Оглавление

Основные классы. 1
Общая логика работы 2
Формат описания эксперимента 5
Параметры источника сигнала 7
Параметры генератора состояний 8
Параметры стадии обработки 8
Логика выполнения стадии обработки 12
Создание собственного класса обработки сигнала 16
Описание классов 16
Описание тестового эксперимента 17

Основные классы

(Жирным выделены классы, для которых сейчас есть работающая реализация).

t_sample_buf – буфер для хранения данных по каналам и номеров сэмплов

t_sigproc_base – родительский класс для обработки данных (препроцессинга, выделения признаков и распознавания)

Наследники t_sigproc_base:
t_sigproc_iir – выполняет частотную фильтрацию при помощи IIR-фильтра
t_sigproc_spatfilt – выполняет пространственную фильтрацию
t_sigproc_winpow – вычисляет мощность сигнала и сглаживает с использованием скользящего окна
t_statepred_LDA – осуществляет классификацию наборов признаков при помощи LDA.
t_sigproc_test_1 – умножает входные сигналы на заданные значения

t_eeg_recv_manager_base – базовый класс источника данных, получает последовательность сэмплов и соответствующих меток

Наследники t_eeg_recv_manager_base:
t_eeg_recv_manager_NVX – принимает данные из NVX
t_eeg_recv_manager_file – читает данные из датасета в формате EEGLAB
t_eeg_recv_manager_outer – получает данные от глобально объявленного объекта, имеющего тип t_eeg_recv_manager_base
t_eeg_recv_manager_test_1 – генерирует несколько синусоидальных сигналов с заданными частотами

t_state_generator – генерирует последовательности кодов состояний, соответствующие последовательностям данных

t_state_generator_binary – генерирует последовательность из двух кодов, содержащую периодически чередующиеся блоки
t_state_generator_mark – генерирует последовательность кодов на основе последовательности меток, принятой источником данных.
t_state_generator_null – генерирует последовательность одинаковых кодов

t_visualizer – базовый класс для визуализации данных

Наследники t_visualizer:
t_visualizer_scalp – визуализирует распределение сигнала по голове
t_visualizer_sig – визуализирует временную динамику сигнала

t_sigproc_stage – стадия обработки сигнала; представляет собой обертку над t_sigproc_base

t_sigsrc_stage – стадия получения сигнала; представляет собой обертку над t_eeg_recv_manager_base

t_stategen_stage – стадия получения последовательности кодов состояний; представляет собой обертку над t_state_generator

t_procstage_graph – граф стадий работы с сигналом / последовательностью кодов состояний (получение / генерация последовательности / обработка)

t_exp_performer – класс для выполнения эксперимента

Общая логика работы

Все данные хранятся в объектах типа t_sample_buf. В состав этих объектов входит матрица значений сигнала data (каналы x сэмплы) и вектор уникальных номеров сэмплов sample_idx. Изначально номер присваивается сэмплу при получении из источника данных. В ходе обработки данных номера сэмплов сохраняются. При смене частоты дискретизации сигнала вектор номеров сэмплов модифицируется аналогичным образом, благодаря чему сигналы на любых стадиях обработки могут быть сопоставлены друг с другом. Буфер может быть как линейным, так и кольцевым.

Обработка сигнала определяется последовательностью стадий. На каждой стадии данные из нескольких входных буферов преобразуются и кладутся в один выходной буфер. Число выходных каналов не обязано совпадать с числом входных. Преобразование выполняется при помощи объектов классов, унаследованных от t_sigproc_base.

В ходе работы с данными (выполнение эксперимента или обучение) параллельно осуществляется две ветви обработки данных. Каждая ветвь представляет собой граф стадий обработки, представленный объектом типа t_procstage_graph. На вход первой ветви поступает, собственно, входной сигнал (полученный из источника при выполнении эксперимента или зарегистрированный в ходе последнего эксперимента при обучении). Для получения входного сигнала служат объекты классов, унаследованных от t_eeg_recv_manager_base. На вход второй ветви поступает вектор кодов состояний, соответствующих каждому сэмплу данных. Этот вектор может быть получен либо из источника данных вместе с самими данными, либо сгенерирован в ходе эксперимента. Например, при проведении эксперимента на motor imagery каждому сэмплу будет соответствовать код состояния, указывающий на то, о какой руке было велено думать испытуемому в момент регистрации этого сэмпла. Для получения / генерации вектора кодов состояний служат объекты классов, унаследованных от t_state_generator.

В первой ветви обработки буферы могут содержать следующие виды информации:
- исходный сигнал (на первой стадии);
- предобработанный сигнал (например, после частотной или пространственной фильтрации);
- признаки (например, значения мощности, усредненные по некоторому временному окну);
- ответы (т.е. коды состояний, определенные классификатором)

Во второй ветви обработки на каждой стадии либо не происходит ничего, либо осуществляется смена частоты дискретизации, аналогичная соответствующей стадии обработки данных из первой ветви. Идея заключается в том, чтобы на каждой стадии иметь соответствие между сэмплами обработанных данных и кодами состояний. На выходе последней стадии обработки (классификация) в первой ветви содержатся коды состояний, определенные классификатором, а во второй ветви – «настоящие» коды, которые в идеале должны совпадать с результатами классификации. Объект обработки, который не делает ничего кроме смены частоты дискретизации – это просто t_sigproc_base, без наследования.

Выполнение эксперимента осуществляется при помощи объекта типа t_exp_performer. Обучение осуществляется при помощи объекта типа t_paradigm_trainer. Оба этих класса содержат объекты, определяющие источник данных (sigsrc_stage), генератор состояний (stategen_stage), граф обработки данных (sigproc_graph) и граф обработки вектора состояний (stateproc_graph).

Объект, определяющий источник данных (sigsrc_stage), имеет тип t_sigsrc_stage. Он содержит в себе объект proc_obj, который непосредственно получает входной сигнал и относится к типу, унаследованному от t_eeg_recv_manager_base. Кроме того, объект sigsrc_stage содержит два буфера buf_eeg и buf_mark, в которые помещаются сами данные (т.е. сигнал) и последовательность меток событий соответственно. Каналы, содержащиеся в буфере buf_eeg, определяются при инициализации объекта proc_obj. Буфер buf_mark содержит единственный канал ‘MARKERS’ и играет роль в том случае, когда данные читаются из файла, а состояния генерируются в зависимости от ранее зарегистрированных меток. ((На данный момент готов только тестовый класс t_eeg_recv_manager_test_1, а остальные классы нуждаются в модификации под новую структуру данных)).

Объект, определяющий генератор состояний (stategen_stage), имеет тип t_stategen_stage. Он содержит в себе объект proc_obj, которые непосредственно генерирует вектор состояний и относится к типу, унаследованному от t_state_generator. Кроме того, объект stategen_stage содержит буфер buf_out, в котором размещается вектор состояний. Буфер buf_out содержит единственный канал с именем ‘STATE’. Также stategen_stage содержит ссылку на sigsrc_stage, откуда он при необходимости может получать последовательность принятых меток. ((На данный момент готов только тестовый класс t_state_generator_null, а остальные классы нуждаются в модификации под новую структуру данных)).

Объекты графов обработки (sigproc_graph и stateproc_graph) относятся к типу t_procstage_graph. Объект типа t_procstage_graph содержит вектор stages, составленный из объектов, описывающих стадии обработки и относящихся к типам, унаследованным от t_sigproc_stage. Кроме того, объект типа t_procstage_graph содержит ссылку outer_inp_stage, указывающую на объект, являющийся входом для графа обработки. Для графа обработки сигнала (первая ветвь), в outer_inp_stage записывается t_exp_performer::sigsrc_stage; для графа обработки вектора состояний (вторая ветвь) в outer_inp_stage записывается t_exp_performer::stategen_stage. Также объект типа t_procstage_graph содержит поле outer_inp_buf_name, содержащее имя буфера, связанного с outer_inp_stage, и используемого в качестве входа графа обработки. Для графа обработки сигнала (первая ветвь) outer_inp_buf_name==’buf_eeg’; для графа обработки вектора состояний (вторая ветвь) outer_inp_buf_name==’buf_out’.

Направления передачи сигнала в графе обработки задаются в структуре t_procstage_graph::stage_descs. Имя n-й стадии (относительно массива stages) задано в stage_descs(n).stage_name. Имя стадии, являющейся m-м входом n-й стадии, задано в stage_descs(n).params.inp_descs(m).inp_stage_name. Если в этом поле задана строка '[INPUT]', то в качестве входа будет использоваться объект outer_inp_stage. Если inp_stage_name=='[INPUT]', то будет использоваться входной буфер с именем outer_inp_buf_name; в противном случае будет использоваться входной буфер с именем 'buf_out'. Имена каналов, используемых для m-го входа n-й стадии, задаются в stage_descs(n).params.inp_descs(m).chan_names_in.

Порядок выполнения стадий обработки хранится в t_procstage_graph::stage_order. Этот порядок определяется автоматически при инициализации графа таким образом, чтобы для каждой стадии обработки все стадии, используемые в качестве ее входов, выполнялись раньше ее самой.

Каждый объект стадии обработки относится к типу t_sigproc_stage. Он содержит объект proc_obj, который непосредственно выполняет обработку и относится к типу, унаследованному от t_sigproc_base. Также этот объект содержит объект буфера buf_out, в который помещаются обработанные данные. Кроме того, он содержит вектор ссылок на входные буферы bufs_in. ((На данный момент готов только тестовый класс t_sigproc_base_test_1, а остальные классы нуждаются в модификации под новую структуру данных)).

При выполнении эксперимента общий порядок работы имеет следующий вид:
- инициализация источника данных (sigsrc_stage);
- инициализация генератора состояний (stategen_stage), передача ему ссылки на sigsrc_stage;
- инициализация графа обработки сигнала, передача ему ссылки на sigsrc_stage;
- инициализация графа обработки вектора состояний, передача ему ссылки на stategen_stage;
- последовательная инициализация стадий обработки сигнала, передача каждому объекту стадии обработки ссылок на входные буферы;
- последовательная инициализация стадий обработки вектора состояний, передача каждому объекту стадии обработки ссылок на входные буферы;
- получение порции данных от источника;
- генерация вектора состояний, соответствующего принятым данным;
- последовательное применение стадий обработки к данным;
- последовательное применение стадий обработки к вектору состояний.

При выполнении обучения, в отличие от проведения эксперимента, данные читаются из источника не порциями, а сразу целиком. Инициализация стадий обработки, напротив, осуществляется не заранее перед обработкой данных, а последовательно, так что каждая стадия инициализируется только после обработки данных предыдущей на стадии. Для любой стадии может быть задан специальный объект, содержащий функцию обучения. На вход функции обучения подаются данные, полученные на предыдущей стадии обработки; соответствующий этим данным вектор состояний; статические параметры обучаемой стадии (известные заранее и не требующие настройки в ходе обучения); параметры процедуры обучения. Указанная функция возвращает структуру с параметрами стадии обработки, дополненную полями, настроенными в ходе обучения. Затем уже с использованием этой структуры производится инициализация стадии и дальнейшая обработка данных. ((На данный момент обучение не реализовано, хотя классы для отдельных стадий написаны))

При обучении общий порядок работы имеет следующий вид:
- инициализация источника данных (sigsrc_stage);
- инициализация генератора состояний (stategen_stage), передача ему ссылки на sigsrc_stage;
- инициализация графа обработки сигнала, передача ему ссылки на sigsrc_stage;
- инициализация графа обработки вектора состояний, передача ему ссылки на stategen_stage;

- получение всех данных из источника;
- генерация вектора состояний, соответствующего полученным данным;
- обучение первой стадии обработки сигнала (опционально);
- инициализация первой стадии обработки сигнала;
- инициализация первой стадии обработки вектора состояний;
- выполнение первой стадии обработки сигнала;
- выполнение первой стадии обработки вектора состояний;
- обучение второй стадии обработки сигнала (опционально);
- …

При обучении объект источника данных берет данные из файла, записанного в ходе выполнения эксперимента.

Процедура кросс-валидации на данный момент не продумана.

Объект типа t_exp_performer содержит вектор visualizers, составленный и объектов, реализующих визуализацию данных и относящихся к типам, унаследованным от t_visualizer. Каждый из этих объектов получает ссылку на объект типа t_exp_performer и визуализирует данные, хранящиеся в буферах, связанных со стадиями обработки / получения сигнала / генерации вектора состояний. Yf На данный момент реализовано два класса для визуализации данных. Первый (t_visualizer_sig) предназначен для отображения сигналов, т.е. графиков зависимости значений, полученных из канала, от номера сэмпла. Второй (t_visualizer_scalp) предназначен для отображения распределения сигнала по электродам в конкретный момент времени. ((Реально сейчас работает только t_visualizer_sig, а t_visualizer_scalp нуждается в небольшой модификации под новую структуру данных).

Формат описания эксперимента

Параметры эксперимента задаются в виде структуры exp, которая может читаться из mat-файлов. В дальнейшем можно написать заполнение этой структуры на основе xml-файлов. Структура exp передается в качестве аргумента в функцию выполнения эксперимента t_exp_performer::perform_exp().

1. Информация об эксперименте

exp.exp_info.setup_name – название типа эксперимента
exp.exp_info.parent_procname – имя функции, создавшей файл параметров эксперимента

2. Параметры самого эксперимента

exp.exp_params.exp_duration_t – длительность эксперимента

3. Параметры источника данных

exp.sigsrc_stage.obj_type – тип объекта источника данных (напр., t_eeg_recv_manager_NVX)
exp.sigsrc_stage.params.params_spec – параметры, специфичные для конкретного типа источника данных (напр., адрес сетевого сокета)
exp.sigsrc_stage.params.buf_out_desc.type (опционально) – тип выходного буфера
exp.sigsrc_stage.params.buf_out_desc.len_t (опционально) – длина выходного буфера в секундах. Если не задано, будет использоваться exp.sigsrc_stage.params.buf_out_desc.len_t.

4. Параметры генератора кодов состояний

exp.stategen_stage.obj_type – тип объекта генератора кодов состояний (напр., t_eeg_state_generator_binary)
exp.stategen.params_spec – параметры, специфичные для конкретного типа генератора кодов (напр., период переключения одной задачи на другую)
exp.stategen.params.params_base.state_descs(n).label – код n-го состояния
exp.stategen.params.params_base.state_descs(1).name – имя n-го состояния
exp.stategen.params.params_base.state_id_def – номер начального состояния

5. Параметры стадии обработки сигнала

exp.sigproc_graph.stage_descs(n).stage_name – имя стадии обработки
exp.sigproc_graph.stage_descs(n).obj_type – тип объекта, выполняющего обработку
exp.sigproc_graph.stage_descs(n).params.inp_descs(m).inp_stage_name – имя стадии обработки, выход которой является m-м входом текущей стадии; если задано ключевое слово [INPUT] – в качестве входа будет использоваться источник сигнала
exp.sigproc_graph.stage_descs(n).params.inp_descs(m).chan_names_in (опционально) – список имен каналов m-го входа, поступающих на обработку; если не задано – будут использованы все каналы
exp.sigproc_graph.stage_descs(n).params.params_base.srate_out (опционально) – выходная частота дискретизации; если не задано – будет использована частота первого входа
exp.sigproc_graph.stage_descs(n).params.params_base.timewin_prev (опционально) – временной интервал, предшествующий каждому сэмплу, данные из которого требуются для обработки этого сэмпла; если не задано – используется ноль.
exp.sigproc_graph.stage_descs(n).params.buf_out_desc.type (опционально) – тип выходного буфера; если не задано – используется линейный буфер
exp.sigproc_graph.stage_descs(n).params.buf_out_desc.len_t (опционально) – длина выходного буфера в секундах; если не задано – используется значение exp.exp_params.exp_duration_t.
exp.sigproc_graph.stage_descs(n).params.params_spec – параметры, специфичные для указанного типа объекта обработки.

6. Параметры визуализации

Общие:

exp.visualizers(n).name – имя визуализации
exp.visualizers(n).type – имя класса объекта визуализации

Параметры t_visualizer_sig:

exp.visualizers(n).params.fig_num идентификатор окна, в котором будут отображаться данные; если все объекты визуализации работают с одним и тем же окном, это значение должно быть одинаковым для них
exp.visualizers(n).params.subplot_info = [a,b,c] – область в окне, в которой будут отображаться сигналы; числа a,b,c передаются в функцию subplot()
exp.visualizers(n).params.line_styles = {'r-', 'b--',…} – стили, используемые для отображения сигналов
exp.visualizers(n).params.line_width - толщина линий на графиках сигналов
exp.visualizers(n).params.ylim = [xmin,xmax] – верхняя и нижняя граница значения сигнала на графике; числа xmin,xmax передаются в функцию ylim()
exp.visualizers(n).params.flip90 – если не равно нулю, то график будет развернут на 90 градусов, т.е. ось времени будет идти вверх, а ось значения сигнала – вправо
exp.visualizers(n).params.centval – уровень сигнала, на котором будет проведена нулевая отметка
exp.visualizers(n).params.show_legend – нужно ли отображать легенду (отображение сильно замедляет работу!)

exp.visualizers(n).params.sig_descs – дескрипторы визуализируемых сигналов
exp.visualizers(n).params.sig_descs{n}.data_type – тип ветви, из которой берется визуализируемый сигнал; 'SIGNAL' – ветвь обработки данных, 'STATE' – ветвь обработки вектора состояний
exp.visualizers(n).params.sig_descs{n}.stage_name – имя стадии обработки; если равно '[INPUT]', то сигнал будет взят из выходного буфера объекта t_exp_performer::sigsrc_stage при data_type=='SIGNAL' или из выходного буфера объекта t_exp_performer::stategen_stage при data_type=='STATE'; если не равно '[INPUT]', то сигнал будет взят из выходного буфера стадии обработки, имеющей указанное имя (в зависимости от значения data_type эта стадия будет взята из t_exp_performer::sigproc_graph или из t_exp_performer::stateproc_graph)
exp.visualizers(n).params.sig_descs{n}.chan_names_data – имена визуализируемых каналов; если поле пустое – будут отображаться все каналы из соответствующего буфера
exp.visualizers(n).params.sig_descs{n}.vis_win_t – длина отображаемой порции сигнала в секундах
exp.visualizers(n).params.sig_descs{n}.smooth_len_t – длина сглаживающего окна в секундах
exp.visualizers(n).params.sig_descs{n}.need_square – если не ноль, то сигнал будет возыеден в квадрат перед визуализацией
exp.visualizers(n).params.sig_descs{n}.mult – значение, на которое будет домножен сигнал перед визуализацией

Параметры источника сигнала

Используется 3 вида параметров:

params_base – параметры общего назначения
params_spec – параметры, зависящие от типа объекта обработки
buf_out_desc – описание выходного буфера

params_base.srate_out – выходная частота дискретизации
Определяется объектом источника
Инициализация:
- t_sigsrc_stage::init() -> t_eeg_recv_manager_base::init(). Значение берется из результата вызова функции t_eeg_recv_manager_base::get_srate(), которая определяется в классах-потомках t_eeg_recv_manager_base.

params_base.chan_names_out – имена выходных каналов
Определяется объектом источника
Инициализация:
- t_sigsrc_stage::init() -> t_eeg_recv_manager_base::init(). Значение берется из результата вызова функции t_eeg_recv_manager_base::get_chan_names_out(), которая определяется в классах-потомках t_eeg_recv_manager_base.

buf_out_desc.type – тип выходного буфера
Допустимые значения: 'linear' | 'ring'
По умолчанию: 'linear'
Инициализация:
- файл настроек;
- t_exp_performer::perform_exp() -> t_exp_performer::init_params()

buf_out_desc.len_t – длина выходного буфера в секундах
По умолчанию: немного дольше длительности эксперимента

Инициализация:
- файл настроек;
- t_exp_performer::perform_exp() -> t_exp_performer::init_params()

Параметры генератора состояний

Используется 3 вида параметров:

params_base – параметры общего назначения
params_spec – параметры, зависящие от типа объекта обработки
buf_out_desc – описание выходного буфера

params_base.srate_out – выходная частота дискретизации
Равна выходной частоте источника данных
Инициализация:
- t_exp_performer::perform_exp()

params_base.chan_names_out – имена выходных каналов
Содержит одно имя ‘STATE’.
Инициализация:
- t_exp_performer::perform_exp() -> t_exp_performer::init_params()

params_base.state_descs – вектор дескрипторов состояний

params_base.state_descs(n).name – имя n-го состояния
Инициализация: файл настроек

params_base.state_descs(n).label – код n-го состояния, который будет записываться в буфер, когда это состояние активно
Инициализация: файл настроек

params_base.state_id_def – номер состояния, активного при старте эксперимента (относительно params_base.state_descs)
Инициализация: файл настроек

buf_out_desc.type – тип выходного буфера
Совпадает с типом выходного буфера источника данных
По умолчанию: ‘linear’
Инициализация:
- t_exp_performer::perform_exp() -> t_exp_performer::init_params()

buf_out_desc.len_t – длина выходного буфера в секундах
Совпадает с длиной выходного буфера источника данных
Инициализация:
- t_exp_performer::perform_exp() -> t_exp_performer::init_params()

Параметры стадии обработки

Используется 4 вида параметров:

params_base – параметры общего назначения
inp_descs – массив структур, описывающих входные данные
params_spec – параметры, зависящие от типа объекта обработки
buf_out_desc – описание выходного буфера

params_base.srate_out – выходная частота дискретизации
По умолчанию: частота дискретизации первого входного буфера данной стадии
Инициализация:
- файл настроек;
- t_sigproc_stage::init().

params_base.timewin_prev – длина временного окна, которое должно предшествовать сэмплу во входных буферах для его обработки
По умолчанию: 0
Инициализация:
- файл настроек;
- t_sigproc_stage::init().

params_base.downsamp_type – способ снижения частоты дискретизации.
Допустимые значения: ‘interp’ | ‘thin’ | ‘special’
‘interp’ – интерполяция, не реализовано
‘thin’ – прореживание
‘special’ – с использованием функции t_sigproc_base::proc_samp_fsin_spec()
По умолчанию: ‘thin’
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init() -> t_sigproc_base::init_spec();
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

params_base.upsamp_type – способ повышения частоты дискретизации.
Допустимые значения: ‘interp’ | ‘repeat’
‘interp’ – интерполяция, не реализовано
‘repeat’ – дублирование значений
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init() -> t_sigproc_base::init_spec();
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

params_base.chan_names_out – имена выходных каналов
По умолчанию: объединенный список имен входных каналов
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init() -> t_sigproc_base::init_spec();
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

inp_descs(n).srate_in – входная частота дискретизации
По умолчанию: частота дискретизации соответствующего входного буфера
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init().

(Частота дискретизации самого первого буфера в цепочке обработки берется из объекта источника данных)

inp_descs(n).chan_names_in – имена используемых входных каналов
По умолчанию: все имена каналов соответствующего входного буфера
Инициализация:
- файл настроек;
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init().

inp_descs(n).chan_idx_in – индексы используемых входных каналов (относительно списка каналов входного буфера)
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::set_bufs().

inp_descs(n).k_downsamp – коэффициент деления частоты дискретизации
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

inp_descs(n).k_upsamp – коэффициент умножения частоты дискретизации
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

inp_descs(n).nsamples_prev – количество отсчетов, которое должно предшествовать сэмплу во входных буферах для его обработки
Инициализация:
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init().

buf_out_desc.type – тип выходного буфера
Допустимые значения: ‘linear’ | ‘ring’
По умолчанию: ‘linear’ (а), тип первого входного буфера (б).
Инициализация:
- файл настроек;
- t_exp_performer::perform_exp() -> t_exp_performer::init_params() (а);
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() (б).

buf_out_desc.len_t – длина выходного буфера в секундах
По умолчанию: немного дольше длительности эксперимента (а), длина первого входного буфера (б).
Инициализация:
- файл настроек;
- t_exp_performer::perform_exp() -> t_exp_performer::init_params() (а);
- t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() (б).

Процесс инициализации параметров стадии обработки

1. t_exp_performer::perform_exp()

   Параметры передаются из аргумента exp_params функции perform_exp() в аргумент params_outer функции t_exp_performer::init_params().

2. t_exp_performer::perform_exp() -> t_exp_performer::init_params()

   Параметры копируются из аргумента params_outer в поля params массива stage_descs дескрипторов стадий обработки (это просто структуры), хранящийся в объекте графа обработки сигнала t_exp_performer::sigproc_graph, имеющем тип t_procstage_graph.

   Устанавливается значения по умолчанию:
   stage_descs(n).params.buf_out_desc.type становится равным ‘linear’;
   stage_descs(n).params.buf_out_desc.len_t становится немного дольше длительности эксперимента).

3. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage()

   Вызывается для каждой стадии обработки (n). Параметры копируются из дескриптора стадии обработки t_procstage_graph::stage_descs(n).params в параметры объекта стадии обработки t_procstage_graph::stages(n).params

4. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init()

   Устанавливаются значения по умолчанию:
   - params.inp_descs(m).srate_in становится частоте дискретизации соответствующего входного буфера;
   - params.params_base.srate_out становится равен частоте дискретизации первого входа;
   - params.params_base.timewin_prev становится равен нулю;
   - params.inp_descs(m).chan_names_in становится равен набору имен всех каналов соответствующего входного буфера;
   - params.buf_out_desc.type становится равен типу первого входного буфера;
   - params.buf_out_desc.len_t становится равен длине первого входного буфера;

   Параметры из t_sigproc_stage::params передаются в функцию инициализации объекта обработки t_sigproc_base::init(), вызываемую для объекта t_sigproc_stage::proc_obj. Часть параметров может изменяться внутри этой функции, поэтому возвращаемый ею результат кладется обратно в t_sigproc_stage::params.

5. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init()

   Параметры копируются из аргумента params функции init() в поле t_sigproc_base::params.

   Вызывается функция t_sigproc_base::init_spec(), переопределяемая для классов-потомков t_sigproc_base. Внутри этой функции часть параметров может изменяться.

6. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init() -> t_sigproc_base::init_spec()

   Внутри этой функции могут быть (опционально) выставлены значения следующих параметров:
   params.params_base.downsamp_type
   params.params_base.upsamp_type
   params. params_base.chan_names_out

7. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::init()

   (После вызова init_spec())

   Устанавливаются значения по умолчанию следующих параметров:

   params.params_base.downsamp_type становится равным ‘thin’
   params.params_base.upsamp_type становится равным ‘repeat’
   params.params_base.chan_names_out становится равным набору имен входных каналов

Вычисляются следующие параметры:

params.inp_descs(n).k_downsamp
params.inp_descs(n).k_upsamp
params.inp_descs(n).nsamples_prev

8. t_exp_performer::perform_exp() -> t_procstage_graph::create_stage() -> t_sigproc_stage::init() -> t_sigproc_base::set_bufs()

   Заполняются значения params.inp_descs(m).chan_idx_in: имена входных каналов связываются с номерами этих каналов во входных буферах.

Часть указанных выше функций, осуществляющих инициализацию параметров, относится к графу обработки данных. Для графа обработки вектора состояний параметры каждой стадии инициализируются на основании параметров соответствующей стадии обработки данных.

Эта инициализация осуществляется во время вызова perform_exp()->sigproc_to_stateproc_desc(). При этом из стадии обработки данных берутся следующие параметры:
- stage_name
- params.params_base.srate_out
- params.params_base.timewin_prev
- params.buf_out_desc.type
- params.buf_out_desc.len_t

Также в sigproc_to_stateproc_desc() выполняются следующие действия:
- obj_type инициализируется значением ‘t_sigproc_base’
- определяется вход текущей стадии (params.inp_descs(1).inp_stage_name)
- имя входного канала инициализируется значением ‘STATE’.

В отличие от стадии обработки данных, которая может иметь несколько входов, стадия обработки вектора состояний всегда имеет один вход. Этот вход соответствует входу стадии обработки данных, имеющему наибольшую частоту дискретизации.

Все манипуляции с параметрами, происходящие на уровне t_procstage_graph и глубже аналогичны описанным выше для стадий обработки данных.

Логика выполнения стадии обработки

Обработка порции данных выполняется при помощи функции t_sigproc_base::do_work().

1. Из каждого входного буфера (m) считываются последние данные. Во входном буфере ищется первый сэмпл, индекс которого больше либо равен sample_id_last (базовый сэмпл); затем ищется сэмпл, отстоящий от него влево на params.inp_descs(m).nsamples_prev; этот сэмпл становится первым сэмплом считываемой порции данных; последний сэмпл – конец присутствующих в буфере данных. Если нужного числа предшествующих сэмплов в буфере нет (в начале эксперимента), то данные дополняются нулями слева, а соответствующие сэмплы получают индексы
   [-nprev, …, -2, -1].

2. Для каждого буфера определяется последовательность индексов сэмплов (начиная с базового), получаемая после снижения / повышения частоты дискретизации. Пусть коэффициент понижения частоты дискретизации для m-го входа равен kdown(m), коэффициент повышения частоты дискретизации равен kup(m), а klcm = НОК(kup(1),…,kup(M)). Индексы сэмплов в полученных массивах, расположенные в позициях, кратных klcm, должны совпадать. Последний такой индекс определяет подмножество данных, общее для всех входов, и подлежащее обработке.

   Пример.

   Пусть есть 3 входа с частотами дискретизации 600, 150 и 100, а частота выхода равна 300. Тогда kdown = [2,1,1], kup = [1,2,3], klcm = 6.

   Входные индексы сэмплов:
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
   4 8 12 16 20 24 28
   6 12 18 24 30

   После смены частоты дискретизации:
   2 4 6 8 10 **12** 14 16 18 20 22 **24** 26
   4 4 8 8 12 **12** 16 16 20 20 24 **24** 28 28
   6 6 6 12 12 **12** 18 18 18 24 24 **24** 30 30 30

   Поскольку klcm=6, индексы в 6 и 2*6=12 позициях должны совпадать (выделены жирным). В итоге из каждого входного буфера будет обработана порция сэмплов, имеющих индексы до 24 включительно. После обработки t_sigproc_base::sample_id_last станет равным 24.

3. Для каждого буфера подлежащая обработке порция данных обрабатывается при помощи функции proc_seq_fsin_spec(). Эта функция выполняет обработку входной порции данных как целого до изменения частоты дискретизации. Пример ситуации, когда нужно использовать эту функцию – частотная фильтрация. Число каналов на выходе этой функции не обязано совпадать с числом каналов на входе. Если функция не определена, с данными ничего не происходит.

4. Производится снижение частоты дискретизации.

   Если params.params_base.downsamp_type=’thin’, то производится прореживание данных: сэмплы берутся начиная с базового сэмпла с шагом kdown(m). Число каналов остается прежним. Если kdown(m)=1, то данные не меняются.

   Если params.params_base.downsamp_type=’special’, то определяется последовательность входных сэмплов начиная с базового с шагом kdown(m). Для каждого сэмпла этой последовательности берется порция данных, заканчивающаяся этим сэмплом и имеющая длину params.inp_descs(m).nsamples_prev. Эта порция обрабатывается функцией proc_samp_fsin_spec(); выходной вектор этой функции записывается в позицию выходного буфера, соответствующую текущему сэмплу. Число каналов на выходе этой функции не обязано совпадать с числом каналов на входе. Пример ситуации, когда нужно использовать эту функцию – оконное Фурье-преобразование.

   Пример.

   Пусть kdown = 2, nsamples_prev = 3.

Входные данные:
Сэмплы: 1 2 3 4 5 6 7
Канал 1: 10 20 30 40 50 60 70
Канал 2: 11 22 33 44 55 66 77

Сэмплы, соответствующие выходным: [2 4 6] (т.к. kdown=2), перед каждым нужно взять 3 сэмпла.

Функция proc_samp_fsin_spec() будет вызвана три раза, на вход ее будут поданы следующие матрицы:
X1 = 0 10 20
0 11 22
X2 = 20 30 40
22 33 44
X3 = 40 50 60
44 55 66

Допустим, proc_samp_fsin_spec(X) дает на выходе значения 3-х каналов: f1(X), f2(X), f3(X).

Выходные данные:
Сэмплы: 2 4 6
Канал 1: f1(X1) f1(X2) f1(X3)
Канал 2: f2(X1) f2(X2) f2(X3)
Канал 3: f3(X1) f3(X2) f3(X3)

5. Производится повышение частоты дискретизации. Номера сэмплов и данные просто размножаются в нужное число раз.

   Пример.

   Пусть kup(m)=2

   Вход:
   Сэмплы: 1 2 3 4
   Канал 1: 10 20 30 40

   Выход:
   Сэмплы: 1 1 2 2 3 3 4 4
   Канал 1: 10 10 20 20 30 30 40 40

6. Обработанные данные со всех входов объединяются в общую матрицу. В качестве нового вектора индексов сэмплов выбирается вектор, полученный от входа, имеющего наибольшую частоту дискретизации

   Пример.

   Пусть srate_in(1)=100, srate_in(2)=50, srate_out=100. Тогда kdown=[1 1], kup=[2 1].

   Вход 1 (обработанный):
   Сэмплы: 1 2 3 4
   Канал 1: 10 20 30 40

   Вход 2 (обработанный):
   Сэмплы: 2 2 4 4
   Канал 1: 22 22 44 44

Объединение:
Сэмплы: 1 2 3 4
Канал 1: 10 20 30 40
Канал 2: 22 22 44 44

7. На объединенных данных (с выходной частотой дискретизации) вызывается функция proc_seq_fsout_spec(). Пример ситуации, когда нужно использовать эту функцию – пространственная фильтрация. Число каналов на выходе этой функции не обязано совпадать с числом каналов на входе. Если функция не определена, с данными ничего не происходит.
8. Полученные в итоге данные и последовательность индексов сэмплов добавляются в выходной буфер.

Создание собственного класса источника сигнала

Класс должен быть унаследован от t_eeg_recv_manager_base.

Необходимо правильно определить конструктор (см. t_eeg_recv_manager_test_1)

Необходимо переопределить функции:
- get_nchans() – возвращает пару значений: количество ЭЭГ-каналов и количество каналов с метками событий
- get_srate() – возвращает выходную частоту дискретизации
- get_chan_names_out() – возвращает список имен выходных ЭЭГ-каналов
- get_mark_chan_names_out() – возвращает список имен выходных каналов, содержащих метки событий
- start_recv() – начало приема
- stop_recv() – остановка приема
- restart() – перезапуск приема
- recv_data() – генерирует порцию данных

Также может быть переопределена функция init_spec().

Любая инициализация, которая требуется для корректной работы функций get_...() должна быть реализована внутри init_spec(). Соответственно, внутри init_spec() эти функции вызываться не должны. Никакие поля структуры params (кроме, возможно, params_spec) внутри init_spec() инициализировать не нужно.

Функции start_recv(), stop_recv(), restart() могут содержать любой произвольный код.

Функция recv_data имеет следующий вид:
nsamples_recv = recv_data(this, buf_eeg, buf_mark)
Функция генерирует порцию данных (например, читает из файла или принимает из энцефалографа). Генерируются собственно ЭЭГ-данные (каналы x сэмплы) и последовательность меток событий, имеющая ту же длину что и данные. Для сэмплов, которым не сопоставлены события, следует указывать код метки, равный NaN. ЭЭГ-данные и последовательность меток записываются в конец буферов buf_eeg и buf_mark соответственно. Функция возвращает количество сгенерированных сэмплов.

Создание собственного класса обработки сигнала

Новые классы обработки сигнала нужно наследовать от t_sigproc_base.

Нужно правильно определить конструктор (см. t_sigproc_test_1.m)

Можно переопределить следующие функции:
init_spec()
proc_seq_fsin_spec()
proc_samp_fsin_spec()
proc_seq_fsout_spec()

Если не переопределять proc_seq_fsin_spec() и proc_seq_fsout_spec(), то на выходе этих функций будут те же данные, что и на входе.
Если не переопределять proc_samp_fsin_spec(), то на выходе будет последний столбц входных данных. Эта функция используется только в случае params.params_base.downsamp_type='special'.

Внутри init_spec() можно задать значения следующих полей:
params.params_base.downsamp_type
params.params_base.upsamp_type
params.params_base.chan_names_out

Если не задавать эти параметры, будут использоваться значения по умолчанию (см. выше).

Также внутри init_spec() можно произвести любые подготовительные действия (например, создание фильтров).

Параметры, специфичные для нового класса, следует передавать через params.params_spec.

Переопределенные функции должны удовлетворять следующим требованиям.

1. Функции proc_seq_fsin_spec() и proc_seq_fsout_spec() должны на выходе иметь столько же сэмплов, сколько на входе.
2. Если params.params_base.downsamp_type='special', то число каналов на выходе proc_seq_fsin_spec() должно совпадать с числом каналов, ожидаемых на входе proc_samp_fsin_spec(), а число каналов на выходе proc_seq_fsin_spec() – с числом каналов, ожидаемых на входе proc_seq_fsout_spec().
3. Если params.params_base.downsamp_type='special', то число сэмплов, ожидаемых на входе proc_samp_fsin_spec(), должно быть равно params.inp_descs(m).nsamples_prev.
4. Если params.params_base.downsamp_type='thin', то число каналов на выходе proc_seq_fsin_spec() должно совпадать с числом каналов, ожидаемых на входе proc_seq_fsout_spec().
5. Число каналов, полученное в результате всей процедуры обработки, должно совпадать с числом имен выходных каналов, заданных в params.params_base.chan_names_out.

Описание классов

t_sigproc_base

Базовый класс для преобразования векторов значений сигнала / состояния / признака / предсказания.

Переменные

name - имя объекта, используется для ведения лога

params – параметры объекта. Часть параметров имеет общее назначение, а часть может быть специфичной для класса, являющегося потомком t_sigproc_base.

ready – флаг готовности. Устанавливается в 1 функцией t_sigproc_base::init()

sample_id_last – индекс последнего обработанного сэмпла. Это поле определяет позицию во входных буферах, с которой будет осуществляться чтение во время следующего вызова t_sigproc_base::do_work(). Поле обновляется также в ходе работы t_sigproc_base::do_work().

bufs_in – вектор ссылок на входные буферы

buf_out – ссылка на выходной буфер

Методы

t_sigproc_base (name) – конструктор
name – имя объекта, копируется в t_sigproc_base::name

init(params) – инициализация
params – параметры объекта. Копируется в t_sigproc_base::params

set_bufs(bufs_in, buf_out) – передача ссылок на входные буферы и на выходной буфер
bufs_in – входные буферы. Копируется в t_sigproc_base::bufs_in
buf_out – выходной буфер. Копируется в t_sigproc_base::buf_out

do_work() – чтение из входных буферов порции данных, обработка и копирование в выходной буфер

Описание тестового эксперимента

Для запуска эксперимента нужно запустить скрипт exp_test. В этом скрипте нужно установить путь к рабочему каталогу: dirpath_exp.

Используются следующие тестовые классы:
- t_eeg_recv_manager_test_1 < t_eeg_recv_manager
Генерирует несколько синусоидальных сигналов, частоты берутся из params_spec.sig_freqs.
- t_state_generator_null < t_state_generator
Всегда выдает одно и то же состояние.
- t_sigproc_test_1
Умножает входные данные на константы из массива, заданного в params_spec.mult_vals. Число выходных каналов превышает число входных каналов в length(mult_vals) раз.

1. Источник данных

   Источник генерирует сигнал, содержащий 3 канала:
   CHAN1 – синусоида с частотой 8 Гц;
   CHAN2 – синусоида с частотой 4 Гц;
   CHAN3 – синусоида с частотой 2 Гц.

Частота дискретизации источника: 600 Гц.

2. Стадия 1 (SIGPROC_STAGE_1)

   На вход поступает канал CHAN2 источника данных

   Входной сигнал умножается на -1

   Выходные каналы:
   CHAN2_(-1)

   Частота дискретизации: 150 Гц

3. Стадия 2 (SIGPROC_STAGE_2)

   На вход поступает канал CHAN1 источника данных

   Входной сигнал умножается на значения 1 и 2 (т.е. выходных кагалов 2, а не 1)

   Выходные каналы:
   CHAN1_(1)
   CHAN1_(2)

   Частота дискретизации: 100 Гц

4. Стадия 2 (SIGPROC_STAGE_3)

   На вход поступают следующие каналы:
   CHAN2_(-1) от SIGPROC_STAGE_1
   CHAN1_(2) от SIGPROC_STAGE_2
   CHAN1_(1) от SIGPROC_STAGE_2
   CHAN3 от источника данных

   Выходные каналы:
   CHAN2_(-1)_(1)
   CHAN1_(2) _(1)
   CHAN1_(1) _(1)
   CHAN3_(1)

   Частота дискретизации: 300 Гц

5. Используются два объекта визуализации. Первый (связан с верхним графиком) отображает каналы источника данных. Второй (связан с нижним графиком) отображает выходные каналы стадии 3.

Проверяется следующее:

- коэффициенты масштабирования (-1, 2, 1, 1)
- смена частоты дискретизации

График для канала CHAN3_(1) самый гладкий, т.к. сигнал претерпевает только понижение частоты дискретизации в 2 раза (600 -> 300)

Остальные графики для стадии 3 ступенчатые, т.к. сигналы претерпевают сначала сильное снижение частоты дискретизации, а потом ее повышение (600 -> 150 -> 300 и 600 -> 100 -> 300). Графики CHAN1_(2) _(1) и CHAN1_(1) _(1) более грубые, т.к. частота дискретизации этих сигналов снижалась наиболее сильно.

В каталог, заданный в dirpath_exp, после каждого запуска эксперимента будет записан лог, содержащий подробную информацию о стадиях обработки сигнала.

Визуализацию можно сильно ускорить, если отключить текстовые подписи к графикам: exp.visualizers(n).params.show_legend = 0

t_sigproc_iir

Параметры

params_spec.freq_bands – полосы частот, в которых осуществляется фильтрация (каждая полоса соответствует отдельному набору выходных каналов).
params_spec.freq_bands{n} = [fmin, fmax] – нижняя и верхняя граница n-й полосы. Если fmin==0, используется фильтр нижних частот; если fmax==Inf – используется фильтр верхних частот.
params_spec.filt_order – порядок фильтров (используется при вызове функции butter).
params_spec.chan_names_in_filt – имена входных каналов, которые подлежат фильтрации; остальные каналы будут скопированы в выходной буфер без изменений; если массив пуст – фильтруются все входные каналы.