МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра теоретической и прикладной информатики



**КУРСОВАЯ РАБОТА**

**по дисциплине: «***Планирование и анализ эксперимента***»**

**на тему:** **«***Синтез непрерывных D-планов эксперимента для нечетких однофакторных моделей с тремя подобластями определения***»**

Выполнил:Проверил:

Студент гр. *ПМ-41*, *ФПМИ* Профессор

Полохин С.Н. *Попов А.А.*

«21» марта 2018г.«21» марта 2018 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск

2018

# Перечень вопросов к разработке

1. Ознакомиться с математическим аппаратом построения регрессионных моделей в рамках концепции нечетких систем, вопросами оптимального планирования систем.
2. Разработать программное приложение синтеза непрерывных D-оптимальных планов экспериментов для однофакторных моделей.
3. Работа приложения должна быть продемонстрирована на нескольких тестовых примерах. Оптимальность полученных планов должна быть подтверждена выполнением соответствующих условий.

# Требования

Приложение должно осуществлять синтез непрерывных D-оптимальных планов эксперимента для нечетких регрессионных моделей с одним вещественным фактором. В правых частях нечетких правил

Локальные модели линейные или квадратичные. Область определения вещественной переменной (отрезок [-1;+1]) при фаззификации разбивается на три нечеткие партиции с функциями принадлежности:

Программное приложение должно быть переносимым.

# Решение

В соответствии с [1] регрессионная модель в рамках концепции нечетких систем может быть интерпретирована как взвешенная линейная комбинация моделей, определенных на нечетких множествах, с весами, равными вероятности модели в рассматриваемой точке. Очевидно, что в качестве вероятности выступает функция принадлежности модели, нормированная на сумму функций принадлежности всех моделей в данной точке. Таким образом итоговая модель выгляди следующим образом:

Поскольку в рассматриваемой задаче , то итоговую модель можно переписать в виде:

В случае если на i-ом нечетком множестве определена линейная модель, то

Если на i-ом нечетком множестве определена квадратичная модель, то

Для построения непрерывного оптимального плана эксперимента запишем используемый при построении информационной матрицы вектор f(x):

Построение плана будем производить в соответствии с последовательным алгоритмом [3, ст.28]. Алгоритм выбран при наличии в качестве альтернативы комбинированного алгоритма, поскольку первый является гораздо менее ресурсозатратным на этапе реализации, и при этом несущественно проигрывает в скорости сходимости второму.

Для глобальной оптимизации на одном из этапов алгоритма построения плана, будем использовать алгоритм Дифференциальной эволюции (реализованный в библиотеке SciPy для Python 3.5).

Процедуру очистки для построенного плана, в предположении что кластеры можно выделить явно, будем производить по следующему алгоритму:

Для каждой i-ой точки в спектре плана

создать i-ое множество , содержащие все точки из r-окрестности i-ой точки

Объединить все пересекающиеся множества в множества

План

Для каждого множества , содержащего более одного элемента

Добавить в среднюю по всему множеству точку с весом, равным сумме весов точек

Для каждого множества , содержащего один элемент

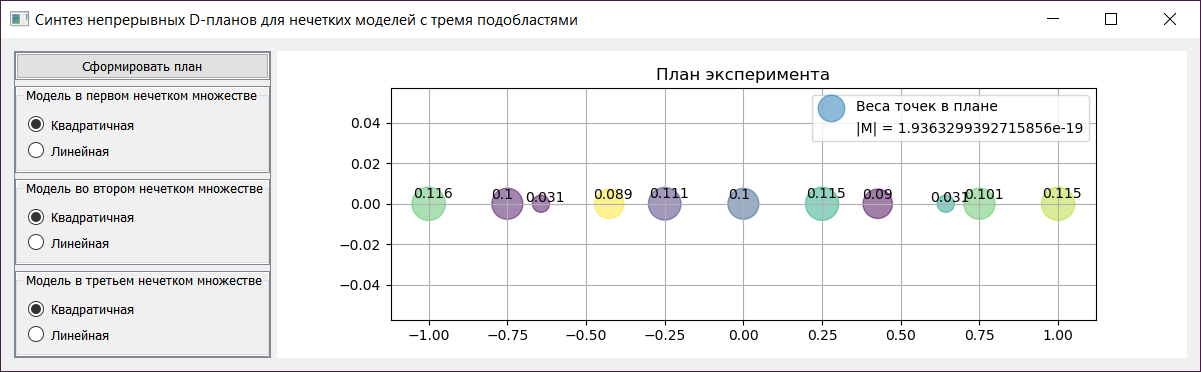
К каждой точке добавить (вес единственной точки в )/(количество точек в спектре )

*Алгоритм очистки плана*

# Результаты

Далее решения ищутся с точностью 1e-04.

|  |  |  |
| --- | --- | --- |
| Первая модель | Вторая модель | Третья модель |
| Квадратичная | Квадратичная | Квадратичная |

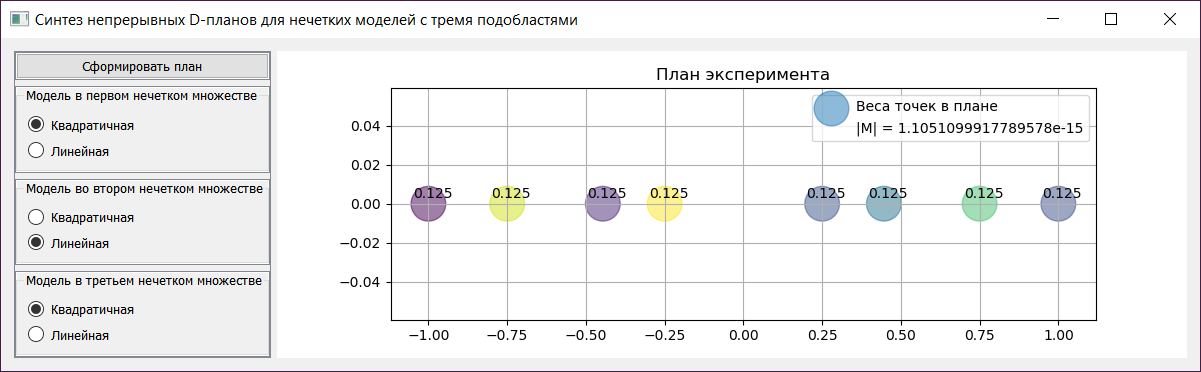


Полученный план:

|  |  |  |
| --- | --- | --- |
| *i* | Координата () | Вес () |
| 1 | -1,0000000 | 0,110788 |
| 2 | -0,7500000 | 0,095905 |
| 3 | -0,2500000 | 0,10768 |
| 4 | 0,0000000 | 0,091577 |
| 5 | 0,2500000 | 0,107997 |
| 6 | 0,7500000 | 0,096218 |
| 7 | 1,0000000 | 0,110731 |
| 8 | -0,4343319 | 0,106154 |
| 9 | 0,4342370 | 0,105828 |
| 10 | -0,6364388 | 0,033603 |
| 11 | 0,6351892 | 0,033518 |

=> условие оптимальности выполнено.

|  |  |  |
| --- | --- | --- |
| Первая модель | Вторая модель | Третья модель |
| Квадратичная | Линейная | Квадратичная |

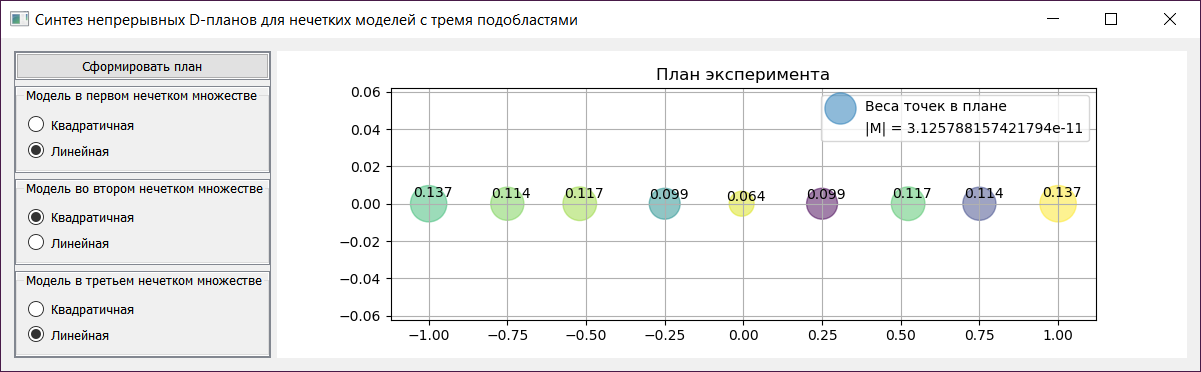


Полученный план:

|  |  |  |
| --- | --- | --- |
| *i* | Координата () | Вес () |
| 1 | -1,00000 | 0,124811 |
| 2 | -0,7498725 | 0,125032 |
| 3 | -0,4460891 | 0,124999 |
| 4 | 0,4460676 | 0,125008 |
| 5 | 0,7499098 | 0,125278 |
| 6 | 1,0000000 | 0,125014 |
| 7 | -0,2499995 | 0,125045 |
| 8 | 0,2500000 | 0,124813 |

=> условие оптимальности выполнено.

|  |  |  |
| --- | --- | --- |
| Первая модель | Вторая модель | Третья модель |
| Линейная | Квадратичная | Линейная |

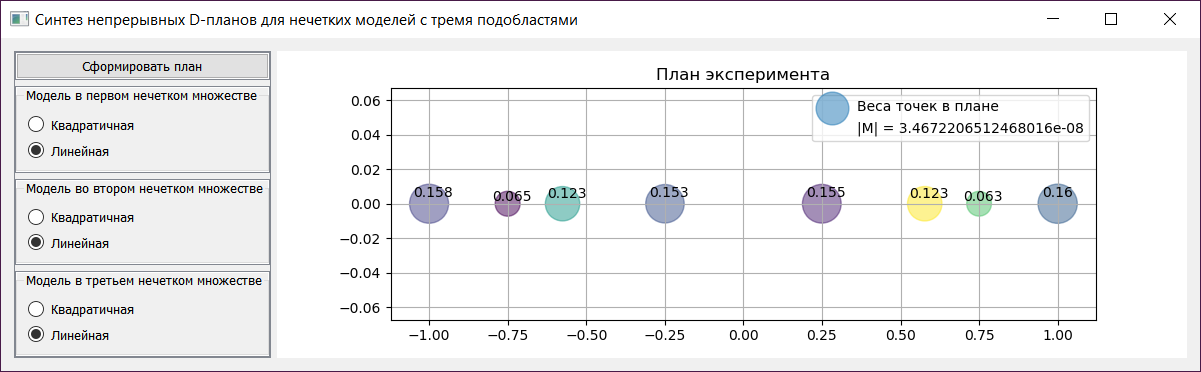


Полученный план:

|  |  |  |
| --- | --- | --- |
| *i* | Координата () | Вес () |
| 1 | -1,00000 | 0,136873 |
| 2 | -0,0055156 | 0,064391 |
| 3 | 0,9999998 | 0,137344 |
| 4 | 0,7500000 | 0,113896 |
| 5 | -0,7500000 | 0,114292 |
| 6 | 0,5235473 | 0,117455 |
| 7 | -0,5198794 | 0,117408 |
| 8 | 0,2500000 | 0,099183 |
| 9 | -0,2500000 | 0,099158 |

=> условие оптимальности выполнено.

|  |  |  |
| --- | --- | --- |
| Первая модель | Вторая модель | Третья модель |
| Линейная | Линейная | Линейная |



Полученный план:

|  |  |  |
| --- | --- | --- |
| *i* | Координата () | Вес () |
| 1 | -1,00000 | 0,157804 |
| 2 | -0,5753235 | 0,122603 |
| 3 | -0,2494565 | 0,153385 |
| 4 | 0,2496000 | 0,155399 |
| 5 | 0,5771090 | 0,123115 |
| 6 | 0,9999999 | 0,159703 |
| 7 | -0,7500000 | 0,064506 |
| 8 | 0,7500000 | 0,063485 |

=> условие оптимальности выполнено.

# Литература

1. Попов А.А. Регрессионное моделирование на основе нечетких правил / А.А. Попов // Сборник научных трудов НГТУ. -2000. -№2(19). -С.49-57.
2. Пегат А. Нечеткое моделирование и управление / А. Пегат; пер.с англ. -2-е изд. –М.: БИНОМ. Лаборатория знаний, 2013. -798 с.: ил.
3. Попов А.А. Планирование и анализ эксперимента. Конспект лекций. Электронный вариант.
4. Попов А.А. Оптимальное планирование эксперимента в задачах структурной и параметрической идентификации моделей многофакторных систем: монография / А.А. Попов. – Новосибирск: Изд-во НГТУ, 2013. – 296 с.

# Текст программы

Python 3.5

***“main v2.0.py”:***

import sys

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import QIcon

from qtasync import AsyncTask, coroutine

from PyQt5.QtCore import QCoreApplication, Qt,QThread

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.figure import Figure

import matplotlib.pyplot as plt

from core import fmaker

import random

from planmaker import make\_plan

import numpy as np

import asyncio

class App(QWidget):

fmaker\_ =0

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.left = 400

self.top = 400

self.title = 'PyQt5 matplotlib example - pythonspot.com'

self.width = 800

self.height = 600

self.fmaker\_ = fmaker()

self.plan=[[],[]]

self.initUI()

def center(self):

qr = self.frameGeometry()

cp = QDesktopWidget().availableGeometry().center()

qr.moveCenter(cp)

self.move(qr.topLeft())

def initUI(self):

self.setWindowTitle(self.title)

self.setGeometry(self.left, self.top, self.width, self.height)

self.center()

hbox = QHBoxLayout(self)

topleft = QSplitter(Qt.Vertical)

topleft.setFrameShape(QFrame.StyledPanel)

splitbutton = QSplitter(Qt.Horizontal)

splitbutton.setFrameShape(QFrame.StyledPanel)

splitter1 = QSplitter(Qt.Horizontal)

splitter1.addWidget(topleft)

topleft.addWidget(splitbutton)

parts= []

for i in range(3):

parts.append(QSplitter(Qt.Horizontal))

parts[i].setFrameShape(QFrame.StyledPanel)

topleft.addWidget(parts[i])

hbox.addWidget(splitter1)

self.setLayout(hbox)

self.setWindowTitle('Синтез непрерывных D-планов для нечетких моделей с тремя подобластями')

self.m = PlotCanvas(splitter1, width=5, height=4)

self.m.move(0,0)

self.radiobutton = []

dic = ["в первом","во втором", "в третьем"]

for i in range(3):

grid = QGridLayout()

group\_box = QGroupBox("Модель "+dic[i]+" нечетком множестве")

group\_box.setLayout(grid)

self.radiobutton.append(QRadioButton("Квадратичная"))

self.radiobutton[i \* 2].setChecked(True)

self.radiobutton[i \* 2].type = "quad"

self.radiobutton[i \* 2].toggled.connect(self.on\_radio\_button\_toggled1)

self.radiobutton.append(QRadioButton("Линейная"))

self.radiobutton[i \* 2 + 1].type = "lin"

self.radiobutton[i \* 2 + 1].toggled.connect(self.on\_radio\_button\_toggled1)

parts[i].addWidget(group\_box)

grid.addWidget(self.radiobutton[i \* 2],1,1)

grid.addWidget(self.radiobutton[i \* 2 + 1],2,1)

button = QPushButton('Сформировать план')

button.clicked.connect(self.start\_calculations)

splitbutton.addWidget(button)

self.show()

def on\_radio\_button\_toggled1(self):

radiobutton = self.sender()

if radiobutton.isChecked():

self.fmaker\_.change\_model((self.radiobutton.index(radiobutton)+1)//3, radiobutton.type)

@coroutine

def start\_calculations(self,arg):

button = self.sender()

button.setText('Производятся вычисления')

button.setEnabled(False)

for rb in self.radiobutton:

rb.setEnabled(False)

self.plan = yield AsyncTask(make\_plan,self.fmaker\_,button)

self.m.plot(self.plan)

file = open('plan.txt','w')

for i in range(len(self.plan[0])):

file.write(str(self.plan[0][i]) + '\t' + str(self.plan[1][i]) + '\n')

file.close()

button.setText('Сформировать план')

button.setEnabled(True)

for rb in self.radiobutton:

rb.setEnabled(True)

class PlotCanvas(FigureCanvas):

def \_\_init\_\_(self, parent=None, width=5, height=4, dpi=100):

fig = Figure(figsize=(width, height), dpi=dpi)

self.axes = fig.add\_subplot(111)

FigureCanvas.\_\_init\_\_(self, fig)

self.setParent(parent)

FigureCanvas.setSizePolicy(self,

QSizePolicy.Expanding,

QSizePolicy.Expanding)

FigureCanvas.updateGeometry(self)

def plot(self, plan):

data = [random.random() for i in range(25)]

self.axes.cla()

self.axes.scatter( x = plan[0], y = [0 for \_ in plan[0]],

s = 5e3 \* np.array(plan[1]), c = np.random.rand(len(plan[0])),

alpha = 0.5,

label = 'Веса точек в плане')

self.axes.scatter( x = [0], y = [0],

s = 0,

alpha = 0.0,

label = '|M| = ' + str(plan[2]))

plt.ylim(-1,1)

self.axes.legend()

for i, num in enumerate(plan[1]):

self.axes.annotate(round(num,3), (plan[0][i]-0.05,plan[1][i]/40))

self.axes.set\_title('План эксперимента')

self.axes.grid()

self.draw()

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = App()

sys.exit(app.exec\_())

***“planmaker.py”:***

import numpy as np

import scipy as sp

import scipy.linalg as lng

import scipy.optimize as opt

import sys

import multiprocessing as mp

from scipy.spatial import distance

import pickle

from core import fmaker

# Поиск информационной матрицы для известного плана

def Meps():

global plan

global f

global m

M=np.zeros((m,m))

for i in range(len(plan[0])):

fu=f.make(plan[0][i])

M+=plan[1][i]\*fu\*fu.T

return M

# phi(x,eps) = f.T \* d(psi)d(M) \* f = [ в случае psi == ln|M| (D-оптимальный план)]=

# = tr(M.inv(eps) \* M(x))

# M.inv - это матрица M в -1 степени

def phi (x):

global plan

global f

global Mep

fux=f.make(x)

Mx=fux\*fux.T

return -np.trace(np.dot(lng.inv(Mep),Mx))

# Создание нового в соответствии с алгоритмом

def make\_new\_plan(alpha,x):

global plan

for i in range(len(plan[0])):

plan[1][i]=(1-alpha)\*plan[1][i]

plan[0].append(x)

plan[1].append(alpha)

# Жутко неэффективно, зато не запарно

# Просто отменяет действие функции make\_new\_plan

def return\_old\_plan(alpha):

global plan

for i in range(len(plan[0])):

plan[1][i]=plan[1][i]/(1-alpha)

plan[0].pop()

plan[1].pop()

# Процедура очистки

# необходима для того, что окончательный спектр не содержал

# огромного числа близко расположенных точек.

# Точки объеденяются по группам в зависимости от близости

# к точкам изначального спектра

def clear(plan: list, threshold: int = 1e-2) -> list:

clear\_plan = [[],[]]

# ДАЖЕ НЕ ПЫТАЙТЕСЬ ПОНЯТЬ ОЧИСТКУ

free\_p = 0

while len(plan[0])>0:

que = [plan[0][0]]

amount = 1

middle\_point = plan[0][0]

sump = plan[1][0]

del plan[1][0]

del plan[0][0]

while len(que)>0:

j = 0

while j < len(plan[0]):

dist = distance.euclidean(que[0],plan[0][j])

if dist < threshold:

que.append(plan[0][j])

middle\_point += plan[0][j]

sump += plan[1][j]

del plan[1][j]

del plan[0][j]

amount += 1

j -= 1

j += 1

del que[0]

if amount > 1:

clear\_plan[0].append(middle\_point / amount)

clear\_plan[1].append(sump)

else:

free\_p+=sump

# Перераспределение весов свободных точек

len\_cp = len(clear\_plan[0])

free\_p /= len\_cp

for i in range(len\_cp):

clear\_plan[1][i]+=free\_p

return clear\_plan

plan=[]

m = 0

Mep = 0

f = 0

def make\_plan(fmake: fmaker, button, precision: float = 1e-4,

maxiter: int = 1000, iters\_to\_update: int = 10,

threshold:float = 5e-2) -> list:

global f

global plan

global m

global Mep

f = fmake

m = f.get\_m( )

start\_points = list(np.linspace(-1,1, m))

startn = len(start\_points)

# Создание изначального спектра

# В plan[0] хранятся точки плана

# В plan[1] хранятся веса точек плана

plan = []

plan.append([])

plan[0] = start\_points.copy()

# Задание весов плана

plan.append([])

for i in range(startn):

plan[1].append(1.0/startn)

# ПОдготовка к циклу

Mep=Meps()

psiold=np.log(lng.det(Mep))

gamma=2

iter\_=1

while True:

#2 Поиск глобального максимума

res = opt.differential\_evolution(phi,bounds=[(-1,1)])

point = res.x

val = -res.fun

#3 Проверка необходимых и достаточных условий выхода из алгоритма

if abs(-val+m)<precision or iter\_>maxiter:

break

# Инициализация alpha в соответствии с советами в методичке

alpha=2.0/(len(plan[0]))

while True:

#4 Создание нового плана

make\_new\_plan(alpha, point[0])

#5 Проверка допустимости плана

Mep=Meps()

psinew=np.log(lng.det(Mep))

# В отличии от оригинального алгоритма добавлена проверка значимости alpha

# Для избежания ситуаций зацикливания

if psinew<=psiold and alpha>1e-10:

return\_old\_plan(alpha)

alpha=alpha/gamma

else:

break

if iter\_ % iters\_to\_update == 0:

button.setText('Iteration: '+str(iter\_))

if iter\_ % (m\*40) == 0:

plan = clear(plan,threshold = threshold)

iter\_=iter\_+1

psiold=psinew

button.setText('Clearing...')

plan = clear(plan,threshold = threshold)

plan.append(lng.det(Meps()))

return plan

#make\_plan(iters\_to\_update = 10)

***“core.py”:***

import numpy as np

def mu1(x: float) -> float:

if x<-0.75:

return 1

if x<-0.25:

return -0.5-2\*x

return 0

def mu2(x: float) -> float:

if x<-0.75:

return 0

if x<-0.25:

return 1.5+2\*x

if x<0.25:

return 1

if x<0.75:

return 1.5-2\*x

return 0

def mu3(x: float) -> float:

if x<0.25:

return 0

if x<0.75:

return 2\*x-0.5

return 1

def flin(x):

return np.array([1, x])

def fquad(x):

return np.array([1, x, x\*\*2])

class fmaker:

# Доступные для использования модели

models\_avlb = {'lin': flin,'quad':fquad}

# Функции для вычиления вектора f

# и используемые для этого функции принадлежности

models\_used = [{'type':'quad', 'mu':mu1},

{'type':'quad', 'mu':mu2},

{'type':'quad', 'mu':mu3}]

def \_\_init\_\_(self):

return

def get\_avlb\_models(self):

return self.models\_avlb.keys()

def get\_m(self):

return len(self.make(1))

def change\_model(self,index: int, type\_: str, muf = None):

self.models\_used[index]['type'] = type\_

if muf!= None:

self.models\_used[index]['mu'] = muf

def make(self,x: float) -> float:

fs = []

for fparam in self.models\_used:

fs.append(fparam['mu'](x) \* self.models\_avlb[fparam['type']](x))

fs = np.concatenate(fs)

return fs[np.newaxis].T

if \_\_name\_\_ == '\_\_main\_\_':

fm = fmaker()

fm.change\_model(0,'lin')

fm.change\_model(1,'lin')

fm.change\_model(2,'lin')

a = fm.make(0.5)

print(a, fm.get\_m())