ISSN: 2658–5782

Номер 3–4

2021

МНОГОФАЗНЫЕ СИСТЕМЫ

mfs.uimech.org

ИМех
им. Р.Р. Мавлютова

ISSN 2658–5782 Том 16 (2021), № 3–4, с. 121–143

Многофазные системы

http://mfs.uimech.org/mfs2021.3.016
DOI: 10.21662/mfs2021.3.016
УДК 004.432.2, 004.942, 004.92, 517.938

Получена: 17.06.2021
Принята: 9.09.2021

Использование свободного ПО для визуализации результатов моделирования динамических процессов[1]

Насибуллаев И.Ш.

Институт механики им. Р.Р. Мавлютова УФИЦ РАН, Уфа

В работе представлен обзор современных свободных инструментов динамической визуализации и даны рекомендации для выбора инструментов в зависимости от метода исследования, формы представления исходных данных и специфики исследуемого явления. Для удобства использования материалов работы приводятся исходные коды как для программ вычислительных экспериментов классических задач, так и для создания графического отображения результатов моделирования. Для анимации процесса, описываемого аналитическими формулами, предлагается использовать терминал gif программы gnuplot или Python-библиотеки визуализации. Приведен пример применения данного подхода для решения модифицированного уравнения Ферхюльста–Пирла, описывающего изменение популяции при периодическом внешнем влиянии. При изучении нестационарных распределенных в пространстве явлений результаты могут быть представлены в видеоформате. Проведено моделирование задачи естественной конвекции в горизонтальном слое жидкости или газа в программе решения дифференциальных уравнений методом конечных элементов FreeFem++ с конвертацией результатов с помощью программ GhostScript и MEncoder в видеоформат. Приведен пример использования метода конечных разностей при моделировании автоколебательных химических реакций в среде Qt с сохранением кадров анимации в виде графических файлов. Для отображения результатов моделирования трехмерных динамических процессов предлагается использовать программу компьютерной графики Blender. Представлены моделирование и визуализация колебаний упругого маятника с помощью встроенного интерпретатора Blender Python API. Показан подход разделения вычислительного эксперимента и визуализации его результатов, позволяющий повысить эффективность использования вычислительных ресурсов. Предложен универсальный Python-скрипт для построения трехмерных траекторий движения объектов по внешним исходным данным.

Ключевые слова: свободное ПО, Gnuplot, Matplotlib, FreeFem++, Qt, MEncoder, Blender

1. Введение

Многие природные явления являются нестационарными и пространственно распределенными: ламинарное, вихревое или турбулентное течения в механике жидкости и газа [1, 2]; динамические процессы в многофазных и дисперсных средах; автоколебательные процессы в химии (например, реакция Белоусова–Жаботинского [3]); взаимодействие популяций в распределенных ареалах обитания в нелинейной популяционной динамике [4]. Для изучения и понимания этих явлений могут потребоваться инструменты для динамической визуализации в виде анимации. В связи с этим особый интерес представляет свободное программное обеспечение (ПО).

Широкое распространение получила программа интерактивной обработки и визуализации данных *ParaView* [5–7] (например, результаты мониторинга климатических изменений [8], космологических наблюдений [9, 10], моделирования задач механики сплошных сред, инженерного проектирования). Программа может работать с большими массивами данных, используя распределенную память и одно- и многопроцессорные системы, позволяет анализировать и отображать расчетные сетки, по-

[1] Работа выполнена за счет средств государственного задания № FWGZ-2019-0089.

ля переменных, их срезы в статическом и динамическом видах. *ParaView* может использоваться как постпроцессор во многих программах, например:

- в открытой интегрируемой платформе для численного моделирования механики сплошных сред *OpenFOAM* [11];
- в параметрической системе автоматизированного проектирования *FreeCAD* [12] (экспорт в формат *VTK* результатов моделирования встроенным решателем *CalculiX* [13]);
- в модульной открытой платформе численного моделирования *Salome* [14] (входит как стандартный модуль);
- в математическом пакете для решения междисциплинарных физических задач методом КЭ *Elmer* [15] (входит как стандартный модуль).

Существует большое количество библиотек для создания статической, динамической и интерактивной визуализаций данных, написанных для языка программирования *Python* [16, 17], например:

- *Matplotlib* [18–20] поддерживает двумерные и трехмерные графики и диаграммы, анимацию, отображение математических формул в стандарте LaTeX; может быть интегрирована в приложения *PyQt* [21, 22], *GTK+* [23], *wxWidgets* [24];
- *Seaborn* [20, 25] основана на *Matplotlib* и предлагает более удобный для использования синтаксис высокого уровня, но с меньшим количеством параметров настройки;
- *Plotly* [26] поддерживает анимацию и интерактивные элементы управления (кнопка, ползунок), интерактивное управление графикой (масштабирование, выделение области данных), LaTeX, интеграцию в вебстаницы, подключение в приложениях *Python*, *JavaScript*, *R*, *Julia*;
- библиотека декларативной статистической визуализации *Altair* [27, 28] создает интерактивную визуализацию (панорамирование, масштабирование), связанные графики, содержит инструменты преобразования данных;
- *Folium* [29] позволяет привязывать данные к локациям на интерактивных картах с возможностью подключения дополнительных плагинов (например, *Altair*).

Во многих свободных интегрированных средах разработки программ (например, *Qt* [30], *wxWidgets* [24], *Lazarus* [31]) имеются инструменты для отображения и сохранения в графическом формате статических изображений. Анимация достигается перерисовкой окна приложения (для контроля скорости анимации используется таймер). Для работы с трехмерной графикой подключают библиотеки (например, анимация *openGL* [32] с помощью библиотек *GLUT* [33], *SDL* [34] или *VTK* [35]).

Приведем примеры использования этих сред в научных исследованиях:

- *Qt* (язык программирования *C++*): рентгеновская навигационная система [36], многозадачная и многопоточная встаиваемая система управления автомобилем с графическим терминалом [37], модуль визуализации аэрогазодинамических расчетов [38], программа для отслеживания движения сердца по изображением магнитно-резонансной томографии [39];
- *Lazarus* (язык программирования *Object Pascal*): построение трехмерной модели кровеносной системы [40].

Некоторые свободные программы моделирования физических и инженерных задач содержат либо встроенные инструменты визуализации, либо эти инструменты реализованы в постпроцессоре: программа решения дифференциальных уравнений, записанных в вариационной форме, методом конечных элементов (КЭ) *FreeFem++* [41, 42] имеет возможность двумерной и трехмерной визуализации (анимации) результатов (КЭ-сеток, полей КЭ-функций) непосредственно в процессе моделирования [43], а также их сохранения в виде файлов в формате *EPS*; *CalculiX CrunchiX* [13], предназначенный для решения задач механики сплошных сред, содержит постпроцессор *CalculiX GraphiX* [13] для трехмерной визуализации геометрии, расчетных сеток, полей КЭ-функций и их сечений с возможностью сохранения изображений в графическом формате, а для автоматизации можно использовать командный файл.

Приведем примеры использования этих программ в исследованиях динамических процессов:

- *FreeFem++* — трехмерная модель регулирования течения жидкости сжатием упругой трубки [44] и осесимметричная модель пьезоэлектрического микронасоса [45, 46]; модель генерации течения жидкости колебаниями погруженного в нее пьезопривода с поперечным изгибом [47]; моделирование взаимодействия жидкости с конструкцией и силы удара по язычковому клапану [48]; моделирование системы охлаждения микрозахвата при нестационарном течении охлаждающей жидкости и

аналитический анализ смены рабочего режима [49, 50]; параллельные вычисления при моделировании фазового перехода твердого тела в жидкость с естественной конвекцией [51]; адаптивное численное моделирование генерации и распространения волн цунами [52];

- *CalculiX* — термомеханический анализ переходных процессов в модели двигатель–система подачи вторичного воздуха [53]; автоматическая методика аэромеханического анализа лопаток турбомашин [54]; анализ влияния типа КЭ на результаты трехмерного моделирования биомеханической модели человеческого оптического нерва и глазного дна [55].

Для преобразования графических файлов из одного формата в другой можно использовать конвертеры *GhostScript* [56] или *ImageMagick* [57], или программы для работы с растровой графикой *Gimp* [58]. Для создания научной графики высокого качества можно использовать пакеты *Tikz* или *PGFPlots* настольной издательской системы *LaTeX* [59]. Для сборки набора изображений в видеоформат применяется конвертер *MEncoder* [60].

При изучении трехмерных объектов многие исследователи используют программу компьютерной графики Blender [61]. Встроенная поддержка интерпретатора языка *Python* (*Blender Python API*) позволяет не только отображать статические трехмерные объекты, но и проводить компьютерное моделирование динамических процессов с последующей визуализацией. В качестве примера приведем применение *Blender* для исследования кинематики робототехнических систем: кинематическая модель манипулятора, состоящего из четырех сегментов с одной степенью свободы между сегментами [62]; управление техническими устройствами по данным, полученным с внешних датчиков [62]; модель многосегментного манипулятора, позволяющая с помощью видеокамеры и сенсоров отслеживать движение человека [63]; моделирование движения роботизированной руки робота с параллельным управлением, через USB порт, реальным роботом [64]; трехмерная модель конфигурирования модульного колесного робота–инспектора в трубе [65]. Отметим, что в области моделирования робототехнических систем набирает популярность робототехническая операционная система (ROS) [66, 67] с модулями моделирования, управления и визуализации Gazebo [68].

Blender используется и в других областях исследований: кинематическая модель человеческого тела, основанная на данных трехмерного сканирования, предназначенная для разработки персонализированной одежды [69]; инструмент для создания шаблона хирургической реконструкции, собранный по данным сканирования [70]; создание компьютерной модели трубы по данным лазерного сканера [71]; трехмерный анализ камнепада в зависимости от формы камней и сравнение результатов моделирования с реальным процессом [72]; динамические молекулярные модели и модели фрагмента дна Северного Ледовитого океана, построенные по расчетным и экспериментальным данным, представленным как численно, так и в форме графических изображений [73]. В работе [74] проверяется эффективность распараллеливания на 16 GPU различных модификаций модуля трассировщика пути для массового рендеринга *Blender Cycle*.

В настоящей работе представлены три набора инструментов для визуализации данных:

- *Gnuplot* и *Matplotlib* для построения научных графиков с анимацией;
- *Ghostsctipt* и *MEncoder* для конвертации результатов моделирования двумерных физических процессов в видеоформат;
- *Blender* для построения трехмерных динамических моделей и визуализации траекторных задач по исходным данным из внешнего источника.

Представлены примеры проведения вычислительных экспериментов методом конечных элементов в программе *FreeFem++* и методом конечных разностей в программе *Qt*, имеющие инструменты для визуализации в графическом окне в процессе моделирования с возможностью сохранения результатов в виде графических файлов.

2. Анимация графиков в Gnuplot и Matplotlib

При проведении параметрического анализа эволюционных систем удобно отображать полученную информацию в виде последовательно сменяющихся при изменении параметров графиков, т.е. провести визуализацию результатов исследования в виде анимации.

В качестве тестовой задачи выберем эволюционное дифференциальной уравнение первого порядка с начальным условием

$$\frac{dx}{dt} = ax\left(1 - \frac{x}{K}\right)(1 + c\sin(bt)), \quad x(0) = x_0. \qquad (1)$$

Смысл данного уравнения следующий: в начальный момент времени имеется популяция плотностью равной x_0; со временем t плотность популя-

ции $x(t)$ будет изменяться в зависимости от параметров a (скорость роста популяции) и K (поддерживающая емкость среды обитания, максимальная плотность популяции); скорость роста популяции зависит от времени так, что отклонения от среднего значения гармонические с амплитудой c и периодом T; параметр $b = 2\pi/T$ означает частоту колебаний изменения численности.

Разделяя переменные и интегрируя уравнение (1), получим решение в следующем виде:

$$x(t) = K \Big/ \left(1 + \frac{K - x_0}{x_0} \exp\left(-at - \frac{ac}{b}(1 - \cos(bt))\right)\right). \quad (2)$$

При отсутствии колебаний скорости рождаемости при $c = 0$ или $T \to \infty$ (соответственно $b \to 0$), с учетом того, что

$$\lim_{b \to 0} \frac{1 - cos(bt)}{b} = 0,$$

уравнение (1) переходит в логистическое уравнение Ферхюльста–Пирла [4] для популяции в ограниченной среде с внутренней конкуренцией.

Проведем параметрический анализ решения (2) для различных значений a, используя *Gnuplot*-терминал для создания *GIF* анимации *term gif*. Результат показан на рис. 1. Настройка терминала содержит следующие опции: графический файл поддерживает анимацию *animate*; задержка между кадрами *delay* (100 соответствует 1 секунде); размер графика *size*; шрифт текста *font*; обрезка пустых полей *crop*. Подключаем отображение сетки *set grid*:

```
set term gif animate delay 200 \
 size 600, 600 font "Times-Roman,16"
set grid
```

Определяем параметры модели:

```
k=1;a=0.4;T=2.0;b=2*3.14/T;c=0.5;x0=0.5;
n=4;t1=15;
```

Задаем функцию $f(t,i) = x(t)$, соответствующую решению (2) и параметру скорости роста a для i-го расчета:

$$a = a_m i/n, \quad i = (-n, n),$$

где a_m — амплитуда измерения параметра $a \in [-a_m, a_m]$; общее количество кадров равно $2n + 1$. Отрицательные значения скорости роста a могут означать, например, ухудшение экологии среды обитания или увеличение промысла изучаемой популяции. Дополнительная функция $g(i)$ используется для цветового обозначения *set style line* характера зависимости (2) (полное вымирание, снижение популяции, ее рост или насыщение).

```
f(x,i)=k/(1.0+(k/x0-1)*exp(-a*(1.0*i/n)*\
 (x+(cos(b*x)-1)/b)))
g(i)=(i==-n)?1:(i<x0)?2:(i==n)?4:3
set style line 1 lw 3 lc rgb 'red'
set style line 2 lw 3 lc rgb 'pink'
set style line 3 lw 3 lc rgb 'sea-green'
set style line 4 lw 3 lc rgb 'green'
set style line 5 lw 3 lc rgb 'blue'
```

Указываем имя выходного файла *output*; диапазон значений по осям координат *xrange* и *yrange*, подписи осей *xlabel* и *ylabel* и текст *label*:

```
set output "animate.gif"
set size square
set xrange[0:20]
set yrange[0:1.29]
set xlabel "{/Times-Italic t}, годы"
set ylabel "{/Times-Italic x}"
set label 2 at 1,1.05 'Gnuplot'
set label 3 at 1,1.15 '{/Times-Italic x}\
({/Times-Italic t})' tc rgb 'blue'
```

Отследим плотность популяции в момент времени $t_1 = 15$ лет (стрелка *set arrow 1* и подпись к ней *set label 1* со значением $x(t_1)$). Кадры формируются в цикле *do for*. Дополнительно отобразим асимптоты для полного вымирания $x = 0$, нулевого изменения плотности популяции $x = x_0$ и полного насыщения $x = K$:

```
do for [i=-n:n] {
 set arrow 1 from t1,x0 to t1,f(t1,i) ls 5 front
 set arrow 2 from t1,0 to t1,1 ls 5 lw 1 nohead
 set label 1 at x1+0.2,0.5*(x0+f(x1,i))+0.02\
  sprintf("{/Times-Italic x} = %5.3f", f(x1,i))
 plot 0 w l ls 1 title 'x = 0',\
  0.99*x0 w l ls 2 notitle,\
  x0 w l ls 3 title 'x = x_0',\
  k w l ls 4 title 'x = K',\
  f(x,i) w l ls g(i) title sprintf(\
   "x(t), a = %4.2f",a*(1.0*i/n))}
```

В результате выполнения скрипта создается анимированный файл *animate.gif* (отдельные кадры файла показаны на рис. 1). Формат *GIF* поддерживает одновременное использование не более 256 цветов в формате *(A)RGB* (по два байта на значения прозрачности, красного, зеленого и синего цветов). Прозрачный фон создается при указании опции *transparent*. Этот файл можно открыть в большинстве программ для просмотра графических файлов, а так же в веб-браузерах.

Анимацию можно встроить в *PDF*-файл с помощью *pdfLaTeX*. Для этого разбиваем файл с анимацией на графические файлы с отдельными кадрами консольной командой:

```
convert -coalesce animate.gif ani.png
```

В документе LaTeX подключаем пакет *animate*:

```
\usepackage{animate}
```

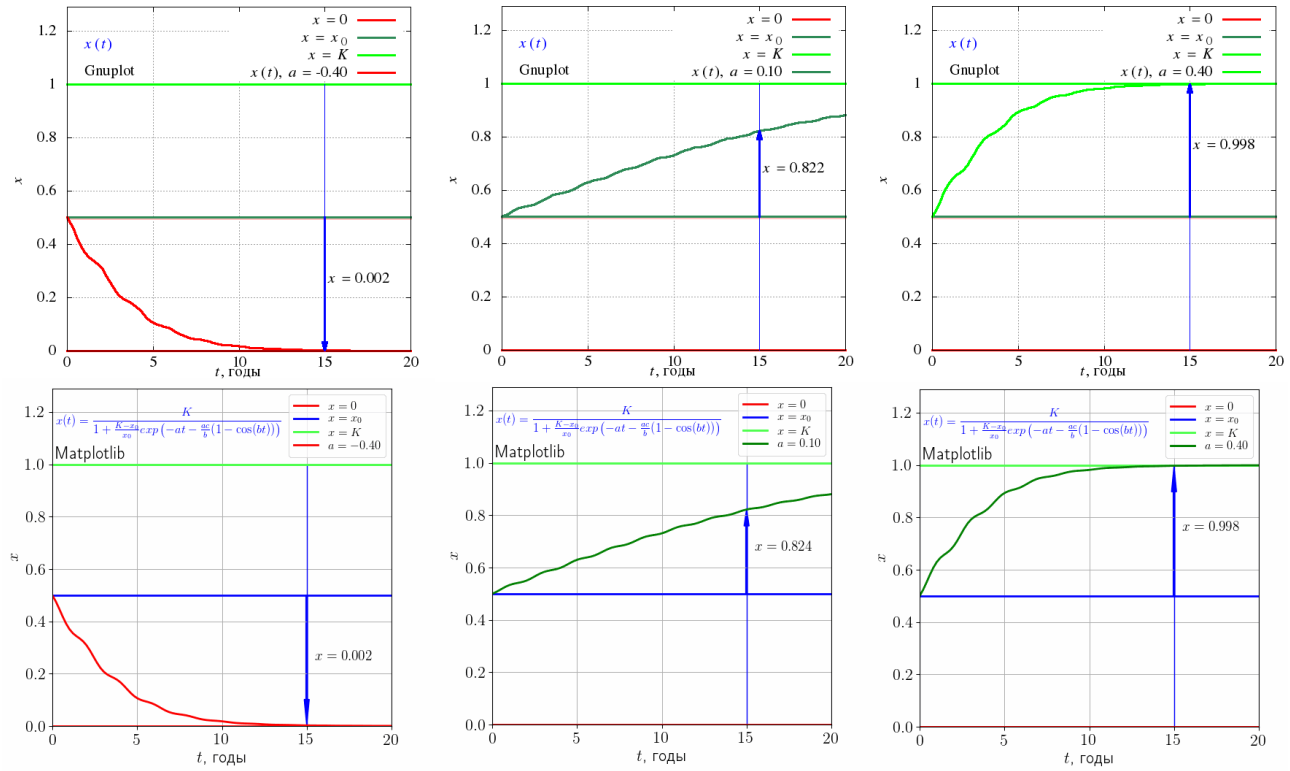

Рис. 1. Кадры анимации, построенные в *Gnuplot* (сверху) и *Matplotlib* (снизу)

Анимацию подключаем с помощью команды *animategraphics* со следующими опциями: непрерывное воспроизведение *loop*; автозапуск *autoplay*; вывод кнопок управления *controls*. Обязательные параметры: число кадров в секунду; неизменяемая часть имени файлов; номера первого и последнего кадров (можно указать подинтервал для воспроизведения группы файлов):

```
\begin{figure}[h]
\centering
\animategraphics[loop,autoplay,
 width=0.49\textwidth,controls]{10}{ani/ani-}
 {0}{80}
\captionВстроенная{ в PDF анимациядля  $n=40$}
\label{fig:gnuplot:anim}
\end{figure}
```

Использование анимации в *PDF*-файле позволяет повысить уровень восприятия материала в презентациях и электронных лекциях. Воспроизведение анимации полностью поддерживает только программа *Adobe Acrobat* (в других программах будет показан только первый кадр анимации).

Дополнительная информация по включению аудио и видео в учебные *PDF*-файлы (например, лекции) доступна в работе [75].

Для сравнения с *Gnuplot* построим анимацию графика, используя *Python*-библиотеку визуализации *Matplotlib* (результат показан на рис. 1). В текстовом редакторе создаем файл *animate.py*. Подключаем библиотеки, поддержку LaTeX и кириллических символов:

```
# coding: utf-8
import matplotlib.pyplot as plt
from matplotlib import rc
rc('figure', figsize=[6,6])
rc('xtick', labelsize=16)
rc('ytick', labelsize=16)
rc('text', usetex=True)
rc('text.latex',unicode=True)
rc('text.latex',\
preamble=r'\usepackage[utf8]{inputenc}')
rc('text.latex',\
preamble=r'\usepackage[russian]{babel}')
from matplotlib.patches import Arrow
import matplotlib.animation as animation
from math import exp, cos
import numpy as np
```

Задаем параметры:

```
k=1
a0 = 0.4
da=0.1
a = -a0-da
T=2.0
b=2*3.14/T
c=0.5
x0=0.5
t1=15.0
t2=20.0
m=100
```

Запишем функцию, определяющую диапазон данных по осям, и подписи осей, используя LaTeX формулы:

```
def init():
 ax.set_xlim(0, 20)
 ax.set_ylim(0, 1.29)
 ax.set_xlabel(r'$t$'u', годы', fontsize=16)
 ax.set_ylabel(r'$x$', fontsize=16)
 line.set_data(xd,yd)
 return line,
```

Здесь одномерные массивы x_d и y_d задают точки кривой в виде $(x_d[i], y_d[i])$, $i = 0, m$.

Запишем функцию построения графика для текущего значения параметра роста $a \in [-0.4, 0.4]$, меняющегося с шагом da. Новые точки кривой обновляются в функции *set_data()*:

```
def run(j):
 global a
 global arr
 a = a+da
 for i in range(0,len(xd)):
  yd[i]=k/(1.0+(k/x0-1)*\
  exp(-a*(xd[i]+(cos(b*xd[i])-1)/b)))
 line.set_data(xd, yd)
```

Для положительного параметра роста кривая (и соответствующая линия в легенде) будет иметь зеленый цвет, а для отрицательного — красный:

```
 if a > 0:
  line.set_color('g')
  legs.legendHandles[3].set_color('g')
 if a < 0:
  line.set_color('r')
  legs.legendHandles[3].set_color('r')
```

Обновляем стрелку, пояснительный текст со значениями популяции $x(t_1)$ и значение параметра a в легенде:

```
 arr.remove()
 arr = ax.add_patch(Arrow(t1,x0,0,\
 yd[75]-x0,width=0.5, color='b'));
 txt.set_y((x0+yd[75])/2.0)
 txt.set_text(r'$x ='+str("{:5.3f}".\
 format(yd[75]))+'$')
 legs.texts[3].set_text(r'$a = '+\
 str("{:4.2f}".format(a))+r'$')
 return line,
```

В основном коде программы создаем рисунок *fig*, оси *ax*, кривую *line*, стрелку *arr*, вспомогательные линии с постоянным значением для нулевой, средней и максимальной популяций, пояснительный текст со значением популяции $x(t_1)$ *txt* и надпись, содержащую большую формулу в формате LaTeX. Все кривые дополняются меткой *label* для автоматического построения легенды *legs*:

```
fig, ax = plt.subplots()
ax.grid()
ax.plot([t1, t1], [0, 2*x0], 'b', lw=1)
ax.plot([0, t2], [0, 0], 'r', lw=2,\
label='$x=0$')
ax.plot([0, t2], [x0, x0], 'b', lw=2,\
label='$x=x_0$')
ax.plot([0, t2], [2*x0, 2*x0], '#44FF44',\
lw=2, label='$x=K$')
line,=ax.plot([],[],color='0',lw=2,label='$a$')
arr = ax.add_patch(Arrow(0,0,t1,x0, color='b'))
txt = ax.text(t1+0.5,x0+0.01,'0', fontsize=14)
legs = ax.legend(loc=1, prop={'size': 12},\
labelspacing=0.0)
ax.text(0.2,1.10,r'$\displaystyle x(t)='\
 r'\frac{K}{1+\frac{K-x_0}{x_0}exp\left('\
 r'-at-\frac{ac}{b}\bigl(1-\cos(bt)\bigr)'\
 r'\right)}$',color='b',fontsize=12)
ax.text(0.2,1.02,r'Matplotlib',\
 fontsize=16)
```

Создаем массив с линейным распределением x_d и соответствующий массив y_d:

```
xd = np.linspace(0,t2,m)
yd = [0] * m
```

Задаем аргументы для функции построения анимации: имя рисунка *fig*; функция, которая вызывается в начале каждого кадра (обновление координат точек кривой для нового параметра a) *run*; количество кадров; время задержки между кадрами в миллисекундах *interval*; функция инициализации графика *init* и запрет циклического воспроизведения *repeat*:

```
ani = animation.FuncAnimation(fig, run,\
abs(int(2*a/da))-1, interval=10,\
init_func=init, repeat=False)
plt.show()
```

Анимацию сохраняем в файл *ani.gif*:

```
a=-a0-da
writergif = animation.PillowWriter(fps=10)
ani.save("ani.gif", writer=writergif)
plt.close()
exit()
```

Скрипт запускаем в консоли:

```
python animate.py
```

Представленные скрипты для *Gnuplot* и *Matplotlib* создают графику высокого качества. *Matplotlib* за счет поддержки математических формул LaTeX может создавать более сложное и качественное текстовое содержание (в *Gnuplot* формулы LaTeX используются в терминале *term epslatex*, который не поддерживает анимацию). В отличие от *Gnuplot* (близкого по функциональности скрипта) *Matplotlib* требует значительно больше строк кода (77 вместо 27) и имеет более сложный синтаксис.

Для сравнения используемых вычислительных ресурсов были запущены оба скрипта для построения анимационного файла, содержащего 801 кадр с разрешением 600×600 точек. *Gnuplot* создал файл анимации размером 6.8 МБ за 6.6 с, используя 3.4 МБ оперативной памяти (ОЗУ). *Matplotlib* создал файл размером 31.7 МБ за 115 с, используя 1.5 ГБ ОЗУ. При запуске скрипта *Matplotlib* без создания файла (только анимация в графическом окне) продолжительность визуализации (процессорное время, уменьшенное на величину суммарной задержки между кадрами) составила 34 с с использованием 65.9 МБ ОЗУ. Таким образом, для динамической анимации данных, полученных из аналитических зависимостей или текстовых файлов с данными, *Gnuplot* использует вычислительные ресурсы более эффективно при сопоставимом качестве визуализации.

Отметим, что вычислительные возможности *Gnuplot* ограничены. Для анимации более сложных наборов данных (результаты численных расчетов, базы данных экспериментов) библиотека *Matplotlib* или другие *Python* библиотеки визуализации имеют преимущество, поскольку базовые возможности этих библиотек значительно расширяются функциональностью языка программирования *Python* (например, использованием численных и научных модулей). Важной особенностью этих библиотек является возможность их встраивания в другие приложения (например, в графические интерфейсы, написанные на *Qt*, *GTK*, *Jupiter* и др.).

3. Динамическая визуализация FreeFem++/GhostScript/MEncoder

Рассмотрим нестационарную пространственную задачу естественной тепловой конвекции в горизонтальном плоском слое жидкости или газа — конвекцию Рэлея–Бенара [76]. Область имеет размер $L_x \times L_y$. Геометрия задачи показана на рис. 2.

В безразмерном виде уравнения для скорости жидкости $\mathbf{u} = (u_x, u_y)$ имеют вид [77]:

$$\frac{1}{\mathrm{Pr}}\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \mathrm{Ra}T\mathbf{e}_y + \Delta\mathbf{u}, \qquad \nabla\mathbf{u} = \varepsilon_p p, \tag{3}$$

где t — время; ∇ и Δ — операторы набла и Лапласа; p — давление; T — температура среды относительно температуры окружающей среды; $\mathbf{e}_y$ — единичный орт вдоль оси Oy; $\varepsilon_p = 10^{-6}$ — малый параметр, используемый для стабилизации численной схемы; безразмерные параметры Pr и Ra — числа

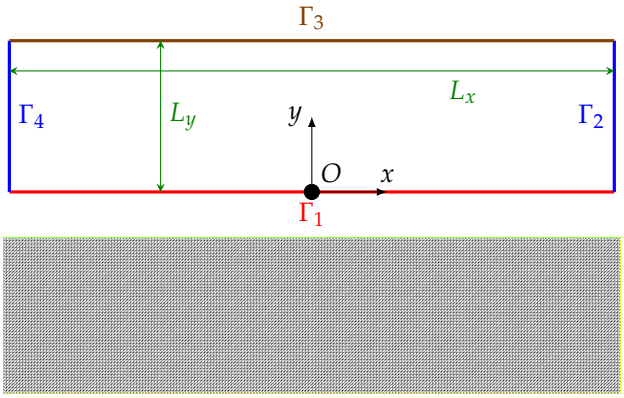

Рис. 2. Геометрия задачи и граничные условия (сверху) и расчетная сетка (снизу)

Прандтля и Рэлея:

$$\mathrm{Pr} = \frac{\nu}{\alpha}, \quad \mathrm{Ra} = \frac{g\beta L^3}{\nu\alpha}\delta T.$$

Здесь ν — кинематическая вязкость; α — коэффициент температуропроводности; g — ускорение свободного падения; β — коэффициент объемного расширения; L — характерный размер (высота слоя); δT — перепад температуры между нижней и верхней границами слоя. Плотность ρ зависит от температуры по закону (приближение Буссинеска)

$$\rho(T) = \rho_0(1 - \beta T).$$

Уравнение переноса тепла в безразмерной форме имеет вид:

$$\frac{\partial T}{\partial t} + (\mathbf{u}\cdot\nabla)T = \Delta T. \tag{4}$$

Начальные условия соответствуют неподвижной жидкости $\mathbf{u} = 0$ с температурой $T = T_0$. Граничные условия для жидкости определяются условием залипания $\mathbf{u} = 0$ на нижней границе Γ_1 и на боковых границах Γ_2 и Γ_4. На верхней границе Γ_3 вертикальная компонента скорости $u_y = 0$. Температура на верхней границе равна температуре окружающей среды $T = T_0$, а на нижней — $T = T_0 + \delta T$.

Решим уравнение с помощью программы для решения дифференциальных уравнений методом конечных элементов *FreeFem++* [42]. Результаты моделирования показаны на рис. 3.

Уравнения запишем в вариационной форме, дискретизацию по времени проведем по неявной схеме Эйлера.

Задаем длину L_x и высоту L_y слоя и количество узлов n_x и n_y в направлении Ox и Oy:

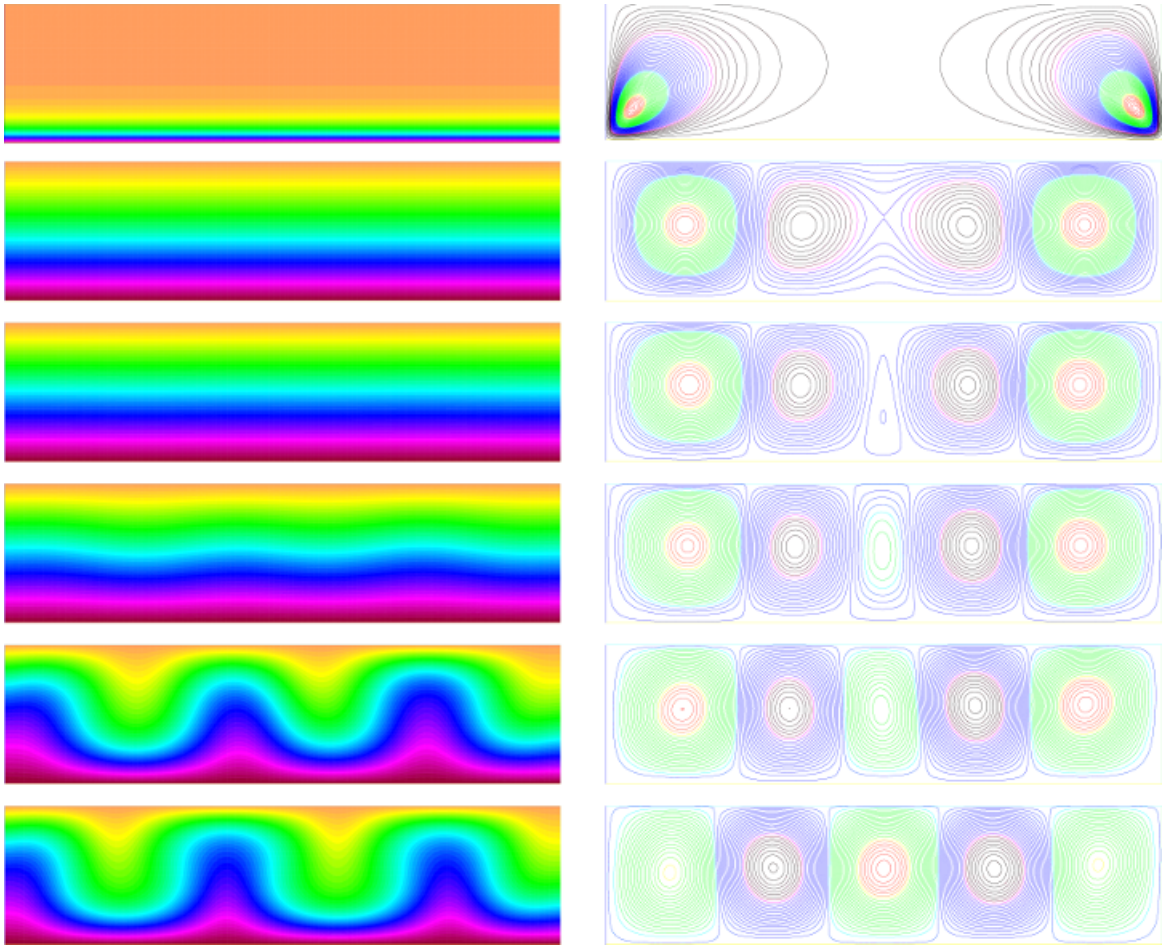

Рис. 3. Распределение температуры (слева) и линии тока (справа) для кадров $\{1, 5, 10, 15, 20, 60\}$ (сверху вниз)

```
real lx=4, ly=1;
int nx=200,ny=50;
```

Создаем расчетную сетку размером $L_x \times L_y$ и записываем ее изображение в графический файл *mesh.eps* (см. рис. 2):

```
mesh Th=square(nx,ny,[lx*x,ly*y]);
plot(Th,ps="mesh.eps");
```

Задаем следующие параметры: числа Прандтля и Рэлея; коэффициент искусственной сжимаемости ε_p в уравнении неразрывности (второе уравнение (3)); шаг по времени δt и коэффициент при полной производной по времени в уравнении Навье–Стокса $\tau = 1/(\delta t \mathrm{Pr})$ (левая часть первого уравнения (3)); температуры нижней T_1 и верхней T_2 границ; количество шагов по времени *steps* (обычно полное время вычислительного эксперимента $steps \cdot \delta t$ имеет тот же порядок, что и параметры, характеризующие процесс, например, характерное время релаксации скорости). Результаты расчетов будем записывать в файл тогда, когда текущий шаг по времени кратен переменной *save*:

```
real pr = 0.8, ra=3000.0, epsp=1.e-6;
real dt=0.01,tau=1/(dt*pr);
real T0=0,T1=1.0;
real  nc,np,eps;
cout << "Pr=" << pr << ", Ra=" << ra << endl;
int steps=300, save=5, k=0;
```

Определяем переменные и начальные условия: температура, давление, функция линии тока и соответствующие им пробные функции задаются конечно-элементными функциями первого порядка *P1*, а компоненты скорости течения жидкости и соответствующие им пробные функции — функциями второго порядка *P2*:

```
fespace Mh(Th,P1);
Mh T=T0,Tp=T0,dT=0,vT,p,vp,s,vs;
fespace Xh(Th,P2);
Xh ux=0,uy=0,uxp=0,uyp=0,ua,vx,vy;
```

Определяем количество интервалов в легенде *nb* и их значения *visou* для модуля скорости, а также область отображения для графических файлов:

```
int nb=51;
real[int] visou(nb);
for (int j = 0; j < visou.n; j++)
  visou[j] = 14.0*j/(visou.n-1);
func bbs = [[-0.01,-0.01],[1.01*lx,1.01*ly]];
func bbu = [[-0.01,-0.01],[1.2*lx,1.01*ly]];
string filePost;
```

В главном цикл программы решаем уравнения теплопроводности (4)

```
int i;
for (i=0; i<=steps; i++){
 solve Heat(T, vT, solver=UMFPACK,init=i) =
  int2d(Th) ( tau*T*vT
  +(dx(T)*dx(vT) + dy(T)*dy(vT)))
  +int2d(Th)(-tau*convect([uxp, uyp],-dt,Tp)*vT)
  +on(1,T=T1)+on(3,T=T0);
 Tp=T;
```

и уравнения гидродинамики (3):

```
solve NSPicard(ux, uy, p, vx, vy, vp,
 solver=UMFPACK,init=i) =
  int2d(Th) (
   tau*( ux*vx + uy*vy )
   + ( dx(ux)*dx(vx) + dy(ux)*dy(vx)
   + dx(uy)*dx(vy) + dy(uy)*dy(vy) )
   - p*(dx(vx)+dy(vy))
   + epsp *p*vp + (dx(ux)+dy(uy))*vp)
  - int2d(Th) (
   tau*( convect([uxp,uyp],-dt,uxp)*vx
   + convect([uxp,uyp],-dt,uyp)*vy))
   - int2d(Th) (ra * Tp * vy )
   +on(1,2,4,ux=0,uy=0)+on(3,uy=0);
uxp=ux;uyp=uy;
```

Определяем модуль скорости по формуле

$$u_a = \sqrt{u_x^2 + u_y^2},$$

норму модуля скорости в виде

$$n_c = \max(|u_a|)$$

и сходимость к стационарному решению с помощью относительного изменения n_c за один шаг по времени:

```
ua=sqrt(ux^2+uy^2);
np=nc;
nc=ua[].linfty;
if (nc) eps=100.0*abs(1.0-np/nc);
cout << "i=" <<i<< ", Ra=" <<ra<< ", ua.max="
 << ua[].max << ", eps=" << eps << "%" << endl;
```

Функция тока определяется решением уравнения [80]

$$\frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x} = \Delta s,$$

с граничными условиями на твердых стенках $s = 0$:

```
solve streamlines(s,vs,solver=UMFPACK) =
 int2d(Th)( dx(s)*dx(vs) + dy(s)*dy(vs))
  +  int2d(Th)( vs*(dy(ux)-dx(uy)))
  +  on(1,2,3,4,s=0.0);
```

На каждом *save*-шаге по времени сохраняем результаты для температуры, модуля скорости, вектора скорости и линии тока в графические файлы (перед запуском программы необходимо создать соответствующие папки):

```
 if (i % save == 0) {
  filePost=(k<10?"0":"")+(k<100?"0":"")
   +k+".eps";
  plot(T,fill=1, nbiso=nb, bb=bbs,
   ps="t/"+filePost);
  plot(ua,fill=1,value=1, viso=visou,
   nbiso=visou.n, ps="a/"+filePost);
  plot([ux,uy],coef=0.5, ps="c/"+filePost);
  plot(s,nbiso=nb, bb=bbs, ps="s/"+filePost);
  k++;}
}
```

После того, как программа создала набор *eps*-файлов, их можно объединить в видеофайл с помощью скрипта (конвертируем *eps* файлы в формат *png* с помощью *GhostScript*, а затем собираем видеофайл из *png* фреймов с помощью *MEncoder*):

```
cd ./t/
for i in $( ls *.eps ); do
 eps2eps $i $i.2eps
 gs -dSAFER -dBATCH -dNOPAUSE -dEPSCrop -r300 \
  -sDEVICE=pngalpha -dBackgroundColor=16#FFFFFF\
  -sOutputFile=$i.png $i.2eps
done
mencoder mf://*.png -mf fps=5:type=png -ovc \
 lavc -lavcopts vcodec=mpeg4:vbitrate=500000 \
 -o t.avi
rm -f *.2eps
rm -f *.png
mv t.avi ../t.avi
cd ..
```

Величина *save*dt* определяет промежуток времени процесса между двумя кадрами. Параметр *fps=N* означает, что видео будет показывать N кадров в секунду, т.е. время процесса t_s соотносится с реальным временем t_r как $t_s = save \cdot dt / N \cdot t_r$.

4. Динамическая визуализация Qt/MEncoder

Рассмотрим инструменты визуализации *Qt* (среды разработки кроссплатформенного программного обеспечения на языке программирования C++) на примере классической автоколебательной химической реакции Белоусова–Жаботинского [78]. Простейшей математической формулировкой, допускающей колебательный режим, является модель брюсселятора [79], которая

имеет следующий вид:

$$X_t = A - (B+1)X + X^2Y = f_1(X,Y),$$
$$Y_t = BX - X^2Y = f_2(X,Y), \quad (5)$$

где A и B — постоянные концентрации реагентов, а X и Y — переменные; $X_t = \partial X/\partial t$ и $Y_t = \partial Y/\partial t$ — частные производные по времени t.

Уравнения реакции (5) имеют одну особую точку: $X_f = A$, $Y_f = B/A$. При $B \leqslant 1 + A^2$ данная точка устойчива (при $t \to \infty$ концентрации $X \to X_f$ и $Y \to Y_f$). При $B > 1 + A^2$ особая точка неустойчива и на фазовой плоскости (X,Y) решение уравнений (5) образует предельный цикл (автоколебательная химическая реакция).

Уравнения (5) с начальными условиями $X(0) = 0.5$, $Y(0) = 2$ решались явным методом Эйлера на промежутке времени $[0,t]$:

$$X_{k+1} = X_k + \tau f_1(X_k, Y_k),$$
$$Y_{k+1} = Y_k + \tau f_2(X_k, Y_k), \quad (6)$$

где индексы k и $k+1$ относятся к предыдущему и текущему моменту времени; τ — шаг по времени.

Для изучения поведения пространственно распределенной системы строилась сетка размером $N \times N$ с узлами (x_i, y_j), $i = 0,\ldots,N-1$, $j = 0,\ldots,N-1$. Для каждого узла записывались уравнения реакции-диффузии:

$$X_t(x_i,y_j) = f_1(X(x_i,y_j),Y(x_i,y_j)) + D_1(X_{xx}(x_i,y_j) + X_{yy}(x_i,y_j)),$$
$$Y_t(x_i,y_j) = f_2(X(x_i,y_j),Y(x_i,y_j)) + D_2(Y_{xx}(x_i,y_j) + Y_{yy}(x_i,y_j)), \quad (7)$$

где D_1 и D_2 определяют скорость диффузии реагентов X и Y; $X_{xx} = \partial^2 X/\partial x^2$ и $X_{yy} = \partial^2 X/\partial y^2$ — частные производные по координатам x и y.

Начальные условия для X и Y в каждом узле (x_i, y_j) задавались следующим образом:

$$X(x_i,y_j) = X(t_r),\ Y(x_i,y_j) = Y(t_r), \quad (8)$$

где $t_r = \mathrm{rand}(0,t)$ — случайный момент времени на интервале $[0,t]$, а $X(t_r)$ и $Y(t_r)$ — соответствующие этому моменту времени концентрации, полученные решением уравнений реакции (6). Для каждого узла значение t_r определялось заново.

В качестве краевых условий заданы периодические граничные условия:

$$x_{-1} = x_{N-1},\ x_N = x_0,\ y_{-1} = y_{N-1},\ y_N = y_0. \quad (9)$$

Уравнения (7) с начальными (8) и граничными условиями (9) решались методом конечных разностей (правая разность для производной по времени; центральная разность для второй производной по координате) в два этапа.

На первом этапе рассчитывалась диффузионная часть уравнений (7):

$$X_{k,D}(x_i,y_j) = \tau\frac{D_1}{L^2}\left(X_k(x_{i-1},y_j) + X_k(x_{i+1},y_j) + X_k(x_i,y_{j-1}) + X_k(x_i,y_{j+1}) - 4X_k(x_i,y_j)\right),$$
$$Y_{k,D}(x_i,y_j) = \tau\frac{D_2}{L^2}\left(Y_k(x_{i-1},y_j) + Y_k(x_{i+1},y_j) + Y_k(x_i,y_{j-1}) + Y_k(x_i,y_{j+1}) - 4Y_k(x_i,y_j)\right), \quad (10)$$

где L — расстояние между соседними узлами по x и y.

На втором этапе решались уравнения реакции с добавлением ранее рассчитанной диффузии (10):

$$X_{k+1}(x_i,y_j) = X_k(x_i,y_j) + \tau\left(f_1(X_k(x_i,y_j),Y_k(x_i,y_j)) + X_{k,D}(x_i,y_j)\right),$$
$$Y_{k+1}(x_i,y_j) = Y_k(x_i,y_j) + \tau\left(f_2(X_k(x_i,y_j),Y_k(x_i,y_j)) + Y_{k,D}(x_i,y_j)\right). \quad (11)$$

Результаты моделирования показаны на рис. 4. В начале, за счет диффузии происходит локальное выравнивание концентраций. Затем образуются области с четко выраженной структурой. Со временем происходит рост этих структур. После выхода реакции на периодический режим образуются динамические волны концентрации реагентов.

Подготовим проект в *Qt*, который будет показывать анимацию в графическом окне, а также сохранять кадры в заданном графическом формате. Создадим текстовый файл *main.cpp*. Подключаем библиотеки:

```
#include <QtGui/QApplication>
#include <QWidget>
#include <QPainter>
#include <QImage>
#include <QString>
#include <cmath>
#include <iostream>
```

Создаем производный от класса *QWidget* класс окна приложения *iW*:

```
class iW : public QWidget{
public:
 iW(QWidget *parent = 0);
 ~iW();
```

Событие *paintEvent* отвечает за перерисовку графического окна:

```
protected:
 void paintEvent(QPaintEvent *event);
```

Задаем параметры и переменные:

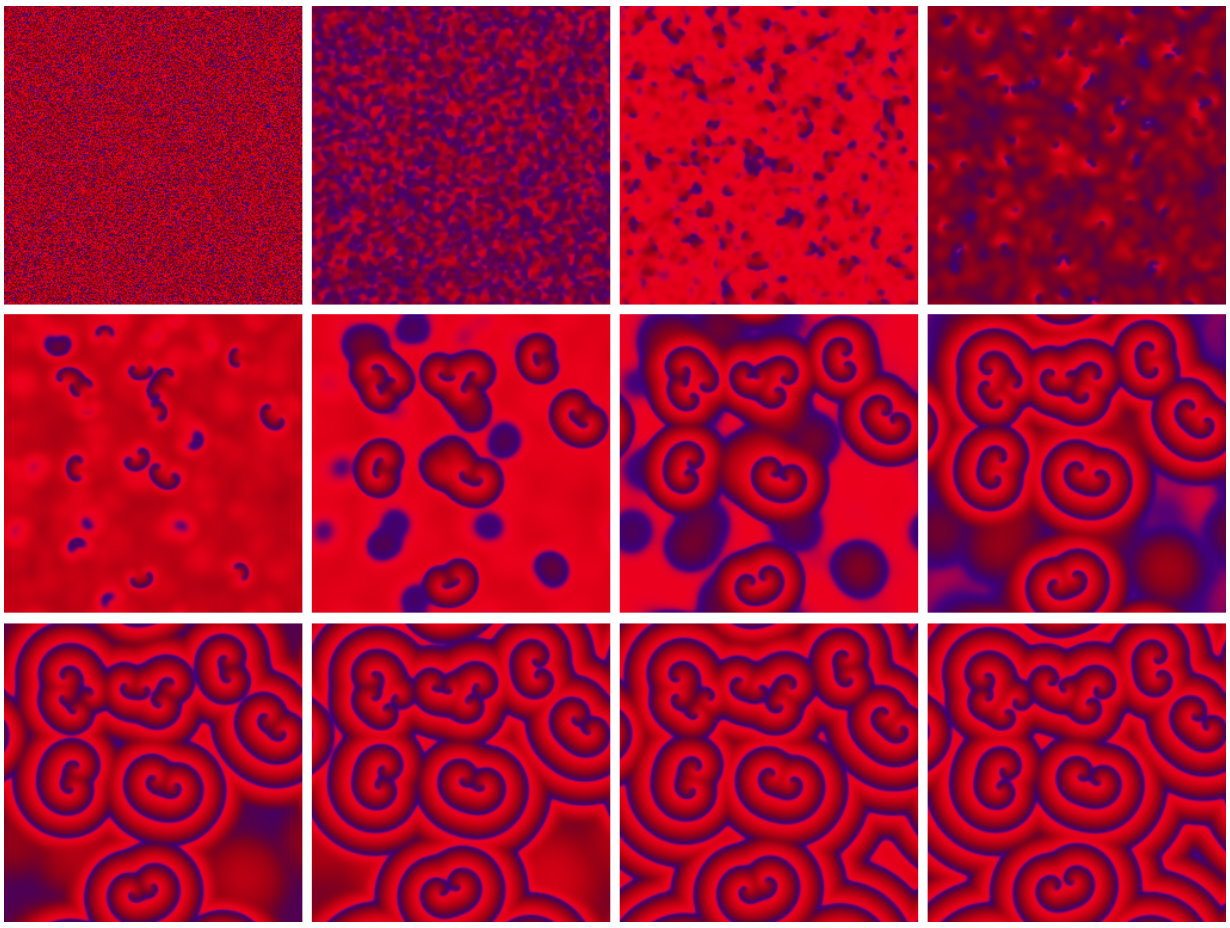

Рис. 4. Распределение концентрации реагентов X (синий) и Y (красный) в модели брюсселятора для кадров $\{1, 5, 10, 20, 50, 100, 150, 200, 250, 300, 400, 500\}$

```
private:
 int N=250,M=5000,P=1000,S1=10,S2=10;
 double A=1,B=3,D1=1,D2=0.05,S=50,dt=0.1;
 double** X;
 double** Y;
 double** XD;
 double** YD;
```

Создаем объект класса *QImage*, отвечающий за инструмент рисования *Qt*:

```
 QImage *image;
```

Определяем функции *fx*, *fy* для вычисления реакционной части уравнений (5), функции вычисления полной системы уравнений в текущий момент времени *solve* и полного цикла анимации *animate*:

```
 double fx(double x,double y)
   {return A-(B+1)*x+x*x*y;};
 double fy(double x,double y)
   {return B*x-x*x*y;};
 void solve();
 void animate();
};
```

Поскольку для переменных X, Y, X_D, Y_D используется динамическое выделение памяти, опишем деструктор класса:

```
iW::~iW(){
 for(int i=0; i<N; ++i) {
  delete [] X[i]; delete [] XD[i];
  delete [] Y[i]; delete [] YD[i];
 }
 delete [] X; delete [] XD;
 delete [] Y; delete [] YD;
}
```

В конструкторе выделяем память для динамических переменных:

```
iW::iW(QWidget *parent) : QWidget(parent) {
 X = new double*[N]; X0 = new double*[N];
 Y = new double*[N]; Y0 = new double*[N];
 for(int i=0; i<N; ++i) {
  X[i]=new double[N]; X0[i]=new double[N];
  Y[i]=new double[N]; Y0[i]=new double[N];
 }
```

Задаем начальное пространственное распределение концентраций реагентов (9):

```
 double xx[P],yy[P]; xx[0]=0.5; yy[0]=2.0;
 for (int i = 0; i < P-1; i++){
  xx[i+1] = xx[i] + 0.01*fx(xx[i],yy[i]);
  yy[i+1] = yy[i] + 0.01*fy(xx[i],yy[i]);
 };
 int c;
 for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++){
   c=rand()%P; X[i][j]=xx[c]; Y[i][j]=yy[c];
  };
```

Задаем размеры изображения, создаваемого объектом класса *QImage*, и размеры главного окна *setFixedSize*:

```
 image = new QImage(N,N,QImage::Format_RGB32);
 setBackgroundRole(QPalette::Base);
 setAutoFillBackground(true);
 setFixedSize(N,N);
 show();
 animate();
 this->setAttribute(Qt::WA_DeleteOnClose);
}
```

Функция *show* отображает главное окно. Эту функцию нужно вызывать до начала рисования результатов моделирования в окне приложения, иначе окно станет видимым только после отработки всего цикла моделирования. Параметр *Qt::WA_DeleteOnClose* означает, что при закрытии окна будет запущен деструктор класса.

Каждый раз, когда возникает событие *paintEvent*, окно будет перерисовываться с учетом текущего распределения концентраций реагентов X (доля синего) и Y (доля красного) с масштабным множителем S:

```
void iW::paintEvent(QPaintEvent *) {
 QPainter painter(this);
 for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++) {
   painter.setPen(
    QColor(Y[i][j]*S,0,X[i][j]*S));
   painter.drawPoint(i,j);
  }
}
```

В цикле по времени вызываем функцию *solve* и отображаем полученные результаты следующим образом: каждый $S2$ шаг по времени вызываем функцию repaint, что создает событие paintEvent; каждый $S1$ шаг по времени создаем изображение *QImage* с результатом моделирования для текущего шага по времени и сохраняем в файл формата *PNG* в папку *data*:

```
void iW::animate() {
 QString str;
 for (int f = 0; f < M; f++) {
 solve();
 if (f%S2 == 0) repaint();
 if (f%S1 == 0) {
  for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
    image->setPixel(i,j,
     qRgb(Y[i][j]*S,0,X[i][j]*S));
   str.sprintf("./data/%04d.png", f/S1);
   image->save(str, "PNG", 100);
  }
 }
}
```

В функции *solve* сначала вычисляем диффузионную часть уравнений (10):

```
void iW::solve() {
 for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++) {
   int ip=(i==N-1)?0:i+1, jp=(j==N-1)?0:j+1;
   int im=(i==0)?N-1:i-1, jm=(j==0)?N-1:j-1;
   X0[i][j]=D1*(X[ip][j]+X[im][j]+X[i][jp]
    +X[i][jm]-4.0 * X[i][j]);
   Y0[i][j]=D2*(Y[ip][j]+Y[im][j]+Y[i][jp]
    +Y[i][jm]-4.0 * Y[i][j]);
  }
```

Затем реакционную часть с добавлением диффузии (11):

```
 for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++) {
   X[i][j]+=(fx(X[i][j],Y[i][j])+X0[i][j])*dt;
   Y[i][j]+=(fy(X[i][j],Y[i][j])+Y0[i][j])*dt;
  }
}
```

Основная функция создает главное окно приложения в виде объекта класса *iW*:

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    iW *w = new iW;
    w->show();
    return app.exec();
}
```

Для компиляции программы создаем файл *brusselator.pro*:

```
TEMPLATE = app
SOURCES += main.cpp
```

Создаем *Makefile*, содержащий инструкции для компиляции программы в консоли (запускается один раз):

```
qmake
```

Компиляция проводится командой (запускается после каждого изменения исходного кода программы)

```
make
```

Сборка видеофайла осуществляется скриптом

```
cd data
mencoder mf://*.png -mf fps=25:type=png -ovc\
 lavc -lavcopts vcodec=mpeg4:vbitrate=5000000\
 -o brusselator.avi
mv brusselator.avi ..
```

5. Трехмерная анимация в Blender

Рассмотрим следующую динамическую задачу: груз массой m подвешен в поле силы тяжести с ускорением свободного падения g на невесомом растяжимом стержне с коэффициентом упругости k и длиной ℓ_0. При $k \to \infty$ система представляет собой математический маятник с собственной частотой колебаний $f_g = \sqrt{g/\ell_0}$. При конечном значении k в поле силы тяжести система представляет собой пружинный маятник с частотой колебаний в вертикальном направлении $f_k = \sqrt{k/m}$. При отклонении стержня в вертикальной плоскости на угол α движение груза будет представлять суперпозицию колебаний угла отклонения $\alpha(t)$ и длины стержня $\ell_0 + \ell(t)$, где t — время; ℓ — изменение первоначальной длины стержня ℓ_0.

При математическом описании данного динамического процесса выберем в качестве обобщенных координат угол $\alpha(t)$ и удлинение стержня $\ell(t)$. Лагранжиан $\mathcal{L}$ механической системы представляет собой разность кинетической и потенциальной энергии:

$$\mathcal{L} = \frac{1}{2} m \left(\dot{\ell}^2 + (\ell_0 + \ell)^2 \dot{\alpha}^2 \right) - \\ - \frac{1}{2} k \ell^2 + mg(\ell_0 + \ell) \cos\alpha, \tag{12}$$

где $\dot{x}$ обозначает частную производную по времени. Уравнения движения представляют собой систему уравнений Эйлера–Лагранжа [81]:

$$\frac{\partial \mathcal{L}}{\partial \ell} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{\ell}} = 0, \quad \frac{\partial \mathcal{L}}{\partial \alpha} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{\alpha}} = 0. \tag{13}$$

Подставляя (12) в (13) получим уравнения движения:

$$\ddot{\ell} = (\ell_0 + \ell)\dot{\alpha}^2 - f_k^2 \ell + g\cos\alpha, \\ \ddot{\alpha} = -\frac{g}{\ell_0 + \ell} \sin\alpha - \frac{2\dot{\ell}}{\ell_0 + \ell} \dot{\alpha}, \tag{14}$$

где $\ddot{x}$ означает вторую частную производную по времени.

Начальные условия задают величины начального угла отклонения и угловой скорости, начальное удлинение стержня и скорости растяжения или сжатия:

$$\alpha(0) = \alpha_0, \ \dot{\alpha}(0) = \dot{\alpha}_0, \ \ell(0) = \ell_1, \ \dot{\ell}(0) = \dot{\ell}_1. \tag{15}$$

Дискретизация уравнений (15) по времени выполнялась по схеме Эйлера первого порядка с введением новых переменных для снижения порядка дифференциальных уравнений

$$y_i = \dot{x}_i = \frac{x_i - x_{i-1}}{\tau}, \ \ddot{x}_i = \dot{y}_i,$$

где индексы $i-1$ и i означают предыдущий и текущий момент времени, а τ — шаг по времени.

Моделирование уравнений (14) с граничными условиями (15) проведем на языке программирования *Python* с визуализацией динамики процесса в программе трехмерного моделирования *Blender*. Результаты моделирования показаны на рис. 5.

Приведем код *Python*-скрипта с комментариями. Вначале подключаем библиотеки для работы с объектами *Blender* и математические функции:

```
import bpy
from math import pi, sin, cos
from mathutils import Vector
```

Во время отладки приходится запускать скрипт несколько раз, что приводит к созданию объектов–клонов, засоряющих рабочее пространство. Этого можно избежать, если поместить в начале скрипта код, удаляющий все объекты заданного типа, т.е. очищающий рабочее пространство от результатов предыдущего запуска:

```
for o in scene.objects:
 if o.type == 'MESH':
  o.select = True
 else:
  o.select = False
bpy.ops.object.delete()
```

Описываем переменные и задаем начальные условия (15) с $\alpha_0 = \pi/6$, $\dot{\alpha}_0 = 0$, $\ell_1 = 0$, $\dot{\ell}_1 = 0$, $\tau = 0.01$:

```
g = 9.8
w = 100.0
l0 = 1
l = 0
lp = l
l1 = 0
l1p = l1
a = pi/6.0
ap = a
a1 = 0
a1p = a1
r0 = 0.01
r1 = 0.005
r2 = 0.01
dt = 0.01
```

Определяем продолжительность процесса в кадрах:

```
st = 1000
scene = bpy.context.scene
scene.frame_start = 0
scene.frame_end = st
```

Строим модель маятника, состоящего из сферы (точка подвеса), стержня и сферы, обозначающей груз. Все три элемента объединяем в один объект. Порядок создания объектов важен, поскольку

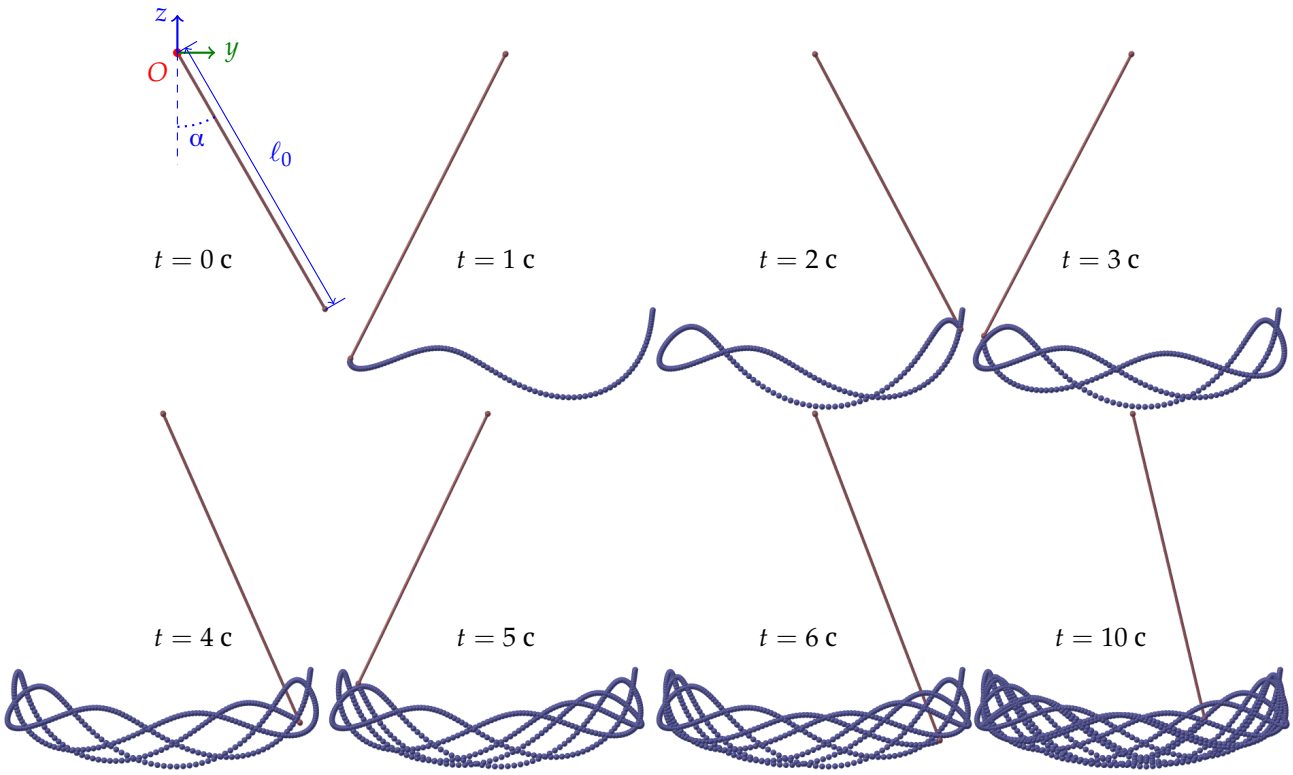

Рис. 5. Траектория движения груза и конфигурация маятника в различные моменты времени

начало отсчета в объекте определяется геометрическим центром последнего созданного объекта. Начальная конфигурация объекта устанавливается поворотом относительно оси Ox на угол α_0. Колебания маятника будут проходить в плоскости Oyz. Маятнику задаем материал красного цвета:

```
m = bpy.ops.mesh
c0 = Vector((0,0,0))
c1 = Vector((0,0,-l0/2))
c2 = Vector((0,0,-l0))

m.primitive_cylinder_add(radius=r1, depth=l0,\
location=c1)
cyl = bpy.context.object
m.primitive_uv_sphere_add(size=r2,location=c2)
s2 = bpy.context.object
m.primitive_uv_sphere_add(size=r0,location=c0)
s0 = bpy.context.object

cyl.select=True
s2.select=True
s0.select=True
bpy.ops.object.join()
p = bpy.context.object
p.rotation_euler.rotate_axis("X", a)
red = bpy.data.materials.new('Red')
red.diffuse_color = (0.6, 0.2, 0.2)
red.specular_hardness = 200
p.data.materials.append(red)
```

Для того чтобы отображать положения груза создаем пустой массив сфер синего цвета (при определении координат груза в каждый момент времени будем добавлять сферу как новый элемент массива):

```
ball = []
blue = bpy.data.materials.new('Blue')
blue.diffuse_color = (0.2, 0.2, 0.6)
blue.specular_hardness = 200
```

В цикле по времени (реализуемом как итерации $i = (0, st - 1)$ по кадрам анимации) решаем уравнения (14):

```
for i in range (0, st):
 scene.frame_set(i)
 ap = a
 a1p = a1
 lp = l
 l1p = l1
 ll = l0+lp
 l1 = l1p + dt * (ll*l1p*l1p-w*lp+g*cos(ap))
 l = lp + l1*dt
 print(l)
 a1 = a1p - dt * (g*sin(ap)-2.0*l1p*a1p)/ll
 a = ap + a1*dt
```

Поворачиваем маятник на величину изменения угла α за текущий шаг по времени и растягиваем стержень на величину $1 + \ell/\ell_0$:

```
 p.rotation_euler.rotate_axis("X", a1*dt)
 p.scale[2]=1+l/l0
```

Отметим, что в данном коде реализуется вращение объекта в связанной с ним локальной системе координат. Данный вариант является «безопасным» способом поворота в отличие от вращения объекта относительно глобальной системы координат, где поворот на эйлеровы углы может привести к блокировке оси (если при повороте на угол ось объекта совпадает с глобальной осью, то они становятся неразличимыми, т.е. последующий поворот одной оси ведет к аналогичному повороту другой; данный эффект называется Gimbal Lock или шарнирный клин).

Координаты груза определяются формулами

$$y = (\ell_0 + \ell)\sin\alpha,\ z = -(\ell_0 + \ell)\cos\alpha.$$

Создаем сферу с центром в данных координатах и делаем ее невидимой; включаем видимость сферы, созданной в предыдущий момент времени:

```
yz = Vector((0,ll*sin(a),-ll*cos(a)))
m.primitive_uv_sphere_add(size=r0,location=yz)
ball.append(bpy.context.object)
ball[i].data.materials.append(blue)
ball[i].hide = True;
if i>0:
 ball[i-1].hide = False;
```

Поясним необходимость управления видимостью сфер. При выполнении скрипта *Blender* создает новые или трансформирует существующие объекты и связывает их параметры с кадрами анимации. После выполнения скрипта созданные объекты становятся доступны на всех кадрах анимации. В процессе анимации путем управления видимостью сфер достигается их последовательное отображение.

Включаем в кадр поворот маятника, его перемещение (удлинение) и статус видимости сфер, отображающих положение груза в предыдущие моменты времени:

```
p.keyframe_insert(data_path='rotation_euler')
p.keyframe_insert(data_path='scale')
ball[i-1].keyframe_insert(data_path='hide')
ball[i].keyframe_insert(data_path='hide')
```

При моделировании динамического процесса можно повысить эффективность использования вычислительных и временных ресурсов с помощью разделения вычислительного этапа и этапа обработки и визуализации результатов моделирования. Для этого используются различные программные средства, при выборе которых учитываются следующие особенности:

- для вычислительного модуля используются быстрые программы-компиляторы с оптимизированными под быстродействие библиотеками (например, *C++*, *Fortran*, *BLAS/Lapack*), в которых инструменты визуализации отсутствуют или могут быть ограничены по функциональности;
- программы визуализации обычно используют более медленные интерпретаторы (обычно, *Python*) и обладают широким набором инструментов, оптимизированных для работы с графикой (что компенсирует скорость интерпретации кода для работы с графикой, но при интерпретации численных алгоритмов быстродействие может значительно снижаться);
- при моделировании с малым шагом по времени получаются избыточные для визуализации данные, поэтому сохраняются результаты только в отдельные моменты времени (например, для создания 24 кадров в секунду);
- программы трехмерного моделирования используют большие объемы памяти и процессорного времени (особенно во время рендеринга, т.е. создания плоского изображения трехмерной модели), поэтому их оптимально использовать для визуализации заранее обработанных данных;
- для повышения скорости расчетов в вычислительном модуле могут использоваться технологии параллельного программирования [82], позволяющие запускать на процессорах с общей (*OpenMP* для языков программирования *Fortran 77/90*, *C* и *C++*) или распределенной (*MPI* для языков программирования *Fortran 77/90*, *Java*, *C* и *C++*) памятью одновременно несколько потоков. Для распараллеливания вычислений на графическом процессоре используются специальные библиотеки, ориентированные на графические карты (например, библиотека *CUDA* для графических процессоров *Nvidia* доступная для языков программирования *C*, *C++*, *Fortran*, *Python* и *MATLAB*).

Приведем пример раздельной реализации вычислительного эксперимента на примере моделирования траектории движения модулей колесного робота [65]. Моделирование проводилось программой, написанной на языке программирования *C++*, которая сохраняла каждую 2000-ю точку траекторий в файл *trajectoryN*, где N — номер модуля ($N = (0,9)$) в следующем формате: время, координаты геометрического центра, координаты крайнего контакта правого и левого колес с поверхностью.

Визуализация проводилась в программе *Blender* с помощью *Python*-скрипта (рис. 6). Для корректной работы скрипта рекомендуется запускать *Blender* из консоли, открытой в той же папке,

где расположены файлы с исходными данными. В консоль будут выводиться сообщения об ошибках (при их наличии) и результаты выполнения команды *print*.

Подключаем основную библиотеку *Blender*:

```
import bpy
```

На время отладки удаляем лишние объекты:

```
for o in bpy.context.scene.objects:
 o.select = False
 if o.type == 'MESH':
  o.select = True
 if o.type == 'CURVE':
  o.select = True
bpy.ops.object.delete()
```

В начале определим функцию рисования кривой cj со следующими аргументами: имя создаваемого объекта, его цвет, индекс модуля робота и позиция первой координаты в файле данных (номер колонки). Создаем трехмерную кривую шириной *bevel_depth*:

```
def MakeLine(on, c, j, s):
 cd=bpy.data.curves.new(name=on,type='CURVE')
 cd.dimensions = '3D'
 cd.bevel_depth = 0.01
```

Создаем объект *Blender* и присваиваем ему цвет по следующему принципу: для первого модуля черный цвет ($j = 0$), а для остальных — цвет, передаваемый в функцию:

```
 od = bpy.data.objects.new(on, cd)
 if ( j ):
  od.data.materials.append(c)
 else:
  od.data.materials.append(black)
 bpy.context.scene.objects.link(od)
```

Определяем тип кривой *POLY* для кусочно-линейной интерполяции. Для сглаживания кривой можно использовать кривые Безье *BEZIER* [83] или B-сплайн *NURBS* [84]. Задаем координаты в формате 4-х мерного вектора (x, y, z, w), где w — весовой коэффициент, используемый при сглаживании кривой:

```
 pl = cd.splines.new('POLY')
 pl.points.add(len(d[j])-1)
 for i in range(len(d[j])):
  pl.points[i].co = (d[j][i][s],\
   d[j][i][1+s], d[j][i][2+s], 1)
 pl.order_u = len(pl.points)-1
 pl.use_endpoint_u = True
```

Для разделения групп кривых определим следующие цвета: черный (траектория первого модуля), красный (траектория геометрического центра), синий и зеленый (крайние точки контакта правого и левого колес с поверхностью):

```
red = bpy.data.materials.new('Red')
red.diffuse_color = (1,0,0)
black = bpy.data.materials.new('Black')
black.diffuse_color = (0,0,0)
blue = bpy.data.materials.new('Blue')
blue.diffuse_color = (0,0,1)
green = bpy.data.materials.new('Green')
green.diffuse_color = (0,1,0)
```

Зададим количество модулей и загрузим координаты из файла данных:

```
n = 10
d = []
for i in range(n):
 with open('./trajectory'+str(i)) as f:
  d.append( [[float(x) for x in line.split()] \
   for line in f] )
```

Отобразим траектории для каждого из модулей:

```
for j in range(n):
 MakeLine('mC'+str(j), red, j, 1)
 MakeLine('mCR'+str(j), blue, j, 6)
 MakeLine('mCL'+str(j), green, j, 9)
```

Отметим, что на проведение моделирования полностью в среде *Blender* с достаточной точностью потребовалось 55 с. После разделения вычислительная часть на *C++* с записью координат траекторий движения модулей не превышала 1 с, а визуализация этих траекторий в *Blender* по данным, полученным из файлов, — 7 с. Экономия в использовании оперативной памяти также была значительной: скомпилированная программа на *C++* использовала 144 КБ ОЗУ; программа *Blender* — 72 МБ, полный вычислительный скрипт — 120 МБ, а скрипт визуализации — 20 МБ.

В данной задаче можно оценить затраты процессорного времени вычислительной части и визуализации при изменении параметров модели. При увеличении количества ведомых модулей или длины траектории ведущего модуля в k раз время расчета при неизменном шаге по времени также возрастет в k раз. При этом время визуализации увеличится более, чем в k раз, поскольку при росте размерности модели или количества создаваемых кадров возрастают затраты на выделение оперативной памяти. Для больших задач может потребоваться разделение рассчитанной траектории на несколько частей, создание для каждой части отдельной анимации с экспортом в видеофайл и с последующей сшивкой видеофрагментов. При уменьшении шага по времени в k раз погрешность вычисления траекторий снизится в k^2 раз, а время расчета возрастет в k раз, т.к. используется алгоритм второго порядка по временному шагу. Модель позволяет провести дополнительную оптимизацию быстродействия, используя зависимость точности расчета от кривизны траектории:

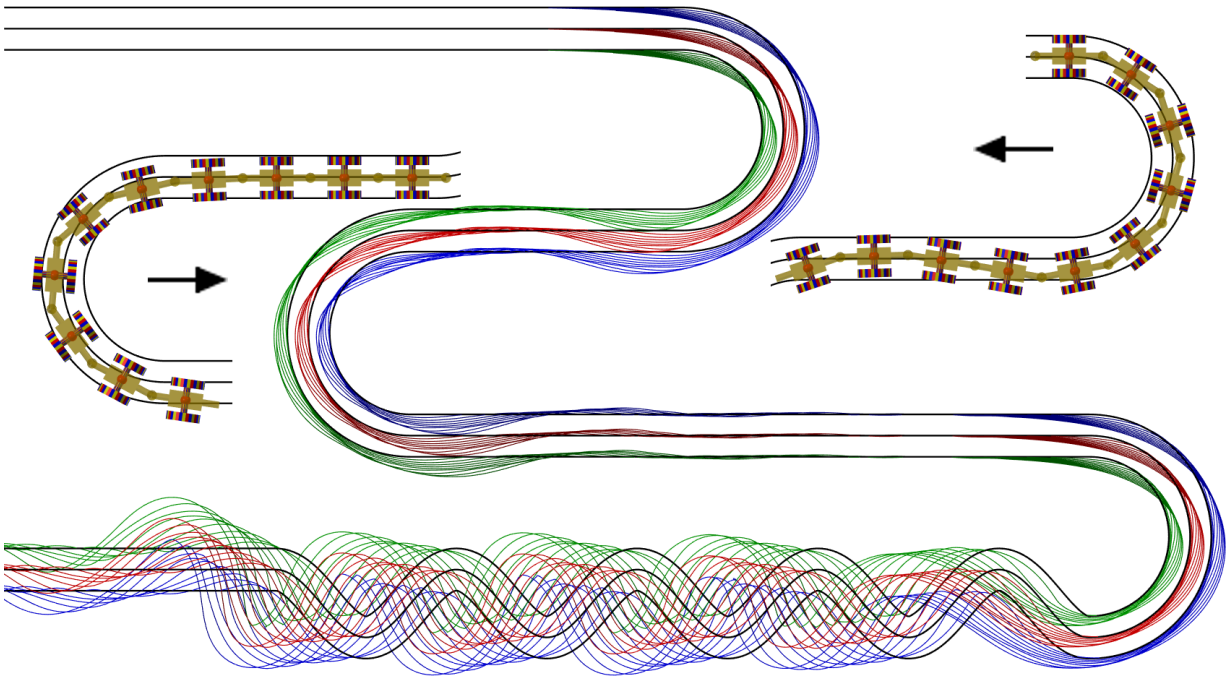

Рис. 6. Траектория движения геометрический центров (красные линии) и крайних точек контакта колес с поверхностью (синие и зеленые линии) модулей колесного робота. Траектория первого модуля обозначена черными линиями

на участках с меньшей кривизной можно использовать больший шаг по времени без потери точности. На время визуализации это не влияет, поскольку количество кадров не меняется. Алгоритм рассчитывает положение ведомых модулей последовательно, от первого до последнего, что не позволяет использовать параллельные вычисления. Однако, при моделировании одновременного движения нескольких модульных роботов вычисление траектории для каждого может быть организовано в виде отдельного потока.

В данный скрипт можно добавить обработку исходных данных и визуализацию объекта исследования (на рис. 6 представлены конфигурации робота на разных участках траектории). Так как траектории ведомых модулей не совпадают с траекторией ведущего модуля, то за одно и то же время модули будут проходить разный путь. Выведем в консоль минимальную и максимальную скорости для каждого модуля:

```
from math import sqrt, pow
a = []
for i in range(n):
 a.append([])
 for j in range(1,len(d[i])):
  a[i].append(sqrt(pow(d[i][j][1]-\
   d[i][j-1][1],2)+pow(d[i][j][2]-\
   d[i][j-1][2],2)+pow(d[i][j][3]-\
   d[i][j-1][3],2))/(d[i][j][0]-d[i][j-1][0]))
 print(i,min(a[i]),max(a[i]))
```

Предложенный скрипт является универсальным инструментом для визуализации трехмерных траекторий и может применяться не только в изучении особенностей кинематики робота, но и в других исследованиях (трехмерные течения жидкости или газа, вязко-пластические деформации тела, процесс переноса тепла, молекулярная динамика, движение небесных тел и т.д.).

Отметим, что *Blender* позволяет сохранять анимацию в виде графических файлов и в видеоформате. Есть два режима (меню Render): Render Animation (сохранение анимации, в области видимости камеры, медленный режим) и OpenGL Render (изображение в том виде, как оно отображено в окне программы, быстрый режим).

6. Заключение

Проведен анализ свободных программных средств для динамической визуализации результатов моделирования нестационарных физических явлений. На примере решения пяти физических задач (модифицированное логистическое уравнение Ферхюльста–Пирла для популяционной динамики; естественная тепловая конвекция Рэлея–Бенара; автоколебательная химическая реакция Белоусова–Жаботинского на примере модели брюсселятора; колебание груза на упругом растяжимом стержне в поле тяжести; кинематика модульного колесного

мобильного робота) показаны способы анимации параметрических аналитических зависимостей в *Gnuplot*, отображения двумерных пространственно распределенных динамических процессов с помощью видеофайла, построения трехмерной модели в программе компьютерной графики *Blender*. Дополнительно представлен способ повышения эффективности использования вычислительных ресурсов с помощью разделения вычислительного этапа и этапа обработки и визуализации результатов моделирования на примере универсального *Python*-скрипт для построения трехмерных траекторий по внешним исходным данным в *Blender*.

Список литературы

[1] Фрик П.Г. Турбулентность: модели и подходы. Курс лекций. Часть I. Пермь, ПГТУ, 1998. 108 с.

[2] Шлихтинг Г. Теория пограничного слоя, 5-е изд, М.: Наука. 1974.

[3] Белоусов Б.П. Периодически действующая реакция и ее механизм. Сб.: Автоволновые процессы в системах с диффузией. Горький: Институт прикладной физики АН СССР. 1981. 287 с.

[4] Базыкин А.Д. Математическая биофизика взаимодействующих популяций. Москва: Наука. 1985. 181 с.

[5] ParaView homepage.
https://www.paraview.org/ (accessed: 01.04.2021)

[6] Ahrens J., Geveci B., Law Ch. ParaView: An End-User Tool for Large Data Visualization. Visualization Handbook. 2005.
DOI: 10.1016/B978-012387582-2/50038-1

[7] Ayachit U. The ParaView Guide: A Parallel Visualization Application. Kitware. 2015.

[8] Jucke M. Scientific Visualisation of Atmospheric Data with ParaView // Journal of Open Research Software. 2014. Vol. 2, No. 1. P. e4.
DOI: 10.5334/jors.al

[9] Thooris B., Pomaréde D. Visualization of Large Scientific Datasets – Analysis of Numerical Simulation Data and Astronomical Surveys Catalogues // In Proceedings of the 6th International Conference on Information Visualization Theory and Applications – IVAPP, (VISIGRAPP 2015). 2015. Pp. 117–122.
DOI: 10.5220/0005300901170122

[10] Woodring J., Heitmann K., Ahrens J., Fasel P., Hsu C.-H., Habib S., Pope A. Analyzing and Visualizing Cosmological Simulations with ParaView // The Astrophysical Journal Supplement Series. 2011. Vol. 195, No. 1.
DOI: 10.1088/0067-0049/195/1/11

[11] OpenFOAM homepage.
https://www.openfoam.com/ (accessed: 01.04.2021)

[12] FreeCAD: Your own 3D parametric modeler.
https://www.freecadweb.org/ (accessed: 01.04.2021)

[13] CalculiX: A Free Software Three-Dimensional Structural Finite Element Program.
http://www.dhondt.de/ (accessed: 01.04.2021)

[14] Salome Platform — The opem source platform for numerical simulation.
https://www.salome-platform.org/ (accessed: 01.04.2021)

[15] Elmer homepage.
https://www.csc.fi/web/elmer (accessed: 01.04.2021)

[16] Welcome to Python.org.
https://www.python.org/ (accessed: 01.04.2021)

[17] Milliken C.P. Python Projects for Beginners. Apress Berkeley, CA. 2020.
DOI: 10.1007/978-1-4842-5355-7

[18] Matplotlib: Visualization with Python.
https://matplotlib.org/ (accessed: 01.04.2021)

[19] Vaingast Sh. Beginning Python Visualization. Crafting Visual Transformation Scripts. Apress Berkeley, CA. 2014. 416 p.
DOI: 10.1007/978-1-4842-0052-0

[20] Rajagopalan G. A Python Data Analyst's Toolkit. Learn Python and Python-based Libraries with Applications in Data Analysis and Statistics. Apress Berkeley, CA. 2021.
DOI: 10.1007/978-1-4842-6399-0

[21] PyQt homepage.
http://www.riverbankcomputing.com/software/pyqt/ (accessed: 01.04.2021)

[22] Willman J. Modern PyQt. Create GUI Applications for Project Management, Computer Vision, and Data Analysis. Apress Berkeley, CA. 2021.
DOI: https://doi.org/10.1007/978-1-4842-6603-8

[23] The GTK Project — a free open-source cross-platform widget toolkit.
https://www.gtk.org/ (accessed: 01.04.2021)

[24] wxWidgets. Cross-platform GUI Library.
https://www.wxwidgets.org/ (accessed: 01.04.2021)

[25] Seaborn: statistical data visualization.
https://seaborn.pydata.org/ (accessed: 01.04.2021)

[26] Plotly Python Open Source Graphing Library.
https://plotly.com/python/ (accessed: 01.04.2021)

[27] Altair: Declarative Visualization in Python.
https://altair-viz.github.io/ (accessed: 01.04.2021)

[28] Unpingco J. Python Programming for Data Analysis. Springer Cham. 2021.
DOI: 10.1007/978-3-030-68952-0

[29] Folium homepage.
https://python-visualization.github.io/folium/ (accessed: 01.04.2021)

[30] Qt. Cross-platform software development for embedded and desktop.
https://www.qt.io/ (accessed: 01.04.2021)

[31] Lazarus homepage.
https://www.lazarus-ide.org/ (accessed: 01.04.2021)

[32] OpenGL — the industry standart for high performance graphics.
https://www.opengl.org/ (accessed: 01.04.2021)

[33] GLUT — the OpenGL utility toolkit.
https://www.opengl.org/resources/libraries/glut/glut_downloads.php (accessed: 01.04.2021)

[34] Simple DirectMedia Layer homepage.
https://www.libsdl.org/ (accessed: 01.04.2021)

[35] VTK — the visualization toolkit.
https://vtk.org/ (accessed: 01.04.2021)

[36] Vernikouskaya I., Bertsche D., Rottbauer W. Rasche V. 3D-XGuide: open-source X-ray navigation guidance system // Int. J. CARS. 2021. Vol. 16. Pp. 53–63.
DOI: 10.1007/s11548-020-02274-0

[37] Zhang J., Tian X., Duan Q. Design and Implementation of Vehicle Terminal Graphic Interface Based on QT // Journal of Physics: Conference Series. 2020. Vol. 1486. Art. 072010.
DOI: 10.1088/1742-6596/1486/7/072010

[38] Silvestrov P., Bessonov O. Development of a visualization module for aerogasdynamic computations // Journal of Physics: Conference Series. 2018. Vol. 1009. P. 012035. DOI: 10.1088/1742-6596/1009/1/012035

[39] Xiao P., Zhao X., Leng S., Tan R.S., Wong P., Zhong L. A Software Tool for Heart AVJ Motion Tracking Using Cine Cardiovascular Magnetic Resonance Images // IEEE Journal of Translational Engineering in Health and Medicine. 2017. Vol. 5. Pp. 1–12. Art. No. 1900412. DOI: 10.1109/JTEHM.2017.2738623

[40] Ritraksa S. Mekchay K. 3D Structural Model and Visualization of Blood Vessels Based on L-System // Trends in Sciences. 2021. Vol. 18, No. 24. P. 1407. DOI: 10.48048/tis.2021.1407

[41] FreeFEM — an open-source PDE solver using finite element method. https://freefem.org/ (accessed: 01.04.2021)

[42] Hecht F. New development in FreeFem++ // Journal of Numerical Mathematics. 2012. V. 20, No. 3–4. Pp. 251–265. DOI: 10.1515/jnum-2012-0013

[43] Насибуллаев И.Ш. Применение свободных программ FreeFem++/Gmsh и FreeCAD/CalculiX для моделирования статических структурных задач // Многофазные системы. 2020. Т. 15, № 3–4. С. 183–200. DOI: 10.21662/mfs2020.3.129

[44] Насибуллаев И.Ш., Насибуллаева Э.Ш., Даринцев О.В. Моделирование течения жидкости через деформируемый пьезоэлементом эластичный микроканал системы охлаждение микрозахвата // Мехатроника, автоматизация, управление. 2019. Т. 20, № 12. С. 740–750. DOI: 10.17587/mau.20.740-750

[45] Nasibullayev I.Sh., Darintsev O.V., Nasibullaeva E.Sh. and Bogdanov D.R. Piezoelectric Micropumps for Microrobotics: Operating Modes Simulating and Analysis of the Main Parameters of the Fluid Flow Generation // In: Ronzhin A. and Shishlakov V. (eds) Proceedings of 15th International Conference on Electromechanics and Robotics "Zavalishin's Readings". Smart Innovation, Systems and Technologies. 2021. V. 187. Pp. 525–536. Springer, Singapore. DOI: 10.1007/978-981-15-5580-0_43

[46] Nasibullayev I.Sh., Nasibullaeva E.Sh., Darintsev O.V. Computer Axisymmetric Model of a Piezoelectric Micropump // Journal of Engineering Science and Technology Review. 2021. V. 14, No. 2. Pp. 152–164. DOI: 10.25103/jestr.142.19

[47] Насибуллаев И.Ш., Даринцев О.В. Двумерная динамическая модель взаимодействия жидкости и пьезоэлектрического привода с поперечным изгибом в плоском канале // Многофазные системы. 2019. Т. 14, № 4. С. 220–232. DOI: 10.21662/mfs2019.4.029

[48] Castrillo P., Schillaci E., Rigola J. Simulation of Fluid-Structure Interaction and Impact Force on a Reed Valve // WCCM-ECCOMAS2020. 2021. DOI: 10.23967/wccm-eccomas.2020.305

[49] Насибуллаев И.Ш., Даринцев О.В. Компьютерное двумерное моделирование системы жидкостного охлаждения микрозахвата // Вычислительные технологии. 2021. Т. 26, № 2. С. 4–20. DOI: 10.25743/ICT.2021.26.2.002

[50] Насибуллаев И.Ш. Аналитический анализ переключения рабочего режима в двумерной модели системы жидкостного охлаждения микрозахвата // Вестник УГАТУ. 2021. Т. 25, № 3(93). С. 120–131. DOI: 10.54708/19926502_2021_25393120

[51] Sadaka G., Rakotondrandisa A., Tournier P.-H., Luddens F., Lothodé C., Danaila I. Parallel finite-element codes for the simulation of two-dimensional and three-dimensional solid–liquid phase-change systems with natural convection // Computer Physics Communications. 2020. Vol. 257. P. 107492. DOI: 10.1016/j.cpc.2020.107492

[52] Sadaka G., Dutykh D. Adaptive Numerical Modeling of Tsunami Wave Generation and Propagation with FreeFem++ // Geosciences. 2020. Vol. 10, No. 9. Art. no. 351. DOI: 10.3390/geosciences10090351

[53] Giuntini S., Andreini A., Cappuccini G., Facchini B. Finite element transient modelling for whole engine-secondary air system thermomechanical analysis // Energy Procedia. 2017. Vol. 126. Pp. 746–753. DOI: 10.1016/j.egypro.2017.08.231

[54] Agnolucci A., Vanti F, Pinelli L., Arnone A. Automatic procedure for aeromechanic analysis of turbomachinery blade-rows // AIP Conference Proceedings. 2019. Vol. 2191. Art. no. 020003. DOI: 10.1063/1.5138736

[55] Karimi A., Grytz R., Rahmati S.M., Girkin C.A., Downs J.C. Analysis of the effects of finite element type within a 3D biomechanical model of a human optic nerve head and posterior pole // Computer Methods and Programs in Biomedicine. 2021. Vol. 198. Art.no. 105794. DOI: 10.1016/j.cmpb.2020.105794

[56] GhostScript homepage. https://www.ghostscript.com/ (accessed: 01.04.2021)

[57] ImageMagick — convert, edit, or compose digital images. https://imagemagick.org/ (accessed: 01.04.2021)

[58] GIMP — GNU image manipulation program. https://www.gimp.org/ (accessed: 01.04.2021)

[59] Насибуллаев И.Ш. Использование свободных программ для обработки и визуализации результатов научных исследований // Многофазные системы. 2021. Т. 16, № 2. С. 58–71. DOI: 10.21662/mfs2021.2.009

[60] MEncoder http://www.mplayerhq.hu/ (accessed: 01.04.2021)

[61] Blender. 3D Content Creation Noob to Pro. Wikibooks. https://upload.wikimedia.org/wikipedia/commons/b/b4/BlenderDocumentation4.pdf (accessed: 01.04.2021)

[62] Bruyninckx H. Blender for robotics and robotics for Blender. 2004. http://download.blender.org/documentation/bc2004/Herman_Bruyninckx/robot-blender.pdf (accessed: 01.04.2021)

[63] Buys K., De Laet T., Smits R., Bruyninckx H. Blender for Robotics: Integration into the Leuven Paradigm for Robot Task Specification and Human Motion Estimation. Pp. 15–25. 2010. DOI: 10.1007/978-3-642-17319-6_5

[64] Díaz-Andrade A., Álvarez-Cedillo J., Herrera-Lozada J., Rivera-Zarate I. Robotic Arm Control with Blender. Journal of Emerging Trends in Computing and Information Sciences. Vol. 4, No. 4. Pp. 382–386. 2013.

[65] Nasibullayev I., Darintsev O., Bogdanov D. In-Pipe Modular Robot: Configuration, Displacement Principles, Standard Patterns and Modeling // In: Ronzhin A., Shishlakov V. (eds) Electromechanics and Robotics. Smart Innovation, Systems and Technologies. Vol. 232, pp. 85–96. Springer, Singapore. 2022. DOI: 10.1007/978-981-16-2814-6_8

[66] ROS homepage. https://www.ros.org/ (accessed: 01.04.2021)

[67] Robot Operating System (ROS). The Complete Reference (Volume 1). Ed. Anis Koubaa. Springer Cham. 2016. DOI: 10.1007/978-3-319-26054-9

[68] Gazebo simulator homepage. https://gazebosim.org/home (accessed: 01.04.2021)

[69] Rudolf A., Stjepanovič Z., Cupar A. Study Regarding the Kinematic 3D Human-Body Model Intended for Simulation of Personalized Clothes for a Sitting Posture // Materials. 2021. Vol. 14, No. 18. Art. No. 5124.
DOI: 10.3390/ma14185124

[70] Simić L., Kopačin V., Mumlek I., et al. Improved technique of personalised surgical guides generation for mandibular free flap reconstruction using an open-source tool // Eur. Radiol. 2021. Exp. Vol. 5. Art. No. 30.
DOI: 10.1186/s41747-021-00229-x

[71] Lind M., Ystgaard P. Mesh-based tool path calculations for tubular joints // Advances in Mechanical Engineering. 2020. Vol. 12, No. 6. Pp.1–13.
DOI: 10.1177/1687814020933383

[72] Bonneau D.A., Hutchinson D.J., DiFrancesco P.-M., Coombs M. Sala Z. Three-dimensional rockfall shape back analysis: methods and implications // Nat. Hazards Earth Syst. Sci. 2019. Vol. 19. Pp. 2745–2765.
DOI: 10.5194/nhess-19-2745-2019

[73] Филиппов С.В. Программная платформа Blender как среда моделирования объектов и процессов естественно-научных дисциплин // Препринты ИПМ им. М.В.Келдыша. 2018. № 230. 42 с.
DOI: 10.20948/prepr-2018-230

[74] Milan J., Lubomír Ř., Petr S., Matěj Š. GPU Accelerated Path Tracing of Massive Scenes // ACM Trans. Graph. 2021. Vol. 40, No. 2. Art. No. 16. Pp. 1–17.
DOI: 10.1145/3447807

[75] Yi T. Making PDFs with Animation and Lecture Video for Beginner. WSEAS Transactions on Advances in Engineering Education. 2017. Vol. 14. Pp. 76–80.
https://www.wseas.org/multimedia/journals/education/2017/a185810-086.php

[76] Гетлинг А.В. Конвекция Рэлея-Бенара. Структуры и динамика. М: Эдиториал УРСС. 247 с.

[77] Гершуни Г.З., Жуховицкий Е.М. Конвективная устойчивость несжимаемой жидкости. М.: Наука. 1972. 392 с.

[78] Белоусов Б.П. Периодически действующая реакция и ее механизм. Сборник рефератов по радиационной медицине за 1958 г. М: Медгиз. 1959.

[79] Glandsdorff P., Prigogine I. Thermodynamic theory of structure, stability and fluctuations. Wiley, New York. 1971.

[80] Batchelor G.K. An Introduction to Fluid Dynamics, Cambridge University Press. 1967.
DOI: 10.1017/CBO9780511800955

[81] Goldstein H., Poole C., Safko J. Classical Mechanics, 3rd ed. American Journal of Physics. 2002. Vol. 70, pp. 782.
DOI: 10.1119/1.1484149

[82] Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А. Садовничий. М.: Издательство Московского университета, 2012. 344 с.

[83] Hoffmann G. Bézier Curves.
https://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf
(accessed: 01.04.2021)

[84] Piegl L., Tiller W. The NURBS Book (2. ed.). Berlin: Springer. 1997.

ISSN 2658–5782 16 (2021), **3–4**, 121–143

Multiphase Systems

http://mfs.uimech.org/mfs2021.3.016
DOI: 10.21662/mfs2021.3.016

Received: 17.06.2021
Accepted: 9.09.2021

Application of free software to visualize the results of simulation of dynamic processes

Nasibullayev I.Sh.

Mavlyutov Institute of Mechanics, UFRC RAS, Ufa, Russia

The paper presents an overview of modern free tools for dynamic visualization and gives recommendations for choosing tools depending on the research method, the form of presentation of the initial data, and the specifics of the phenomenon under study. For the convenience of using the materials of the work, the source codes are given both for the programs of computational experiments of classical problems, and for creating a graphical visualization of the simulation results. To animate the process described by analytical formulas, it is proposed to use the gif terminal of the gnuplot program or the Python visualization library. An example of applying this approach to solving the modified Verhulst-Pearl equation, which describes the change in the population under periodic external influence, is given. When studying non-stationary phenomena distributed in space, the results can be presented in video format. The problem of natural convection in a horizontal layer of fluid or gas was simulated in the program FreeFem++ for solving differential equations by the finite element method with the conversion of the results using the GhostScript and MEncoder programs into a video format. An example of using the finite difference method in modeling self-oscillating chemical reactions in the Qt environment with saving animation frames as graphic files is given. To display the results of modeling three-dimensional dynamic processes, it is proposed to use the Blender computer graphics program. Modeling and visualization of oscillations of an elastic pendulum using the built-in Blender Python API interpreter are presented. An approach is shown for dividing a computational experiment and visualizing its results, which makes it possible to increase the efficiency of the use of computational resources. A universal Python-script is proposed for constructing three-dimensional object motion trajectories based on external source data.

Keywords: free software, Gnuplot, Matplotlib, FreeFem++, Qt, MEncoder, Blender

References

[1] Frick P.G. [Turbulence: models and approaches. Lecture course. Part I] *Turbulentnost': modeli i podkhody. Kurs lektsiy. Chast' I.* Perm, PSTU, 1998. 108 p. (In Russian).

[2] Schlichting G. [Theory of the boundary layer, 5th ed.] *Teoriya pogranichnogo sloya, 5-ye izd.* M.: Nauka. 1974. (In Russian).

[3] Belousov B.P. [Periodically acting reaction and its mechanism. Autowave processes in systems with diffusion] *Periodicheski deystvuyushchaya reaktsiya i yeye mekhanizm.* Sb.: Avtovolnovyye protsessy v sistemakh s diffuziyey. Gorky: Institute of Applied Physics, Academy of Sciences of the USSR. 1981. 287 p. (In Russian).

[4] Bazykin A.D. [Mathematical biophysics of interacting populations] *Matematicheskaya biofizika vzaimodeystvuyushchikh populyatsiy.* Moskva: Nauka. 1985. 181 p. (In Russian).

[5] ParaView homepage. https://www.paraview.org/ (accessed: 01.04.2021)

[6] Ahrens J., Geveci B., Law Ch. ParaView: An End-User Tool for Large Data Visualization. Visualization Handbook. 2005. DOI: 10.1016/B978-012387582-2/50038-1

[7] Ayachit U. The ParaView Guide: A Parallel Visualization Application. Kitware. 2015.

[8] Jucke M. Scientific Visualisation of Atmospheric Data with ParaView. Journal of Open Research Software. 2014. Vol. 2, No. 1. P. e4. DOI: 10.5334/jors.al

[9] Thooris B., Pomaréde D. Visualization of Large Scientific Datasets – Analysis of Numerical Simulation Data and Astronomical Surveys Catalogues. In Proceedings of the 6th International Conference on Information Visualization Theory and Applications – IVAPP, (VISIGRAPP 2015). 2015. Pp. 117–122. DOI: 10.5220/0005300901170122

[10] Woodring J., Heitmann K., Ahrens J., Fasel P., Hsu C.-H., Habib S., Pope A. Analyzing and Visualizing Cosmological Simulations with ParaView. The Astrophysical Journal Supplement Series. 2011. Vol. 195, No. 1. DOI: 10.1088/0067-0049/195/1/11

[11] OpenFOAM homepage. https://www.openfoam.com/ (accessed: 01.04.2021)

[12] FreeCAD: Your own 3D parametric modeler. https://www.freecadweb.org/ (accessed: 01.04.2021)

[13] CalculiX: A Free Software Three-Dimensional Structural Finite Element Program. http://www.dhondt.de/ (accessed: 01.04.2021)

[14] Salome Platform — The opem source platform for numerical simulation.
https://www.salome-platform.org/ (accessed: 01.04.2021)

[15] Elmer homepage.
https://www.csc.fi/web/elmer (accessed: 01.04.2021)

[16] Welcome to Python.org.
https://www.python.org/ (accessed: 01.04.2021)

[17] Milliken C.P. Python Projects for Beginners. Apress Berkeley, CA. 2020.
DOI: 10.1007/978-1-4842-5355-7

[18] Matplotlib: Visualization with Python.
https://matplotlib.org/ (accessed: 01.04.2021)

[19] Vaingast Sh. Beginning Python Visualization. Crafting Visual Transformation Scripts. Apress Berkeley, CA. 2014. 416 p.
DOI: 10.1007/978-1-4842-0052-0

[20] Rajagopalan G. A Python Data Analyst's Toolkit. Learn Python and Python-based Libraries with Applications in Data Analysis and Statistics. Apress Berkeley, CA. 2021.
DOI: 10.1007/978-1-4842-6399-0

[21] PyQt homepage.
http://www.riverbankcomputing.com/software/pyqt/ (accessed: 01.04.2021)

[22] Willman J. Modern PyQt. Create GUI Applications for Project Management, Computer Vision, and Data Analysis. Apress Berkeley, CA. 2021.
DOI: https://doi.org/10.1007/978-1-4842-6603-8

[23] The GTK Project — a free open-source cross-platform widget toolkit.
https://www.gtk.org/ (accessed: 01.04.2021)

[24] wxWidgets. Cross-platform GUI Library.
https://www.wxwidgets.org/ (accessed: 01.04.2021)

[25] Seaborn: statistical data visualization.
https://seaborn.pydata.org/ (accessed: 01.04.2021)

[26] Plotly Python Open Source Graphing Library.
https://plotly.com/python/ (accessed: 01.04.2021)

[27] Altair: Declarative Visualization in Python.
https://altair-viz.github.io/ (accessed: 01.04.2021)

[28] Unpingco J. Python Programming for Data Analysis. Springer Cham. 2021.
DOI: 10.1007/978-3-030-68952-0

[29] Folium homepage.
https://python-visualization.github.io/folium/ (accessed: 01.04.2021)

[30] Qt. Cross-platform software development for embedded and desktop.
https://www.qt.io/ (accessed: 01.04.2021)

[31] Lazarus homepage.
https://www.lazarus-ide.org/ (accessed: 01.04.2021)

[32] OpenGL — the industry standart for high performance graphics.
https://www.opengl.org/ (accessed: 01.04.2021)

[33] GLUT — the OpenGL utility toolkit.
https://www.opengl.org/resources/libraries/glut/glut_downloads.php (accessed: 01.04.2021)

[34] Simple DirectMedia Layer homepage.
https://www.libsdl.org/ (accessed: 01.04.2021)

[35] VTK — the visualization toolkit.
https://vtk.org/ (accessed: 01.04.2021)

[36] Vernikouskaya I., Bertsche D., Rottbauer W. Rasche V. 3D-XGuide: open-source X-ray navigation guidance system. Int. J. CARS. 2021. Vol. 16. Pp. 53–63.
DOI: 10.1007/s11548-020-02274-0

[37] Zhang J., Tian X., Duan Q. Design and Implementation of Vehicle Terminal Graphic Interface Based on QT. Journal of Physics: Conference Series. 2020. Vol. 1486. Art. 072010.
DOI: 10.1088/1742-6596/1486/7/072010

[38] Silvestrov P., Bessonov O. Development of a visualization module for aerogasdynamic computations. Journal of Physics: Conference Series. 2018. Vol. 1009. P. 012035.
DOI: 10.1088/1742-6596/1009/1/012035

[39] Xiao P., Zhao X., Leng S., Tan R.S., Wong P., Zhong L. A Software Tool for Heart AVJ Motion Tracking Using Cine Cardiovascular Magnetic Resonance Images. IEEE Journal of Translational Engineering in Health and Medicine. 2017. Vol. 5. Pp. 1–12. Art. No. 1900412.
DOI: 10.1109/JTEHM.2017.2738623

[40] Ritraksa S. Mekchay K. 3D Structural Model and Visualization of Blood Vessels Based on L-System. Trends in Sciences. 2021. Vol. 18, No. 24. P. 1407.
DOI: 10.48048/tis.2021.1407

[41] FreeFEM — an open-source PDE solver using finite element method.
https://freefem.org/ (accessed: 01.04.2021)

[42] Hecht F. New development in FreeFem++. Journal of Numerical Mathematics. 2012. V. 20, No. 3–4. Pp. 251–265.
DOI: 10.1515/jnum-2012-0013

[43] Nasibullayev I.Sh. [Application of free software FreeFem++/Gmsh and FreeCAD/CalculiX for simulation of static elasticity problems] *Primeneniye svobodnykh programm FreeFem++/Gmsh i FreeCAD/CalculiX dlya modelirovaniya staticheskikh strukturnykh zadach*. Multiphase Systems [*Mnogofaznyye sistemy*]. 2020. V. 15, No. 3–4. Pp. 183–200 (In Russian).
DOI: 10.21662/mfs2020.3.129

[44] Nasibullayev I.Sh., Nasibullaeva E.Sh., Darintsev O.V., [Simulation of fluid flow through a elastic microchannel deformed by a piezoelement in microgrip cooling systems] *Modelirovaniye techeniya zhidkosti cherez deformiruyemyy p'yezoelementom elastichnyy mikrokanal sistemy okhlazhdeniye mikrozakhvata*. Mechatronics, automation, control [*Mekhatronika, Avtomatizatsiya, Upravlenie*]. 2019. V. 20, No. 12. Pp. 740–750 (In Russian).
DOI: 10.17587/mau.20.740-750

[45] Nasibullayev I.Sh., Darintsev O.V., Nasibullaeva E.Sh. and Bogdanov D.R. Piezoelectric Micropumps for Microrobotics: Operating Modes Simulating and Analysis of the Main Parameters of the Fluid Flow Generation // In: Ronzhin A. and Shishlakov V. (eds) Proceedings of 15th International Conference on Electromechanics and Robotics "Zavalishin's Readings". Smart Innovation, Systems and Technologies. 2021. V. 187. Pp. 525–536. Springer, Singapore.
DOI: 10.1007/978-981-15-5580-0_43

[46] Nasibullayev I.Sh., Nasibullaeva E.Sh., Darintsev O.V. Computer Axisymmetric Model of a Piezoelectric Micropump // Journal of Engineering Science and Technology Review. 2021. V. 14, No. 2. Pp. 152–164.
DOI: 10.25103/jestr.142.19

[47] Nasibullayev I.Sh., Darintsev O.V., [Two-dimensional dynamic model of the interaction of a fluid and a piezoelectric bending actuator in a plane channel] *Dvumernaya dinamicheskaya model' vzaimodeystviya zhidkosti i p'yezoelektricheskogo privoda s poperechnym izgibom v ploskom kanale*. Multiphase Systems [*Mnogofaznyye sistemy*]. 2019. V. 14, No. 4. Pp. 220–232 (in Russian).
DOI: 10.21662/mfs2019.4.029

[48] Castrillo P., Schillaci E., Rigola J. Simulation of Fluid-Structure Interaction and Impact Force on a Reed Valve. WCCM-ECCOMAS2020. 2021.
DOI: 10.23967/wccm-eccomas.2020.305

[49] Nasibullayev I.Sh., Darintsev O.V. [Computer 2D modelling of a micro-grip fluid cooling system] *Komp'yuternoye dvumernoye modelirovaniye sistemy zhidkostnogo okhlazhdeniya mikrozakhvata*. Computational technologies [*Vychislitel'nyye tekhnologii*]. 2021. V. 26. No. 2. Pp. 4–20 (in Russian).
DOI: 10.25743/ICT.2021.26.2.002

[50] Nasibullayev I.Sh. [Analytical analysis of operating mode switching in a 2D model of a fluid cooling system of the microgripper] *Analiticheskiy analiz pereklyucheniya rabochego rezhima v dvumernoy modeli sistemy zhidkostnogo okhlazhdeniya mikrozakhvata*. Vestnik USATU [*Vestnik UGATU*]. 2021. Vol. 25, N. 3 (93). Pp. 120–131. (in Russian).
DOI: 10.54708/19926502_2021_25393120

[51] Sadaka G., Rakotondrandisa A., Tournier P.-H., Luddens F., Lothodé C., Danaila I. Parallel finite-element codes for the simulation of two-dimensional and three-dimensional solid–liquid phase-change systems with natural convection. Computer Physics Communications. 2020. Vol. 257. P. 107492. DOI: 10.1016/j.cpc.2020.107492

[52] Sadaka G., Dutykh D. Adaptive Numerical Modeling of Tsunami Wave Generation and Propagation with FreeFem++. Geosciences. 2020. Vol. 10, No. 9. Art. no. 351. DOI: 10.3390/geosciences10090351

[53] Giuntini S., Andreini A., Cappuccini G., Facchini B. Finite element transient modelling for whole engine-secondary air system thermomechanical analysis // Energy Procedia. 2017. Vol. 126. Pp. 746–753. DOI: 10.1016/j.egypro.2017.08.231

[54] Agnolucci A., Vanti F, Pinelli L., Arnone A. Automatic procedure for aeromechanic analysis of turbomachinery blade-rows. AIP Conference Proceedings. 2019. Vol. 2191. Art. no. 020003. DOI: 10.1063/1.5138736

[55] Karimi A., Grytz R., Rahmati S.M., Girkin C.A., Downs J.C. Analysis of the effects of finite element type within a 3D biomechanical model of a human optic nerve head and posterior pole. Computer Methods and Programs in Biomedicine. 2021. Vol. 198. Art.no. 105794. DOI: 10.1016/j.cmpb.2020.105794

[56] GhostScript homepage. https://www.ghostscript.com/ (accessed: 01.04.2021)

[57] ImageMagick — convert, edit, or compose digital images. https://imagemagick.org/ (accessed: 01.04.2021)

[58] GIMP — GNU image manipulation program. https://www.gimp.org/ (accessed: 01.04.2021)

[59] Nasibullayev I.Sh. [Application of free software for processing and visualization of scientific research results] *Ispol'zovaniye svobodnykh programm dlya obrabotki i vizualizatsii rezul'tatov nauchnykh issledovaniy*. Multiphase Systems [*Mnogofaznyye sistemy*]. 2021. V. 16, No 2. Pp. 58–71 (In Russian). DOI: 10.21662/mfs2021.2.009

[60] MEncoder http://www.mplayerhq.hu/ (accessed: 01.04.2021)

[61] Blender. 3D Content Creation Noob to Pro. Wikibooks. https://upload.wikimedia.org/wikipedia/commons/b/b4/BlenderDocumentation4.pdf (accessed: 01.04.2021)

[62] Bruyninckx H. Blender for robotics and robotics for Blender. 2004. http://download.blender.org/documentation/bc2004/Herman_Bruyninckx/robot-blender.pdf (accessed: 01.04.2021)

[63] Buys K., De Laet T., Smits R., Bruyninckx H. Blender for Robotics: Integration into the Leuven Paradigm for Robot Task Specification and Human Motion Estimation. Pp. 15–25. 2010. DOI: 10.1007/978-3-642-17319-6_5

[64] Díaz-Andrade A., Álvarez-Cedillo J., Herrera-Lozada J., Rivera-Zarate I. Robotic Arm Control with Blender. Journal of Emerging Trends in Computing and Information Sciences. Vol. 4, No. 4. Pp. 382–386. 2013.

[65] Nasibullayev I., Darintsev O., Bogdanov D. In-Pipe Modular Robot: Configuration, Displacement Principles, Standard Patterns and Modeling // In: Ronzhin A., Shishlakov V. (eds) Electromechanics and Robotics. Smart Innovation, Systems and Technologies. Vol. 232, pp. 85–96. Springer, Singapore. 2022. DOI: 10.1007/978-981-16-2814-6_8

[66] ROS homepage. https://www.ros.org/ (accessed: 01.04.2021)

[67] Robot Operating System (ROS). The Complete Reference (Volume 1). Ed. Anis Koubaa. Springer Cham. 2016. DOI: 10.1007/978-3-319-26054-9

[68] Gazebo simulator homepage. https://gazebosim.org/home (accessed: 01.04.2021)

[69] Rudolf A., Stjepanovič Z., Cupar A. Study Regarding the Kinematic 3D Human-Body Model Intended for Simulation of Personalized Clothes for a Sitting Posture // Materials. 2021. Vol. 14, No. 18. Art. No. 5124. DOI: 10.3390/ma14185124

[70] Simić L., Kopačin V., Mumlek I., et al. Improved technique of personalised surgical guides generation for mandibular free flap reconstruction using an open-source tool. Eur. Radiol. 2021. Exp. Vol. 5. Art. No. 30. DOI: 10.1186/s41747-021-00229-x

[71] Lind M., Ystgaard P. Mesh-based tool path calculations for tubular joints. Advances in Mechanical Engineering. 2020. Vol. 12, No. 6. Pp.1–13. DOI: 10.1177/1687814020933383

[72] Bonneau D.A., Hutchinson D.J., DiFrancesco P.-M., Coombs M. Sala Z. Three-dimensional rockfall shape back analysis: methods and implications. Nat. Hazards Earth Syst. Sci. 2019. Vol. 19. Pp. 2745–2765. DOI: 10.5194/nhess-19-2745-2019

[73] Filippov S.V. [Blender software platform as an environment for modeling objects and processes in natural sciences] *Programmnaya platforma Blender kak sreda modelirovaniya ob"yektov i protsessov yestestvenno-nauchnykh distsiplin*. Preprints of IAM im. M.V. Keldysh [*Preprinty IPM im. M.V.Keldysha*]. 2018. No. 230. 42 p (In Russian). DOI: 10.20948/prepr-2018-230

[74] Milan J., Lubomír Ř., Petr S., Matěj Š. GPU Accelerated Path Tracing of Massive Scenes // ACM Trans. Graph. 2021. Vol. 40, No. 2. Art. No. 16. Pp. 1–17. DOI: 10.1145/3447807

[75] Yi T. Making PDFs with Animation and Lecture Video for Beginner. WSEAS Transactions on Advances in Engineering Education. 2017. Vol. 14. Pp. 76–80. https://www.wseas.org/multimedia/journals/education/2017/a185810-086.php

[76] Getling A.V. [Rayleigh-Benard convection. Structures and dynamics] *Konvektsiya Releya-Benara. Struktury i dinamika*. M: Editorial URSS. 247 p. (In Russian).

[77] Gershuni G.Z., Zhukhovitsky E.M. [Convective stability of an incompressible fluid] *Konvektivnaya ustoychivost' neszhimayemoy zhidkosti* M.: Nauka. 1972. 392 p. (In Russian).

[78] Belousov B.P. [Periodically acting reaction and its mechanism] *Periodicheski deystvuyushchaya reaktsiya i yeye mekhanizm*. Collection of abstracts on radiation medicine for 1958 [*Sbornik referatov po radiatsionnoy meditsine za 1958 g.*]. M: Medgiz. 1959 (In Russian).

[79] Glandsdorff P., Prigogine I. Thermodynamic theory of structure, stability and fluctuations. Wiley, New York. 1971.

[80] Batchelor G.K. An Introduction to Fluid Dynamics, Cambridge University Press. 1967. DOI: 10.1017/CBO9780511800955

[81] Goldstein H., Poole C., Safko J. Classical Mechanics, 3rd ed. American Journal of Physics. 2002. Vol. 70, pp. 782. DOI: 10.1119/1.1484149

[82] Antonov A.S. Technologies of parallel programming MPI and OpenMP: Proc. allowance. Foreword: V.A. Sadovnichiy [*Tekhnologii parallel'nogo programmirovaniya MPI i OpenMP: Ucheb. posobiye. Predisl.: V.A. Sadovnichiy. M.*]. Moscow: Moscow University Press, 2012. 344 p. (In Russian).

[83] Hoffmann G. Bézier Curves. https://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf (accessed: 01.04.2021)

[84] Piegl L., Tiller W. The NURBS Book (2. ed.). Berlin: Springer. 1997.