Требуемые пакеты

cmake gfortran mpich

Сборка

Сборка автоматическая с помощью CMake.

1. Перейти в Build.
2. Перейти в требуемый вариант сборки(Релиз или отладка)
3. Запустить sh файл.
4. После завершения процесса, в каталоге bin будут находиться исполняемые файлы тестов и замеров времени, в inc будут находиться mod-файлы, сама библиотека находится в lib.
5. Библиотека подключается как обычная библиотека Fortran - указывается путь к библиотеке и mod-файлам. Если положить по системному пути - указывать пути будет не нужно.

Структура

В каталоге src находятся исходные коды функций, разложенные по отдельным каталогам.
В каталоге tst находятся тестовые программы используемые для тестирования функций.
В каталоге bnch находятся более сложные тестовые программы, использующиеся для замера времени выполнения функций.

Функции

Основные

Все функции имеют 3 варианта, если не указано иного - последовательный и параллельный, имеющие одинаковый набор аргументов.
Распараллеленные с помощью технологии OpenMP имеют постфикс _omp
Распараллеленные с помощью технологии MPI имеют постфикс _mpi
Пример:

- cells - последовательный
- cells_omp - openmp
- cells_mpi - mpi
  Из данных функций будет явно указана только cells

Нахождение интеграла

Различные численные методы нахождения интеграла.
Пока в рамках отладки методы нахождения интеграла проводят только одну итерацию.

В дальнейшем будет включен цикл по ошибке.

Метод Ячеек

real function cells(nx, ny, ax, bx, ay, by, eps, f) result(res)
integer :: nx, ny - количество разбиений по оси x/y
real ::
    ax, ay - начало отрезка оси x/y
    bx, by - конец отрезка оси x/y
    eps - погрешность расчёта
    f - функция z=f(x,y)

Метод Гаусса

real function gauss_integrate(a, b, n, p, eps, f) result(res)
integer ::
    n - количество разбиений
    p - количество точек в формуле гаусса
real ::
    a - начало отрезка
    b - конец отрезка
    eps - погрешность расчёта
    f - функция y=f(x)

Метод Монте-Карло

Одномерный:
real function montekarlo_one(counter, a, b, eps, f) result(res)
integer :: counter - количество испытаний
real ::
    a - начало отрезка
    b - конец отрезка
    eps - погрешность расчёта
    f - функция y=f(x)
Двумерный:
real function montekarlo_two(counter, ax, bx, ay, by, eps, f) result(res)
integer :: counter - количество испытаний
real ::
    ax, ay - начало отрезка оси x/y
    bx, by - конец отрезка оси x/y
    eps - погрешность расчёта
    f - функция z=f(x,y)

Метод Симпсона

real function simpson(a, b, n, eps, f) result(res)
integer :: n - количество разбиений
real ::

a - начало отрезка
b - конец отрезка
eps - погрешность расчёта
f - функция y=f(x)

Методы Прямоугольников

Левых:
real function left_square(a, b, n, eps, f) result(res)
Центральных:
real function central_square(a, b, n, eps, f) result(res)
Правых:
real function right_square(a, b, n, eps, f) result(res)
integer :: n - количество разбиений
real ::
a - начало отрезка
b - конец отрезка
eps - погрешность расчёта
f - функция y=f(x)

Нахождение обратной матрицы

Метод Гаусс-Жордана

function invgj(A, n) result(inv)
integer :: n - размер матрицы
real, dimension(n,n) ::
A - Матрица, для которой нужна обратная
inv - найденная обратная матрица

Метод нахождения обратной матрицы при помощи решения СЛАУ с различной правой частью

function invslau(A, n) result(inv)
integer :: n - размер матрицы
real, dimension(n,n) ::
A - Матрица, для которой нужна обратная
inv - найденная обратная матрица

Метод Шульца

Примечание: *Должно работать, но не до конца протестирован*
function shultz(A, n, alpha, eps) result(Inv)
integer :: n - размер матрицы
A - Матрица, для которой нужна обратная
alpha - начальный множитель
eps - допустимая погрешность измерений
inv - найденная обратная матрица

Умножение Матриц

Метод Кэннона

Только mpi
Запускать на квадратном числе процессов/узлов
function cannon(i_A, i_B, i_n) result(res)
integer :: i_n - размер матрицы
real, dimension(i_n,i_n) ::
    i_A, i_B - матрицы для умножения
    res - результат

Простой блочный метод

function cblock(A, B, n, ichunk) result(C)
integer ::
    n - размер матрицы
    ichunk - размер блока
real, dimension(i_n,i_n) ::
    A, B - матрицы для умножения
    res - результат

Классическое умножение

function classic(A, B, n) result(C)
integer :: n - размер матрицы
real, dimension(i_n,i_n) ::
    A, B - матрицы для умножения
    res - результат

Метод Фокса

Только mpi
Запускать на квадратном числе процессов/узлов
function fox(A, B, i_n) result(res)
integer :: i_n - размер матрицы
real, dimension(i_n,i_n) ::
    A, B - матрицы для умножения
    res - результат

Методы линейного программирования

До конца не протестированы, поэтому пока более стабильные последовательные варианты.

Симплекс-метод

function simplex_method(func_vec, constraints, eq_type, var_num, constraint_num)
result(res_vector)

integer ::
    var_num - количество переменных
    constraint_num - количество ограничений
real, dimension(var_num) :: func_vec - вектор функции, содержит последовательные множители для вычисления результата.
real, dimension(var_num+1,constraint_num) :: constraints - набор векторов ограничения в формате: var_num последовательных множителей переменных и значение правой стороны.
integer, dimension(constraint_num) :: eq_type - содержит целочисленное значение - индикатор типа уравнения между правой и левой частью соответствующего ограничения.
Виды:
    >= => -1
    <= => 1
     = => 0
real, dimension(:), allocatable :: res_vector - вектор выходных значений.

Решение СОДУ

Метод Эйлера

function eiler_system_solve_matrix(h, n, matrix, eps) result(res) integer :: n - размер матрицы
real ::
    h - размер шага
    eps - допустимая погрешность
real, dimension(n) :: res - выходной вектор
real, dimension(n,n) :: matrix - СОДУ в матричной форме
Следующие методы имеют аналогичные аргументы вызова
Метод Рунге-Кутты 4
function rk_system_solve_matrix(h, n, matrix, eps) result(res)
Метод Адамса 5
function adams5_system_solve_matrix(h, n, matrix, eps) result(res)

Решение СЛАУ

Метод Гаусса

function gauss(a, b, n) result(res)
integer :: n - размер матрицы
real, dimension(n) ::
    b - правые значения
    res - результат
real, dimension(n,n) ::
    a - матрица системы линейных алгебраических уравнений

Метод Якоби или простой итерации

function jacobi(a, b, n, eps) result(res)
integer :: n - размер матрицы
real :: eps - допустимая погрешность

real, dimension(n) ::
    b - правые значения
    res - результат
real, dimension(n,n) :: a - матрица системы линейных алгебраических уравнений

Метод LU-разложения

function lud(A, b, n) result(res)
integer :: n - размер матрицы
real, dimension(n) ::
    b - правые значения
    res - результат
real, dimension(n,n) :: a - матрица системы линейных алгебраических уравнений

Метод Зейделя

function zeidel(A, b, n, eps) result(res) integer :: n - размер матрицы
real :: eps - допустимая погрешность
real, dimension(n) ::
    b - правые значения
    res - результат
real, dimension(n,n) :: a - матрица системы линейных алгебраических уравнений

Метод Прогонки

Применим только для матриц трёхдиагонального вида
function tda_m(a, b, n) result(res)
integer :: n - размер матрицы
real, dimension(n) ::
    b - вектор свободных членов
    res - результат
real, dimension(n,n) :: a - трёхдиагональная матрица
function tda_v(c, a, b, d, n) result(res)
integer :: n - размер векторов
real, dimension(n) ::
    c - диагональ
    a - поддиагональ
    b - наддиагональ
    d - вектор свободных членов
    res - результат
Данные функции можно вызвать через интерфейс tda. Он автоматически выберет требуемую функцию в зависимости от переданных аргументов.

Решение СНЛУ методом Ньютона

function newton_func_system(func, xstart, eps, n) result(xnew)
integer :: n - количество уравнений
real :: eps - допустимая погрешность

real, dimension(n) :: xstart - вектор начальных значений
procedure(efunc) :: func - входящая векторная функция вида efunc
function efunc(x) result(result) - явный интерфейс векторной функции efunc - result=f(x)
    real, dimension(:), intent(in) :: x
    real, dimension(size(x)) :: result

Дополнительные функции

Умножение матрицы на вектор

function matrixvector(a, b, n) result(res)
integer :: n - размер матрицы и вектора
real, dimension(n) ::
    res - произведение
    b - вектор
real, dimension(n,n) :: a - матрица

Скалярное произведение векторов

real function scalar_mult(a, b) result(res)
real :: res - произведение
real, dimension(:) ::
    a - меньший по размеру вектор
    b - больший по размеру вектор

Дополнительные вспомогательные функции

Сортировка слиянием

function merge_sort(array, n) result(res)
integer :: n - размерность массива
real, dimension(n) :: array, result - векторы входа/выхода

Нормы

Корень из разности суммы квадратов всех элементов вектора для двух вектороы

real function v_norm2(a, b, n)
integer :: n - размер вектора
real, dimension(n) :: a, b - векторы

Корень из суммы квадратов всех элементов матрицы

real function m_norm4(A, n) result(res) integer :: n - размер матрицы
real :: res - результат real, dimension(n,n) :: a - матрица

Численное дифференцирование

Функции численного дифференцирования имеют следующий шаблон именования:
d<номер производной>_<Количество точек рассчёта>_<Дополнительная информация о функции>_<Тип функции> Все функции одного типа имеют однаковый набор аргументов

Тип b - базовый

real function d1x_2p_fd_b(x, h, f) result(res) - передняя разность
real function d1x_2p_cd_b(x, h, f) result(res) - центральная разность
real function d1x_5p_b(x, h, f) result(res)
real function d2x_5p_b(x, h, f) result(res)
real function d3x_5p_b(x, h, f) result(res)
real function d4x_5p_b(x, h, f) result(res)
real ::
    x - точка в которой находится производная
    h - шаг дифференцирования
    f - дифференцируемая функция
    res - результат

Тип mult - градиент по данной формуле численного дифференцирования

function d1x_2p_fd_mult(x, h, f, n, d_var) result(res)
integer ::
n - размер вектора
d_var - позиция в векторе, по которой производить дифференцирование
real :: h - шаг дифференцирования real, dimension(n) ::
    x - вектор точки
    hv - вектор шага
    res - вектор результата
    procedure(efunc) - см метод Ньютона

Тип mult_select - найти градиент и вернуть значение в данной позиции

function d1x_2p_fd_mult_select(x, h, f, n, pos) result(res)
pos определяет позицию в векторе, по которой произвести дифференцирование, затем возвращается дифференциал res в данной позиции
pos - integer
res - real
Остальные аргументы аналогичны предыдущему типу.

Градиент

Данная функция является обёрткой над функциями типа select и mult_select
function gradient(x, h, f, n, pos, approx) result(res)
approx - численный метод нахождения градиента
В последовательной версии требуется функция типа select
Параллельные для ускорения требуют функцию типа mult_select

Матрица Якоби

function jacobi_matrix(exfunc, vect, step, n, approx) result(res)
integer :: n - размерность вектора
real ::
exfunc - векторная функция y=f(x)

approx - численный метод нахождения градиента
step - шаг численного метода
real, dimension(n) :: vect - ветор точек
real, dimension(n,n) :: res - якобиан

Сумма элементов вектора

real function psum(a) result(res)
real :: res - сумма real, dimension(:) :: a - вектор

Определитель

function determinant(a, n) result(res)
integer :: n - размер матрицы
real, dimension(n,n) ::
    a - матрица, для которой надо найти определитель
    res - выходная матрица

На основе метода Гаусса

Установка треугольника для обнуления методом гаусса
Данная подпрограмма только последовательная
subroutine set_triangle_type(vertical, horizontal, n)
integer :: n - размер матрицы
character :: vertical, horizontal - обозначение треугольника для обнуления. Пара значений.
Допустимые значения - 'd', 'l' и 'u', 'r'
После установки треугольника можно использовать следующие функции:
Прямой ход метода гаусса
subroutine null_triangle_gauss(a, b)
real, dimension(:) :: b - вектор свободных членов
real, dimension(:,:) :: a - матрица
Обратный ход метода гаусса
subroutine null_triangle_gauss_b(a, b)
Приведение матрицы к треугольному виду
subroutine null_triangle_matrix(a)
То же и вернуть произведение диагональных множителей
real function null_triangle_matrix_get_normalise_mult(a) result(res)
При приведении матрицы к треугольному виду повторять все действия на второй матрице
subroutine null_triangle_matrix_mirror(A, mirror)
Для варианта MPI добавляется параметр в конце n - размерность матрицы:
subroutine null_triangle_matrix_mpi(a, n)
LU-разложение матрицы
subroutine lu_decomposition(a, n, l, u)
integer :: n - размерность матрицы
real, dimension(n,n) :: a, l, u - матрицы, входная и L, U

Дополнительные вспомогательные функции имеющие только последовательный вариант

Формирование на основе вектора матрицы с диагональю из данного вектора
function vectortodiagonal(v) result(d)
real, dimension(:) :: v - вектор
real, dimension(size(v),size(v)) :: d - матрица

Возведение диагональной матрицы в степень
function diagonalmatrixpower(matrix, power) result(powered)
integer :: power - степень
real, dimension(:,:) :: matrix - исходная матрица powered - результат

Генерация единичной матрицы заданного размера
function genidenmat(n) result(res)
integer :: n - размер матрицы real, dimension(n,n) :: res - единичная матрица

Возведение любой матрицы в степень
function matpow(matrix, i_n, power) result(res)
integer ::
    i_n - размерность матрицы
    power - степень
real, dimension(i_n,i_n) ::
    matrix - исходная матрица
    res - результат

Нахождение суммы квадратов матрицы
function matsumpow(matrix, i_n, power) result(res)
Аналогично matpow

ОДУ

Решение уравнения ОДУ
real function <метод>_function_solve(a, b, h, f) result(y)
real ::
    a - начало отрезка
    b - конец отрезка
    h - шаг
    f - функция z=f(x)
    y - результат

Решение векторного уравнения ОДУ
real function <метод>_function_solve_vector(a, b, h, n, vf, vs) result(y)
integer :: n - размерность вектора
real ::
    a - начало отрезка
    b - конец отрезка
    h - шаг
real, dimension(n) ::

f - векторная функция z=f(x)
vs - начальный вектор
y - вектор результата

НЛУ

Решение нелинейного уравнения с одной переменной
real function newton_func_onev(func, xstart, eps) result(xnew)
real ::
func - функция
xstart - начальная точка
eps - допустимая погрешность
xnew - результат
Решение нелинейного уравнения с исключением уже найденных значений
real function newton_func_manyv(func, xstart, eps, xfound) result(xnew) real ::
func - функция
xstart - начальная точка
eps - допустимая погрешность
xnew - результат
real, dimension(:) :: xfound - вектор уже найденных корней
Решение нелинейного уравнения с одной переменной и несколькими корнями
function newton_func_onev_all_ione(func, xstart, eps, n) result(xnew)
integer :: n - количество решений
real ::
func - функция
eps - допустимая погрешность xstart - начальная точка
real, dimension(n) :: xnew - результат
Поиск нескольких корней нелинейного уравнения
Не гарантирует отсутствия дубликации корней
function newton_func_onev_imany(func, xstart, eps, n) result(xnew)
integer :: n - количество решений
real ::
func - функция
eps - допустимая погрешность
real, dimension(n) ::
xstart - начальная точка
xnew - результат
Поиск нескольких корней нелинейного уравнения с исключением уже найденных значений
function newton_func_onev_all_imany(func, xstart, eps, n) result(xnew)
integer :: n - количество решения
real :: eps - допустимая погрешность
real, dimension(n) ::
func - векторная функция
xnew - результат
xstart - начальная точка

Функции по модулям

Подключение с помощью use. Перечисление функций по модулям.

cells_lib

real function cells(nx, ny, ax, bx, ay, by, eps, f) result(res)

gaussi_lib

real function gauss_integrate(a, b, n, p, eps, f) result(res)

montekarlo_lib

real function montekarlo_one(counter, a, b, eps, f) result(res)
real function montekarlo_two(counter, ax, bx, ay, by, eps, f) result(res)

simpson_lib

real function simpson(a, b, n, eps, f) result(res)

squares_lib

real function left_square(a, b, n, eps, f) result(res)
real function central_square(a, b, n, eps, f) result(res)
real function right_square(a, b, n, eps, f) result(res)

invgj_lib

function invgj(A, n) result(inv)
integer :: n - размер матрицы

invslau_lib

function invslau(A, n) result(inv)

shultz_lib

function shultz(A, n, alpha, eps) result(Inv)

cannon_lib

function cannon(i_A, i_B, i_n) result(res)

cblock_lib

function cblock(A, B, n, ichunk) result(C)

classic_lib

function classic(A, B, n) result(C)
function matrixvector(a, b, n) result(res)
real function scalar_mult(a, b) result(res)

fox_lib

function fox(A, B, i_n) result(res)

strassen_lib

function strassen(A, B, n) result(C)
function recursive_strassen(A, B, n) result(C)

simplex_lib

function simplex_method(func_vec, constraints, eq_type, var_num, constraint_num) result(res_vector)

eiler_lib

function eiler_system_solve_matrix(h, n, matrix, eps) result(res) real function eiler_function_solve(a, b, h, f) result(y)
real function eiler_function_solve_vector(a, b, h, n, vf, vs) result(y)

rk_lib

function rk_system_solve_matrix(h, n, matrix, eps) result(res)
real function rk_function_solve(a, b, h, f) result(y)
real function rk_function_solve_vector(a, b, h, n, vf, vs) result(y)

adams5_lib

function adams5_system_solve_matrix(h, n, matrix, eps) result(res)
real function adams5_function_solve(a, b, h, f) result(y)
real function adams5_function_solve_vector(a, b, h, n, vf, vs) result(y)

gauss_lib

function gauss(a, b, n) result(res)
subroutine set_triangle_type(vertical, horizontal, n)
subroutine null_triangle_gauss(a, b)
subroutine null_triangle_gauss_b(a, b)
subroutine null_triangle_matrix(a)
real function null_triangle_matrix_get_normalise_mult(a) result(res)
subroutine null_triangle_matrix_mirror(A, mirror)
subroutine null_triangle_matrix_mpi(a, n)

jacobi_lib

function jacobi(a, b, n, eps) result(res)

lud_lib

function lud(A, b, n) result(res)
subroutine lu_decomposition(a, n, l, u)

zeidel_lib

function zeidel(A, b, n, eps) result(res)

newton_lib

function newton_func_system(func, xstart, eps, n) result(xnew)
real function newton_func_onev(func, xstart, eps) result(xnew)
real function newton_func_manyv(func, xstart, eps, xfound) result(xnew) function
newton_func_onev_all_ione(func, xstart, eps, n) result(xnew)
function newton_func_onev_imany(func, xstart, eps, n) result(xnew)
function newton_func_onev_all_imany(func, xstart, eps, n) result(xnew)

norm_lib

real function v_norm2(a, b, n)
real function m_norm4(A, n) result(res)

num_diff_lib

real function d1x_2p_fd_b(x, h, f) result(res) real function d1x_2p_cd_b(x, h, f) result(res) real function d1x_5p_b(x, h, f) result(res)
real function d2x_5p_b(x, h, f) result(res)
real function d3x_5p_b(x, h, f) result(res)
real function d4x_5p_b(x, h, f) result(res)
real function d1x_2p_fd_mult(x, h, f) result(res) real function d1x_2p_cd_mult(x, h, f) result(res)
real function d1x_5p_mult(x, h, f) result(res)
real function d2x_5p_mult(x, h, f) result(res)
real function d3x_5p_mult(x, h, f) result(res)
real function d4x_5p_mult(x, h, f) result(res) real function d1x_2p_fd_mult_select(x, h, f) result(res) real function d1x_2p_cd_mult_select(x, h, f) result(res) real function d1x_5p_mult_select(x, h, f) result(res)
real function d2x_5p_mult_select(x, h, f) result(res)
real function d3x_5p_mult_select(x, h, f) result(res)
real function d4x_5p_mult_select(x, h, f) result(res) function gradient(x, h, f, n, pos, approx) result(res)
function jacobi_matrix(exfunc, vect, step, n, approx) result(res)

psum_lib

real function psum(a) result(res)

determinant_lib

function determinant(a, n) result(res)

utilites_matrix_vector

function vectortodiagonal(v) result(d)
function diagonalmatrixpower(matrix, power) result(powered)
function genidenmat(n) result(res)
function matpow(matrix, i_n, power) result(res)
function matsumpow(matrix, i_n, power) result(res)

tda_lib

function tda_m(a, b, n) result(res)
function tda_v(c, a, b, d, n) result(res)
inteface tda

sort_merge

function merge_sort(array, n) result(res)