Библиотека Krylov, версия 0.1
Руководство пользователя

Содержание

Назначение

Библиотека Krylov предназначена для решения систем линейных алгебраических уравнений (СЛАУ) большой размерности с разреженными квадратными невырожденными вещественными матрицами. Решение осуществляется итерационными методами в подпространствах Крылова. Целевой платформой библиотеки Krylov являются многопроцессорные вычислительные системы с распределенной памятью (кластеры) с установленной реализацией MPI.
Распараллеливание вычислений возможно как за счет использования многопоточных вычислений, так и на основе алгебраической декомпозиции областей. В последнем случае вспомогательные СЛАУ в подобластях решаются итерационными или прямыми методами, реализуя аддитивный метод Шварца, выступающий в роли переобуславливателя исходной матрицы в методе FGMRes.
Для решения СЛАУ необходимо задать матрицу в координатном формате (COO) или в формате CSR с тремя массивами. В случае использования алгебраической декомпозиции матрица задается в одном из MPI-процессов определенного коммуникатора. Этот процесс мы будем называть *основным*. Остальные (задействованные в решении) процессы могут не содержать никаких данных о СЛАУ и их множество, вообще говоря, может быть пустым. Эти процессы мы будем называть *рабочими*.
Правая часть и начальное приближение также задаются в основном процессе в виде одномерных векторов с двойной точностью.

Ограничение: на данный момент MPI-процесс может участвовать в решении только одной СЛАУ в качестве как основного, так и рабочего процесса.

Обозначения

Функции, процедуры, переменные и другие синтаксические элементы языков программирования выделяются моноширинным шрифтом, подпрограммы дополняются круглыми скобками после имени, например:
`function()`
`variable`
Знаком "`?`" обозначается произвольный символ, а знаком "`*`" — произвольный набор символов, включая пустой. Если, например, есть функции `func()`, `func1()`, `func2()`, `func12()`, то `func*()` подразумевает их все, в то время как `func?()` означает, что речь ведется о `func1()` и `func2()`, но не `func()` или `func12()`.

Порядок работы

В начале работы следует с помощью функции `kr_new_handle()` выделить дескриптор, который в дальнейшем будет описывать СЛАУ и все связанные с ней параметры. Затем, используя функции `kr_set_*param()`, можно изменить желаемые параметры, например, требуемую точность, максимальное количество итераций и т.п. После этого к дескриптору необходимо привязать матрицу системы, вызвав `kr_setup` или `kr_schwarz_setup()` в основном процессе (в рабочих этому шагу логически соответствует вызов `kr_schwarz_worker()`). Решение СЛАУ выполняется с помощью функции `kr_solve()`. По окончанию работы следует высвободить ресурсы, вызвав `kr_release_handle()`. Функция `MPI_Init()` должна вызываться до `kr_schwarz_setup()` и `kr_schwarz_worker()`, а функция `MPI_Finalize()` — после вызова `kr_release_handle()` в основном процессе и после возвращения управления из `kr_schwarz_worker()`, соответственно.

Интерфейс библиотечных подпрограмм для языка C

Все интерфейсные функции для языка C возвращают 0 в случае корректного завершения всех действий и ненулевое значение, если произошла ошибка. В последнем случае следует уточнить причину ее возникновения с помощью `kr_strerror()`. Коды ошибок и их значения пока не перечислены, но, вероятно, будут. Все функции объявлены в `krylov.h`. Аргументы, если не оговорено иное, являются входными. Во избежание путаницы в данном разделе *строкой* и *ссылкой на строку* будем называть указатель на первый элемент массива символов, заканчивающийся нулем (т.н. нуль-терминированная или C-строка).

kr_new_handle

```
int kr_new_handle(int *handle);
```

Данная функция выделяет дескриптор для решения СЛАУ, использующийся в последующих вызовах библиотечных функций. `handle` является выходным параметром и должен указывать на доступную для записи переменную. На данный момент возможно выделение лишь одного дескриптора, при нарушении этого ограничения функция вернет ненулевое значение.

kr_release_handle

```
int kr_release_handle(int handle);
```

`kr_release_handle()` завершает работу с СЛАУ, описываемой дескриптором `handle`, и высвобождает все дополнительные данные. Дескриптор становится недействительным.

После того, как данная функция возвращает управление, возможно решение СЛАУ с другой матрицей.

kr_set_*param

```
int kr_set_param(int handle, char *param, void *avalue);
int kr_set_sparam(int handle, char *param, char *svalue);
int kr_set_iparam(int handle, char *param, int ivalue);
int kr_set_dparam(int handle, char *param, double dvalue);
```

`kr_set_*param()` задают опции для построения переобуславливателя и решения СЛАУ, присваивая параметру `param` дескриптора `handle` значение `avalue` (`svalue`, `ivalue`, `dvalue`). Использование функции `kr_set_param()` рекомендуется опытным разработчикам. Указатель `avalue` должен ссылаться на строку, 32-разрядное целое или 64-разрядное вещественное число. Тип значения, на которое ссылается `avalue`, определяется из строки `param`.
Функции `kr_set_sparam()`, `kr_set_iparam()`, `kr_set_dparam()` выполняют аналогичную функцию, но имеют более строгий синтаксис и принимают на вход строки, целые и вещественные числа, соответственно. Список параметров приводится ниже.

kr_setup

```
int kr_setup(int handle, int n, int nnz, int *ia, int *ja, double
*a);
```

`kr_setup()` выполняет построение переобуславливателя и создание всех дополнительных структур данных. На данном этапе выполняется привязка матрицы СЛАУ к дескриптору. Параметры, заданные с помощью `kr_set_*param()`, вступают в силу. Дальнейшее их изменение не оказывает влияния на процесс решения СЛАУ. Матрица может быть представлена в одном из двух форматов: COO или CSR. Если `nnz` не равно 0, то считается, что матрица задана в формате COO, в противном случае данные интерпретируются как CSR-матрица. Индексы должны начинаться с 0.

kr_schwarz_setup

```
int kr_schwarz_setup(int handle, MPI_Comm comm, int n, int nnz,
int *ia, int *ja, double *a);
```

`kr_schwarz_setup()` выполняет построение переобуславливателя Шварца и создание всех дополнительных структур данных. На данном этапе выполняется привязка матрицы СЛАУ к дескриптору. Переданный в функцию MPI-коммуникатор `comm` дублируется с помощью `MPI_Comm_dup()` и его использование возможно после возвращения управления в вызывающую программу. Параметры, заданные с помощью `kr_set_*param()`, вступают в силу. Дальнейшее их изменение не оказывает влияния на процесс решения СЛАУ. Матрица может быть представлена в одном из двух форматов: COO или CSR. Если `nnz` не равно 0, то считается, что матрица задана в формате COO, в противном случае данные интерпретируются как CSR-матрица. Индексы должны начинаться с 0.
Перед обращением к `kr_schwarz_setup()` программа должна вызвать `MPI_Init()`.

kr_schwarz_worker

```
int kr_schwarz_worker(MPI_Comm comm);
```

`kr_schwarz_worker()` запускает часть решателя для рабочего процесса. Эта функция

должна вызываться всеми рабочими процессами коммуникатора comm, в противном случае программа зависнет. Перед обращением к `kr_schwarz_worker()` вызывающая программа должна вызвать `MPI_Init()`.

kr_solve

```
int kr_solve(int handle, double *f, double *u);
```

`kr_schwarz_solve()` выполняет решение СЛАУ с заданными правой частью `f` и начальным приближением `u`. Эта функция может вызываться столько раз, сколько потребуется, однако не должна вызываться повторно до завершения ее работы. Результат расчетов записывается в u, даже если критерии сходимости не удовлетворены. В последнем случае также возвращается ненулевой код ошибки.

kr_strerror

```
const char * kr_strerror(int error);
```

Данная функция возвращает строку с описанием ошибки `error`. Строка является постоянной, и изменять ее содержимое и высвобождать занимаемую ей память нельзя.

kr_get_*stat

```
int kr_get_stat(int handle, char *stat, void *avalue);
int kr_get_istat(int handle, char *stat, int *ivalue);
int kr_get_dstat(int handle, char *stat, double *dvalue);
```

Функции `kr_get_*stat()` позволяют выяснить некоторые характеристики итерационного процесса: количество выполненных итераций, использование памяти, процессорного и календарного времени.
`kr_get_stat()` определяет тип данных, на которые указывает `avalue`, из строки `stat`. `kr_get_istat()` и `kr_get_dstat()` предназначены для считывания целочисленных и вещественных характеристик, соответственно. Список характеристик приводится ниже.

Имена параметров для kr_set_*param() и их значения

В данном разделе перечислены параметры, их смысл, допустимые значения и значения по умолчанию.

cgc:interval	целый	10	Количество итераций решателя между шагами грубосеточной корректировки. Если данному параметру присвоено значение 0, грубосеточная корректировка не производится.
solver:type	строковый	"fgmres"	Используемый внешний итерационный решатель. На данный момент доступно лишь значение "fgmres", соответствующее итерационному методу FGMRes.
solver:nmax	целый	500	Максимальное количество внешних итераций.
solver:nres	целый	100	Период рестарта внешних итераций.
solver:tol	вещественный	1e-8	Желаемая точность решения СЛАУ; указывает целевое отношение нормы невязки к норме правой части.

precond:type	строковый	"eisen"	В случае, когда не используется алгебраическая декомпозиция областей, указывает используемый переобуславливатель. Доступные значения: "eisen" — модификация Айзенштата; "id" — тождественный оператор (это соответствует решению системы без переобуславливания).
subsv:type	строковый	"fgmres"	Используемый в подобластях решатель. Доступные значения: "pardiso" позволяет использовать решатель PARDISO из библиотеки Intel® MKL. При использовании данного решателя доступен для изменения целочисленный параметр "pardiso:nth", указывающий количество потоков, используемых при факторизации матриц в подобластях. "fgmres" позволяет задействовать итерационный метод FGMRes. При использовании итерационных решателей имеют значение параметры subsv:nmax, subsv:tol, subpc:type. Если же задействован прямой решатель PARDISO, эти параметры не обрабатываются.
subsv:nmax	целый	10	Указывает максимальное количество итераций при решении СЛАУ в подобластях итерационным методом. Рекомендуется оставить это значение неизменным.
subsv:tol	вещественный	0.1	Параметр, аналогичный solver:tol, однако отвечающий за точность решения СЛАУ в подобластях. Рекомендуемое значение: 0.1. Дальнейшее уменьшение этого параметра, как правило, лишь увеличивает трудоемкость, т.к. скорость сходимости внешнего метода остается прежней.
subpc:type	строковый	"eisen"	Используемый в подобластях предобуславливатель. Доступны те же значения, что и для параметра precond:type.
pardiso:nth	целый	1	Указывает количество потоков, используемых решателем PARDISO из библиотеки Intel® MKL.

Имена параметров для kr_get_*stat() и их значения

В данном разделе перечислены коды статистических данных, которые можно прочитать с помощью функций kr_get_*stat().

niter	целый	Количество итераций внешнего решателя.
walltime	вещественный	Календарное время в секундах, затраченное решателем. Суммируется все время, относящееся к одному дескриптору.
cputime	вещественный	Процессорное время в секундах, затраченное решателем. Суммируется все время, относящееся к одному дескриптору.
memusage	вещественный	Количество использованной всеми MPI-процессами памяти, в мегабайтах.