Задача-проект по курсу «Применение высокопроизводительных вычислительных технологий для моделирования задач сейсморазведки»

Цыбулин Иван
tsybulinhome@gmail.com

1 Постановка задачи

1.1 Задача сейсморазведки

Сейсморазведкой называются методы исследования земной коры, основанные на изучении искусственно возбуженных упругих волн. С ее помощью можно изучить глубинное строение Земли и обнаружить места возможных скоплений полезных ископаемых.

В основе сейсморазведки лежит возбуждение с помощью специального источника упругих волн в геологический среде, которые затем регистрируются сейсмоприемниками.

Различают прямую и обратную задачу сейсморазведки. Под прямой задачей понимается нахождение отклика от заданного воздействия в известной среде, обратная задача состоит в нахождении параметров среды по полученным откликам.

Обратная задача крайне сложна, но именно она является основной задачей сейсморазведки. Практически все методы решения обратной задачи так или иначе опираются на решение прямой задачи.

1.2 Математическая модель

Распространение волн по геологической среде можно описать уравнениями динамики упругих сред, которые выражаются в совокупности уравнения импульса и закона Гука.

Состояние упругой среды в точке можно описать вектором смещений $\mathbf{u}$ и тензором деформаций $\hat{e} = \frac{1}{2}\left(grad\mathbf{u} + (grad\mathbf{u})^{\mathrm{T}} \right)$. Закон Гука связывает тензор деформаций с тензором упругих напряжений

$$\hat{\sigma} = \lambda \hat{I} Tr\hat{e} + 2\mu\hat{e}, \tag{1}$$

где λ, μ — коэффициенты Ламэ, зависящие от среды. Запишем закон сохранения импульса

$$\rho\ddot{\mathbf{u}} = div\hat{\sigma}, \tag{2}$$

где ρ — плотность среды. Подставляя (2) в (1) и считая коэффициенты постоянными, получаем

$$\rho\ddot{\mathbf{u}} = \mu\Delta\mathbf{u} + (\mu + \lambda) graddiv\mathbf{u}. \tag{3}$$

Обозначим $\mathbf{u} \equiv (u_x, u_y)$,

$$\begin{cases} \rho\ddot{u}_x = \mu\left(\dfrac{\partial^2 u_x}{\partial x^2} + \dfrac{\partial^2 u_x}{\partial y^2} \right) + (\mu + \lambda)\left(\dfrac{\partial^2 u_x}{\partial x^2} + \dfrac{\partial^2 u_y}{\partial y \partial x} \right) \\ \rho\ddot{u}_y = \mu\left(\dfrac{\partial^2 u_y}{\partial x^2} + \dfrac{\partial^2 u_y}{\partial y^2} \right) + (\mu + \lambda)\left(\dfrac{\partial^2 u_y}{\partial y^2} + \dfrac{\partial^2 u_x}{\partial x \partial y} \right) \end{cases} \tag{4}$$

Данная система уравнений имеет сходство с обычным волновым уравнением, но в этой

системе есть два типа волн с разными скоростями, так называемые волны сжатий (p-волны), которые распространяются со скоростью $c_p = \sqrt{\frac{\lambda + 2\mu}{\rho}}$ и волны сдвигов (s-волны), которые распространяются со скоростью $c_s = \sqrt{\frac{\mu}{\rho}} < c_p$.

Перегруппируем производные

$$\begin{cases} \rho \ddot{u}_x = (\lambda + 2\mu)\frac{\partial^2 u_x}{\partial x^2} + \mu \frac{\partial^2 u_x}{\partial y^2} + (\lambda + \mu)\frac{\partial^2 u_y}{\partial y \partial x} \\ \rho \ddot{u}_y = (\lambda + 2\mu)\frac{\partial^2 u_y}{\partial y^2} + \mu \frac{\partial^2 u_y}{\partial x^2} + (\lambda + \mu)\frac{\partial^2 u_x}{\partial x \partial y} \end{cases} \quad (5)$$

и перепишем (5) пользуясь c_p и c_s:

$$\begin{cases} \ddot{u}_x = c_p^2 \frac{\partial^2 u_x}{\partial x^2} + c_s^2 \frac{\partial^2 u_x}{\partial y^2} + (c_p^2 - c_s^2)\frac{\partial^2 u_y}{\partial y \partial x} \\ \ddot{u}_y = c_p^2 \frac{\partial^2 u_y}{\partial y^2} + c_s^2 \frac{\partial^2 u_y}{\partial x^2} + (c_p^2 - c_2^2)\frac{\partial^2 u_x}{\partial x \partial y}. \end{cases} \quad (6)$$

Уравнения в данном виде будут использоваться для численного метода.

1.3 Начально-краевая задача

В рамках проекта, Вам будет предложено решить систему уравнений (6) в квадратной области $G = [-1,1]\times[-2,0]$ со следующими начальными условиями:

$$\begin{cases} \mathbf{u}|_{t=0} = \frac{25\mathbf{r}}{\sqrt{x^2 + y^2}} y^2 e^{-25(x^2+y^2)} \\ \frac{\partial \mathbf{u}}{\partial t}|_{t=0} = 0 \end{cases} \quad (7)$$

и граничным условием $\mathbf{u}|_{\partial G} = 0$ до момента времени $t = 2$.

1.4 Численный метод

Чтобы построить численный метод необходимо сделать два действия:

- Построить систему разностных уравнений, приближающих(аппроксимирующих) исходную систему дифференциальных уравнений. Это можно сделать многими способами, мы воспользуемся простейшим.
- Убедиться, что система разностных уравнений устойчива(то есть численные ошибки не влияют катастрофически на решение)

1.4.1 Аппроксимация исходных уравнений

Для построения разностных уравнений заменим каждую производную ее разностным

аналогом. Для вторых производных $\frac{\partial^2}{\partial x^2}$ и $\frac{\partial^2}{\partial y^2}$ воспользуемся стандартными аппроксимациями

$$\frac{\partial^2 u}{\partial x^2} \sim \frac{u^p_{n-1,m} - 2u^p_{n,m} + u^p_{n+1,m}}{h^2} \equiv D_{xx} u^p_{n,m} \tag{8}$$

$$\frac{\partial^2 u}{\partial y^2} \sim \frac{u^p_{n,m-1} - 2u^p_{n,m} + u^p_{n,m+1}}{h^2} \equiv D_{yy} u^p_{n,m} \tag{9}$$

$$\frac{\partial^2 u}{\partial t^2} \sim \frac{u^{p-1}_{n,m} - 2u^p_{n,m} + u^{p+1}_{n,m}}{\tau^2} \equiv D_{tt} u^p_{n,m}. \tag{10}$$

Здесь нижние индексы n, m у u означают номера узлов пространственной сетки, верхний индекс p нумерует шаг по времени, h - шаг сетки, для простоты взятый одинаковым по всем направлениям, а τ — шаг по времени. Операторы D_{xx}, D_{yy} и D_{tt} введены для краткости. Смешанную производную также несложно аппроксимировать

$$\frac{\partial^2 u}{\partial x \partial y} \sim \frac{u^p_{n-1,m-1} - u^p_{n-1,m+1} + u^p_{n+1,m+1} - u^p_{n+1,m-1}}{h^2} \equiv D_{xy} u^p_{n,m} = D_{yx} u^p_{n,m}. \tag{11}$$

Отметим, что эта аппроксимация сохранила свойство смешанной производной $\frac{\partial^2 u}{\partial x \partial y} = \frac{\partial^2 u}{\partial y \partial x}$.

Заменив производные в (6) разностными аппроксимациями, получим

$$\begin{cases} D_{tt}(u_x)^p_{n,m} & = c_p^2 D_{xx}(u_x)^p_{n,m} + c_s^2 D_{yy}(u_x)^p_{n,m} + (c_p^2 - c_s^2) D_{xy}(u_y)^p_{n,m} \\ D_{tt}(u_y)^p_{n,m} & = c_p^2 D_{yy}(u_y)^p_{n,m} + c_s^2 D_{xx}(u_y)^p_{n,m} + (c_p^2 - c_2^2) D_{xy}(u_x)^p_{n,m}. \end{cases} \tag{12}$$

Данная система разностных уравнений имеет второй порядок аппроксимации уравнений (6). Это значит, что в случае устойчивости схемы, разность между точным решением и численным будет стремится к нулю как величина $O(h^2 + \tau^2)$.

1.4.2 Устойчивость системы разностных уравнений

Воспользуемся простым, но не вполне строгим методом спектральной устойчивости. Поскольку мы хотим, чтобы разностная схема не усиливала малые ошибки округления, поведение схемы изучается на ошибках специального вида $f^p_{n,m} = \lambda^p e^{i\alpha m + i\beta n}$. Данный вид хорош тем, что эти функции являются собственными функциями операторов D_{xx}, D_{yy}, D_{tt} и D_{xy}:

$$D_{xx} f^p_{n,m} = -\frac{4}{h^2} \sin^2 \frac{\alpha}{2} f^p_{n,m}$$

$$D_{yy} f^p_{n,m} = -\frac{4}{h^2} \sin^2 \frac{\beta}{2} f^p_{n,m}$$

$$D_{tt} f^p_{n,m} = -\frac{\lambda - 2 + \lambda^{-1}}{\tau^2} f^p_{n,m}$$

$$D_{xy} f^p_{n,m} = -\frac{1}{h^2} \sin\alpha \sin\beta f^p_{n,m}.$$

Пусть $(u_x)^p_{n,m} = af^p_{n,m}, (u_y)^p_{n,m} = bf^p_{n,m}$. Найдем λ из условия, что u_x, u_y удовлетворяют (12). Несколько громоздкие выкладки приводят к

$$\lambda - 2 + \lambda^{-1} = -\frac{\tau^2}{h^2}\left(2(c_p^2 + 2c_s^2)\left(\sin^2\frac{\alpha}{2} + \sin^2\frac{\beta}{2}\right) \pm (c_p^2 - c_s^2)\,|\cos\alpha\cos\beta - 1|\right) \qquad (13)$$

Легко видеть, что если λ — решение этого уравнения, то и λ^{-1} — тоже решение. Если одно из решений меньше единицы по модулю, то второе обязательно больше. Если модуль λ больше 1, то данное решение будет экспоненциально возрастать со временем как λ^p, что и вызывает неустойчивость. Остается только вариант, когда оба λ по модулю равны 1, то есть $\lambda = e^{\pm\gamma}$. Это случай комплексно-сопряженных корней, который соответствует отрицательному дискриминанту квадратного уравнения $\lambda^2 - 2\lambda + 1 = -\mu\lambda$. Это происходит при $0 < \mu < 4$. Выпишем μ

$$\mu = \frac{\tau^2}{h^2}\left(2(c_p^2 + c_s^2)\left(\sin^2\frac{\alpha}{2} + \sin^2\frac{\beta}{2}\right) \pm (c_p^2 - c_s^2)\,|\cos\alpha\cos\beta - 1|\right) \qquad (14)$$

Условие $\mu < 4$ накладывает ограничение на максимальный шаг τ, при котором схема будет устойчива для всех возмущений при любых α и β. Выражение в скобках всегда неотрицательно, а его максимальное значение не больше $4(c_p^2 + c_s^2)$. Получается ограничение

$$\tau < \frac{h}{\sqrt{c_p^2 + c_s^2}}. \qquad (15)$$

При выполнении этого условия малые возмущения не будут расти экспоненциально, а следовательно, схема будет устойчива. В практическом расчете можно положить $\tau = 0.9\frac{h}{\sqrt{c_p^2 + c_s^2}}$.

2 Последовательный программный код

Вам предлагается для ознакомления последовательная программа на языке C, которая решает данную задачу. Программа состоит из файлов

- Makefile — файл для сборки командой make,
- real.h — заголовочный файл, в котором определен тип real. Используется для удобства быстрого изменения между одинарной(float) и двойной(double) точностью вычислений,
- vtk.h — заголовочный файл с функцией для записи результатов вычисления в формате VTK совместимом с программой ParaView
- vtk.c — реализация записи результатов в файл
- main.c — основной код программы

2.1 Команда make и Makefile

Команда make является весьма удобной для сборки программы из нескольких файлов. Ее поведение зависит от специального файла Makefile, в котором содержатся правила сборки. Разберем предлагаемый Makefile

2.1.1 Секция переменных

OBJS = vtk.o main.o CFLAGS = -O2 -Wall LDFLAGS = -lm NVCCFLAGS = -O2 TARGET = seismic

CC = gcc LD = $(CC)

В этом блоке происходит обьявление переменных, используемых в Makefile.

Переменная OBJS содержит имена всех объектных файлов (файлов, полученных компиляцией каждого отдельного файла с исходным кодом), из этих объектных файлов необходимо будет собрать (“слинковать”) один исполняемый файл.

Далее идут переменные, в которых определяются флаги компиляции и линковки. В данном случае для компиляции (переменная CFLAGS) применяется -O2, что означает “оптимизировать скорость выполнения” и -Wall, что означает "выдавать все предупреждения".

В опциях линковки (переменная LDFLAGS) указана опция -lm, которая указывает линковать вместе со стандартной математической библиотекой (без этой опции программа не смогла бы вызывать функции sqrt, exp и подобные).

Переменная NVCCFLAGS здесь не используется, но является аналогом CFLAGS для компилятора nvcc.

Переменная TARGET обозначает имя полученной программы.

Переменная CC традиционно определяет используемый компилятор C, в данном случае, это gcc.

Переменная LD определяет программу для линковки (линкер), в данном случае испольуется gcc, который умеет правильно вызывать системный линкер.

2.1.2 Правила

Работа команды make основана на правилах. Правила имеют вид

целевой файл : файлы-источники [табуляция]команды для получения целевого файла из файлов-источников

Единственное полноценное правило в нашем Makefile

$(TARGET): $(OBJS) $(LD) $(LDFLAGS) -o $@ $^

говорит, что файл с именем TARGET получится из файлов OBJS после выполнения команды $(LD) $(LDFLAGS) -o $@ $^, в которой $@ означает целевой файл, то есть TARGET, а $^ — файлы-источники. Данное правило описывает шаг линковки.

В Makefile нет правил для компиляции, то есть преобразования c-файлов в обьектные файлы. Дело в том, что эти правила стандартны для make и уже содержатся в виде

$(CC) $(CFLAGS) -c -o $@ $<

Это следует читать как: чтобы получить o-файл из c-файла с тем же именем нужно вызвать CC с опцией CFLAGS с флагом компиляции (-c), в качестве выходного (-o) файла использовать o-файл, а входного — c-файл. Именно для этого встроенного правила определяются переменные CC и CFLAGS.

2.1.3 Зависимости

Команда make разработана так, чтобы при модификации исходного кода вызывать по-возможности как можно меньше команд. Для этого make перестраивает только те файлы, у которых файлы-источнки более новые, чем целевой файл. Таким образом, если Вы измените файл main.c, обьектный файл vtk.o перестроен не будет, поскольку main.c не указан у него в файлах-источниках.

Но в файл main.c включен заголовочный файл vtk.h, который *может* влиять на сборку

main.o, но программе make это изначально не известно. Для этого в Makefile указано следующее “урезанное” правило main.o : real.h vtk.h Оно говорит, что main.o следует перестроить, если real.h или vtk.h изменились.

2.1.4 Прочее

Дополнительно, в Makefile включена цель clean и all. Это специальные целевые файлы. Их особенность в том, что правила для них не создают файлы с названием all и clean, то есть эти правила будут выполняться всегда, когда последует запрос на построение clean или all. Такие фиктивные цели помечаются в Makefile с помощью слова PHONY:.

Цель all традиционно присутствует во всех Makefile’ах и является первой целью, которая имеется в файле. Когда Вы вызваете команду make без аргументов вызвается процедура построения самой первой цели, то есть all. В смою очередь, all инициирует процедуру постройки TARGET. Таким образом, чтобы собрать всю программу достаточно выполнить команду make или make all.

Цель clean нужна, чтобы удалить все обьектные файлы и файл TARGET, например, если вы захотите собрать программу “начисто”. Чтобы вызвать эту цель достаточно ввести make clean.

Если Вы не знакомы с Makefile’ами и затрудняетесь написать Makefile для Вашей задачи, напишите об этом на e-mail, указанный в самом верху документа.

2.2 Основная часть программы

Схема, построенная в (12) является явной трехслойной схемой. Это значит, что решение на слое $p+1$ по времени можно вычислить по явной формуле зная решение на слоях p и $p-1$. Для того, чтобы начать вычисления необходимо заполнить слои с $p=0$ и $p=1$. Эти слои можно заполнить, используя начальные данные для функции $\mathbf{u}$ и ее производной $\mathbf{u}_t$. Слой $p=0$ получается просто заполением решением в момент времени $t=0$, Слой $p=1$ заполняется с помощью некоторой операции (она называется схемой предиктор-корректор), которая позволяет получить решение в момент $t=\tau$ с погрешностью порядка $O(\tau^2+h^2)$, то есть вторым порядком, как и сама схема. Далее, схема второго порядка с начальными условиями, точными со вторым порядком, будет вычислять решение с погрешностью также второго порядка.

Для проведения вычислений, достаточно хранить только слои по времени с номерами $p-1$, p и $p+1$. После расчета слоя $p+1$ слой $p-1$ становится не нужен и его можно использовать для хранения слоя $p+2$. То есть после каждого рассчитанного слоя производится сдвиг

- слой $p-1$ более не нужен и называется теперь слоем $(p+1)+1$
- слой p называется теперь слоем $(p+1)-1$
- слой $p+1$ называется теперь слоем $(p+1)$

Если слои назвать относительно текущего значния p как $previous, current$ и $next$, то данная процедура выглядит как

- $temp := next$
- $next := previous$
- $previous := current$

- $current := temp$

При этом данные, хранящиеся на слоях никуда не перемещаются, перемещаются лишь ярлыки (или указатели) $previous, current$ и $next$.

Программа может выводить промежуточные результаты вычислений в файлы. Однако, это сильно тормозит процесс вычислений, поэтому рекомендуется делать вывод каждые 10 или 100 шугов по времени, либо вообще отказаться от вывода промежуточных результатов и смотреть лишь на финальный резульат расчетов.

При запуске программа требует указать два аргумента командной строки — число узлов по каждой координате и количество итераций между выводом промежуточных результатов (или -1, чтобы не выводить промежуточные результаты).

Например, вызов ./seismic 101 -1 запустит расчет на сетке из 101 на 101 узла и выведет лишь первую и последнюю итерации решения.

После работы программа выводит статистику быстродействия. Используется несколько необычная единица измерения быстродействия — “шаблонов в секунду” (или “гигашаблонов в секунду”, как в программе). Эта единица тесно связана с более-менее стандартной единицей Флопс (операций с плавающей точкой в секунду). Операция “применение шаблона” подразумевает вычисление одного элемента на верхнем слое по времени, но не всегда легко правильно подсчитать количество операций с плавающей точкой, которые занимает вычисление одного элемента на верхнем слое. Также, это число может зависеть от настроек компилятора. С величиной “шаблонов в секнуду” таких проблем не возникает. Грубо можно сказать, что для данной задачи 1 шаблон в секунду ≈ 50 Флопс.