

Санкт-Петербургский государственный университет
Математическое обеспечение и администрирование
информационных систем

Тонких Артем Андреевич

Разработка системы расчета
характеристик объектов
по данным УЗИ – томографии

Отчёт по учебной практике (научно-исследовательской работе)

Научный руководитель
д.ф.-м.н., профессор, О.Н.Граничин

Санкт-Петербург
2021

Оглавление

Введение	2
Постановка задачи	3
Обзор решений	4
Основные результаты практики	6
Заключение	8
Список использованных литературных источников и инфор- мационных материалов	9
Приложение	10

Введение

Ультразвуковое исследование (УЗИ) – один из известных методов медицинского обследования. Такой метод намного безопаснее для пациента, так как не нужно подвергать пациента какому-либо излучению. Также этот метод быстрее, так как можно получать результаты прямо на приёме у врача. Наконец такая диагностика дешевле по сравнению с остальными. Самым известным применением УЗИ является обследование плода беременных женщин. В современном мире УЗИ не ограничивается только этим. Неуклонно растёт интерес применения УЗИ-аппаратов в диагностике рака молочной железы.

К сожалению, ещё нет способов диагностировать опухоли до 2 мм, а УЗИ может решить эту проблему. Это позволит врачам и пациентам за куда меньшие затраты диагностировать заболевания и вовремя назначить лечение.

Работа совершается в сотрудничестве с Китайской Ультразвуковой Лабораторией.

В задаче требуется найти такое распределение скоростей звука, которое минимизирует разницу между экспериментальными и расчётными данными. Поиск распределения включает такие этапы: с текущим распределением скоростей звука решается система уравнений; из решения системы извлекаются времена прихода сигнала; минимизируется квадрат нормы ошибки между настоящими временами (экспериментальные данные) и расчётными временами. Минимизация осуществляется методом Левенберга – Марквардта.

Постановка задачи

Цель работы: разработать систему, определяющую распределение скоростей звука в исследуемой области путём минимизации функционала ошибки.

Задачи работы:

- обзор существующих решений;
- реализовать алгоритм моделирования распространений ультразвуковых лучей в исследуемой области;
- добавить распараллеливание, чтобы ускорить расчёты;
- реализовать метод Левенберга – Марквардта;
- минимизировать разницу между экспериментальными данными и расчётом.

Обзор решений

Существует несколько способов ультразвуковой визуализации: использование отраженных сигналов, использование сквозных сигналов, затухание сигнала. В дальнейшем будут применяться сквозные и отражённые сигналы. В работе [1] использовались сквозные и отраженные сигналы, но в ней проводился поиск плотностей с заранее известными расположениями интересующих областей.

В данной работе предлагается изменить подход: разбить всю область на ячейки, в каждой ячейке будет своя собственная скорость прохождения звука. Далее, испуская сигнал, в каждой ячейке будет решаться следующая векторная система уравнений [2]:

$$\begin{cases} \frac{d\mathbf{r}}{dt} = c\mathbf{n}, \\ \frac{d\mathbf{b}}{dt} = \frac{-c_0\nabla c}{c} \end{cases}$$

и из неё будут определяться вектор распространения фронта волны (\mathbf{b}) и радиус-вектор (\mathbf{r}) сигнала на текущий момент времени. В этой системе \mathbf{n} – вектор нормали к фронту распространения волны, $\mathbf{n} = \frac{\mathbf{b}}{\|\mathbf{b}\|}$; c_0 – скорость распространения звука в воде, равная 1000 м/с; c – распределение скорости звука в области. Принято решение сделать свою реализацию метода Рунге – Кутты 4 порядка для этой системы дифференциальных уравнений, а не использовать готовые из открытого доступа, так как нужно проводить расчёт траектории до достижения определенного модуля искомого радиус-вектора (\mathbf{r}).

Вся область (внутри и снаружи круга) разбивается равномерной сеткой 1000×1000 . Для начала полагаем, что скорость звука равномерно распределена в области со значением 1200 м/с. После определения времён $\hat{t}_{ij}(c_0, \dots, c_{1000000})$, минимизируется функционал ошибки:

$$F(c_0, \dots, c_{1000000}) = \sum_{i,j} (\hat{t}_{ij}(c_0, \dots, c_{1000000}) - t_{ij})^2,$$

где t_{ij} – экспериментальные времена прихода сигнала от источника i к приёмнику j . Этот функционал будет минимизироваться мето-

дом Левенберга – Марквардта, который будет подбирать более оптимальные скорости звука в рассматриваемой области. Этот метод также решено реализовать самим с дальнейшим модифицированием, так как с таким большим количеством параметров (1000000) готовая реализация может просто не справиться. По своему опыту работы в среде Matlab, даже с 20 параметрами этот метод не всегда справлялся. Имеется идея модификации метода: делать просчёт траекторий не на каждой итерации метода Левенберга – Марквардта, а один в несколько итераций (например, раз в 5 итераций).

Основным языком программирования для реализации вышеописанного алгоритма на данный момент выбран Python.

Основные результаты практики

На данный момент реализован алгоритм, который решает систему дифференциальных уравнений

$$\begin{cases} \frac{dr}{dt} = c\mathbf{n}, \\ \frac{d\mathbf{b}}{dt} = \frac{-c_0 \nabla c}{c} \end{cases}$$

методом Рунге – Кутты 4 порядка, для области с равномерным и произвольным распределениями скоростей звука. Для начала считается, что имеется 16 датчиков, которые поочередно распространяют сигналы всем остальным датчикам, которые в свою очередь их измеряют. Визуализация распространений сигналов (трассировка лучей) с равномерным распределением 1200 м/с для одного датчика показана на рисунке 1. Визуализация распространений сигналов с неравномерным распределением для одного датчика показана на рисунке 2. На рисунке 2 присутствует круговая область с центром в точке $(-0.025, -0.025)$, радиусом 0.025 и скоростью звука 100 м/с. Как можно заметить траектории некоторых сигналов изменилась. В перспективе планируется провести расчёт для 2048 датчиков, что неминуемо потребует работы по экономии памяти, так как количество данных колоссальное: каждый из 2048 датчиков испускает сигнал, все остальные датчики принимают его, что в общей мере даёт 2048×2048 времен распространения сигнала. В связи с этим планируется распараллелить вычисления, используя библиотеку PyTorch для дальнейшего распараллеливания на GPU.

В настоящее время продолжаются:

- изучение языка программирования Python;
- изучение библиотеки PyTorch путём консультаций с командой руководителя, имеющей опыт работы с этой библиотекой в подобных задачах;
- работа в направлении модификации метода Левенберга – Марквардта.

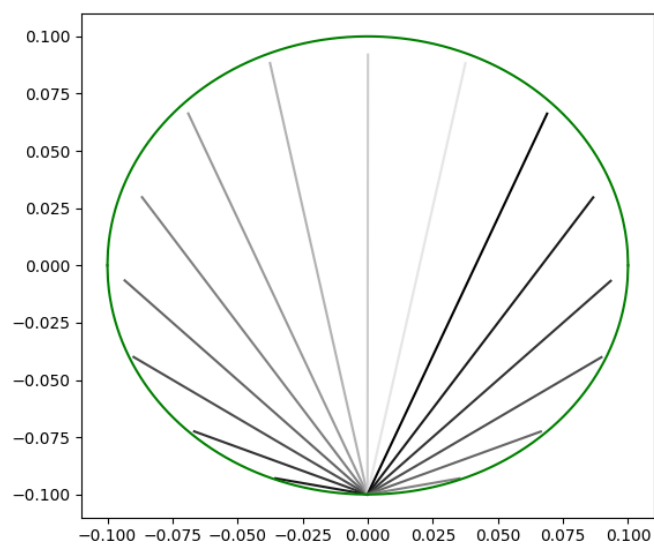


Рис. 1: Сигналы от одного датчика (равномерное распределение скоростей звука)

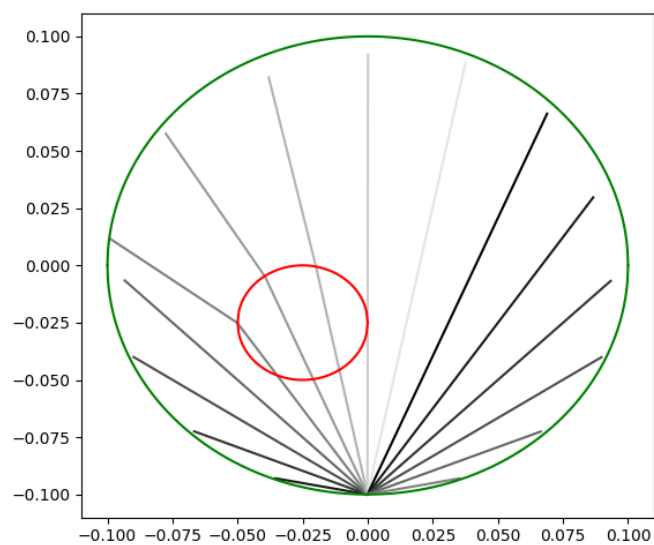


Рис. 2: Сигналы от одного датчика (неравномерное распределение скоростей звука)

Заключение

В рамках практики было выполнено:

- сделан обзор решений
- реализован алгоритм трассировки ультразвуковых лучей
- реализован метод Левенберга – Марквардта (без модификации)

Список использованных литературных источников и информационных материалов

- [1] Леонова А.В. Система для расчета скоростей звука в особых областях по данным УЗИ – томографии. Санкт-Петербург, 2020
- [2] Glide-Hurst C. K., Duric N., Littrup P. Volumetric breast density evaluation from ultrasound tomography images // Medical physics. 2008. Vol. 35, no. 9. P. 3988–3997.
- [3] Jovanovic I. Inverse problems in acoustic tomography. EPFL, 2008. – №4165
- [4] Li C. et al. In vivo breast sound-speed imaging with ultrasound tomography //Ultrasound in medicine biology. – 2009. – Т. 35. – №. 10. – С. 1615-1628.
- [5] Matthews T. P. et al. Regularized dual averaging image reconstruction for full-wave ultrasound computed tomography //IEEE transactions on ultrasonics, ferroelectrics, and frequency control. – 2017. – Т. 64. – №. 5. – С. 811-825.
- [6] Roy O. et al. Robust array calibration using time delays with application to ultrasound tomography //Medical Imaging 2011: Ultrasonic Imaging, Tomography, and Therapy. – International Society for Optics and Photonics, 2011. – Т. 7968. – С. 796806

Приложение

Код на языке программирования Python, реализующий трассировку лучей:

```
import matplotlib.pyplot as plt
import numpy as np
import torch
import math

def func(id_equation, massive_c, axis, y1, y2, x):
    #общая функция системы
    #id_equation - номер уравнения в системе, может быть 1 или 2
    #massive_c - массив скоростей звука
    #axis - оси (сетка равномерная)
    #y1 - искомый вектор в 1 уравнении
    #y2 - искомый вектор во 2 уравнении
    #x - точка в которой вычисляется значение функции
    #output - значение функции в точке x

    if(id_equation==1):
        func=func1(massive_c, axis, y1, y2, x)
    else:
        func=func2(massive_c, axis, y1, y2, x)
    return func

def func1(massive_c, axis, y1, y2, x):
    #функция 1 в системе
    #massive_c - массив скоростей звука
    #axis - оси (сетка равномерная)
    #y1 - искомый вектор в 1 уравнении
    #y2 - искомый вектор во 2 уравнении
    #x - точка в которой вычисляется значение функции
    #output - значение функции в точке x

    #func1=torch.zeros(2)
    func1=np.zeros(2)
    #n=axis.size(dim=0) #размер сетки
    n=len(axis)
    j=find(axis, x[0])
    i=find(axis, x[1])
```

```

func1=massive_c[n*i+j]*y2/norma(y2)
return func1

def func2(massive_c, axis, y1, y2, x):
    #функция 2 в системе
    #massive_c - массив скоростей звука
    #axis - оси (сетка равномерная)
    #y1 - искомый вектор в 1 уравнении
    #y2 - искомый вектор во 2 уравнении
    #x - точка в которой вычисляется значение функции
    #output - значение функции в точке x

    c0=1000.0 #скорость звука в воде
    #func2=torch.zeros(2)
    func2=np.zeros(2)
    #n=axis.size(dim=0) #размер сетки
    n=len(axis)
    j=find(axis, x[0])
    i=find(axis, x[1])
    c=massive_c[n*i+j]
    func2=-c0*gradient(massive_c, axis, y2, x)/massive_c[n*i+j]
    return func2

def gradient(massive_c, axis, y2, x):
    #градиент в точке x
    #massive_c - массив скоростей звука
    #axis - оси (сетка равномерная)
    #y2 - второй искомый вектор в системе
    #нужен для учёта направления при расчёте градиента
    #x - точка в которой вычисляется значение градиента
    #output - значение градиента в точке x

    h=axis[1]-axis[0]
    gradient_c=np.zeros(2)
    #n=axis.size(dim=0)
    n=len(axis) #размер сетки
    j=find(axis, x[0])
    i=find(axis, x[1])
    sgn_x=int(np.sign(y2[0])) #учёт направления при расчёте по вектору y2
    sgn_y=int(np.sign(y2[1]))

```

```

gradient_c[0]=(massive_c[n*i+j+sgn_x*1]-massive_c[n*i+j])
gradient_c[1]=(massive_c[n*(i+sgn_y*1)+j]-massive_c[n*i+j])
return gradient_c

def runge_kutta(id_equation, massive_c, axis, y1_old, y2_old, x):
    #метод Рунге - Кутты 4 порядка
    #id_equation - номер уравнения в системе, может быть 1 или 2
    #massive_c - массив скоростей звука
    #axis - оси (сетка равномерная)
    #y1_old - значение искомого вектора в 1 уравнении на текущий момент
    #y2_old - значение искомого вектора в 1 уравнении на текущий момент
    #x - точка, в которой находимся в данный момент
    #output - новое значение функции
    delta_t=1e-5
    h=axis[1]-axis[0]
    #perturbation=torch.ones(x.size(dim=0))
    perturbation=np.ones(np.size(x,0))
    k1=func(id_equation, massive_c, axis, y1_old, y2_old, x)
    k2=func(id_equation, massive_c, axis, y1_old+k1*h/2.0, y2_old+k1*h/2.0,
            x+perturbation*h/2.0)
    k3=func(id_equation, massive_c, axis, y1_old+k2*h/2.0, y2_old+k2*h/2.0,
            x+perturbation*h/2.0)
    k4=func(id_equation, massive_c, axis, y1_old+k3*h, y2_old+k3*h,
            x+perturbation*h)

    if(id_equation==1):
        vector_old=y1_old
    else:
        vector_old=y2_old
    vector_new=vector_old+delta_t/6.0*(k1+2*k2+2*k3+k4)
    return vector_new

def norma(vector):
    #норма вектора
    return math.sqrt(vector.dot(vector.T))

def find(axis, point):
    #поиск ближайшего к точке узла сетки
    #axis - рассматриваемая ось
    #point - точка, которую требуется найти на оси axis

```

```

#output - индекс ближайшей к точке point точки на оси axis
index=0
found=axis[0]
#n=axis.size(dim=0)
n=len(axis)
for i in range(n):
    if (math.fabs(axis[i]-point)<math.fabs(found-point)):
        found=axis[i]
        index=i
return index

n=1000      #количество узлов сетки
c=np.zeros(n*n) #массив скоростей звука
#if torch.cuda.is_available():
#    c = c.to('cuda')

D=0.2      #диаметр УЗИ-прибора
h=D/(n-1)  #шаг сетки
Radius=D/2.0 #радиус УЗИ-прибора
axis=np.linspace(-Radius,Radius,n)
for i in range(n):
    for j in range(n):
        #if (math.sqrt((axis[j]+0.025)**2+(axis[i]+0.025)**2)<=0.025):
        #c[n*i+j]=100.0
        #else:
        c[n*i+j]=1200.0
sensor_count=16
sensors=np.zeros((sensor_count,2))
#if torch.cuda.is_available():
#    sensors = sensors.to('cuda')
phi=np.linspace(0,2*math.pi,sensor_count+1)
#if torch.cuda.is_available():
#    sens_x = sens_x.to('cuda')
dimen=np.size(sensors,0)
for k in range(dimen):
    sensors[k,0]=axis[find(axis, Radius*np.cos(3*math.pi/2-phi[k]))]
    sensors[k,1]=axis[find(axis, Radius*np.sin(3*math.pi/2-phi[k]))]
Y1=np.zeros(2)
Y2=np.zeros(2)
fig = plt.figure(1)
ax = fig.add_subplot(111)
for j in range(dimen):

```

```

for k in range(dimen):
    if(j!=k):
        del Y1
        del Y2
        Y2=sensors[k,:]-sensors[j,:]
        Y1=sensors[j,:]
        time_step=0
        while(True):
            time_step+=1
            if(time_step>1):
                y1_old=Y1[time_step-1,:]
                y2_old=Y2[time_step-1,:]
            else:
                y1_old=Y1
                y2_old=Y2
            if(k==8):
                k=k
                y2_new=runge_kutta(2, c, axis, y1_old, y2_old, y1_old)
                y1_new=runge_kutta(1, c, axis, y1_old, y2_new, y1_old)

            if(norma(y1_new)>=Radius and time_step>1):
                rgb=np.array([k%10,k%10,k%10])/float(10)
                ax.plot(Y1[:,0], Y1[:,1], color=rgb)
                break
            Y1=np.vstack((Y1, y1_new))
            Y2=np.vstack((Y2, y2_new))
ylist1=np.zeros(np.size(axis))
ylist2=np.zeros(np.size(axis))
for i in range(np.size(axis)):
    ylist1[i]=math.sqrt(Radius**2-axis[i]**2)
    ylist2[i]=-math.sqrt(Radius**2-axis[i]**2)
#plt.plot(axis, sensors[:,1], 'g')
plt.plot(axis, ylist1, 'g')
plt.plot(axis, ylist2, 'g')
plt.show()
plt.show()

```

Код на языке программирования Python, реализующий метод Ле-венберга – Марквардта

```

import pylab
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

```

```

from numpy import linalg
import matplotlib.pyplot as plt
import math

def vector_function(x):
    F=[]
    with open("F_values.txt") as f:
        for line in f:
            F.append([float(x) for x in line.split()])
    m=len(F)
    vector_f=np.zeros(m)
    n=len(x)
    if(m==1):
        x=x[0]
        y=x[1]
        vector_f[0]=0.26*(x**2+y**2)-0.48*x*y
    else:
        for i in range(m):
            vector_f[i]=(x[i]-F[i])**2
    return vector_f

def F(x):
    return norm(vector_function(x))

def levenverg_marquardt(x_initial):
    maxiteration=20
    eps=1e-4
    lambda_k=0.0001
    nu=1.0001
    for i in range(maxiteration):
        if(i>0):
            x_initial=x
            f_initial=vector_function(x_initial)
            J=Jacob(x_initial)
            A=J.T.dot(J)+lambda_k*np.diag(np.diag(J.T.dot(J)))
            grad=2*J.T.dot(f_initial)
            b=-J.T.dot(f_initial)
            p=linalg.inv(A).dot(b)
            x=x_initial+p
            if (F(x)>=F(x_initial)):
                while(F(x)>=F(x_initial)):

```



```

        lambda_k*=nu
        f_initial=vector_function(x_initial)
        J=Jacob(x_initial)
        A=J.T.dot(J)
        A+=lambda_k*np.diag(np.diag(A))
        b=-J.T.dot(f_initial)
        p=linalg.inv(A).dot(b)
        x=x_initial+p
    else:
        lambda_k/=nu
        if(norm(p)<eps):
            break
    return x

def norm(vector):
    return math.sqrt(vector.dot(vector.T))

def Jacob(x):
    delta=1e-2
    f=vector_function(x)
    m=len(f)
    n=len(x)
    Jacobian=np.zeros((m,n))
    perturbation_vector=np.zeros(n)
    f_perturbation=np.zeros(m)
    for j in range(n):
        perturbation_vector[j]=delta
        f_perturbation=vector_function(x+perturbation_vector)
        for i in range(m):
            Jacobian[i,j]=(f_perturbation[i]-f[i])/delta
        perturbation_vector[j]=0
    return Jacobian

x_v=np.array([1.0,5.0])
x_v=levenverg_marquardt(x_v)
f=vector_function(x_v)
np.savetxt('test1.txt', x_v)
np.savetxt('test2.txt', f)

"""
x = np.arange(0., 5., 0.2)

```

```
y = np.arange(0., 5., 0.2)
xgrid, ygrid = np.meshgrid(x, y)
zgrid = (xgrid+2*ygrid-7)**2+(2*xgrid+ygrid-5)**2
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(xgrid, ygrid, zgrid)

pylab.show()
"""
```