6%AF%8D" \t "_blank)；后续单字的首字母亦用大写字母，例：userAaaBbb。

* 1. 除非必要，不要用数字或较奇怪的字符来定义标识符
  2. 用正确的反义词组命名具有互斥意义的变量或相反动作的函数等

说明：下面是一些在软件中常用的反义词组。

add / remove begin / end create / destroy insert / delete first / last get / release increment / decrement put / get cut / paste

add / delete lock / unlock open / close min / max old / new start / stop next / previous source / target show / hide send / receive source / destination up / down

1. 可读性
   1. 注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级

例如：

a+b:c+d; 和(a+b):(c+d)实现效果一样，因为matlab中“：”运算符优先级较低。编程时使用后者。

* 1. 避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替

示例：如下的程序可读性差。

if Trunk[index].trunk\_state == 0

Trunk[index].trunk\_state = 1;

... % program code

end

应改为如下形式。

global TRUNK\_IDLE;

global TRUNK\_BUSY;  
define TRUNK\_IDLE 0;

define TRUNK\_BUSY 1;

if Trunk[index].trunkState == TRUNK\_IDLE

Trunk[index].trunkState = TRUNK\_BUSY;

... % program code

end

* 1. 源程序中关系较为紧密的代码应尽可能相邻

例如：

fs = 96000;

**频率参数放在一起**

b = 6000;

fc = 12000;

noiseVar = 1e-10;

Nfft = 1024;

* 1. **不要使用难懂的技巧性很高或者很复杂的语句，除非很有必要时**

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。

1. 变量、结构
   1. 去掉没必要的公共变量

说明：公共变量是增大模块间耦合的原因之一，故应减少没必要的公共变量以降低模块间的耦合度。

* 1. 严禁使用未经初始化的变量作为右值

说明：赋值语句右边的值为右值。

另外，循环中用到的变量也要进行初始化。

* 1. 结构的功能要单一，是针对一种事务的抽象

说明：设计结构时应力争使结构代表一种现实事务的抽象，而不是同时代表多种。结构中的各元素应代表同一事务的不同侧面，而不应把描述没有关系或关系很弱的不同事务的元素放到同一结构中。

* 1. 不要设计面面俱到、非常灵活的数据结构

说明：面面俱到、灵活的数据结构反而容易引起误解和操作困难。

* 1. 不同结构间的关系不要过于复杂

说明：若两个结构间关系较复杂、密切，那么应合为一个结构。示例：如下两个结构的构造不合理。

* 1. 结构中元素的个数应适中。若结构中元素个数过多可考虑依据某种原则把元素组成不同 的子结构，以减少原结构中元素的个数

1. 函数、过程
   1. 对所调用函数的错误返回码要仔细、全面地处理
   2. 明确函数功能，精确（而不是近似）地实现函数设计
   3. 防止将函数的参数作为工作变量

说明：将函数的参数作为工作变量，有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。

* 1. 函数的规模尽量限制在200行以内

说明：不包括注释和空格行。

* 1. 一个函数仅完成一件功能

说明：禁止多个功能放在一个函数，这样后期修改维护麻烦。多功能集于一身的函数，很可能使函数的理解、测试、维护等变得困难。

* 1. 为简单功能编写函数

说明：虽然为仅用一两行就可完成的功能去编函数好像没有必要，但用函数可使功能明确化，增加程序可读性，亦可方便维护、测试。

* 1. 函数的功能应该是可以预测的，也就是只要输入数据相同就应产生同样的输出

说明：带有内部“存储器”的函数的功能可能是不可预测的，因为它的输出可能取决于内部存储器（如某标记）的状态。这样的函数既不易于理解又不利于测试和维护。

* 1. 尽量不要编写依赖于其他函数内部实现的函数

说明：此条为函数独立性的基本要求。

* 1. 避免设计多参数函数，不使用的参数从接口中去掉

说明：目的减少函数间接口的复杂度。

* 1. 函数名应准确描述函数的功能

使用动宾词组为执行某操作的函数命名，避免使用无意义或含义不清的动词为函数命名。

* 1. 函数的返回值要清楚、明了，让使用者不容易忽视错误情况

说明：函数的每种出错返回值的意义要清晰、明了、准确，防止使用者误用、理解错误或忽视错误返回码。

* 1. 避免函数中不必要语句，防止程序中的垃圾代码

说明：程序中的垃圾代码不仅占用额外的空间，而且还常常影响程序的功能与性能，很可能给程序的测试、维护等造成不必要的麻烦。

* 1. 防止把没有关联的语句放到一个函数中

说明：防止函数或过程内出现随机内聚。随机内聚是指将没有关联或关联很弱的语句放到同一个函数或过程中。随机内聚给函数或过程的维护、测试及以后的升级等造成了不便，同时也使函数或过程的功能不明确。使用随机内聚函数，常常容易出现在一种应用场合需要改进此函数，而另一种应用场合又不允许这种改进，从而陷入困境。在编程时，经常遇到在不同函数中使用相同的代码，许多开发人员都愿把这些代码提出来，并构成一个新函数。若这些代码关联较大并且是完成一个功能的，那么这种构造是合理的，否则这种构造将产生随机内聚的函数。

* 1. 如果多段代码重复做同一件事情，那么在函数的划分上可能存在问题

说明：若此段代码各语句之间有实质性关联并且是完成同一件功能的，那么可考虑把此段代码构造成一个新的函数。

* 1. 功能不明确较小的函数，特别是仅有一个上级函数调用它时，应考虑把它合并到上级 函数中，而不必单独存在

说明：模块中函数划分的过多，一般会使函数间的接口变得复杂。所以过小的函数，特别是扇入很低的或功能不明确的函数，不值得单独存在。

* 1. 减少函数本身或函数间的递归调用

说明：递归调用特别是函数间的递归调用（如 A->B->C->A），影响程序的可理解性；递归调用一般都占用较多的系统资源（如栈空间）；递归调用对程序的测试有一定影响。故除非为某些算法或功能的实现方便，应减少没必要的递归调用。

* 1. 仔细分析模块的功能及性能需求，并进一步细分，同时若有必要画出有关数据流图，据此来进行模块的函数划分与组织

说明：函数的划分与组织是模块的实现过程中很关键的步骤，如何划分出合理的函数结构，关系到模块的最终效率和可维护性、可测性等。根据模块的功能图或/及数据流图映射出函数结构是常用方法之一。

* 1. 改进模块中函数的结构，降低函数间的耦合度，并提高函数的独立性以及代码可读性、效率和可维护性。优化函数结构时，要遵守以下原则：

（1）不能影响模块功能的实现。

（2）仔细考查模块或函数出错处理及模块的性能要求并进行完善。

（3）通过分解或合并函数来改进软件结构。

（4）考查函数的规模，过大的要进行分解。

（5）降低函数间接口的复杂度。

（6）不同层次的函数调用要有较合理的扇入、扇出。

（7）函数功能应可预测。

（8）提高函数内聚。（单一功能的函数内聚最高） 说明：对初步划分后的函数结构应进行改进、优化，使之更为合理。

1. 程序效率
   1. 编程时要经常注意代码的效率

说明：代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

* 1. 在保证软件系统的正确性、稳定性、可读性的前提下，提高代码效率。

说明：不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

* 1. 循环体内工作量最小化

说明：应仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率。

在matlab中，可以用矩阵实现的尽量不要用循环。

* 1. 在多重循环中，应将最忙的循环放在最内层

说明：最忙的循环即循环次数最多的循环。

* 1. 避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中

说明：目的是减少判断次数。循环体中的判断语句是否可以移到循环体外，要视程序的具 体情况而言，一般情况，与循环变量无关的判断语句可以移到循环体外，而有关的则不可以。

* 1. 代码不用很紧凑

说明：因为紧凑的代码并不代表高效的机器码。相反，过于紧凑会使代码的可读性变差。