МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Национальный исследовательский
Нижегородский государственный университет им. Н. И. Лобачевского

А. В. Маслов

РЕШЕНИЕ ЭЛЕКТРОДИНАМИЧЕСКИХ ЗАДАЧ МЕТОДОМ КОНЕЧНЫХ РАЗНОСТЕЙ ВО ВРЕМЕННО́Й ОБЛАСТИ

Учебно-методическое пособие

Рекомендовано научно-методическим советом исследовательской школы «Лазерная физика» для аспирантов ННГУ, обучающихся по направлению подготовки 03.06.01 «Физика и астрономия», и для магистрантов ННГУ, обучающихся по направлениям подготовки 03.04.03 «Радиофизика», 02.04.02 «Фундаментальная информатика и информационные технологии»

Нижний Новгород
2016

УДК 537.87
ББК 22.336
М-31

М-31 Маслов А.В. РЕШЕНИЕ ЭЛЕКТРОДИНАМИЧЕСКИХ ЗАДАЧ МЕТОДОМ КОНЕЧНЫХ РАЗНОСТЕЙ ВО ВРЕМЕННО́Й ОБЛАСТИ: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2016. – 42 с.

Рецензент: д.т.н., профессор Оболенский С. В.

В пособии изложены основы и приёмы численного подхода к решению электродинамических задач методом конечных разностей во временно́й области (КРВО). Для практического усвоения материала рассмотрены конкретные примеры задач распространения импульсов в неоднородных средах, в том числе имеющих временну́ю дисперсию. Представлены коды программ численного моделирования на языке С. Описаны организация файлов и директорий для оптимизации процесса моделирования, а также процесс компиляции программ в системах Unix/Linux. Пособие способствует приобретению практических навыков решения электродинамических задач методом КРВО, который лежит в основе многих коммерческих пакетов электродинамического моделирования.

Пособие предназначено для студентов, магистрантов и аспирантов радиофизического факультета ННГУ, специализирующихся в области электродинамики, нелинейной оптики, лазерной физики, и также для слушателей спецкурса «Вычислительная электродинамика».

Подготовлено в соответствии с Планом мероприятий по реализации программы повышения конкурентоспособности ННГУ среди ведущих мировых научно-образовательных центров на 2013-2020 годы.

Ответственный за выпуск:
Председатель научно-методического совета
д.ф.-м.н., профессор Бакунов М.И.

УДК 537.87
ББК 22.336

Содержание

1 Введение

Пособие предназначено для приобретения практических навыков решения электродинамических задач методом конечных разностей во временно́й области (КРВО). Обозначение КРВО является переводом на русский термина FDTD method (finite-difference time-domain method), используемого в англо-язычной литературе. Метод КРВО является одним из наиболее распространённых методов моделирования. В пособии изложены основы метода КРВО на примере решения одномерных (1D) задач. Приобретённые навыки помогут при дальнейшем изучении метода и его применении для решения двумерных (2D) и трехмерных (3D) задач.

В пособии делается упор на следующих вопросах:

1. Теоретические основы метода КРВО и особенности вычислительных сеток.

2. Разработка программ с модульной структурой.

3. Организация файлов, директорий и процесса создания выполняемых файлов для оптимизации моделирования.

4. Проработка и усвоение теоретического материала с использованием полного текста программ моделирования.

5. Формирование базовых теоретических знаний и практических навыков для дальнейшего использования метода КРВО при решении 2D и 3D задач.

В пособие включены основные приемы метода КРВО, позволяющие использовать среды с дисперсией и поглощением, моделировать открытые системы с помощью идеально согласованных слоёв на границе вычислительной области, получать частотные характеристики (коэффициенты отражения и прохождения).

Для овладения материалом в пособии предполагается знание электромагнитной теории в рамках курса общей физики, основ математического анализа и наличие навыков программирования. В пособии представлены программы на языке C и даны практические советы по их компиляции и использованию для работы на системах Unix/Linux. Программы были написаны в текстовом редакторе XEmacs, который облегчает редактирование исходных кодов на множестве языков программирования, включая C. Для построения графиков была использована программа Gnuplot. Алгоритмы, разработанные в пособии, могут быть реализованы читателем в конкретные программы и на других языках программирования. А программы могут выполняться и под другими операционными системами. Тем не менее следует отметить, что выбор языка C и систем Unix/Linux обусловлен их широким использованием как в коммерческих компаниях, так и в различных научно-исследовательских лабораториях и университетах во всем мире.

2 Рассматриваемые электродинамические задачи

Поведение электромагнитных волн описывается уравнениями Максвелла. К сожалению, аналитическое решение этих уравнений возможно только в отдельных идеализированных случаях. Поэтому для решения практических задач приходится использовать численное решение этих уравнений. Метод КРВО зарекомендовал себя как достаточно надежный и универсальный метод, который к тому же довольно прост на практике. Широкое распространение метода КРВО обусловленно его применимостью к объектам произвольной формы и к средам с произвольными свойствами, включая временну́ю дисперсию и анизотропию.

Многие коммерческие пакеты электродинамического моделирования используют метод КРВО, например, FullWAVE компании Synopsys (США), CST Microwave studio компании CST (США), OptiFDTD компании Optiwave Systems (Канада), FDTD Solutions компании Lumerical (Канада), OmniSim FDTD компании Photon Design (Великобритания), EM-Suite компании Panoramic Technology (США) и другие.

Метод КРВО применяется для дизайна различных приборов и моделирования процессов, например:

- Приборы (резонаторы) для спектрального уплотнения каналов (WDM).
- Дифракционные решётки.
- Светоизлучающие диоды и лазеры.
- Антенны.
- Процессы литографии, создание и оптимизация масок.
- Взаимодействие света с биологическими объектами (биофотоника).
- Рассеяние света.
- Изучение свойств метаматериалов.

Многие из перечисленных выше приложений метода КРВО можно представить в виде упрощенных типичных задач, проиллюстрированных на рис. 1:

- Излучение заданных источников тока в среде.
- Распространение электромагнитных импульсов в различных средах.
- Рассеяние импульсов на различных объектах.

К решению таких задач мы теперь можем приступить.

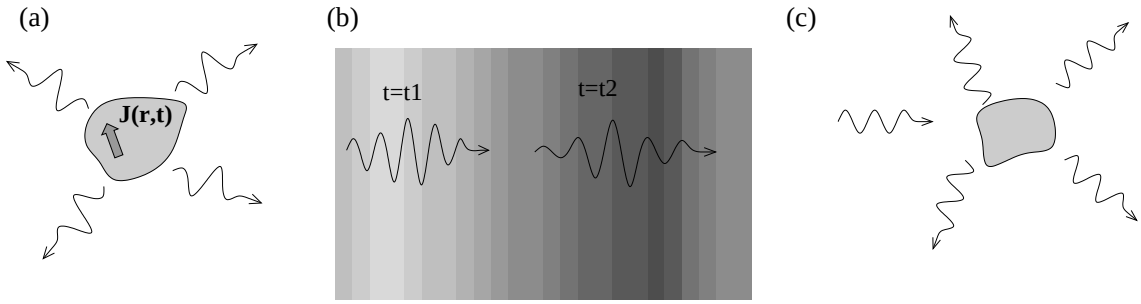

Рис. 1: Иллюстрация различных электродинамических задач. (a) Излучение волн источниками тока с произвольной пространственной и временной зависимостью. (b) Распространение волн в неоднородной среде. (c) Рассеяние волн на некотором объекте.

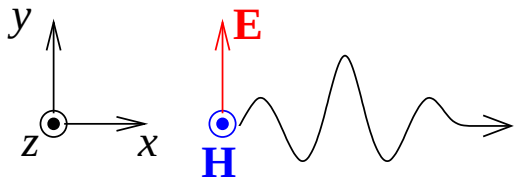

Рис. 2: Распространение импульса вдоль оси x в свободном пространстве в 1D случае.

3 Основные принципы решения

Электромагнитное поле описывается уравнениями Максвелла:

$$\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{B}}{\partial t}, \tag{1a}$$

$$\nabla \times \mathbf{H} = \frac{1}{c}\frac{\partial \mathbf{D}}{\partial t} + \frac{4\pi}{c}\mathbf{J}. \tag{1b}$$

В этих уравнениях:
E – напряжённость электрического поля,
H – напряжённость магнитного поля,
D – электрическая индукция,
B – магнитная индукция,
J – плотность электрического тока.
Заметим, что уравнения (1) записаны в гауссовой системе, в которой поля **E**, **D**, **H**, **B** имеют одинаковые размерности. Такой выбор удобен не только для теоретических выкладок, но и для численного расчёта, так как величины полей имеют сравнимые значения. Часто также записывают сначала уравнения Максвелла в системе СИ, а затем домножают поля на определенные коэффициенты, чтобы привести их к соизмеримым величинам. После численного расчёта полученные поля переводят обратно в единицы СИ. Мы не будем пользоваться системой СИ в данном пособии. Уравнения (1) следует дополнить материальными соотношениями, которые будут обсуждаться позднее.

Рассмотрим распространение электромагнитного импульса в свободном пространстве в 1D случае, см. рис. 2. Мы ограничимся только компонентами E_y,

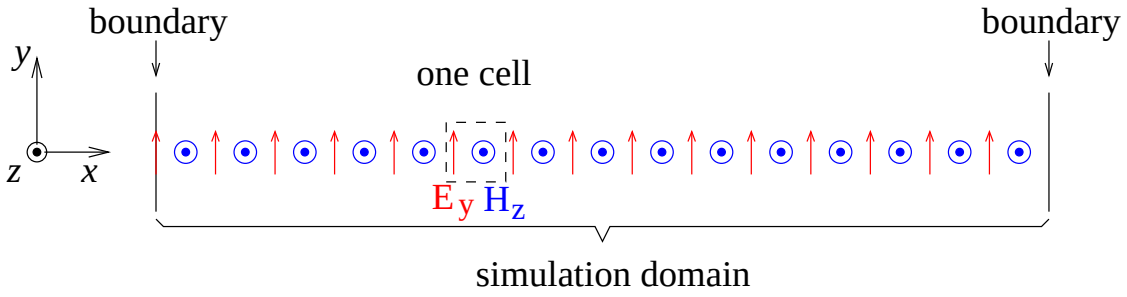

Рис. 3: Разбиение пространства на ячейки и расположение точек (узлов), в которых заданы поля E_y и H_z.

D_y, J_y, H_z, B_z. Тогда уравнения Максвелла сводятся к

$$\frac{\partial B_z}{\partial t} = -c\frac{\partial E_y}{\partial x}, \tag{2a}$$

$$\frac{\partial D_y}{\partial t} = -c\frac{\partial H_z}{\partial x} - 4\pi J_y. \tag{2b}$$

Далее мы опустим индексы x и y, показывающие ориентацию полей, так как каждое поле имеет только одну векторную компоненту. Мы получим

$$\frac{\partial B}{\partial t} = -c\frac{\partial E}{\partial x}, \tag{3a}$$

$$\frac{\partial D}{\partial t} = -c\frac{\partial H}{\partial x} - 4\pi J. \tag{3b}$$

Уравнения (3) – основа построения численного алгоритма. В свободном пространстве $D = E$ и $B = H$.

Для расчёта изменения поля во времени и пространстве производные в (3) представляются через конечные разности. Рассмотрим внимательно одно из уравнений, например, (3a). Временно́е изменение поля B в произвольной точке определяется пространственной производной от поля E, которую можно выразить через значения E в двух соседних точках. С другой стороны, временно́е приращение поля B можно записать, используя его значения сдвинутые по времени. Такие рассуждения приводят к тому, что в пространстве удобно задавать значения полей E и B на одномерных сетках, которые смещены по отношению друг к другу, см. рис. 3. Таким же образом надо выбирать и моменты времени. Поле D задается в тех же точках и в те же моменты времени, что и E, а поле H – где и B.

Обычно считается, что полная вычислительная сетка состоит из ячеек размера Δx, в каждой из которых заданы поля E и H в смещенных точках. Предположим, что Δt - шаг по времени. Тогда мы можем задать поле E в моменты времени $t = n\Delta t$, где $n = 0, 1, 2, \ldots$, а поле H – в моменты $t = (n + 1/2)\Delta t$. Будем считать, что E_i^n – значение поля E в i-ой ячейке в момент времени $t = n\Delta t$ и $H_i^{n+1/2}$ - значение поля H в i-ой ячейке в момент времени $t = (n + 1/2)\Delta t$. Подчернем еще раз, что в такой записи поля E и H имеют одинаковый пространственный индекс и принадлежат одной ячейки i, но в действительности они задаются в точках, отстоящих на $\Delta x/2$ друг от друга.

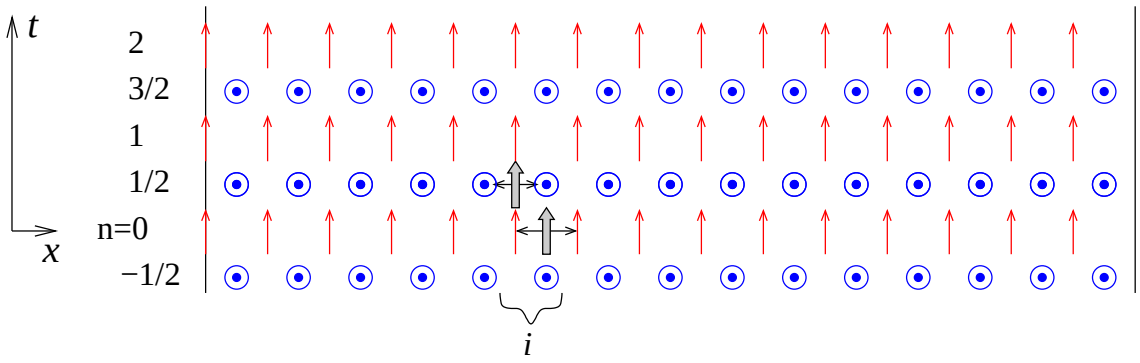

Рис. 4: Схема алгоритма пересчета поля.

Мы можем записать частные производные с помощью конечных разностей с учётом введённых обозначений. Уравнения (3) становятся дискретными и выглядят (без учета тока) как

$$B_i^{n+1/2} - B_i^{n-1/2} = -\frac{c\Delta t}{\Delta x}\left(E_{i+1}^n - E_i^n\right), \tag{4a}$$

$$D_i^{n+1} - D_i^n = -\frac{c\Delta t}{\Delta x}\left(H_i^{n+1/2} - H_{i-1}^{n+1/2}\right). \tag{4b}$$

Далее, уравнения (4) можно переписать в следующем виде:

$$B_i^{n+1/2} = B_i^{n-1/2} - \xi\left(E_{i+1}^n - E_i^n\right), \tag{5a}$$

$$D_i^{n+1} = D_i^n - \xi\left(H_i^{n+1/2} - H_{i-1}^{n+1/2}\right), \tag{5b}$$

где был введён параметр $\xi = c\Delta t/\Delta x$.

С учетом $D = E$ и $B = H$ мы получаем из (5) уравнения для свободного пространства:

$$H_i^{n+1/2} = H_i^{n-1/2} - \xi\left(E_{i+1}^n - E_i^n\right), \tag{6a}$$

$$E_i^{n+1} = E_i^n - \xi\left(H_i^{n+1/2} - H_{i-1}^{n+1/2}\right), \tag{6b}$$

Уравнения (5) являются основными дискретными уравнениями для моделирования распространения электромагнитного импульса в свободном пространстве. Алгоритм продвижения во времени можно представить в виде схемы

$$\cdots \to H^{n+1/2} \to E^{n+1} \to H^{n+3/2} \to E^{n+2} \to \cdots \tag{7}$$

Эта схема проиллюстрирована на рис. 4. Например, для нахождения $H_i^{1/2}$ берётся его предыдущее значение $H_i^{-1/2}$ и значения полей E_i^0, E_{i+1}^0 в соседних точках. Тоже делается и для нахождения E_i^1.

Как видно из (5a), изменение поля B в определённой точке зависит от значений E в точках справа и слева от нее. Тоже справедливо и для полей D и H в (5b). Таким образом, возникает вопрос, как ограничить область моделирования. Для этого требуются некоторые граничные условия. Для простоты мы предположим, что область ограничена стенками, на который зануляется поле

E, то есть имеется идеальный металлический проводник. Такое условие ведёт к отражению импульсов от границ области. Далее будут рассмотрены граничные условия, которые позволяют избежать такое отражение и получить выход импульсов из области моделирования, как это и происходит в неограниченном пространстве.

Для начала моделирования по схеме (7) требуется также задать начальные поля. Предположим, что имеется некоторый импульс, который распространяется вдоль x. Тогда мы знаем, что в свободном пространстве

$$E_y(x,t) = H_z(x,t) = f(t - x/v), \tag{8}$$

где f – произвольная функция и $v = c$ – скорость распространения импульса. Функция (8) позволяет рассчитать поля $E_y(x, t = 0)$, $H_z(x, t = -\Delta t/2)$ и создать распределения E_i^0, $H_i^{-1/2}$, которые обеспечивают возможность продвижения во времени согласно (7). Далее в пособии мы будем использовать

$$f(t) = \cos(\omega_0 t) e^{-t^2/\tau^2} \tag{9}$$

где ω_0 - угловая частота импульса и τ – его продолжительность.

Таким образом, с учётом начальных и граничных условий мы можем теперь применить разностную схему (6) для моделирования распространения импульса.

4 Пример 1: Распространение импульса в свободном пространстве

4.1 Разработка программы моделирования

Рассмотрим конкретный пример моделирования распространения импульса в свободном пространстве. Остановимся сначала подробно на том, как воплотить алгоритмы, разработанные в разделе 3, в программы на языке C. В общем случае программы на C состоят из отдельных функций, которые взаимодействуют друг с другом. Программы численного моделирования могут использовать десятки и сотни различных функций. Поэтому при написании программ особое внимание требуется уделять выделению операций в функции и разбивке функций по файлам. Оптимальная разбивка позволяет собирать различные программы из уже существующих функций и файлов, как будет видно по мере того, как мы будем совершенствовать алгоритмы моделирования в последующих разделах.

Алгоритм моделирования (7) подразумевает хранение значений E и H в виде массивов, содержимое которых изменяется последовательно во времени. При этом заранее неизвестно, сколько требуется осуществить шагов для того, чтобы импульс прошел определенное расстояние. Поэтому удобно предоставить пользователю возможность задания начального количества шагов, их добавления или остановки выполнения программы. Такое решение будет приниматься в зависимости от прохождения импульсом какого-либо расстояния. Удобно расположить функцию управления в одном файле, а функции, отвечающие за ал-

горитм – в другом. Функция управления будет отвечать за выделение памяти, вывод данных и взаимодействие с пользователем.

Разработанная программа состоит из следующих файлов:

free_space.c: функция управления.

fdtd_1d_maxwell.c и **fdtd_1d_maxwell.h:** функции алгоритма и их описание.

pulse.c и **pulse.h:** функции создания импульса и их описание.

output.c и **output.h:** функции вывода в файлы и их описание.

constants.c: задание постоянных величин.

Давайте теперь изучим содержимое файлов.

Файл `free_space.c` с функцией управления `free_space()`:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

#include"fdtd_1d_maxwell.h"
#include"pulse.h"
#include"output.h"

extern const double cspeed;

/*****************************************************************************/
void free_space(void){

  char *tag="v1";     // used to label output files

  /** Optical pulse ***/
  double tau = 1.0;    // fs, width of the pulse
  double w0=0;         // no oscillating component

  /*** Computational parameters ***/
  int     Nx = 4000;   // number of cells along x
  double  dx = 20.0;    // nm
  double  xi = 0.9;
  int    ix0 = 1000;    // cell number of the center of the pulse at t=0
  int     No = 1000;    // defines the output rate

  /*** start execution ***/
  double  dt = xi*dx/cspeed;  // in fs
  printf("dx=%.12e nm, dt=%.12e fs\n", dx, dt);

  /*** arrays for the fields ***/
  double  *fields = malloc(2*Nx*sizeof(double));
  double   *Hz = fields+0*Nx;
  double   *Ey = fields+1*Nx;

  int  T=0; // total steps
```

```
38
39    create_initial_dist(Nx, Ey, Hz, dx, dt, cspeed, ix0, tau, w0);
40    output_Ey_vs_x(Nx, Ey, 0,  dx, tag);
41
42    for(;;){
43      printf("Number of steps to run (<=0 to exit) -> ");  fflush(stdout);
44      int steps;
45      scanf("%d", &steps);
46      if(steps<=0) break;
47
48      printf("Making %d steps\n", steps);
49      for(int n=0; n<steps; n++, T++){
50        update_Bz(Nx, Hz, Ey, xi);    // find Bz at n+1/2
51        update_Dy(Nx, Ey, Hz, xi);    // find Ey at n+1
52
53        /* output of Hz */
54        if((T+1)%No == 0){
55          printf("Elapsed time -> %g fs (%d steps)\n", dt*(T+1), T+1);
56          output_Ey_vs_x(Nx, Ey, T+1,  dx, tag);
57        }
58
59      }// end of n loop
60    }// end of global loop
61
62
63    free(fields);
64  }
65
66  /*****************************************************************************/
67  int main(void){
68
69    free_space();
70
71    return 0;
72  }
73
74  /*** END OF FILE ************************************************************/
```

В функции `free_space()` задаются основные параметры: `Nx` – количество ячеек вдоль оси x, `dx` - пространственный шаг, `xi` – параметр ξ, `ix0` – номер ячейки, где расположен пик импульса в начальный момент времени, `tau` и `w0` – параметры импульса. Переменная `T` содержит полное количество шагов. Программа спрашивает пользователя, сколько шагов требуется осуществить и дает возможность остановки после их выполнения. После каждых `No` шагов программа записывает состояние поля E в файл. Имя файла содержит значение `T`, что позволяет хранить информацию о динамике в различные моменты времени. Имя файла также имеет дополнительную метку, заданную переменной `tag`, что позволяет отличать файлы при запуске программы с изменёнными параметрами. Вверху файла включаются файлы с описанием функций, которые расположены в других файлах.

Файл `fdtd_1d_maxwell.c` с основными функциями алгоритма (5):

```
1  /*****************************************************************************/
```

```
void update_Bz(int Nx, double *Bz, const double *Ey, double xi){

  for(int i=0; i<Nx-1; i++){// except the last point
    Bz[i] +=-xi*(Ey[i+1]-Ey[i]);
  }
  int i=Nx-1; // last point
  Bz[i] +=-xi*(0.0 - Ey[i]);
}

/*****************************************************************************/
void update_Dy(int Nx, double *Dy, const double *Hz, double xi){

  // Ey[0]=0 always and not updated
  for(int i=1; i<Nx; i++){
    Dy[i] +=-xi*(Hz[i]-Hz[i-1]);
  }
}
/*** END OF FILE ************************************************************/
```

Функции `update_Bz` и `update_Dy` измененяют поля во времени. Заметим, что мы предположили, что происходит изменение полей B и D, а не E и H, что удобнее для дальнейшего использования этих функций.

Файл `fdtd_1d_fspace_basic.h` с описанием функций из `fdtd_1d_maxwell.c`:

```
void update_Bz(int Nx, double *Bz, const double *Ey, double xi);
void update_Dy(int Nx, double *Dy, const double *Hz, double xi);
```

Файл `pulse.c` с функциями создания импульса:

```
#include<stdio.h>
#include<math.h>

/*****************************************************************************/
double pulse(double x, double t, double speed, double tau, double w0){
  double t1 = t - x/speed;
  double  a = t1/tau;
  return exp(-a*a)*cos(w0*t1);
}

/*****************************************************************************/
void create_initial_dist(int Nx, double *Ey, double *Hz, double dx, double dt,
                         double speed, int ix0, double tau, double w0){
  /*
   * Creates Ey[T=0] and Hz[T=-0.5]
   */

  printf("Pulse parameters: w0=%g rad/fs, tau=%g fs\n", w0, tau);

  for(int i=0; i<Nx; i++){
    Ey[i] = pulse(dx*(i+0.0-ix0),    0*dt, speed, tau, w0);
    Hz[i] = pulse(dx*(i+0.5-ix0), -0.5*dt, speed, tau, w0);
  }
  Ey[0]=0;  // to satisfy b.c.
```

```
}

/*** END OF FILE ************************************************************/
```

Функция `create_initial_dist()` задает начальное распределение полей.

Файл `pulse.h` с описанием функций из `pulse.c`:

```
double pulse(double x, double t, double speed, double tau, double w0);
void create_initial_dist(int Nx, double *Ey, double *Hz, double dx, double dt,
                         double speed, int ix0, double tau, double w0);
```

Файл `output.c` с функциями вывода в файлы:

```
#include<stdio.h>
#include<string.h>

/*****************************************************************************/
void output_Ey_vs_x(int Nx, const double *Ey, int T, double dx, const char *tag){

  char fname[100];
  sprintf(fname, "Ey_vs_x_T=%d_%s.dat", T, tag);
  FILE *fp=fopen(fname, "w");
  fprintf(fp, "# 1: x (micron) | 2 : Ey (a.u.) \n");
  for(int i=0; i<Nx; i++){
    double x=i*dx;
    fprintf(fp, "% .12e  % .12e\n", 1e-3*x, Ey[i]);
  }
  fclose(fp);
  printf("Completed writing to file \"%s\"\n", fname);
}
/*** END OF FILE ************************************************************/
```

Файл `output.h` с описанием функций из `output.c`:

```
void output_Ey_vs_x(int Nx, const double *Ey, int T, double dx, const char *tag);
```

Файл `constants.c` задает скорость света c и значение π:

```
const double cspeed=299.792458;      // nm/fs
const double  pi=3.141592653589793;
```

После разбивки функций по файлам, обсудим теперь структуру директорий (каталогов), которая удобна для хранения исходных файлов, компиляции программы и выполнения расчётов. Можно использовать следующую иерархическую структуру директорий:

```
free_space/
|-- bin
```

```
|   `-- fdtd_1d.exe
|-- data [137 entries exceeds filelimit, not opening dir]
|-- makefile
|-- obj
|   |-- constants.o
|   |-- fdtd_1d_maxwell.o
|   |-- free_space.o
|   |-- output.o
|   `-- pulse.o
`-- src
    |-- constants.c
    |-- fdtd_1d_maxwell.c
    |-- fdtd_1d_maxwell.h
    |-- free_space.c
    |-- output.c
    |-- output.h
    |-- pulse.c
    `-- pulse.h

4 directories, 15 files
```

В директории `src/` хранятся исходные файлы, в `obj/` – объектные файлы, в `bin/` – выполняемые файлы, а в `data/` – файлы с результатами расчётов.

Для создания выполняемого файла рекомендуется использовать стандартную команду `make`, которая выполняет инструкции из файла `makefile`. Преимущество использования команды `make` заключается в том, что не надо запоминать все опции компиляции файлов, а только один раз их записать в `makefile`. Создание выполняемого файла состоит из двуй шагов. Сначала компиляция файлов с исходным кодом и получение из них объектных файлов. Затем соединение объектных файлов и получение из них выполняемого файла.

Содержание файла `makefile`:

```
CC=gcc
GCCFLAGS= -O3 -Wall -std=c99

all : free_space

OBJFS = obj/free_space.o  obj/fdtd_1d_maxwell.o  obj/pulse.o  \
        obj/output.o      obj/constants.o
free_space:  $(OBJFS)
        $(CC) $(GCCFLAGS) -o bin/free_space.exe  $(OBJFS)  -lm
        @echo -e 'Created executable "bin/free_space.exe"\n'

### creation of object files
obj/%.o : src/%.c
        $(CC) -c $(GCCFLAGS) $< -o $@

clean:
        rm -f obj/*.o bin/*.exe
```

В файле используется следующая структура:

```
цель: зависимости
[tab] команда
```

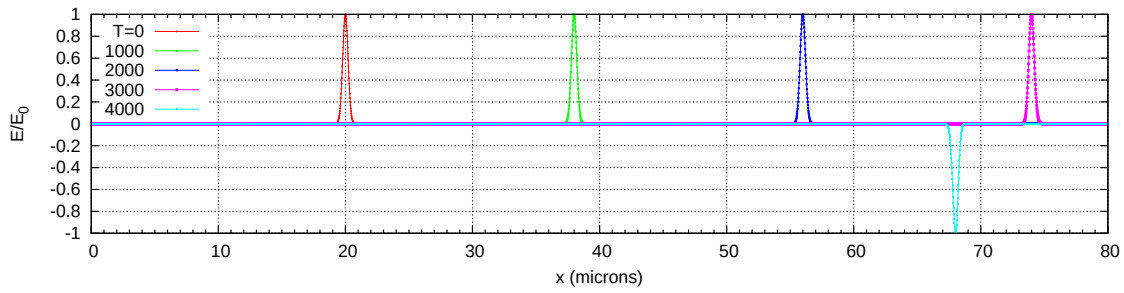

Рис. 5: Распространение импульса в свободном пространстве в одномерном случае при $\tau = 1$ фс, $\Delta x = 20$ нм, $N_x = 4000$, $\xi = 0.9$.

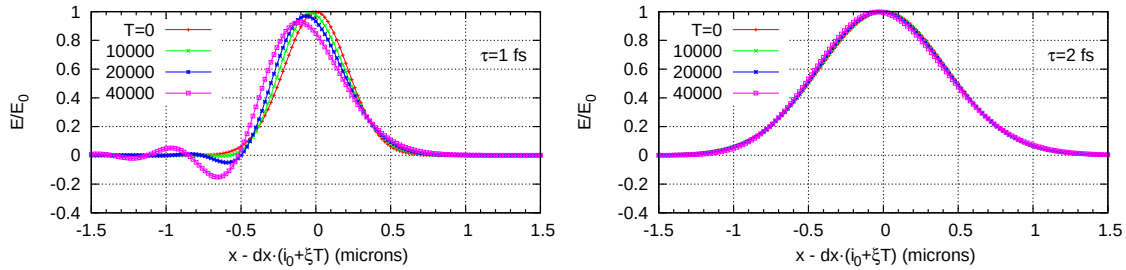

Рис. 6: Изменение формы импульса в процессе его распространения при $\Delta x = 20$ нм, $\xi = 0.9$ и $\tau = 1,\ 2$ фс.

Обратите внимание, что строка с командой должна всегда начинаться с табуляции.

Выполняемый файл создается в директории `bin/` после выполнения `make` из директории `free_space/`. Выполнять программу удобно из директории `data/`, куда и будут записываться все файлы с данными. После запуска программы используя `../bin/fdtd_1d.exe` появляется строка, запрашивающая требуемое количество шагов. Затем программа выполнит введённое количество шагов и спросит снова. Закончить выполнение можно введя число ≤ 0. При выполнении программа осуществляет вывод данных в файлы, что позволяет сразу же анализировать результаты.

4.2 Результаты моделирования и их обсуждение

Возьмём сначала область моделирования с `Nx=4000` ячейками и запустим гауссов импульс. Типичные распределения полей в различные моменты времени показаны на рис. 5. Как и ожидалось, созданный в начальный момент импульс распространяется вправо. Достигнув границы, он отражается с изменением знака и начинает бежать влево.

Давайте проследим за изменением формы импульса более внимательно. Для этого мы увеличим область расчёта, чтобы дать импульсу больше места для распространения. Так как в моделируемой среде нет дисперсии, то импульс не должен меняться в процессе распространения. Для проверки этого результата удобно строить зависимость поля от величины $x - (i0 + \xi T)\Delta x$, то есть в системе координат, совпадающей с пиком импульса. Такие результаты представлены

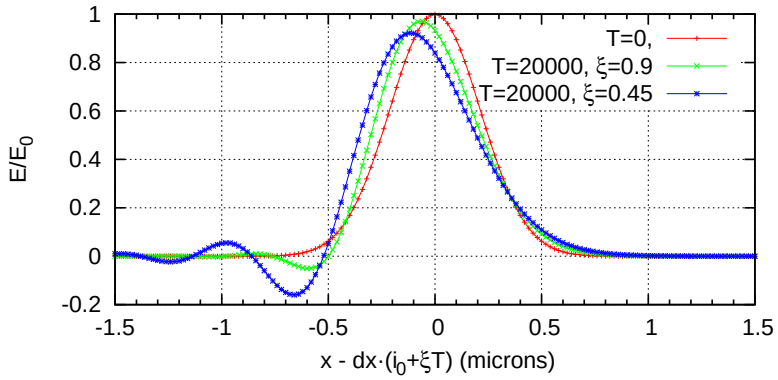

Рис. 7: Изменение формы импульса в процессе его распространения при $\Delta x = 20$ нм, $\xi = 0.9,\ 0.45$ и $\tau = 1$ фс.

на рис. 6. Видно, что при большом количестве шагов импульс деформируется. Причём, деформация становится более заметной с уменьшением ширины импульса τ.

Давайте теперь попробуем изменить временно́й шаг и посмотреть, как это скажется на результате, см. рис. 7. Мы видим, что при меньшем значении параметра ξ, а значит и меньшем временно́м шаге Δt, импульс деформируется больше при прохождении одинакового расстояния. При меньшом ξ при этом требуется, естественно, выполнить и больше шагов.

Таким образом, мы разработали простейшую программу для моделирования распространения электромагнитного импульса и продемонстрировали ее функциональность. При достаточно малых значениях времени мы получили хорошее сохранение формы импульса при его распространении, что соответствует точному решению уравнений Максвелла. При большом количестве шагов импульс начинает искажаться. Степень его искажения при прохождении определённого расстояния возрастает как с уменьшением его ширины τ, так и временного шага Δt. Причина такого поведения будет рассмотрена в следующем разделе.

5 Выбор пространственного и временного шага

Для выяснения причин искажения импульса рассмотрим систему дискретных уравнений (6). Она описывает распространение сигнала между заданными точками. Любой сигнал можно представить в виде собственных функций такой системы. Найдём свойства этих функции. Для этого подставим общую форму решения в комплексном виде

$$E(x,t) = E_0 e^{-i\omega t + ikx}, \tag{10a}$$
$$H(x,t) = H_0 e^{-i\omega t + ikx}, \tag{10b}$$

в систему (6). Мы получим

$$H_0 e^{ik\Delta x/2}\left(e^{-i\omega\Delta t/2} - e^{i\omega\Delta t/2}\right) = -\xi E_0\left(e^{ik\Delta x} - 1\right), \tag{11a}$$
$$E_0\left(e^{-i\omega\Delta t} - 1\right) = -\xi H_0 e^{-i\omega\Delta t/2}\left(e^{ik\Delta x/2} - e^{-ik\Delta x/2}\right), \tag{11b}$$

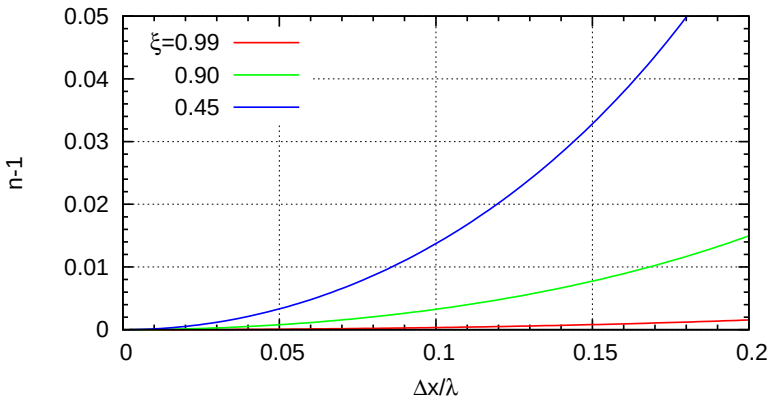

Рис. 8: Зависимость эффективного индекса n от величины пространственного шага Δx при различных значениях параметра ξ.

или

$$\begin{pmatrix} \xi \sin\frac{k\Delta x}{2} & -\sin\frac{\omega\Delta t}{2} \\ \sin\frac{\omega\Delta t}{2} & -\xi\sin\frac{k\Delta x}{2} \end{pmatrix} \begin{pmatrix} E_0 \\ H_0 \end{pmatrix} = 0. \tag{12}$$

Система (12) имеет решение, если детерминант равен нулю. Это условие приводит к

$$\xi^2 \sin^2\frac{k\Delta x}{2} = \sin^2\frac{\omega\Delta t}{2}. \tag{13}$$

Понятно, что два знака при решении уравнения (13) соответствуют волнам, распространяющимся в противоположные стороны. Для анализа изкажения достаточно взять одну волну, бегущую вдоль $+x$ и удовлетворяющую

$$\xi \sin\frac{k\Delta x}{2} = \sin\frac{\omega\Delta t}{2}. \tag{14}$$

Уравнение (13) – дисперсионное уравнение дла волн в дискретной системе (6).

Если мы создадим волну с некоторым волновым числом k, то ее изменение во времени определяется частотой, получаемой из (14). Видно, что при $\xi > 1$, решение может быть только при комплексных частотах ω. Волна может нарастать со временем и, в конце концов, достигать бесконечно больших значений. Такое поведение не соответствует модели, описываемой непрерывными уравнениями (3). Таким образом, выбор $\xi > 1$ приводит к расхождению алгоритма.

Теперь предположим, что $\xi \leq 1$. Для анализа распространения удобно ввести эффективный индекс n, определяющий фазовую скорость волны: $k = n\omega/c$. Обозначая $\delta = \Delta x\omega/c$, мы получаем

$$n = \frac{2}{\delta}\arcsin\left(\frac{1}{\xi}\sin\left(\delta\frac{\xi}{2}\right)\right). \tag{15}$$

Зависимость $n-1$ от величины пространственного шага при различных значения параметра ξ показана на рис. 8. В непрерывной системы (3) все волны распространяются с $n = 1$. В дискретной системе, описываемой уравнениями

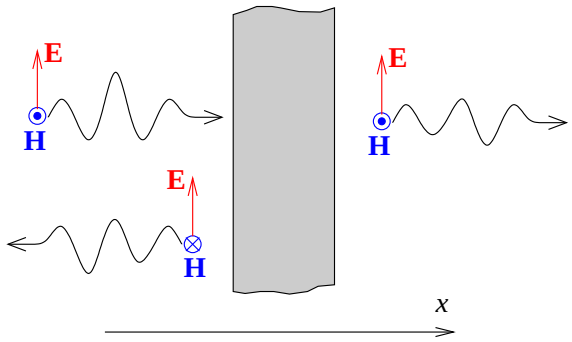

Рис. 9: Геометрия задачи о падении импульса на диэлектрический слой.

(6), фазовые скорости волн будут зависеть от пространственной и временной дискретизации. Как видно из рис. 8, для приближения к правильному результату необходимо выбирать $\Delta x/\lambda \ll 1$. То есть, шаг дискретизации должен быть много меньше, чем минимальная длина волны в разложении импульса по пространственным гармоникам. При этом рис. 8 показывает, что дисперсия уменьшается при $\xi \to 1$, то есть уменьшение временного шага ведёт к увеличению численной дисперсии. Это согласуется и с результатами расчёта, показанными на рис. 7.

Таким образом, при выборе шага необходимо пользоваться параметрами моделируемого импульса. Его ширина позволяет получить максимальнуя длину волны в его спектре. Пространственный шаг должен быть выбран значительно меньше, чем эта длина. На практике выбирают $\Delta x/\lambda \lesssim 20$. Временной шаг надо выбирать из соображения $\xi \leq 1$. При этом надо иметь в виду проявление эффектов численной дисперсии при большом количестве шагов.

6 Моделирование сред без дисперсии

Изучив распространение импульса в свободном пространстве в разделах 3 и 4, мы можем теперь приступить к изучению его прохождения через различные неоднородности. Для этого надо ввести распределение диэлектрической проницаемости (или индекса преломления) среды, то есть дополнить уравнения Максвелла материальными соотношениями. Конкретная задача, которую мы будем рассматривать, показана на рис. 9, где импульс падает на слой диэлектрика с заданной проницаемостью. Помимо моделирования распространения импульса, попробуем найти зависимости коэффициентов прохождения и отражения от частоты.

Естественно, при наличии неоднородности уравенения Максвелла и их дискретная форма остаются прежними. Нам лишь надо дополнить их соотношением для нахождения E через D. Предполая, что среда описывается материальным уравнением

$$D(x,t) = \varepsilon(x)E(x,t), \tag{16}$$

дискретная форма принимает вид

$$E_i^n = \frac{D_i^n}{\varepsilon_i}, \tag{17}$$

где ε_i - значение проницаемости в точке задания поля. Таким образом, наличие диэлектрической проницаемости приводит к незначительному усложнению алгоритма пересчета полей:

$$\cdots \to H^{n+1/2} \to D^{n+1} \to E^{n+1} \to H^{n+3/2} \to D^{n+2} \to E^{n+2} \to \cdots \qquad (18)$$

Следует отметить, что мы могли бы полностью исключить поле D из уравнений Максвелла, используя (16), и работать только с полями E и H. В этом случае проницаемость вошла бы в уравнения Максвелла. Такой подход часто применяется, так как даёт возможность исключить материальное уравнение для сред без дисперсии из численного алгоритма. С другой стороны, алгоритм (18) является более общим и применим даже для сред с дисперсией, если заменить уравнение (17) на соответствующее материальное уравнение. Для сред без дисперсии использование уравнения (17) в алгоритме (18) практически не приводит к существенному увеличению времени расчёта, а поэтому имеет смысл пользоваться сразу более общим подходом.

7 Получение частотных характеристик

На практике часто требуется получить свойства среды при падении монохроматического сигнала, то есть частотные характеристики. К ним относятся, например, коэффициенты отражения и прохождения для задачи, показанной на рис. 9. Так как расчёт ведётся во временно́й области, то для этого можно применить преобразование Фурье. Преобразование Фурье задаёт соотношение между некоторой функцией во времени $f(t)$ и ее спектром $\tilde{f}(\omega)$. Мы будем пользоваться следующими соотношениями для непрерывных функций:

$$f(t) = \int_{-\infty}^{+\infty} \mathrm{d}\omega \ \tilde{f}(\omega) e^{-i\omega t}, \quad \tilde{f}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \mathrm{d}t \ f(t) e^{i\omega t}. \qquad (19)$$

Для получения спектра сигнала, заданного через равные промежутки времени, мы должны выбрать набор требуемых частот ω_j и просто просуммировать вклады от сигнала с экспоненциальным коэффициентами:

$$\tilde{f}(\omega_j) = \frac{\Delta t}{2\pi} \sum_{n=0}^{T} f(t_n) e^{-i\omega_j t_n}. \qquad (20)$$

Расчёт по формуле (20) осуществляется после выбора точки в пространстве, где требуется найти спектр электрического или магнитного поля. Для правильности расчёта временно́й интервал, ограниченный значениями $n = 0$ и $n = T$, в сумме (20) должен быть таким, чтобы импульс полностью в него помещался, то есть значение импульса должно быть пренебрежимо мало как в начальный, так и в конечный моменты времени. При этом Фурье спектр достигает стационарного значения и не меняется с увеличением количества шагов T.

Следует отметить, что для расчёта спектра нет необходимости сначала сохранять все значения функции $f(t_n)$, а потом применять (20). Достаточно просто добавлять их вклад во все компоненты $\tilde{f}(\omega_j)$ в каждый момент времени.

При сохранении значений во времени возможно также использование алгоритма быстрого преобразования Фурье. При этом набор частот будет определяться величиной временного шага и числом шагов. Надо также учитывать, что для нахождения, например, коэффициента отражения требуется знать не только спектр отражённого импульса, но и спектр падающего.

8 Пример 2: Падение импульса на диэлектрический слой

8.1 Разработка программы моделирования

Теперь применим методики, разработанные в разделах 6 и 7, для численного нахождения коэффициентов отражения и прохождения при падении волны на однородный слой с проницаемостью ε, находящийся в свободном пространстве. Для этого надо изменить файл управления, а также написать новые функции для материального уравнения и преобразования Фурье.

Файл `slab.c` с функцией управления:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<complex.h>

typedef double complex dcomplex;

#include"fdtd_1d_maxwell.h"
#include"pulse.h"
#include"output.h"
#include"mater.h"
#include"set_mem.h"
#include"rfourier.h"

extern const double cspeed, pi;

/*****************************************************************************/
void slab(void){

  char *tag="v1";     // used to label output files
  double eslab = 4.0;  // permittivity of the slab

  /** Optical pulse ***/
  double lambda0 = 1000; // nm
  double     tau = 8;    // fs, width of the pulse

  /*** Computational parameters ***/
  double  dx = 20.0;    // nm
  int     Nx = 20000;   // number of cells along x
  int    ix0 = 9000;    // center of the pulse at t=0

  int  Nslab = 200;           // width of the slab
```

```
34  int    si1 = 10000;         // start of the slab
35  int    si2 = si1+Nslab-1;   // end of the slab
36
37  int    fi1 = 7500;    // location of fourier transform
38  int    fi2 = si2+10;   // location of fourier transform
39
40  double  xi = 0.9;
41  int     No = 200;     // defines the output rate
42
43  /*** start execution ***/
44  double w0 = 2*pi*cspeed/lambda0; // rad/fs
45  double  dt = xi*dx/cspeed;  // in fs
46  printf("dx=%.12e nm, dt=%.12e fs\n", dx, dt);
47
48  /*** arrays for the fields ***/
49  double  *fields = malloc(3*Nx*sizeof(double));
50  double   *Hz = fields+0*Nx;
51  double   *Ey = fields+1*Nx;
52  double   *Dy = fields+2*Nx;
53
54  double  *eps = malloc(Nx*sizeof(double));
55
56  create_slab(Nx, eps, si1, si2, eslab);
57  output_eps_x(Nx, eps, dx, tag);
58
59  create_initial_dist(Nx, Dy, Hz, dx, dt, cspeed, ix0, tau, w0);
60
61  update_Ey(Nx, Ey, Dy, eps);
62  output_Ey_vs_x(Nx, Ey, 0,  dx, tag);
63
64  double wmin = 0.8*w0; // rad/fs
65  double wmax = 1.2*w0; // rad/fs
66  int Nw=200;
67
68  dcomplex *ftall=malloc(2*Nw*sizeof(dcomplex));
69  dcomplex *ft1 = ftall + 0*Nw;
70  dcomplex *ft2 = ftall + 1*Nw;
71  zset_mem(2*Nw, ftall, 0.0+I*0.0); // both parts of complex
72
73  int  T=0; // total steps
74  for(;;){
75    printf("Number of steps to run (<=0 to exit) -> ");  fflush(stdout);
76    int steps;
77    scanf("%d", &steps);
78    if(steps<=0) break;
79
80    printf("Making %d steps\n", steps);
81    for(int n=0; n<steps; n++, T++){
82      update_Bz(Nx, Hz, Ey, xi);    // find Bz at n+1/2
83      update_Dy(Nx, Dy, Hz, xi);    // find Dy at n+1
84      update_Ey(Nx, Ey, Dy, eps);   // find Ey at n+1
85
86      /* output of Ey */
87      if((T+1)%No == 0){
88        printf("Elapsed time -> %g fs (%d steps)\n", dt*(T+1), T+1);
89        output_Ey_vs_x(Nx, Ey, T+1,  dx, tag);
```

```
      }

      /*** take running fourier ***/
      double time=dt*(T+1); // for Ey
      rfourier2(wmin, wmax, Nw, ft1, ft2, Ey[fi1], Ey[fi2], dt, time);

    }// end of n loop

    char fname1[100], fname2[100];
    sprintf(fname1, "ft_fi=%d_%s.dat", fi1, tag);
    sprintf(fname2, "ft_fi=%d_%s.dat", fi2, tag);
    four_out(Nw, ft1, wmin, wmax,  fname1);
    four_out(Nw, ft2, wmin, wmax,  fname2);

  }// end of global loop

  free(fields); free(eps); free(ftall);
}

/*****************************************************************************/
int main(void){

  slab();

  return 0;
}

/*** END OF FILE ************************************************************/
```

В новой функции управления `slab()` мы задаём дополнительно границы слоя, его проницаемость, выбираем интервал частот и точки, где берётся преобразование Фурье. Также мы осуществляем вызов функций, отвечающих за материальные свойства в моделируемой области и за преобразование Фурье.

В файле `mater.c` мы поместим функции, задающие распределение проницаемости во всей области и рассчитывающие поле E. Файл `mater.c`:

```
#include<stdio.h>

/*****************************************************************************/
void output_eps_x(int Nx, const double *eps, double dx, const char *tag){

  char fname[100];
  sprintf(fname, "eps_vs_x_%s.dat", tag);
  FILE *fp=fopen(fname, "w");
  fprintf(fp, "# 1: x (micron) | 2 : eps \n");
  for(int i=0; i<Nx; i++){
    double x=i*dx;
    fprintf(fp, "% .12e  % .12e\n", 1e-3*x, eps[i]);
  }
  fclose(fp);
  printf("Completed writing to file \"%s\"\n", fname);
}

/*****************************************************************************/
```

```
19 void create_slab(int Nx, double *eps, int j1, int j2, double eslab){
20   /*
21    * Creates array of dielectric permittivity
22    */
23
24   for(int i=0; i<Nx; i++){
25     eps[i] = 1;
26   }
27   for(int i=j1; i<j2+1; i++){
28     eps[i] = eslab;
29   }
30
31 }
32
33 /*****************************************************************************/
34 void update_Ey(int Nx, double *Ey, const double *Dy, const double *eps){
35   /*
36    * Calculates E from D and epsilon
37    */
38   for(int i=0; i<Nx; i++){
39     Ey[i] = Dy[i]/eps[i];
40   }
41 }
42
43 /*** END OF FILE ************************************************************/
```

Файл `mater.h` с описанием функций из `mater.c`:

```
1 void update_Ey(int Nx, double *Ey, const double *Dy, const double *ebkg);
2 void create_slab(int Nx, double *eps, int j1, int j2, double eslab);
3 void output_eps_x(int Nx, const double *eps, double dx, const char *tag);
```

Далее мы создаём функции для преобразования Фурье и вывода спектра в файл. Файл `rfourier.c`:

```
1  #include<stdio.h>
2  #include<math.h>
3  #include<complex.h>
4
5  typedef double complex dcomplex;
6  extern const double cspeed, pi;
7
8  /*****************************************************************************/
9  void four_out(int Nw, const dcomplex *ft, double wmin, double wmax,
10                const char *fname){
11
12   FILE *fp=fopen(fname, "w");
13
14   printf("output to \"%s\"\n", fname);
15   double dw=(wmax-wmin)/(Nw-1.0);
16   for(int i=0; i<Nw; i++){
17     double w=wmin+dw*i;
18     fprintf(fp, "% .12e  % .12e % .12e\n", w, creal(ft[i]), cimag(ft[i]));
19   }
```

```
  fclose(fp);
}

/*****************************************************************************/
void rfourier2(double wmin, double wmax, int Nw, dcomplex *ft1, dcomplex *ft2,
               double Ey1, double Ey2, double dt, double time){

  double   dw = (wmax-wmin)/(Nw-1.0);
  double mult=dt/(2*pi);
  for(int n=0; n<Nw; n++){
    double w=wmin+dw*n;
    dcomplex ce=mult*cexp(I*w*time);
        ft1[n] += ce*Ey1;
        ft2[n] += ce*Ey2;
  }
}
/*** END OF FILE ************************************************************/
```

Файл `rfourier.h` с описанием функций из `rfourier.c`:

```
void four_out(int Nw, const dcomplex *ft, double wmin, double wmax,
              const char *fname);
void rfourier2(double wmin, double wmax, int Nw, dcomplex *ft1, dcomplex *ft2,
               double Ey1, double Ey2, double dt, double time);
```

Нам также понадобятся вспомогательные функции, которые осуществляют инициализацию массивов. Файл `set_mem.c`:

```
#include<complex.h>

typedef double complex dcomplex;

/*****************************************************************************/
void zset_mem(int N, dcomplex *A, dcomplex var){
  for(int i=0; i<N; i++){
    A[i]=var;
  }
}
/*****************************************************************************/
void dset_mem(int N, double *A, double var){
  for(int i=0; i<N; i++){
    A[i]=var;
  }
}
/*** END OF FILE ************************************************************/
```

Файл `set_mem.h` с описанием функций из `set_mem.c`:

```
void zset_mem(int N, dcomplex *A, dcomplex var);
void dset_mem(int N, double *A, double var);
```

После разработки всех новых функции мы дополним файл `makefile` возможностью создания выполняемого файла `slab.exe`. При этом компиляция будет также использовать файлы, созданные для моделирования импульса в свободном пространстве в разделе 4.

Файл `makefile`:

```
CC=gcc
GCCFLAGS= -O3 -Wall -std=c99

all : free_space  slab

OBJSL = obj/slab.o    obj/fdtd_1d_maxwell.o  obj/mater.o \
        obj/pulse.o   obj/output.o           obj/constants.o\
        obj/set_mem.o obj/rfourier.o
slab:  $(OBJSL)
        $(CC) $(GCCFLAGS) -o bin/slab.exe  $(OBJSL)  -lm
        @echo -e 'Created executable "bin/slab.exe"\n'

OBJFS = obj/free_space.o  obj/fdtd_1d_maxwell.o  obj/pulse.o  \
        obj/output.o      obj/constants.o
free_space:  $(OBJFS)
        $(CC) $(GCCFLAGS) -o bin/free_space.exe  $(OBJFS)  -lm
        @echo -e 'Created executable "bin/free_space.exe"\n'

### creation of object files
obj/%.o : src/%.c
        $(CC) -c $(GCCFLAGS) $< -o $@

clean:
        rm -f obj/*.o bin/*.exe
```

Приведённый выше `makefile` позволяет создавать как выполняемый файл для моделирования распространения импульса в свободном пространстве (при выполнении команды `make free_space`), так и выполняемый файл для моделирования прохождения через слой (`make slab`). Можно также создать оба файла одновременно при выполнении команды `make` без аргументов.

8.2 Результаты моделирования и их обсуждение

Следует обратить внимание, что одна из точек, где берётся преобразование Фурье, находится слева от начального импульса. Это позволяет наблюдать в этой точке только отраженный от слоя сигнал. Другая точка берётся справа от слоя, где распространяется прошедший сигнал. Для расчёта коэффициентов отражения и прохождения требуется также и спектр начального сигнал. Его проще всего получить, если выполнить программу еще раз и либо положить толщину слоя равной нулю, либо его проницаемость – единице. Полученные файлы с данными позволяют построить требуемые коэффициенты. Типичные спектры прошедшего и падающего импульсов показаны на рис. 10. Видно, что прошедший спектр сильно изменён по сравнению с падающим.

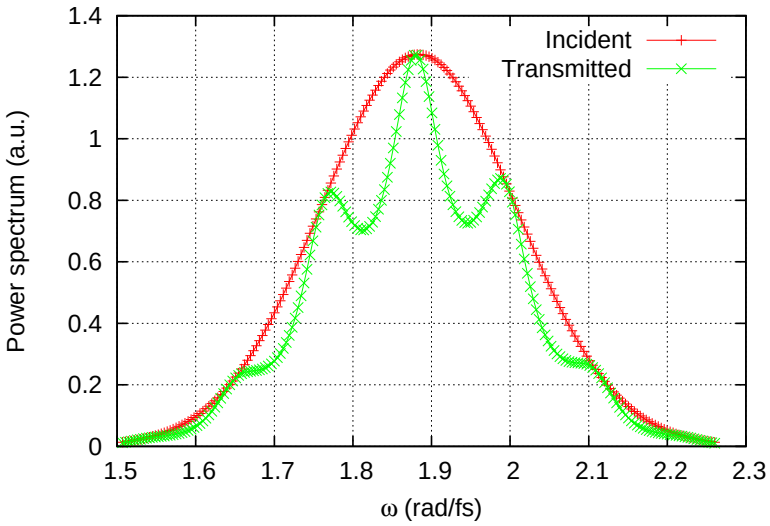

Рис. 10: Спектры падающего и прошедшего импульсов при $\lambda_0 = 1000$ нм, $\tau = 8$ фс, $\varepsilon = 4$, $L = 4000$ нм.

Интересно проверить, насколько правилен расчёт. Для этого можно получить аналитические формулы для коэффициентов. Они имеют следующий вид:

$$r = \frac{(n^2-1)(e^{2i\varphi}-1)}{(n+1)^2-(n-1)^2e^{2i\varphi}}, \quad t = \frac{4ne^{i\varphi}}{(n+1)^2-(n-1)^2e^{2i\varphi}}, \quad \text{где} \quad \varphi = nkL, \quad (21)$$

$n = \sqrt{\varepsilon}$ – индекс преломления для слоя, L – его толщина, $k = \omega/c$ – волновое число в вакууме.

Рисунок 11 показывает коэффициенты отражения и прохождения, рассчитанные численно, и те же самые коэффициенты, построенные на основании формул (21). Видно, что результаты хорошо совпадают.

9 Моделирование диэлектрических сред с поглощением

В предыдущих разделах мы изучили распространение импульсов в свободном пространстве и в диэлектрике без дисперсии и поглощения. На практике же большинство сред имеет дисперсию и поглощение, а их моделирование требует дополнительной работы. Более того, использование временно́го подхода означает, что моделирование проводится в некоторой области частот, и учёт изменения свойств среды в этом диапазоне может быть просто необходим для получения правильного результата. Следует также отметить, что аналитическое решение электродинамических задач в таких средах становится, как правило, очень сложным, а поэтому разработка численных методов представляет особый практический интерес.

Численные подходы к моделированию сред с дисперсией и поглощением зависят от теоретической модели, используемой для описания частотной зависимости. При расчёте распространения импульса в диэлектрике в разделе 6

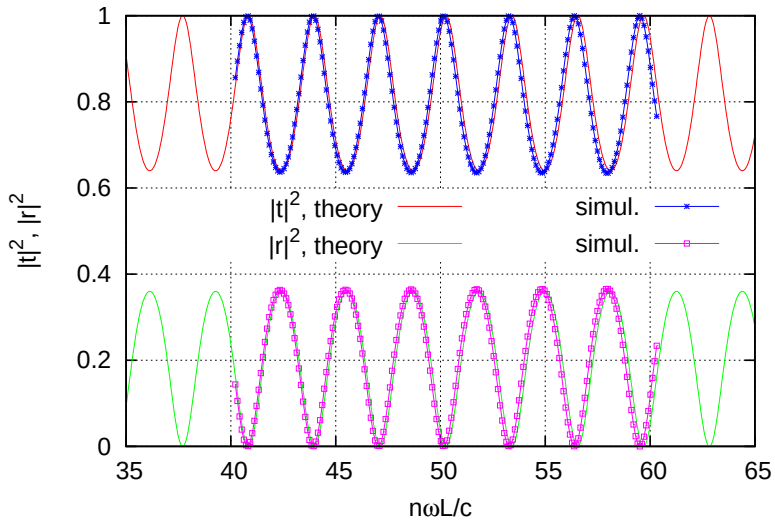

Рис. 11: Коэффициенты прохождения и отражения для слоя диэлектрика, рассчитанные численно (simul.) и аналитически (theory).

мы в некоторых момент $n\Delta t$ находили поле E из D. А поле D получалось из уравнения Максвелла. Давайте теперь обобщим этот подход на случай, когда в диэлекрической среде с некоторым реальным ε имеется поглощение, которое описывается проводимостью σ:

$$J = \sigma E. \tag{22}$$

Будем далее считать, что ток, определяемый уравнением (22), входит в поле D и, таким образом, уравнение Максвелла

$$\frac{\partial D}{\partial t} = -c\frac{\partial H}{\partial x} \tag{23}$$

остается таким же, как и без поглощения. Воспользуемся связью полей

$$D = \varepsilon E + 4\pi P \tag{24}$$

и, продифференцировав по времени это уравнение, мы получим

$$\frac{\partial D}{\partial t} = \varepsilon\frac{\partial E}{\partial t} + 4\pi\sigma E. \tag{25}$$

Используя конечные разности, мы перепишем (25) для каждой ячейки i в виде

$$D_i^{n+1} - D_i^n = \varepsilon\left(E_i^{n+1} - E_i^n\right) + 2\pi\sigma\Delta t\left(E_i^{n+1} + E_i^n\right). \tag{26}$$

Обозначая

$$\eta = 2\pi\sigma\Delta t, \tag{27}$$

мы получим следующую формулу для нахождения поля E_i^{n+1}:

$$E_i^{n+1} = \frac{\varepsilon - \eta}{\varepsilon + \eta}E_i^n + \frac{D_i^{n+1} - D_i^n}{\varepsilon + \eta} \tag{28}$$

Таким образом, для нахождения поля E_i^{n+1} при наличии проводимости σ необходимо знать не только D_i^{n+1}, но и D_i^n, E_i^n, то есть значения полей в предыдущие моменты времени.

Можно поступить и по-другому. Интегрирование уравнения (24) даёт

$$D(t) = \varepsilon E(t) + 4\pi\sigma \int_{-\infty}^{t} \mathrm{d}t' \; E(t'). \tag{29}$$

Используя метод трапеций для оценки интеграла в (29), мы получаем следующее соотношение для полей в момент времени $(n+1)\Delta t$:

$$D_i^{n+1} = \varepsilon E_i^{n+1} + 2\eta \left(\sum_{m=0}^{n} E_i^m + \frac{1}{2} E_i^{n+1} \right). \tag{30}$$

Используя (30), поле E_i^{n+1} можно найти как

$$E_i^{n+1} = \frac{D_i^{n+1} - 2\eta S_i^n}{\varepsilon + \eta}, \quad \text{где} \quad S_i^n = \sum_{m=0}^{n} E_i^m. \tag{31}$$

То есть наличие поглощения привело к необходимости ввести величину S_i^n в уравнение (31), но нет необходимости теперь сохранять D_i^n и E_i^n.

Таким образом, мы получили два разных алгоритма, (28) и (31), для расчёта E_i^{n+1} для сред с поглощением. Оба они одинаково применимы и совместимы со схемой (18). Следует отметить, что полученные формулы работают только в рамках модели поглощения, задаваемой формулой (22). При использовании других моделей конкретные формулы алгоритма будут отличаться от (28) и (31), но могут быть получены методами, изложенными в этом разделе.

10 Безотражательные (поглощающие) граничные условия

Для нахождения коэффициентов прохождения и отражения при падении импульса на слой в разделе 8 нам приходилось брать довольно большую область моделирования, чтобы избежать попадания сигнала, отраженного на границе области, в точки, где бралось преобразование Фурье. Было бы проще, если можно было бы избежать такого отражения. Мы рассмотрим здесь два способа достичь этого.

Сначала разберёмся, почему возникает отражение. Рассмотрим правую границу области моделирования на рис. 3. При нахождении поля $B^{n+1/2}$ в крайней точке, используя уравнение (5а), мы не знаем поле E справа от нее. Поэтому мы использовали условие $E = 0$, как на идеальной металлической границе, что и приводило к отражению импульса. В свободном пространстве импульс распространяется со скоростью света вдоль оси x. Таким образом, если взять $\xi = 1$, то E^n справа от последней точки задания B будет равно значению поля E слева, но в предыдущий момент времени, то есть E^{n-1}. Это можно и использовать

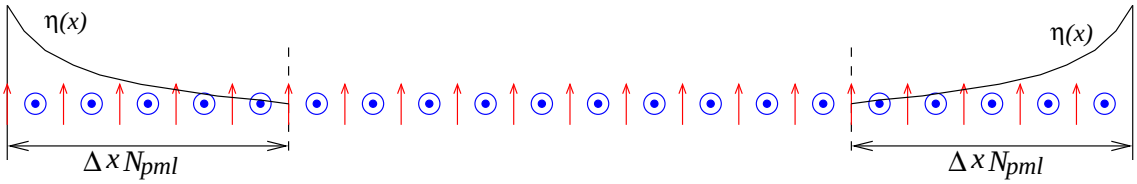

Рис. 12: Применение поглощающего материала на концах области моделирования.

как граничное условие для того, чтобы избежать отражения. Таким образом, уравнение для крайней правой точки становится

$$B_i^{n+1/2} = B_i^{n-1/2} - \xi \left(E_i^{n-1} - E_i^n \right). \tag{32}$$

Если выбрать $\xi = 1/2$, то надо брать значение E^{n-2} вместо E^{n-1}. Такое граничное условие работает хорошо в 1D случае. В 2D и 3D такой подход не работает, так как волны не будут плоскими и нельзя будет так просто экстраполировать их значение.

Теперь мы изучим подход, который в принципе можно обобщить на 2D и 3D задачи. Рассмотрим падение волны из свободного пространства на некоторый материал, который характеризуется диэлектрической проницаемостью ε и магнитной проницаемостью μ. Будем считать, что этот материал обладает поглощением. Поэтому величины ε и μ будут комплексными при нахождении решения в виде $e^{-i\omega t}$. Комплексный коэффициент отражения имеет следующий вид:

$$r(\omega) = \frac{1 - \sqrt{\varepsilon/\mu}}{1 + \sqrt{\varepsilon/\mu}}. \tag{33}$$

Видно, что при соблюдении условия $\sqrt{\varepsilon/\mu} = 1$ отражения не будет. При наличии поглощения слой достаточной толщины из такого материала, помещенный на правый и левый концы области моделирования, может полностью поглотить падающие волны. Граничные условия в таком случае не будут играть никакой роли, так как волны до концов области моделирования доходить не будут. Особенностью такого материала является то, что поглощение связано не только с мнимой частью в ε, но и с мнимой частью в μ. Такое магнитное поглощение можно ввести так же, как и диэлектрическое, рассмотренное в разделе 9. Слой такого материала обычно называют идеально согласованным (perfectly matched layer или PML). Следует отметить, что обычно выбирают параметр η, определённый формулой (27), таким образом, что его величина плавно нарастает от начала слоя, как показано на рис. 12. Это позволяет избежать отражения из-за численных ошибок, связанных с дискретизацией.

11 Моделирование сред с дисперсией

Как мы видели в разделе 6, в средах без дисперсии нахождение поля E из D не вызывает сложностей. Если же есть дисперсия, то, как правило, простое соотношение между E и D существует только в частотной области:

$$\tilde{D}(\omega) = \varepsilon(\omega)\tilde{E}(\omega). \tag{34}$$

Это уравнение можно перевести во временну́ю область и полученный интеграл использовать для нахождения E^{n+1}. Это наиболее общий подход.

На практике, однако, часто встречаются ситуации, в которых можно довольно просто учесть дисперсионные свойства. Мы рассмотрим типичный случай среды, где имеются свободные носители, то есть плазма. Такая среда характеризуется частотно-зависимой диэлектрической проницаемостью

$$\varepsilon(\omega) = \varepsilon_b \left(1 - \frac{\omega_p^2}{\omega^2}\right), \quad \omega_p = \sqrt{\frac{4\pi e^2 N}{\varepsilon_b m_e}}, \tag{35}$$

где ε_b – диэлектрическая проницаемость материала без носителей, ω_p – плазменная частота. Но мы не будем использовать это соотношение. Вместо этого мы возьмём уравнение для динамики свободных носителей

$$m_e \frac{\partial V}{\partial t} = eE. \tag{36}$$

Учитывая, что плотность тока $J_{fc} = eNV$, мы получаем

$$\frac{\partial J_{fc}}{\partial t} = \frac{\varepsilon_b \omega_p^2}{4\pi} E. \tag{37}$$

Удобно задавать J в тех же точках, что и E. При этом брать J в моменты времени, такие же как H. Тогда мы получаем

$$J_i^{n+1/2} = J_i^{n-1/2} + \frac{\varepsilon_b \omega_p^2}{4\pi} \Delta t E_i^n \tag{38}$$

и

$$D_i^{n+1} = D_i^n - \xi \left(H_i^{n+1/2} - H_{i-1}^{n+1/2}\right) - 4\pi \Delta t J_i^{n+1/2}. \tag{39}$$

На практике бывает удобно избежать различных постоянных множителей и ввести, например, $\bar{J} = 4\pi \Delta t J$.

Таким образом, для учёта дисперсии, вызываемой свободными носителями, в основной алгоритм (18) добавляется уравнение для расчёта тока, а в уравнение Максвелла – дополнительный член. При этом структура алгоритма и все разработанные до этого функции можно использовать практически без изменений.

12 Разделение на области полного и рассеяного полей

В разделах 4 и 8 моделирование распространения требовало задания начальных полей $E(x, t = 0)$ и $H(x, t = -\Delta t/2)$. Для этого надо, чтобы область полностью вмещала начальный импульс. На практике это может приводить к необходимости увеличивать размер области моделирования, что особенно критично при обобщении подхода на $2D$ и $3D$ случаи. Чтобы избежать этого, можно воспользоваться подходом, который описывается в этом разделе.

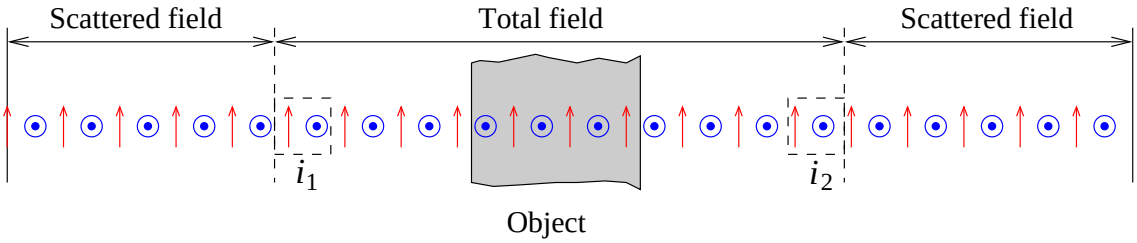

Рис. 13: Разбиение области моделирования на область полного и рассеянного полей.

Представим, что у нас имеется импульс, который падает на некоторый объект. Тогда полное поле всегда можно представить как сумму падающего поля и рассеянного поля, возбуждаемого наведённой поляризацией в объекте. До сих пор мы всегда рассчитывали полное поле, то есть не разделяли его на падающее и рассеянное.

Мы теперь можем разделить область моделирования на две области, см. рис. 13. В первой области, области полного поля, в которую входит и рассеивающий объект, мы будем рассчитывать полное поле, как и раньше. Во второй области, области рассеянного поля, мы будет учитывать только рассеянное поле, то есть без вклада падающего поля. В 1D случае область рассеянного поля состоит из двух частей – справа и слева от области полного поля.

Возникает вопрос, как нам правильно стыковать поля на границе. Давайте преположим, что границы области полного поля заданы ячейками i_1 и i_2, см. рис. 13. Тогда нам надо изменить формулы пересчёта полей по обе стороны от границы, то есть $B_{i_1-1}^{n+1/2}$, $D_{i_1}^{n+1}$, $B_{i_2}^{n+1/2}$, $D_{i_2+1}^{n+1}$. Предположим, что падающие поле задано функциями $E_0(x,t)$ и $H_0(x,t)$. Тогда на левой границе мы получаем

$$B_{i_1-1}^{n+1/2} = B_{i_1-1}^{n-1/2} - \xi\left[\left(E_{i_1}^n - E_0\left(i_1\Delta x, n\Delta t\right)\right) - E_{i_1-1}^n\right], \tag{40a}$$

$$D_{i_1}^{n+1} = D_{i_1}^n - \xi\left[H_{i_1}^{n+1/2} - \left(H_{i_1-1}^{n+1/2} + H_0((i_1-1/2)\Delta x, (n+1/2)\Delta t)\right)\right] \tag{40b}$$

или

$$B_{i_1-1}^{n+1/2} = B_{i_1-1}^{n-1/2} - \xi\left[E_{i_1}^n - E_{i_1-1}^n\right] + \xi E_0\left(i_1\Delta x, n\Delta t\right), \tag{41a}$$

$$D_{i_1}^{n+1} = D_{i_1}^n - \xi\left[H_{i_1}^{n+1/2} - H_{i_1-1}^{n+1/2}\right] + \xi H_0((i_1-1/2)\Delta x, (n+1/2)\Delta t). \tag{41b}$$

Мы видим из (41), что после вычисления полей по формулам без учета существования границы между областями к полям по обе стороны от неё надо добавить члены, зависящие только от падающего поля. На правой границе мы получаем похожие формулы:

$$B_{i_2}^{n+1/2} = B_{i_2}^{n-1/2} - \xi\left[\left(E_{i_2+1}^n + E_0\left((i_2+1)\Delta x, n\Delta t\right)\right) - E_{i_2}^n\right], \tag{42a}$$

$$D_{i_2+1}^{n+1} = D_{i_2+1}^n - \xi\left[H_{i_2+1}^{n+1/2} - \left(H_{i_2}^{n+1/2} - H_0((i_2+1/2)\Delta x, (n+1/2)\Delta t)\right)\right] \tag{42b}$$

или

$$B_{i_2}^{n+1/2} = B_{i_2}^{n-1/2} - \xi\left[E_{i_2+1}^n - E_{i_2}^n\right] - \xi E_0\left((i_2+1)\Delta x, n\Delta t\right), \tag{43a}$$

$$D_{i_2+1}^{n+1} = D_{i_2+1}^n - \xi\left[H_{i_2+1}^{n+1/2} - H_{i_2}^{n+1/2}\right] - \xi H_0((i_2+1/2)\Delta x, (n+1/2)\Delta t). \tag{43b}$$

При разделении областей следует иметь в виду, что в начальный момент времени падающий импульс должен находиться полностью за пределами области полного поля. В противном случае надо также задавать поле начального импульса в области полного поля, чего удобнее избежать. Отличительной особенность 1D случая по сравнению с 2D и 3D является то, что область рассеянного поля разделена на две части. Бывает удобно использовать разделение на области полное/рассеянное поле слева, где находится падающий импульс, а справа такое разделение не использовать. При этом прошедшее поле будет полным.

13 Пример 3: Падение импульса на слой плазмы

13.1 Разработка программы моделирования

Рассмотрение падения импульса на слой плазмы позволяет нам применить методики моделирования, изученные в разделах 9, 10, 11, 12: разделение на области полного и рассеянного поля, создание поглощающих слоев на границе области моделирования и учёт временной дисперсии. Ниже приведены разработанные функции моделирования.

Файл `slab2.c` с новой функцией управления `slab2()`:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<complex.h>

typedef double complex dcomplex;
extern const double cspeed, pi;

#include"fdtd_1d_maxwell.h"
#include"pulse.h"
#include"output.h"
#include"mater.h"
#include"pml.h"
#include"mater_fc.h"
#include"rfourier.h"
#include"set_mem.h"

/*****************************************************************************/
void slab2(void){

  char *tag="v1";     // used to label output files

  /*** Plasma slab ***/
  double   wp = 3.0;   // rad/fs
  int   Nslab = 30;    // number of cells

  /** Optical pulse ***/
  double lambda0 = 2*pi*cspeed/wp; // nm
  double     tau = 5;              // fs, width of the pulse
```

```
31
32  /*** Computational parameters ***/
33  double  dx = 20.0;  // nm
34  int     Nx = 2000;  // number of cells along x
35  int    ix0 = 0;     // center of the pulse at t=0
36  int     i1 = 500;   // scattered/total boundary
37
38  int    si1 = 1000;        // start of the slab
39  int    si2 = si1+Nslab-1; // end of the slab
40
41  int    fi1 = si2+5;   // location of fourier transform
42  int    fi2 =  i1-5;   // location of fourier transform
43
44  double  xi = 0.9;
45  int     No = 200;     // defines the output rate
46
47  /*** start execution ***/
48  double w0 = 2*pi*cspeed/lambda0; // rad/fs
49  double  dt = xi*dx/cspeed;  // in fs
50  double twp2 = wp*wp*dt*dt;
51  printf("dx=%.12e nm, dt=%.12e fs\n", dx, dt);
52  printf("w_p*L/speed=%g\n", wp*dx*Nslab/cspeed);
53
54  /*** Implement pml ***/
55  int    Npml = 20;       // number of PML cell
56  double eta0 = 0.5;      // absorption parameter
57  double  *pml = malloc(4*Npml*sizeof(double));
58  double   *pmlSE1 = pml + 0*Npml, *pmlSE2 = pml + 1*Npml;
59  double   *pmlSH1 = pml + 2*Npml, *pmlSH2 = pml + 3*Npml;
60  dset_mem(4*Npml, pml, 0.0); // initialize PML data
61
62  /*** arrays for the fields ***/
63  double  *fields = malloc(5*Nx*sizeof(double));
64  double   *Hz = fields+0*Nx;
65  double   *Ey = fields+1*Nx;
66  double   *Dy = fields+2*Nx;
67  double   *Bz = fields+3*Nx;
68  double   *nJ = fields+4*Nx;
69  dset_mem(5*Nx, fields, 0.0); // initialize fields
70
71  double  *eps = malloc(Nx*sizeof(double));
72  dset_mem(Nx, eps, 1.0); // sets permittivity
73
74  double wmin = 0.7*w0;  // rad/fs
75  double wmax = 1.3*w0;  // rad/fs
76  int      Nw = 500;
77
78  dcomplex *ftall=malloc(2*Nw*sizeof(dcomplex));
79  dcomplex *ft1 = ftall + 0*Nw;
80  dcomplex *ft2 = ftall + 1*Nw;
81  zset_mem(2*Nw, ftall, 0.0+I*0.0); // initialize Fourier data
82
83  int  T=0; // total steps
84  for(;;){
85    printf("Number of steps to run (<=0 to exit) -> ");  fflush(stdout);
86    int steps;
```

```
87      scanf("%d", &steps);
88      if(steps<=0) break;
89
90      printf("Making %d steps\n", steps);
91      for(int n=0; n<steps; n++, T++){
92        update_Bz(Nx, Bz, Ey, xi);                  // find Bz at n+1/2
93        Bz[i1-1] += xi*pulse((i1-ix0)*dx,T*dt,cspeed,tau,w0);    // t/s boundary
94        memcpy(Hz, Bz, Nx*sizeof(double));          // find Hz at n+1/2
95        update_H_pml(Nx, Hz, Bz, pmlSH1, pmlSH2, Npml, eta0); // find Hz inside PML
96        update_nJ(Nslab, nJ+si1, twp2, Ey+si1);    // find nJ at n+1/2
97        update_Dy(Nx, Dy, Hz, xi);                  // find Dy at n+1
98        Dy[i1] += xi*pulse((i1-0.5-ix0)*dx,(T+0.5)*dt,cspeed,tau,w0); // t/s boundary
99        add_J_to_D(Nslab, Dy+si1, nJ+si1);          // free carrier current
100       update_Ey(Nx, Ey, Dy, eps);                 // find Ey from Dy at n+1
101       update_E_pml(Nx, Ey, Dy, pmlSE1, pmlSE2, Npml, eta0);
102
103       /* output of Ey */
104       if((T+1)%No == 0){
105         printf("Elapsed time -> %g fs (%d steps)\n", dt*(T+1), T+1);
106         output_Ey_vs_x(Nx, Ey, T+1,  dx, tag);
107       }
108
109       /*** take running fourier ***/
110       double time=dt*(T+1); // for Ey
111       rfourier2(wmin, wmax, Nw, ft1, ft2, Ey[fi1], Ey[fi2], dt, time);
112
113     }// end of n loop
114
115     char fname1[100], fname2[100];
116     sprintf(fname1, "ft_fi=%d_%s.dat", fi1, tag);
117     sprintf(fname2, "ft_fi=%d_%s.dat", fi2, tag);
118     four_out(Nw, ft1, wmin, wmax,  fname1);
119     four_out(Nw, ft2, wmin, wmax,  fname2);
120
121   }// end of global loop
122
123   free(fields);   free(eps);  free(pml);
124 }
125
126 /*****************************************************************************/
127 int main(void){
128
129   slab2();
130
131   return 0;
132 }
133
134 /*** END OF FILE *************************************************************/
```

Файл `pml.c` с функциями, отвечающими за поглощающие слои:

```
1  /*****************************************************************************/
2  static void bupdate(const double *D, double *E, double *SE, double shift,
3                      int i, int j, int Npml, double eta0){
4    /*
```

```
   * i - index for the fields E, D; j - index for the SE array
   */

  double tmp = (i+shift)/Npml;
  double eta = tmp*tmp*eta0;
  double bot = 1.0+eta;

  E[j]   = (D[j] - 2*eta*SE[i])/bot;
  SE[i] += E[j];
}

/*****************************************************************************/
void update_H_pml(int Nx, double *H, const double *B, double *SH1, double *SH2,
                  int Npml, double eta0){
  /*
   * Updates H inside pml
   */

  for(int i=0; i<Npml; i++){
    bupdate(B, H, SH1, 0.5, i, Npml-1-i, Npml, eta0); //left
    bupdate(B, H, SH2, 0.5, i, Nx-Npml+i, Npml, eta0);  //right
  }
}

/*****************************************************************************/
void update_E_pml(int Nx, double *E, const double *D, double *SE1, double *SE2,
                  int Npml, double eta0){
  /*
   * Updates E inside pml
   */

  for(int i=0; i<Npml; i++){
    bupdate(D, E, SE1, 1.0, i, Npml-1-i, Npml, eta0); // left
    bupdate(D, E, SE2, 0.0, i, Nx-Npml+i, Npml, eta0);  // right
  }
}

/*** END OF FILE *************************************************************/
```

Файл `pml.h` с описанием функций из `pml.c`:

```
void update_H_pml(int Nx, double *H, const double *B, double *SH1, double *SH2,
                  int Npml, double eta0);
void update_E_pml(int Nx, double *E, const double *D, double *SE1, double *SE2,
                  int Npml, double eta0);
```

Файл `mater_fc.c` с функциями, позволяющими моделировать наличие свободных носителей, то есть дисперсии:

```
/*
 * Contains functions to model free carriers
 */

```

```
5  /*************************************************************************/
6  void add_J_to_D(int Nx, double *D, double *nJ){
7    for(int i=0; i<Nx; i++)
8      D[i] -= nJ[i];
9  }
10
11 /*************************************************************************/
12 void update_nJ(int Nx, double *nJ, double twp2, double *E){
13   for(int i=0; i<Nx; i++)
14     nJ[i] += twp2*E[i];
15 }
16 /*** END OF FILE **********************************************************/
```

Файл `mater_fc.h` с описанием функций из `mater_fc.c`:

```
1  void add_J_to_D(int Nx, double *D, double *nJ);
2  void update_nJ(int Nx, double *nJ, double twp2, double *E);
```

Для создания выполняемого файла мы должны добавить соответствующие команды в `makefile`. Это делается так же, как и в предыдущих примерах.

13.2 Результаты моделирования и их обсуждение

Рисунок 14 показывает распределение поля E_y в разные моменты времени. Сначала ($T = 600$ на рис. 14) импульс входит постепенно в область полного поля. При этом значение E_y меняется скачком на границе областей полного и рассеянного полей. Рассеянное поле еще не образовалось в этот момент. Затем ($T = 1200$ на рис. 14) импульс начинает взаимодействовать со слоем и формируются прошедшее и отраженное поля ($T = 1800$ на рис. 14). В этот момент нету скачка поля слева от слоя на границе областей полного и рассеянного полей, так как существует только рассеянное поле. Достигнув концов области моделирования ($T = 2400$ на рис. 14), прошедшее и отраженное поля поглощаются. На самом деле присутствует очень слабое отражение от идеально согласованного слоя. Коэффициент отражения $|r| \sim 10^{-5}$, что пренебрежимо мало в данном случае. Такие маленькие значения свидетельствуют о правильном выборе алгоритма и параметров поглощения.

Рассмотрим теперь полученные коэффициенты отражения и прохождения, см. рис. 15. При расчёте отраженного сигнала точка для преобразования Фурье берётся в области рассеянного поля, что позволяет исключить вклад падающего импульса. Прошедшее поле не разделяется на падающее и рассеянное. Как и прежде, мы сравниваем численные результаты с аналитическими. Видно, что результаты совпадают, что свидетельствует о правильности расчёта. При низких частотах, $\omega/\omega_p < 1$, проницаемость $\varepsilon < 0$ и поля спадают в слое. Прошедший сигнал получается в результате туннелирования падающего поля. При $\omega/\omega_p > 1$ проницаемость становится $\varepsilon > 0$ и волны в слое могут распространяться. При этом начинают проявляться интерференционные эффекты, которые затем исчезают при $\omega/\omega_p \gg 1$, когда слой становится прозрачным из-за $\varepsilon \approx 1$.

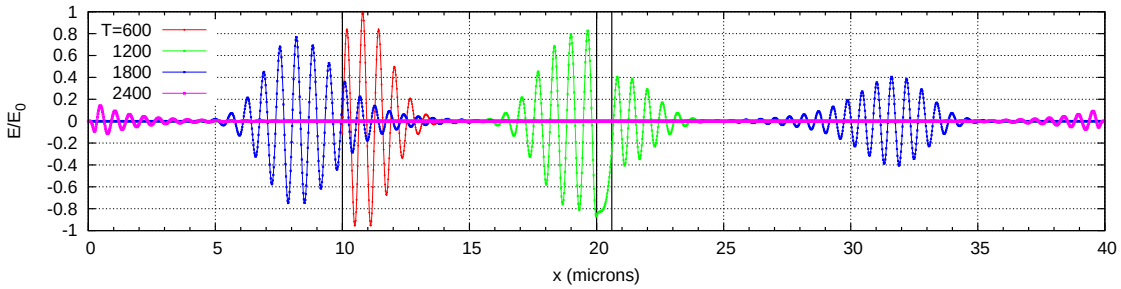

Рис. 14: Распределение E_y в разные моменты времени. Черные сплошные линии показывают границы слоя и границу между областями полного и рассеянного полей.

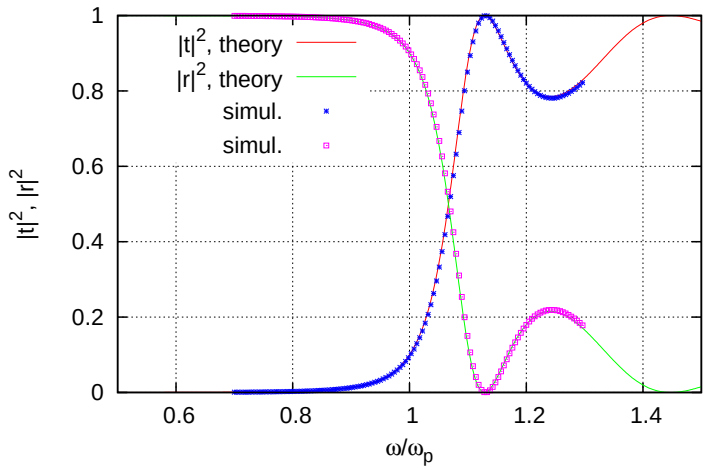

Рис. 15: Коэффициенты прохождения и отражения для слоя плазмы, рассчитанные численно (simul.) и аналитически (theory) при $\omega_p = 3$ рад/фс, $L = 600$ нм.

14 Заключение

В пособии были представлены теоретические основы численного моделирования распространения электромагнитных импульсов в различных средах в 1D случае. Также обсуждались такие темы, как написание программ, организация директорий, получение выполняемых файлов. Теоретические и практические вопросы моделирования иллюстрировались примерами с конкретными программами и с обсуждением результатов.

Особое внимание следует уделять численным ошибкам, которые всегда возникают при расчёте. Правильный выбор сетки и временного шага позволяет их уменьшить. Как правило, на практике достоверность полученного результата проверяется путём варьирования значений пространственных и временных шагов и достижения совпадения получаемых решений. Появление численной дисперсии и необходимость дискретизации всего моделируемого пространства вносят ограничения на область применения метода. В частности, он хорошо работает для моделирования областей с размерами, которые сравнимы с длиной волны или превосходят ее на два-три порядка. Тем не менее, часто выделение малого объема со сложной геометрией бывает достаточно для того, чтобы найти поля и на бóльших масштабах с использование приема пересчёта ближнего поля в дальнее, который также часто используется совместно с методом КРВО.

Отметим, что представленные подходы, как теоретические, так и практические, могут быть использованы не только для решения электродинамических задач, но и, например, для моделирования акустических волн, описываемых системой уравнений для давления и скорости. Также подходы применимы для моделирования квантовой динамики с помощья функции распределения, описываемой уравнением Шрёдингера. Магнитные свойства материалов, не рассматриваемые в пособии, могут быть учтены также, как и диэлектрические. Описание может быть обобщено и на анизотропные среды.

Подходы, использованные для 1D случая, могут быть далее развиты для 2D и 3D задач, причём как в декартовых, так и в цилиндрических или в сферических координатах. Такое обобщение, правда, имеет свои особенности, некоторые из которых мы перечислим:

- Более сложная пространственная сетка, требуемая для решения в 2D и 3D.
- Более жесткие требования на выбор временного шага. В отличие от 1D случая, в 2D и 3D от численной дисперсии нельзя избавиться.
- Создание идеально согласованного слоя требует более сложной теоретической основы и практической реализации в 2D и 3D.
- Наличие больших массивов для хранения полей требует дополнительной работы для сокращения как ресурсов памяти, так и вычислительного времени.

Следует также отметить, что метод КРВО в общем случае применим к широкому спектру электродинамических задач в силу своей простоты как в теоретическом понимании, так и в практическом воплощенни. При этом следует

отметить, что другие методы (метод конечных элементов, решение задач в частотной области и др.) также могут быть применены и, в зависимости от проблемы, давать более точные и быстрые результаты. На практике такие методы, как правило, требуют больше затрат на развитие теоретических основ и практическое воплощение. Для выбора оптимального способа решения конкретной задачи требуется хорошо представлять все возможные методы. Данное пособие ограничивается лишь введением в метод КРВО.

В силу популярности метода КРВО существует большое количество литературы на эту тему: статьи в научных журналах, монографии, учебники и различные пособия. Считается, что основные идеи алгоритма КРВО представлены в 1966 г. в статье [1]. Сильное развитие метод получил в период с конца 1980-х, что связано с прогрессом в вычислительной технике. Пожалуй, одно из наиболее полных описаний метода представлено в книге [2]. Более короткое описание можно найти, например, в учебнике [3].

Список литературы

[1] K. Yee, “Numerical solution of initial boundary value problems involving Maxwell’s equations in isotropic media”, IEEE Transactions on Antennas and Propagation, vol. 14, no. 3, pp. 302–307, 1966.

[2] A. Taflove and S. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd edn. Artech House, 2005.

[3] D. Sullivan, Electromagnetic Simulation Using the FDTD Method, 2nd Edition, Wiley-IEEE Press, 2013.

Об авторе

А. В. Маслов закончил радиофизический факультет ННГУ с получением степени бакалавра физики (с отличием) в 1995 г. В 1995-2001 гг. обучался в магистратуре и аспирантуре физического факультета Университета штата Вашингтон, г. Пульман, штат Вашингтон, США (Washington State University, Pullman, Washington, USA). Получил степени магистра физики в 1997 и доктора философии (Ph.D.) в 2001. Тема диссертации "Оптические свойства квантовой ямы, находящейся под воздействием терагерцового электрического поля". В 2002-2003 гг. работал научным сотрудником на факультете электронного и компьютерного инжиниринга в Технологическом университете штата Джорджия, г. Атланта (School of Electrical and Computer Engineering, Georgia Institute of Technology). В 2003-2006 гг. работал в отделе нанотехнологий научно-исследовательского центра НАСА Эймс (NASA Ames Research Center) в Кремниевой долине США, где разрабатывал теоретические модели и программы численного моделирования перспективных малоразмерных лазеров на основе полупроводниковых нанопроволок. Впоследствии работал старшим научным сотрудником в корпорации Кэнон США (Canon USA), где был ведущим разработчиком различных программ компьютерного моделирования: 3D моделирования полупроводниковых лазеров на квантовых ямах на основе GaN, оптоволоконных лазеров и параметрических усилителей, цифровой интерференционной микроскопии и других. С 2014 г. работает в ННГУ. Автор более 100 научных работ, из них около 70 статей в рецензируемых журналах, в области фотоники, физики лазеров, низкоразмерных полупроводниковых наноструктур, электродинамики нестационарных сред, генерации тергагерцового излучения, магнитных метаматериалов и оптомеханики. Выступал в качестве приглашённого докладчика на ведущих международных научных конференциях, а также на семинарах в университетах США и Европы.

Алексей Владимирович Маслов

РЕШЕНИЕ ЭЛЕКТРОДИНАМИЧЕСКИХ ЗАДАЧ МЕТОДОМ КОНЕЧНЫХ РАЗНОСТЕЙ ВО ВРЕМЕННО́Й ОБЛАСТИ

Учебно-методическое пособие

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский Нижегородский
государственный университет им. Н.И. Лобачевского».
603950, Нижний Новгород, пр. Гагарина, 23.