ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 89 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Иванов А.В.

Использование библиотеки aiwlib на примере численного моделирования стохастического резонанса

Рекомендуемая форма библиографической ссылки: Иванов А.В. Использование библиотеки aiwlib на примере численного моделирования стохастического резонанса // Препринты ИПМ им. М.В.Келдыша. 2018. № 89. 30 с. doi:10.20948/prepr-2018-89
URL: http://library.keldysh.ru/preprint.asp?id=2018-89

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.КЕЛДЫША
Р о с с и й с к о й а к а д е м и и н а у к

А.В. Иванов

Использование библиотеки aiwlib на примере численного моделирования стохастического резонанса

Москва — 2018

Иванов А.В.
e-mail: aivanov@keldysh.ru

Использование библиотеки aiwlib на примере численного моделирования стохастического резонанса

Библиотека `aiwlib` предназначена для разработки высокопроизводительных приложений численного моделирования на языках `C++11` и `Python` для `OS Linux`, организации массовых расчетов и последующего многопараметрического поиска, визуализации и анализа результатов. В последние десять лет библиотека успешно использовалась при создании наукоемкого программного обеспечения и проведения расчетов в самых разных областях.

В данной работе подробно показано применение `aiwlib` при изучении стохастического резонанса в системе с нелинейным броуновским движением. Для численного решения уравнения Фоккера–Планка используется метод стохастического аналога. Описаны создание кода, запуск расчетов и анализ полученных результатов.

Ключевые слова: численное моделирование, уравнение Фоккера–Планка, стохастический резонанс

Anton Valeryevich Ivanov
e-mail: aivanov@keldysh.ru

Using the library aiwlib on the example of numerical modeling of stochastic resonance

`Aiwlib` library is a library for `C++11` and `Python` languages, which is aimed for the development of high-performance computing numerical simulation applications running under `GNU/Linux OS`. It also provides means for batch calculations, search through results using multiparametric filters and visualization of results. In the last ten years, the library has been successfully used to create high-end software and carry out calculations in various fields.

In this paper we show in detail the application of `aiwlib` in the study of stochastic resonance in a system with nonlinear Brownian motion. For the numerical solution of the Fokker–Planck equation, the stochastic analog method is used. This paper describes the creation of the code, the launch of calculations and the analysis of the results.

Keywords: numerical simulation, Fokker-Planck equation, stochastic resonance

Содержание

Введение

Первая версия библиотеки `aiwlib` [1] была написана в 2008 году, а отдельные ее части (утилита для построения графиков `gplt` и система контроля результатов и алгоритмов `RACS` [2]) возникли еще в 2003 году. За прошедшие годы библиотека значительно изменилась [3], стала проще, компактнее, стабильнее, а ее функциональность существенно расширилась.

Библиотека `aiwlib` успешно использовалась в целом ряде проектов: при разработке программных комплексов для нужд сейсморазведки [4–7], физики плазмы [8] и оптики мутных сред [9, 10]; при решении фундаментальных [11–14] и прикладных задач по изучению магнитных материалов и созданию устройств спинтроники [15–18]; при изучении резонансных свойств суперпарамагнетиков [19, 20]; при моделировании газодинамики горения [21], процессов разработки керогеносодержащих нефтяных месторождений с учетом внутрипластового горения, деградации материалов и блистеринга в приповерхностных слоях под влияем ионной бомбардировки [22–24]; при решении задач пороупругости и гидроразрыва пласта [25].

Ядро библиотеки распространяется под лицензией `Apache-2` (что допускает ее применение в коммерческих проектах с закрытым исходным кодом), утилиты визуализации распространяются под лицензией `GPL-3`.

Стохастический резонанс — немонотонная зависимость модуля восприимчивости системы от температуры (шума) при постоянной частоте внешнего переменного поля [26]. Классической постановкой является изучение стохастического резонанса в системах с нелинейным броуновским движением, описываемых уравнением Фоккера–Планка [27]. Изначально предполагалось, что стохастический резонанс возникает при совпадении частоты Крамерса[1] с частотой внешней вынуждающей силы. Впоследствии было установлено, что стохастический резонанс обусловлен нелинейностью системы [28, 29] и возникает даже в нелинейных системах с одним положением равновесия [30], где частота Крамерса отсутствует.

[1] Частота Крамерса — характерная частота перехода частиц через потенциальный барьер в метастабильной системе под влиянием шума.

Данная работа посвящена демонстрации возможностей библиотеки `aiwlib` на примере численного изучения стохастического резонанса, при этом основное внимание уделено процессу написания кода, проведению расчетов под управлением системы `RACS`, анализу и визуализации результатов. Практика показывает, что описанный подход существенно ускоряет и упрощает весь цикл численного моделирования, начиная от программирования и заканчивая построением графиков для печатной работы.

Основные уравнения и численная схема

Ансамбль невзаимодействующих осцилляторов с кубической нелинейностью, находящихся в термостате с температурой T, описывается уравнением Фоккера–Планка для эволюции функции распределения $f(x,v)$, зависящей от координаты x и скорости v [31]:

$$\frac{\partial f(x,v,t)}{\partial t} + v\frac{\partial f}{\partial x} - \left(ax + bx^3 + A\sin\Omega t\right)\frac{\partial f}{\partial v} = \gamma\frac{\partial}{\partial v}\left[T\frac{\partial f}{\partial v} + vf\right], \qquad (1)$$

где a, b — параметры потенциала внешних сил, A, Ω — амплитуда и частота внешней вынуждающей силы, γ — параметр диссипации, T — температура (интенсивность шума) в системе. Здесь и далее мы будем использовать безразмерную систему единиц.

Для численного решения будем использовать метод стохастического аналога [32], в котором исходное параболическое уравнение заменяется системой стохастических дифференциальных уравнений, описывающих эволюцию множества невзаимодействующих траекторий $\{x_i(t), v_i(t)\}$. При интегрировании будем использовать расщепление по физическим процессам (консервативному движению в ангармоническом потенциале и диссипации), тогда на n-ном временном слое изменение координаты и скорости для одной траектории можно записать как

$$\begin{aligned}
x_{n+\frac{1}{2}} &= x_n + \frac{h}{2}v_n,\\
v'_{n+1} &= v_n + h\left[-ax_{n+\frac{1}{2}} - bx^3_{n+\frac{1}{2}} + A\sin\Omega t_{n+\frac{1}{2}}\right],\\
x_{n+1} &= x_{n+\frac{1}{2}} + \frac{h}{2}v'_{n+1},\\
v_{n+1} &= v'_{n+1} + \left(1 - \frac{h\gamma}{2}\right)^{-1}\cdot\left(-h\gamma v'_{n+1} + \sqrt{2h\gamma T}\xi_n\right),
\end{aligned}$$

где h — шаг по времени, ξ_n — случайный источник с нормальным распределением $N(0,1)$.

Восприимчивость χ будем рассчитывать как Фурье–образ средней скорости ансамбля на частоте внешней вынуждающей силы [11]:

$$\chi = \frac{2}{A(t_{\max} - t_{\min})} \left[\int_{t_{\min}}^{t_{\max}} \langle v \rangle (t) \cos \Omega t \, dt + i \int_{t_{\min}}^{t_{\max}} \langle v \rangle (t) \sin \Omega t \, dt \right],$$

где $t_{\min} \gg 1/\gamma$ — время установления вынужденных колебаний, $t_{\max}$ — максимальное время расчета, $t_{\max} - t_{\min} \gg 2\pi/\Omega$.

Реализация численной схемы с использованием aiwlib

Многолетняя практика показывает, что оптимальным вариантом является реализация кода, состоящего из вычислительного ядра, написанного на языке `C++`, и управляющего скрипта (интерфейса), написанного на языке `Python`, для импорта `C++` кода в `Python` используется пакет `SWIG`. Вычислительное ядро обеспечивает хранение данных, реализует численную схему и различные операции, которые должны выполняться с максимально возможной производительностью. Управляющий скрипт обеспечивает задание параметров расчетов, создание объектов и запуск процедур вычислительного ядра, создание файлов и общую диагностику. Такое разделение обусловлено тем, что скорость разработки на языке `Python` на порядок выше скорости разработки на `C++`, в то же время производительность кода на языке `Python` при вычислениях может уступать производительности кода на `C++` на три порядка. При помощи одного вычислительного ядра можно реализовать множество различных постановок задач с различными управляющими скриптами. Управляющий скрипт фактически является и конфигурационным файлом, для изменения параметров расчета достаточно отредактировать управляющий скрипт, не производя перекомпиляцию вычислительного ядра.

Получение исходных кодов библиотеки `aiwlib` и процесс установки подробно описаны в документации [33]. Для решения задачи о стохастическом резонансе нам придется создать четыре файла[2]:

- `Makefile` — файл, описывающий сборку в терминах утилиты `GNU Make`;
- `model.hpp` — заголовочный файл, содержащий описание класса модели;
- `model.cpp` — файл, содержащий непосредственно функции класса модели;
- `run.py` — головной управляющий скрипт на языке `Python`, таких скриптов будет несколько.

[2]Эти и другие файлы с исходным кодом из данной работы доступны по адресу `https://github.com/aivn/aiwlib/tree/master/demo/FPE`

Проще всего выглядит файл для утилиты make:

Makefile:

```
1   # имя модуля
2   name=model
3   # хидеры для обработки утилитой SWIG
4   headers=model.hpp
5   # необходимые .cpp-файлы с кодом
6   modules=model.cpp
7
9   include aiwlib/user.mk
```

Сборка C++ кода в виде модуля для языка Python требует довольно большой работы, но вся она спрятана от пользователя в шаблонном make–файле aiwlib/user.mk. Подробнее об использовании системы сборки aiwlib можно прочитать в документации [33].

Заголовочный файл model.hpp описывает класс, в котором будут хранится данные и параметры для расчета, основные методы и т.д. Использование класса для инкапсуляции не является обязательным, но практика показывает, что это довольно удобно, особенно если речь заходит о совместных расчетах, задействующих несколько различных моделей.

model.hpp:

```
1   #pragma once
2   #include <vector>
3   #include <aiwlib/mesh>
4   using namespace aiw;
5
6   class Model{
7       std::vector<Vec<2> > tracs; // массив траекторий
8   public:
9       double a, b, A, Omega, gamma, T, t, h; // параметры расчета
10
11      Vec<6> av;  // моменты x, v, xx, vv, xv, W
12      // вклад в моменты от одной траектории
13      inline Vec<6> d_av(const Vec<2> &p) const {
14          return p|(p&p)|(p[0]*p[1])|
15              ((.5*a + .25*b*p[0]*p[0])*p[0]*p[0] + .5*p[1]*p[1]);
16      }
17      Mesh<float, 2> f; // функция распределения
18
19      void init(Vec<2> xv0, int N); // инициализация
20      void calc(); // расчет одного шага
21  };
```

Имена параметров задачи в файле model.hpp совпадают с именами параметров в тексте.

Все классы и функции библиотеки `aiwlib` размещаются в пространстве имен `aiw`. В строке 7 объявлен массив траекторий (двумерных векторов), в строке 11 объявлен вектор из шести чисел, хранящий моменты функции распределения $\langle x\rangle$, $\langle v\rangle$, $\langle x^2\rangle$, $\langle v^2\rangle$, $\langle xv\rangle$ и среднюю энергию системы

$$\langle W\rangle(t) = \int\limits_{-\infty}^{\infty}\!\!\int \left[a\frac{x^2}{2} + b\frac{x^4}{4} + \frac{v^2}{2}\right] f(x,v,t)\,dx\,dv.$$

В библиотеке `aiwlib` вектора — это экземпляры шаблонного класса, параметризованного по размеру и опционально типу значений, по умолчанию используется тип `double`. Для векторов перегружены как традиционные операции сложения, вычитания, скалярного умножения и умножения на число, так и ряд нетрадиционных операций, востребованных при реализации численных схем. Применение двух таких операций продемонстрировано в строке 14 — побитового «или» `|` для конкатенации («склейки») векторов и побитового «и» `&` для покомпонентного перемножения.

В строке 17 объявлена двумерная сеточная функция с типом ячейки `float`, которая будет использоваться в качестве функции распределения `f`.

Файл `model.cpp` содержит задание начальных условий в виде δ-функции и реализацию численной схемы. Из модуля `aiwlib/gauss` используются функции `rand_init()` для инициализации псевдослучайной последовательности и `rand_gauss()` для вычисления значения случайного источника. Интерес представляет работа с двумерной сеткой `f` (строки 7 и 23) — операция `[]` на основе пределов и шага сетки пересчитывает координаты (двумерный вектор) в индекс ячейки и обеспечивает доступ к соответствующей ячейке по ссылке.

`model.cpp`:

```
1   #include <aiwlib/gauss> // библиотека со случайным источником
2   #include ''model.hpp''
3
4   void Model::init(Vec<2> xv0, int N){
5       tracs.resize(N, xv0);
6       av = d_av(xv0);
7       f.fill(0.f); f[xv0] = 1./f.step.prod();
8       rand_init(); // инициализация случайного источника
9       t = 0.;
10  }
11  void Model::calc(){
12      double sgT =  sqrt(2*h*gamma*T), stab = 1./(1.-.5*gamma*h);
13      double Fext = A*sin(Omega*(t+.5*h));
14      double df = 1./(tracs.size()*f.step.prod());
15
16      f.fill(0.f); av = vec(0.);
17      for(Vec<2> &p: tracs){ // цикл по траекториям
18          p[0] += p[1]*h*.5; // численная схема
```

```
19          p[1] += h*(-a*p[0]-b*p[0]*p[0]*p[0]+Fext);
20          p[0] += p[1]*h*.5;
21          p[1] += stab*(-gamma*p[1]*h+sgT*rand_gauss());
22
23          f[p] += df; av += d_av(p);  // диагностика
24      }
25      av /= tracs.size(); t += h;
26  }
```

Для сборки созданного модуля достаточно выполнить команду `make` (без параметров).

Наибольший интерес представляют управляющие скрипты на языке `Python`. Для начала напишем скрипт, рассчитывающий релаксацию начального условия

`run1.py`:

```
1   #!/usr/bin/python
2   # -*- coding: utf-8 -*-
3
4   # импортируем необходимые модули
5   from model import *            # модель
6   from aiwlib.vec import *       # вектора
7   from aiwlib.iostream import *  # работа с файлами
8   from aiwlib.MeshF2 import *    # сетка Mesh<float,2>
9
10  # создаем класс модели и задаем параметры
11  M = Model()
12  M.a, M.b = -1, 1
13  M.A, M.Omega = 0., 0.
14  M.gamma, M.T = 0.1, 0.05
15  M.h = 0.1
16
17  # инициализируем функцию распределения
18  M.f.init(ind(100,100), vec(-2.,-1.), vec(2.,1.))
19  M.f.bounds = 0x55 # режим обработки границ функции распределения
20
21  # создаем файлы для записи результатов
22  fout = File('f1.msh', 'w')
23  tvals = open('tvals1.dat', 'w')
24  print>>tvals, '#:t Mx Mv Mxx Mvv Mxv W'
25  for i in 'Mx:x Mv:v Mxx:x^2 Mvv:v^2 Mxv:xv W:W'.split():
26      print>>tvals, r'#:%s.tex=r''\left<%s\right>'''%tuple(i.split(':'))
27
28  # задаем и сохраняем начальные условия
29  M.init(vec(0.,0.), 100000)
30  M.f.head = 't=0'; M.f.dump(fout)
31  print>>tvals, M.t, M.av
32
33  while M.t<50: # цикл по времени
```

```
34      M.calc()
35      print>>tvals, M.t, M.av
36      M.f.head = 't=%g'%M.t; M.f.dump(fout)
```

В строках $5 \div 8$ происходит подключение необходимых модулей, включая созданный нами модуль на языке C++. Если библиотека aiwlib установлена в операционной системе глобально, порядок импорта модулей не имеет значения. Если библиотека aiwlib установлена в операционной системе локально, желательно импортировать пользовательский C++–модуль до импорта остальных модулей библиотеки aiwlib — при этом производится настройка путей для импорта.

Для подключения в Python шаблонных классов C++ необходимо произвести инстанцирование — компиляцию класса с конкретными значениями параметров шаблона. Примером такого модуля является модуль aiwlib.MeshF2, содержащий сетку Mesh<float,2>, создаваемый автоматически при сборке библиотеки. Для векторов такой подход оказывается слишком громоздким — существует множество различных комбинаций размерностей векторов и типов. Поскольку хранение данных в памяти вектора тривиально, в библиотеке aiwlib реализован импорт произвольных векторов из C++ на основе взаимодействия с системой управления типами пакета SWIG. Для включения этого механизма достаточно подключить в Python модуль aiwlib.vec.

В строках 18, 19 производится инициализация функции распределения (задание числа ячеек и пределов) и устанавливается режим обработки выхода за границу — выбранный режим позволяет корректно обрабатывать ситуации, когда траектория не попадает внутрь сетки. За подробностями можно обратиться к документации по библиотеке [33].

В строке 22 создается файл библиотеки aiwlib, в который будет сохраняться функция распределения для последовательных моментов времени (строки 30 и 36) кадр за кадром, при этом момент времени записывается в строковый заголовок функции распределения перед сохранением.

В строках $23 \div 26$ создается обычный текстовый файл Python, в который будет сохраняться эволюция средних значений (в несколько колонок с числами), и настраивается заголовок файла. Заголовок не является обязательным, но существенно упрощает процесс построения графиков с помощью утилиты gplt (утилита входит в состав aiwlib и является высокоуровневой оболочкой для стандартного графопостроителя gnuplot). Заголовок может состоять из нескольких строк, каждая из которых должна начинаться с символов '#:'. В строке 24 записывается название столбцов с числами. В строках 25, 26 задаются имена столбцов в формате LaTeX.

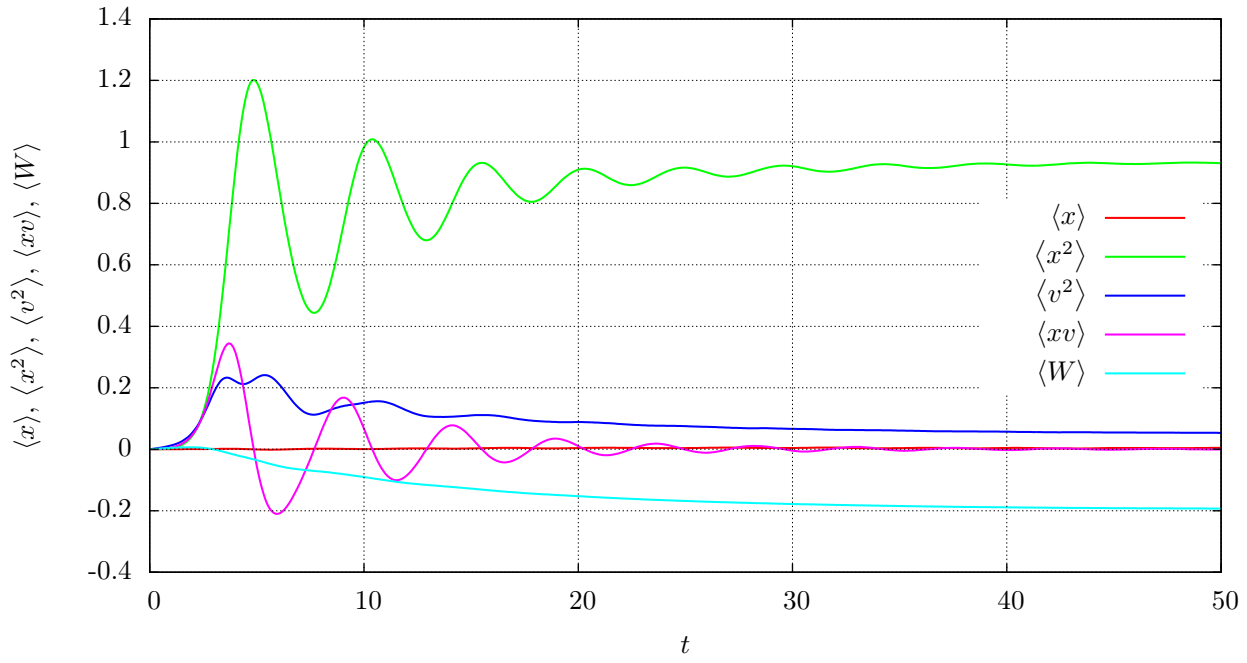


Рис. 1. Зависимости моментов от времени, полученные при запуске скрипта `run1.py`

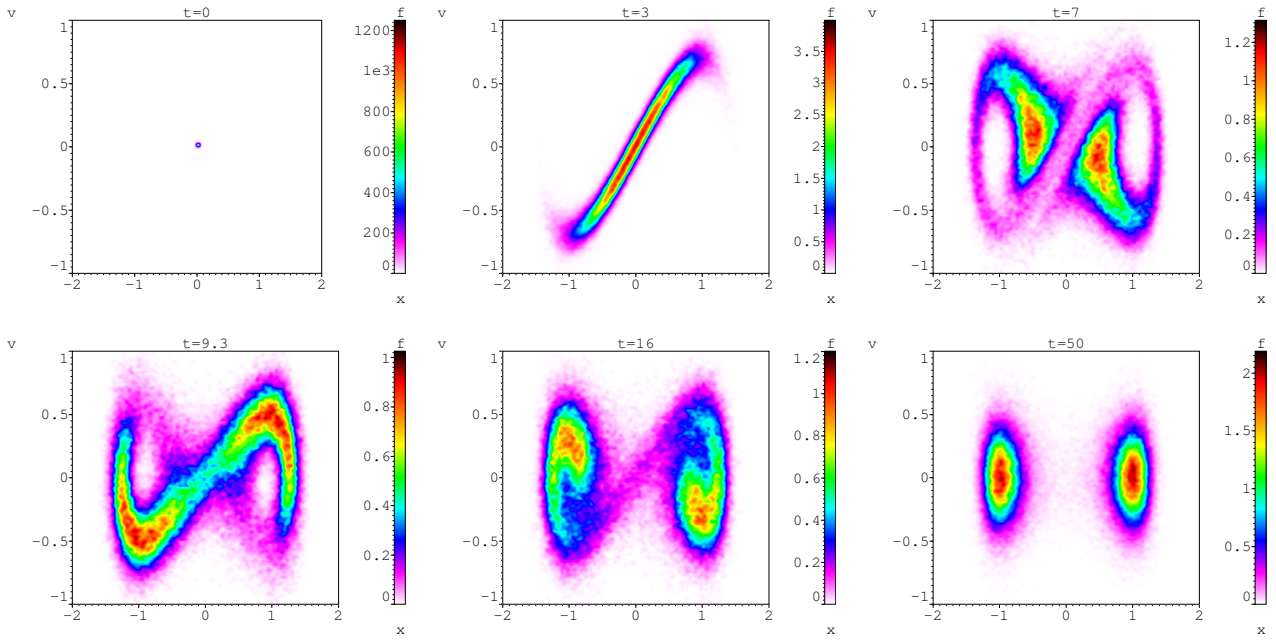


Рис. 2. Эволюция функции распределения, полученная при запуске скрипта `run1.py`

Изменим права доступа для созданного скрипта командой

```
$ chmod a+x run1.py
```

и запустим его командой `./run1.py` — должно появиться два файла `tvals1.dat` и `f1.msh` с результатами, которые представлены на рисунках 1 и 2. Для построения графика на рис. 1 использовалась команда

```
$ gplt -all t,-Mv tvals1.dat -sz 1.3,1 -sk 'right center spacing 1.3' \
                                                    -to picts/tvals1.pdf
```

где опция `-all t,-Mv` означает «построить зависимость всех столбцов из файла от столбца с именем `t`, за исключением столбца с именем `Mv`», опция `-sz 1.3,1` задает увеличенный по горизонтали размер графика, опция `-sk '...'` сдвигает положение так называемых «ключей» (описания кривых на графике) и устанавливает увеличенный интервал между ключами по вертикали, опция `-to picts/tvals1.pdf` указывает на то, что результат построения надо сохранить в файле с именем `picts/tvals1.pdf`. При этом запускается `gnuplot` в который утилита `gplt` передает ряд команд. Поскольку выбрано расширение файла `.pdf`, считается, что результаты должны быть пропущены через LaTeX — для `gnuplot`-а выбирается формат вывода `epslatex`, полученные файлы собираются при помощи `pdflatex`, используются заданные в заголовке файла `tvals1.dat` (LaTeX–имена для столбцов и т.д.). В итоге при минимальном числе нажатия клавиш получается график типографского качества с правильной шрифтовкой и зарамочным оформлением. Кроме файла `picts/tvals1.pdf` создается скрипт (исполняемый файл) `picts/tvals1.gplt`, содержащий инструкции по запуску `gplt` для повторного построения графика — этот скрипт можно использовать при изменении входных данных либо при необходимости отредактировать оформление графика.

Для построения графиков на рис. 2 использовался `aiwlib` вьювер `uplt`.

Доработаем скрипт `run1.py` для вычисления восприимчивости `chi` (показаны лишь изменения по сравнению со скриптом `run1.py`)

`run2.py`:

```
...
5   from math import *
...
14  M.A, M.Omega = 0.01, 1.
...
23  fout = File('f2.msh', 'w')
24  tvals = open('tvals2.dat', 'w')
...
34  chi, t_min, t_max = 0, 10/M.gamma, 10/M.gamma+40*pi/M.Omega
35  while M.t<=t_max: # цикл по времени
36      M.calc()
37      print>>tvals, M.t, M.av
38      M.f.head = 't=%g'%M.t; M.f.dump(fout)
39      if M.t>=t_min:
40          chi += M.av[1]*(cos(M.t*M.Omega)+sin(M.t*M.Omega)*1j)*M.h
41
42  chi *= 2/(M.A*(t_max-t_min))
43  print chi, abs(chi), atan2(chi.imag, chi.real)
```

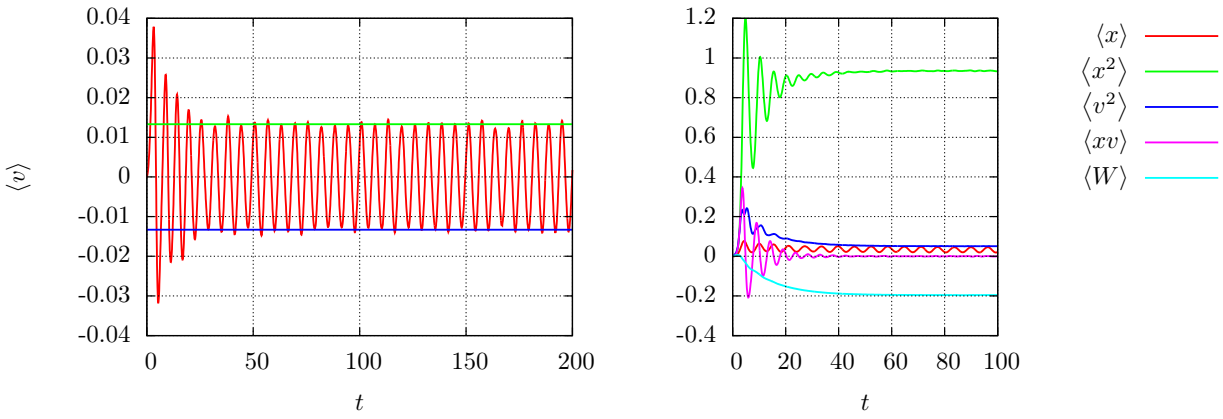


Рис. 3. Эволюция средних значений и амплитуда установившихся колебаний $A \cdot |\chi|$, полученные при запуске скрипта `run2.py`

В строке 5 импортируется стандартный модуль `math`. В строке 14 задаются ненулевые амплитуда и частота внешнего поля. В строках 23, 24 изменены имена файлов для сохранения результатов. В строке 34 создается переменная `chi` для расчета восприимчивости и пределы игнтегрирования по времени. Запустим скрипт

```
$ ./run2.py
  (1.23717556046+0.494007945499j) 1.332158856 0.379905393449
```

Согласно расчету, восприимчивость $\chi = 1.23717556046 + 0.494007945499j$. Построим график с зависимостью $\langle v \rangle\ (t)$ и наложим на него две прямых $\pm A \cdot |\chi|$, кроме того построим зависимости остальных средних величин от времени (рис. 3)

```
$ gplt -U Mv tvals2.dat -fn .0133 -fn -.0133 -nk -lbx t -lby '<%v%>' \
                              -sz .7 -rx :200 -to picts/tvals2a.pdf
$ gplt -U 'Mx Mxx Mvv Mxv W' tvals2.dat -sz .7 -sk 'out spacing 1.3' \
                              -lby '' -rx :100 -to picts/tvals2b.pdf
```

Опция `-U Mv` указывает на то, что необходимо строить столбец с именем `Mv` (по умолчанию строится зависимость от первого столбца). Опции `-fn .0133` и `-fn -.0133` добавляет функции 0.0133 и -0.0133. Опция `-nk` отключает рисование «ключей». Опции `-lbx t` и `-lby '<%v%>'` явно задают метки по осям, причем для задания метки по оси y используется расширенный синтаксис — запись `<%...%>` означает уголковые скобки $\langle ... \rangle$. Опции `-rx :100` и `-rx :200` задают пределы по оси абсцисс. Как видно из рисунка 3, расчет модуля восприимчивости проведен верно.

Подключение системы RACS и запуск массовых расчетов

Если возникнет необходимость детально изучить и сравнить поведение системы при различных наборах параметров, придется несколько раз запускать

отдельные расчеты, редактируя каждый раз управляющий скрипт. При этом необходимо каким то образом обеспечить сохранение результатов расчетов в разные файлы, обязательным требованием является сохранение параметров для каждого расчета. Это не вполне тривиальная задача, которую каждый исследователь решает по-своему, и на решение которой может уходить довольно много времени и сил.

Для расчета зависимости $\chi(\Omega, T)$ необходимо провести серию расчетов, для этого в новой версии скрипта `run.py` необходимо задать двойной цикл по Ω, T и организовать вывод результатов. Если потребуется изучить например зависимость $\chi(\Omega, A)$ придется писать еще один скрипт.

В целом, все эти действия гораздо проще решать на языке `Python` чем на `C++`. Оптимальным решением будет подключение системы `RACS` (`Results & Algorithms Control System`), которая фактически позволяет обойтись одним скриптом на все случаи жизни. При этом обеспечивается автоматическое сохранение результатов и параметров расчетов, массовый параллельный запуск расчетов с циклами по значениям параметров и динамической балансировкой вычислительных мощностей, последующий многопараметрический поиск и анализ результатов моделирования.

При разработке `RACS` делались следующие акценты:

- простота подключения (требуется минимальная модификация отлаженного кода);
- лаконичный и интуитивно понятный синтаксис при запуске расчетов;
- возможность обработки результатов расчетов средствами операционной системы и сторонними утилитами без потери целостности данных;
- интеграция с другими утилитами — вывод полученных зависимостей в формате `gnuplot` с заголовками `gplt`, чтение метаинформации о расчетах другими утилитами.

В случае использования `RACS` для каждого расчета создается уникальная директория, в которой будут храниться параметры расчета, исходные коды и результаты.

Директория создается в репозитории, который по умолчанию имеет имя `repo/`. Название уникальной директории расчета формируется из года, номера недели, дня недели и некоторого трехзначного порядкового номера, уникального для данной даты — таким образом расчеты внутри репозитория автоматически упорядочиваются по дате (например `c18_00_1005` — пятый расчет, проведенный в понедельник первой недели 2018 года). При создании очередной уникальной директории расчета, в текущей директории на нее автоматически создается мягкая ссылка `'_'` (одиночный символ подчеркивания).

Репозиторий может быть структурирован произвольным образом, т.е. являться деревом директорий, в котором расчеты группируются согласно требованиям пользователя (например на основе значений ключевых параметров).

Отдельные расчеты и репозитории могут перемещаться средствами операционной системы, пересылаться по сети и т.д.

Набор параметров расчета образует словарь, который сохраняется в файле `.RACS` уникальной директории расчета, в формате стандартного модуля `pickle` языка `Python`.

Для анализа результатов моделирования используется утилита командной строки `racs`. Утилита позволяет просматривать словарь параметров отдельного расчета (содержимое файла `.RACS`), строить выборки, отвечающие различным критериям, выводить выборки в различных форматах, модифицировать расчеты выборки, удалять расчеты выборки с диска.

Фактически расчеты, помещенные под `RACS`, образуют нереляционную базу данных, при этом отдельные расчеты можно рассматривать как записи в базе, а репозитории — как таблицы.

Создадим очередную версию скрипта `run.py` с подключением `RACS`

`run3.py`:

```
1   #!/usr/bin/python
2   # -*- coding: utf-8 -*-
3
4   # импортируем необходимые модули
5   from math import *
6   from model import *            # модель
7   from aiwlib.vec import *       # вектора
8   from aiwlib.iostream import *  # работа с файлами
9   from aiwlib.MeshF2 import *    # сетка Mesh<float,2>
10  from aiwlib.racs import * #[1] подключаем RACS
11
12  #[2] создаем объект для взаимодействия с RACS
13  calc = Calc(fdump=0,           #@ частота сбросов функции распределения
14              xv0=vec(0.,0.),    #@ начальные условия
15              N=100000,          #@ число частиц
16              f_sz=ind(100,100), #@ размер функции распределения
17              f_min=vec(-2.,-1), #@ левая нижняя и
18              f_max=vec(2.,1))   #@ правая верхняя границы функции распределения
19
20  # создаем класс модели и задаем параметры
21  M = calc.wrap(Model()) #[3] monkey-patch для управления моделью через RACS
22  M.a, M.b = -1, 1          #@ параметры потенциала
23  M.A, M.Omega = 0.01, 1.   #@ параметры внешней вынуждающей силы
24  M.gamma, M.T = 0.1, 0.05  #@ коэффициент затухания и температура
25  M.h = min(.1, .1/M.Omega) #@ автоматически устанавливаем шаг
26
27  # инициализируем функцию распределения
28  M.f.init(calc.f_sz, calc.f_min, calc.f_max)
29  M.f.bounds = 0x55 # режим обработки границ функции распределения
30
31  #[4] создаем файлы для записи результатов в уникальной директории расчета
```

```
32  if calc.fdump: fout = File(calc.path+'f.msh', 'w')
33  tvals = open(calc.path+'tvals.dat', 'w')
34  print>>tvals, '#:t Mx Mv Mxx Mvv Mxv W'
35  for i in 'Mx:x Mv:v Mxx:x^2 Mvv:v^2 Mxv:xv W:W'.split():
36      print>>tvals, r'#:%s.tex=r''\left<%s\right>'''%tuple(i.split(':'))
37
38  # задаем и сохраняем начальные условия
39  M.init(calc.xv0, calc.N)
40  if calc.fdump: M.f.head = 't=0'; M.f.dump(fout)
41  print>>tvals, M.t, M.av
42
43  nt, calc.chi = 0, 0
44  calc.t_min, calc.t_max = 10/M.gamma, 10/M.gamma+40*pi/M.Omega
45  while M.t<=calc.t_max: # цикл по времени
46      M.calc(); nt += 1
47      print>>tvals, M.t, M.av
48      if calc.fdump and not nt%calc.fdump:
49          M.f.head = 't=%g'%M.t; M.f.dump(fout)
50      if M.t>=calc.t_min:
51          calc.chi += M.av[1]*(cos(M.t*M.Omega)+sin(M.t*M.Omega)*1j)*M.h
52      calc.set_progress(M.t/calc.t_max, 'calc') #[5] степень завершенности
53
54  calc.chi *= 2/(M.A*(calc.t_max-calc.t_min))
```

На первый взгляд эта версия значительно отличается от предыдущей, однако ключевыми являются всего лишь пять изменений отмеченных в комментариях как `[...]` — остальные изменения связаны с передачей в `RACS` значений всех параметров расчета и не являются обязательными.

В строке 10 импортируется модуль `aiwlib.racs` содержащий все необходимые классы и функции. В строках 13–18 создается экземпляр класса `Calc` из модуля `aiwlib.racs`, обеспечивающий создание уникальной директории, сохранение параметров расчета и т.д. Аргументы конструктора класса `Calc` становятся полями экземпляра класса, их значения могут быть изменены через аргументы командной строки. Аргументы командной строки разбираются при вызове конструктора, при этом может быть как произведен серийный запуск расчетов (организован цикл по параметрам), так и выведена справка (без запуска расчета).

Обратите внимание на комментарии, начинающиеся с символов `#@` — строки, содержащие такие комментарии, выводятся при запуске расчета с параметрами `-h` или `--help`, при этом сам расчет не запускается. Это позволяет оперативно (не заглядывая в код скрипта) просматривать имена и значения ключевых параметров по умолчанию. Кроме того, в справке содержится памятка по ключевым параметрам `RACS` и синтаксису серийного запуска расчетов.

В строке 21 активизируется механизм так называемого «monkey patch», позволяющий изменять из командной строки атрибуты объекта `M`. Этот механизм

обеспечивает подключение RACS к уже отлаженному коду на языке Python с минимальными модификациями кода — в итоге все параметры объекта M автоматически будут сохраняться на диске, кроме того, значение любого параметра может быть изменено при запуске через аргументы командной строки. Например, запуск с аргументом a=1 приведет к тому, что для параметра потенциала a будет установлено значение 1, несмотря на то, что в коде написано M.a=-1, значение -1 трактуется как значение по умолчанию. Перекрытие значения параметра производится лишь при его первом задании в коде.

В строках 32 и 33 файлы для результатов расчета открываются в уникальной директории расчета calc.path. Директория создается автоматически при первом обращении к полю calc.path в репозитории с именем repo. Имя репозитория может быть изменено через аргументы командной строки -r или --repo, либо через аргумент _repo конструктора класса Calc. Имя репозитория может содержать фрагменты вида %(имя-параметра)s, в этом случае имя репозитория будет формироваться на основе значений параметров, что позволяет автоматически распределять расчеты по различным репозиториям.

При создании уникальной директории расчета автоматически (на основе анализа Makefile) определяется список файлов с исходными кодами расчета, все файлы сохраняются в директории расчета в виде архива .src.tgz. Это может быть полезно при дальнейшем анализе результатов расчета, особенно по прошествии некоторого времени. Кроме самих исходных кодов в файле .src.md5 сохраняются md5-суммы файлов с исходными кодами, а в поле calc.md5sum сохраняется md5-сумма tar-архива с исходными кодами.

В строке 52 устанавливается степень завершенности расчета — это позволяет оценивать общее время выполнения расчета и выводить информацию о ходе расчета в терминал.

Основной результат расчета, параметр χ, вычисляется и задается как поле объекта calc в строке 54. Вся информация, содержащаяся в объекте calc, автоматически сохраняется в файле .RACS по завершении расчета и доступна для последующего анализа.

Запустим скрипт ./run3.py:

```
$ ./run3.py
calc 0:00:14 from 0:00:31 [########################                    ]
```

Благодаря инструкции в строке 52 в процессе расчета в терминале будет выводится «ползунок» (progress bar) с информацией о прошедшем времени и оценочном времени выполнения. После завершения работы будет выведена информация о фактическом времени выполнения, размере расчета и уникальной директории расчета

```
RUNTIME 0:00:32.440 SIZE 236K (.RACS 1.4K) /home/ ... /repo/c18_16_4001/
```

Мы можем посмотреть, что находится в директории расчета, средствами операционной системы

```
$ ls -a _
.  ..  .RACS  .src.md5  .src.tgz  tvals.dat
```

и вывести параметры расчета при помощи утилиты `racs`

```
$ racs _
{'A': 0.1,
 'N': 100000,
 'Omega': 1.0,
 'T': 0.2,
 'a': -1.0,
 'args': ['./run3.py'],
 'av': Vec(0.013792518012332345,0.008717279707804315, ...),
 'b': 1.0,
 'chi': (1.232688377475044+0.48596957942444513j),
 'f_max': Vec(2.0,1.0),
...
```

После того, как мы убедились, что новый скрипт работает, можно запустить серию расчетов

```
$ ./run3.py T=[.1,.2..2] Omega=[.1,.2..2] A1+ -c=4 -d
Start queue for 400 items in 4 threads, master PID=16791, logfile=''/tmp/...
```

Аргументы командной строки вида `T=[`T_1`,`T_2`..`T_N`]` означают, что необходимо провести серию расчетов, в которых параметр `T` должен «пробежаться» по элементам арифметической прогрессии. Можно задать геометрическую прогрессию или просто набор значения через запятую, весь список возможностей можно посмотреть в документации [33] или при помощи команды `./run3.py -h`. Поскольку наборы значения заданы для двух параметров T и Ω, будет запущена серия со всеми возможными комбинациями значений параметров.

Аргумент командной строки `A1+` означает, что все расчеты серии будут помечены тэгом `A1`, что упростит дальнейший анализ результатов. Допускается использование произвольного количества тэгов.

Аргумент `-c=4` означает, что серия будет считаться в четыре потока, при этом `RACS` автоматически обеспечит балансировку загрузки. Аргумент `-d` означает что запуск будет «демонизирован», т.е. терминал будет освобожден, и задачи будут выполняться в фоновом режиме. Это особенно удобно при удаленном запуске по сети.

При серийном запуске, после разбора аргументов командной строки в конструкторе класса `Calc`, расчет порождает при помощи системного вызова `fork` для каждого набора параметров свой дочерний процесс. Дочерние процессы проводят вычисления, родительский процесс ожидает их завершения и при необходимости запускает новые дочерние процессы, но не более числа заданного при помощи аргумента `-c`.

Посмотреть в каком состоянии находятся расчеты можно, как построив выборку по конкретному репозиторию

```
$ racs repo/ started? T Omega state path --ast
-----+-------+---------------------------+------------------
 T   | Omega | state                     | path
-----+-------+---------------------------+------------------
 1.7 | 0.2   | 10.4% 0:00:09 from 0:01:28 | repo/c18_15_6037/
 1.8 | 0.2   | 9.8% 0:00:08 from 0:01:27  | repo/c18_15_6038/
 1.9 | 0.2   | 8.9% 0:00:07 from 0:01:28  | repo/c18_15_6039/
 2.0 | 0.2   | 8.9% 0:00:07 from 0:01:28  | repo/c18_15_6040/
-----+-------+---------------------------+------------------
selected 4 items [0:00:00.013]: 0:00:33.000 total ... completed [0:00:00.000]
```

так и запросив у RACS, какие серии расчетов вообще запущены на данной машине

```
$ racs -q

# user@host:startpath repo='repo' tasks=400 threads=4 PID=16791
# ./run3.py T=[.1,.2..2] Omega=[.1,.2..2] -c=4 -d
#   T: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, ... ]
#   Omega: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, ... ]
--------------------+-------+---------+-----+-------
 date               | PID   | started | T   | Omega
--------------------+-------+---------+-----+-------
 2018.04.14-17:21:50 | 19680 | 21.25%  | 0.5 | 0.5
 2018.04.14-17:21:50 | 19708 | 21.5%   | 0.6 | 0.5
 2018.04.14-17:21:52 | 19743 | 21.75%  | 0.7 | 0.5
 2018.04.14-17:21:53 | 19772 | 22%     | 0.8 | 0.5
--------------------+-------+---------+-----+-------
changed  2018.04.14-17:21:53
finished 2018.04.14-17:21:53
started  2018.04.14-17:17:22
completed 21% (0:04:30.991 from 0:21:30.435) /tmp/racs-started-16791
```

Следует соблюдать осторожность при серийном запуске расчетов — несогласованный одновременный запуск нескольких серий может перегрузить CPU и создать нехватку памяти, что приведет к падению производительности. В настоящий момент RACS не следит за суммарной нагрузкой вычислителя при запуске нескольких серий расчетов.

Полученный код можно использовать при организации массовых расчетов на кластерах в рамках одного узла, например, на кластере k60.kiam.ru — aiwlib установлена на этом кластере локально в директории /home/aiv/aiwlib, при сборке достаточно указать в последней строке Makefile полный путь к библиотеке

```
include /home/aiv/aiwlib/include/aiwlib/user.mk
```

Для запуска используется команда

```
$ mpirun -np 1 -maxtime 20 ./run3.py -r=/nethome/username/FPE \
                           T=[.1,.2..2] Omega=[.1,.2..2] A1+ -c=56
```

где `-np 1` — число MPI–процессов, `-maxtime 20` — время расчета в минутах, `-r=/nethome/username/FPE` — репозиторий, на кластерах, как правило, нельзя размещать репозитории в домашней директории, для этого предназначены специальные хранилища. Аргумент `-c=56` указывает на то, что будет использоваться 56 дочерних процессов — один процессор кластера содержит 28 ядер и поддерживает гипертрейдинг.

Подключение системы RACS на языке C++

Приложение численного моделирования в виде вычислительного ядра на языке C++ и управляющего скрипта на Python является оптимальным с точки зрения соотношения производительность/скорость разработки и модификации кода, но такая гибридная архитектура не всегда может быть реализована. Проблемы могут быть связаны как с переносимостью (на целевой машине могут отсутствовать пакеты `python-dev` и `swig`), так и с запуском из-под MPI на нескольких узлах кластера (в `aiwlib` есть свой модуль `mpi4py`, но для его корректной работы может требоваться специфическая настройка).

Одним из решений может являться отказ от Python и реализация аналога скрипта `run.py` на C++. При этом для организации крупных серий расчетов на кластере система RACS становится особенно востребованной.

Для подключения RACS в C++ придется добавить две строки (отмечены `*`) в файл

`model.hpp`:

```
---------------------------------------------------------------------------
...
3   #include <aiwlib/mesh>
*   #include <aiwlib/objconf> //<=== поддержка конфигурации объектов
4   using namespace aiw;
5
6   class Model{
...
9       double a, b, A, Omega, gamma, T, t, h; // параметры расчета
*       CONFIGURATE(a, b, A, Omega, gamma, T, t, h); //<=== для связи с RACS
...
---------------------------------------------------------------------------
```

Макрос `CONFIGURATE(...)`, объявленный в заголовочном файле `aiwlib/objconf`, добавляет в класс `Model` шаблонный метод `configurate`, обеспечивающий чтение/запись конфигурационных файлов и взаимодействие с RACS. Доступ осуществляется только к тем полям класса `Model`, которые были перечислены в аргументах макроса.

Ниже приводится C++-аналог скрипта `run3.py`, имеющий практически такую же функциональность:

`run.cpp`:

```
1   #include <complex>
2   #include <aiwlib/racs>
3   #include ''model.hpp''
4
5   int main(int argc, const char **argv){
6       RacsCalc calc(argc, argv);
7
8       Model M;
9       M.a = -1; M.b = 1;
10      M.A = 0.01; M.Omega =1.;
11      M.gamma = 0.1; M.T = 0.2;
12      M.h = .1;
13      calc.control(M);
14      if(M.h>.1/M.Omega) M.h = .1/M.Omega; // автоматически устанавливаем шаг
15
16      int fdump = 0, N = 100000;
17      Ind<2> f_sz(100,100);
18      Vec<2> xv0, f_min(-2,-1), f_max(2,1);
19      double t_min = 10/M.gamma, t_max = 10/M.gamma+40*M_PI/M.Omega;
20      std::complex<double> chi;
21      CALC(fdump, N, xv0, f_sz, f_min, f_max, t_min, t_max, chi);
22
23      M.f.init(f_sz, f_min, f_max);
24      M.f.bounds = 0x55; // режим обработки границ функции распределения
25
26      File fout, tvals(''%tvals.dat'', ''w'', calc.path());
27      if(fdump) fout = File(''%f.msh'', ''w'', calc.path());
28      tvals(''#:t Mx Mv Mxx Mvv Mxv W\n'');
29      M.init(xv0, N);
30      if(fdump){ M.f.head = ''t=0''; M.f.dump(fout); }
31      tvals(''% %\n'', M.t, M.av);
32
33      int nt = 0;
34      while(M.t<=t_max){ // цикл по времени
35          M.calc(); nt += 1;
36          tvals(''% %\n'', M.t, M.av);
37          if(fdump && nt%fdump==0){
38              std::stringstream buf; buf<<''t=''<<M.t;
39              M.f.head = buf.str(); M.f.dump(fout);
40          }
41          if(M.t>=t_min)
42              chi += M.av[1]*std::complex<double>(cos(M.t*M.Omega),
43                                                  sin(M.t*M.Omega))*M.h;
44          calc.set_progress(M.t/t_max);
45      }
```

```
46      chi *= 2/(M.A*(t_max-t_min));
47      return 0;
48  }
```

В строке 2 подключается заголовочный файл `aiwlib/racs`, обеспечивающий поддержку `RACS`. В строке 6 создается объект `calc` которому передаются аргументы командной строки для разбора. В строке 13 экземпляр класса `Model` ставится на контроль в объект `calc`, в этот момент при помощи метода `Model::configurate` значения полей объекта `M` могут быть изменены на основе аргументов командной строки. Исходные значения, задаваемые в файле `run.cpp`, трактуются как значения по умолчанию.

В строке 21 параметры расчета, не являющиеся полями класса `Model`, ставятся на контроль в объект `calc` при помощи макроса `CALC`, при этом значения параметров могут быть изменены из командной строки.

В строках 26 и 27 создаются файлы (экземпляры класса `File` библиотеки `aiwlib`) для сохранения результатов расчета. Файлы создаются в уникальной директории расчета, которая, в свою очередь, создается автоматически при первом вызове метода `calc.path()`. Класс `File` поддерживает типобезопасный аналог функции `printf`, реализованный на основе шаблонов с произвольным количеством аргументов (variadic templates), который используется для вывода средних значений в строках 31 и 36. В строке 44 устанавливается степень завершенности расчета, при этом все параметры, стоящие на контроле в объекте `calc`, сохраняются на диск.

При завершении расчета автоматически происходит сохранение всех параметров, контролируемых объектом `calc`.

Для сборки новой версии программы надо добавить в `Makefile` строку

```
cxxmain=run.cpp
```

которая указывает системе сборки `aiwlib` на то, что из файла `run.cpp` можно создать отдельное приложение. Переменная `cxxmain` может содержать несколько `.cpp` файлов со своими функциями `main()`, каждый из них будет компилироваться в отдельное приложение.

Для запуска сборки необходимо выполнить команду

```
$ make run mpi=on
```

опция `mpi=on` указывает на то, что необходимо производить сборку с поддержкой `MPI`.[3]

Запуск полученного приложения практически не отличается от запуска предыдущей версии `run3.py`. В качестве интересного примера рассмотрим запуск полученной программы на кластере `k60.kiam.ru`.

[3] Речь идет о кластерах. Если на Вашей машине `MPI` не установлен, эту опцию указывать не нужно.

После сборки приложения `run` запустим его из под `MPI`

```
$ mpirun -np 28 -maxtime 20 \
  ./run -r=/nethome/username/FPE a=1 T=[.1,.2..2] Omega=[.1,.2..2] A2+ --mpi
```

где `a=1` — параметр потенциала (с параметров `a=-1` мы проводили расчеты в предыдущем разделе), `A2+` — метка для серии расчетов, `--mpi` — опция, указывающая системе `RACS` на то, что серия будет считаться с использованием `MPI`. При этом один из `MPI`-процессов будет играть роль мастер-процесса, создавая уникальные директории расчетов и раздавая задания остальным `MPI`-процессам.

Анализ результатов расчетов

Для поиска и анализа результатов расчетов в репозитории используется утилита командной строки `racs`. Подробное описание ее возможностей выходит за рамки настоящей работы, для получения справки можно выполнить команду `racs` без аргументов или с аргументом `--help`, детали доступны в документации [33].

Построим зависимость $\chi(\Omega, T)$ на основе первой серии расчетов

```
$ racs repo/ A1? ~Omega ~T% chi --ast
-------+-----+--------------------------------------
 Omega | T   | chi
-------+-----+--------------------------------------
 0.1   | 0.1 | (0.110670051057+0.111911601201j)
 0.2   | 0.1 | (0.197872719179+0.142127831895j)
...
 2.0   | 0.1 | (-0.68958373677+0.194997768828j)
-------+-----+--------------------------------------
 0.1   | 0.2 | (0.174600802321+0.123630558973j)
...
 2.0   | 2.0 | (-1.01582930669+0.47935410328j)
-------+-----+--------------------------------------
selected 400 items [0:00:00.046]: 5:39:58.000 total ... completed [0:00:00.019]
```

Аргументы утилиты `racs`, как правило, содержат имена анализируемых репозиториев (заканчиваются на символ `/`, в данном случае это `repo/`), выражения, которые вычисляются для каждого расчета, и опции, задающие действия над результатами (в данном случае это `--ast`, сокращение от `--astable`, вывести результаты в виде таблицы). Выражения задаются в синтаксисе языка `Python` и вычисляются отдельно для каждого расчета в словаре с параметрами расчета. По выражениям может осуществляться фильтрация (символ `?` в конце выражения, здесь мы выбрали лишь те расчеты, которые имеют метку `A1`), сортировка (символ `~` в начале выражения), группировка (символ `%` в конце выражения выводит пустую строку при изменении значения выражения) и ряд

других действий. Последняя строка вывода содержит информацию о числе расчетов в выборке, времени построения выборки и суммарном времени счета для расчетов выборки — более пяти часов.

Для построения графика зависимости $\chi(\Omega, T)$ выведем результаты в формате, пригодном для `gnuplot`, и используем утилиту `gplt`

```
 $ racs repo/ A1? ~Omega ~T% 'chi_abs=abs(chi)' --asd > Omega-T-chi-A1.dat
selected 400 items [0:00:00.046]: 5:39:58.000 total ... completed [0:00:00.018]
 $ less Omega-T-A1.dat
#:Omega  T    chi_abs
0.1      0.1  0.157391444127
0.2      0.1  0.243626627433
...
2.0      0.1  0.716623931966

0.1      0.2  0.213939139203
...
 $ gplt -h3d -pm3d map -cbp invrainbow Omega-T-chi-A1.dat -rx .1: -ry .1:2 \
                                                  -sz .75,1 -to picts/A1-chi.pdf
```

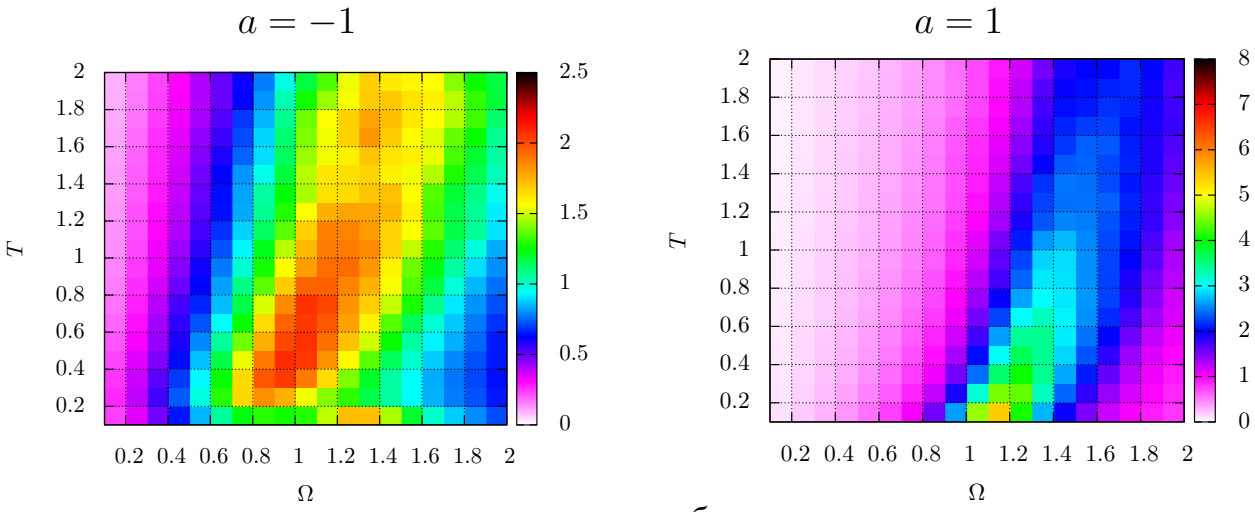
Здесь опция `chi_abs=abs(chi)` задает имя для столбца, содержащего $|\chi|$ (функция `abs` в языке `Python` вычисляет модуль комплексного числа), опция `--asd` (сокращение от `--asdata`) означает вывод результатов в текстовом формате, пригодном для `gnuplot` (аналог формата `CSV MS Windows`, но в качестве разделителей используется пробел или символ табуляции).

При построении графика опции `-h3d -pm3d map` включают режим цветового отображения для функции двух переменных, опция `-cbp invrainbow` устанавливает палитру. Результаты приведены на рисунке 4 *а*. Результаты аналогичных расчетов для одноямного потенциала, полученные на кластере, приведены на рисунке 4 *б*. Из рисунка 4 хорошо видно, что в случае двуямного потенциала стохастический резонанс наблюдается главным образом в диапазоне частот $\Omega \in [0.4 : 1.4]$, в то время как для одноямного потенциала стохастический резонанс смещен в область более высоких частот $\Omega \in [1.4 : 1.6]$ и значительно слабее выражен.

Детально стохастический резонанс виден на зависимостях $|\chi|(T)|_{\Omega=\text{const}}$, являющихся срезами зависимости $|\chi|(\Omega, T)$, рис. 5. Такие зависимости могут быть выгружены в `.dat`–файлы одной командной `racs`:

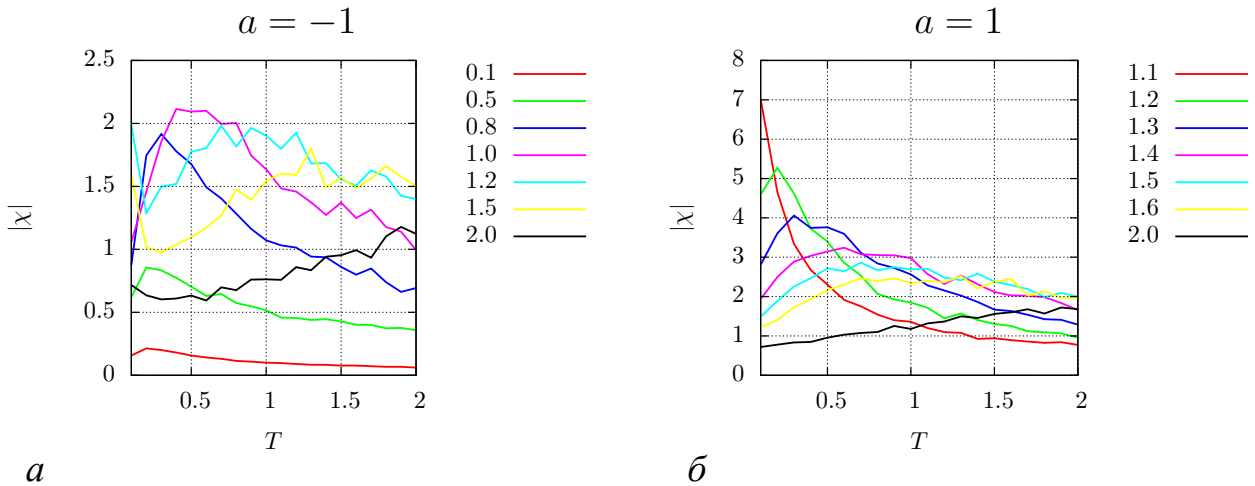
```
$ racs repo/ a: Omega: ~T chi='abs(chi)' --asd 'dat/T-chi-a%(a)g-%(Omega)g.dat'
$ ls dat/T-chi*
  T-chi-a-1-0.1.dat  T-chi-a-1-0.2.dat  T-chi-a-1-0.3.dat ...
  T-chi-a1-0.1.dat   T-chi-a1-0.2.dat T-chi-a1-0.3.dat    ...
```

Здесь опции `a:` и `Omega:` означают «создать множество выборок, различающихся значениями параметров a и Ω». В каждой из выборок производится сортировка



а *б*

Рис. 4. Зависимость $|\chi|(\Omega, T)$ для двуямного (*а*) и одноямного (*б*) потенциалов



а *б*

Рис. 5. Зависимость $|\chi|(T)$ для двуямного (*а*) и одноямного (*б*) потенциалов при различных значениях Ω

по температуре T и добавляется столбец с именем `chi`, содержащий значения $|\chi|$, полученное множество может быть обработано далее различными способами, например разобрано по `.dat`- файлам с именами, зависящими от значений параметров a и Ω.

Для оценки величины стохастического резонанса (немонотонности зависимости $|\chi|(T)$) удобно использовать функцию η

$$\eta(\Omega) = \frac{\max_{\Omega'=\Omega} |\chi(\Omega', T)| - \max_{\Omega'=\Omega} \left[|\chi(\Omega', T_{\min})|, |\chi(\Omega', T_{\max})|\right]}{\max_{\Omega'=\Omega} |\chi(\Omega', T)|},$$

равную нулю, если стохастический резонанс отсутствует на заданной частоте, и стремящуюся к единице по мере усиления эффекта, рисунок 6.

Для построения функции η можно написать отдельный скрипт на языке `Python`, анализирующий файлы `Omega-T-chi-A[12].dat`, но с этой задачей

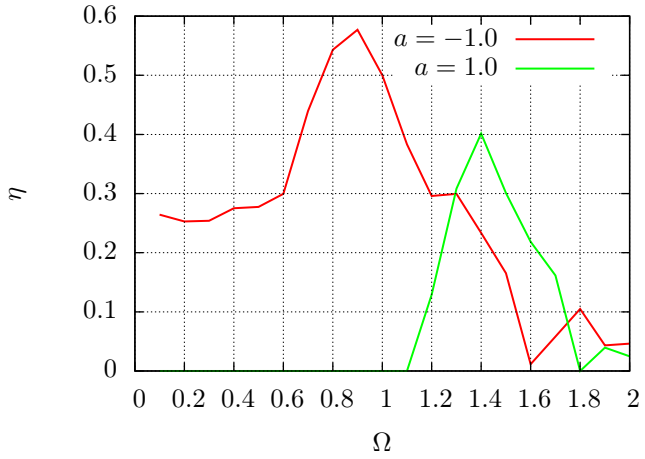


Рис. 6. Зависимость $\eta(\Omega)$ для двуямного ($a = -1$) и одноямного ($a = 1$) потенциалов

вполне может справиться и система `RACS`:

```
$ racs repo/ A1? Omega: ~T chi='abs(chi)' :\
    ~Omega 'eta=(max(chi)-max(chi[0],chi[-1]))/max(chi)' --asd > eta-A1.dat
```

Самое интересное происходит при обработке опции «двоеточие» — все выборки из построенного множества сводятся в одну, при этом в качестве значений столбца `ca` подставляются списки значений исходных выборок. Каждую выборку можно рассматривать как матрицу, составленную из строк–расчетов. Каждая матрица преобразуется в отдельную строку новой выборки с соответствующим значением параметра Ω, при этом для каждой новой строки в столбце с именем `ca` хранится не одно значение $|\chi|$, а список значений, отвечающий исходной выборке.

В объединенной выборке столбцы выражения вычисляются традиционным образом, в частности используется встроенная в `Python` функция `max`.

Сравним частоты $\Omega_{\max}$, на которых наблюдается максимум отклика, и аналитическую оценку для частоты Крамерса для двуямного потенциала [27]

$$\omega_{\text{kr}} \approx \frac{\omega_0^2\sqrt{2}}{\gamma} \exp\left[-\frac{\Delta W}{T}\right],$$

где ω_0 — собственная частота, ΔW — высота барьера, при параметрах потенциала $a = -1$, $b = 1$ получаем $\omega_0 = 1$, $\Delta W = 1/4$. Для получения $\Omega_{\max}$ из результатов расчетов используем `racs`

```
$ racs repo/ a: T: ~Omega ca='abs(chi)' : a: ~T 'argmax(Omega,ca)' \
                                    --asd dat/T-chimax-a%\(a\)g.dat
```

где `argmax(Omega,chi)` — встроенная функция `racs`, возвращающая значения параметра Ω, при котором параметр χ достигает максимума. Результаты

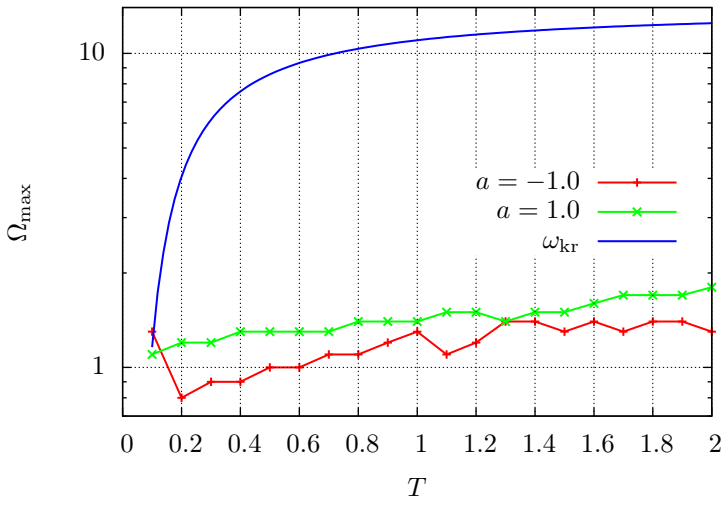


Рис. 7. Сравнение зависмости $\Omega_{\max}(T)$ для двуямного ($a = -1$) и одноямного ($a = 1$) потенциалов с аналитической оценкой для частоты Крамерса ω_{kr}

расчетов изначально разбиваются на выборки, отличающиеся значениями параметров a и T, затем в каждой выборке находится частота $\Omega_{\max}$, полученные результаты разбираются на две выборки, различающиеся значениями параметра a, и выгружаются в файлы `dat/T-chimax-a1.dat` и `dat/T-chimax-a-1.dat`. Как видно из рисунка 7, частоты $\Omega_{\max}$ и ω_{kr} совпадают лишь для малых температур (при которых отклик отнюдь не максимален), а по мере увеличения температуры начинают различаться на порядок и более.

Заключение

В данной работе был рассмотрен относительно простой пример приложения численного моделирования, иллюстрирующий лишь небольшую часть возможностей библиотеки `aiwlib`.

Библиотека `aiwlib` предоставляет развитые средства отладки, оптимизированные для применения в численном моделировании; потоки ввода/вывода, обеспечивающие типобезопасный аналог функции `printf`, мапирование файлов и перегруженные операторы для бинарного ввода/вывода; традиционные вектора и матрицы; традиционные многомерные картезианские сетки, сетки на основе Z-кривой Мортона, обеспечивающие высокую локальность данных, и сферические сетки, построенные на основе рекурсивного разбиения додекаэдра (пентакисдодекаэдра); набор оригинальных вьюверов; систему контроля результатов и алгоритмов для управления численным моделированием; систему сборки на основе `GNU Make`; ряд дополнительных модулей для интерполяции, прогонки, конвертации данных и т.д.

История задачи о стохастическом резонансе насчитывает уже почти сорок лет [26]. Несмотря на то что десять лет назад [28, 29] было показано, что

стохастический резонанс является прямым следствием нелинейности системы, не связан в общем случае с частотой Крамерса и возникает даже в системах с одним положением равновесия, где частота Крамерса отсутствует, до сих пор регулярно выходят работы, объясняющие стохастический резонанс на основе совпадения частоты внешней силы с частотой Крамерса [34–36].

В данной работе в результате прямого численного моделирования было подтверждено возникновение стохастического резонанса в системах с одноямным ангармоническим потенциалом, при этом стохастический резонанс в системе с двуямным потенциалом оказывается значительно более выражен. Для бистабильной системы частота Крамерса совпадает с частотой, на которой наблюдается максимум отклика, лишь при малых температурах. По мере роста температуры эти частоты начинают различаться на порядок и более — таким образом, объяснение стохастического резонанса на основе совпадения частоты внешней силы с частотой Крамерса представляется некорректным.

Список литературы

[1] Иванов А.В., Хилков С.А., Жданов С.А. Библиотека aivlib. — URL: http://a-iv.ru/aivlib/.

[2] Иванов А.В. Система контроля результатов и алгоритмов для задач численного моделирования // Автоматизация. Современные технологии. — 2007. — № 12. — С. 29–34.

[3] Иванов А.В., Хилков С.А. Библиотека aiwlib — инструмент для создания приложений численного моделирования, визуализации и анализа результатов // Научная визуализация. — 2018. — Т. 10, № 1. — С. 110–127.

[4] Высокопроизводительное динамическое 3d моделирование полноволнового сейсмического поля в задачах сейсморазведки. Опыт применения в условиях различных сейсмо–геологических регионов / В.Д. Левченко, А.Ю. Перепёлкина, А.В. Иванов и др. // Суперкомпьютерные технологии в нефтегазовой отрасли. Математические методы, программное и аппаратное обеспечение Материалы научно-практической конференции. — 2017. — С. 49–53.

[5] Высокопроизводительное 3d моделирование полноволнового сейсмического поля для задач сейсморазведки / А.В. Закиров, В.Д. Левченко, А.В. Иванов и др. // Геоинформатика. — 2017. — № 3. — С. 34–45.

[6] Вычисление полного сейсмического волнового поля в геосреде на основе нового метода решения прямых задач сейсмоакустики / А.В. Иванов, С.А. Каплан, А.В. Каракин и др. // Геоинформатика. — 2006. — № 3. — С. 59–61.

[7] Многолучевая 3D глубинная сейсмическая миграция до суммирования с сохранением амплитуд / А.Л. Плешкевич, А.В. Иванов, В.Д. Левченко, С.А. Хилков // Геофизика, спец. выпуск «50 лет ЦГЭ». — 2017. — С. 89–97.

[8] Perepelkina A.Yu., Goryachev I.A., Levchenko V.D. CFHall Code Validation with 3D3V Weibel Instability Simulation // Journal of Physics: Conference Series. IOP Publishing. — 2013. — Vol. 441, no. 1. — P. 012014.

[9] Дмитриев А.В., Иванов А.В., Хохлов А.Р. Численное моделирование распространения света через диффузор // Фундаментальная и прикладная математика. — 2009. — Т. 15, № 6. — С. 33–41.

[10] Dmitriev A.V., Ivanov A.V., Khokhlov A.R. Numerical simulation of light propagation through a diffuser // Journal of Mathematical Sciences. — 2011. — Vol. 172, no. 6. — P. 782–787.

[11] Хилков С.А., Иванов А.В., Зипунова Е.В. Моделирование сильно неравновесных процессов в магнетиках на основе уравнений физической кинетики // Математическое моделирование. — 2016. — Т. 28, № 5. — С. 24–31.

[12] Khilkov S.A., Ivanov A.V., Zipunova E.V. Numerical simulation of strongly nonequilibrium processes in magnets based on physical kinetics equations // Mathematical Models and Computer Simulations. — 2016. — Vol. 8, no. 6. — P. 703–708.

[13] Иванов А.В., Хилков С.А. К вопросу о кинетическом моделировании цепочки фазовых осцилляторов // Препринты ИПМ им. М.В. Келдыша. — 2016. — № 130. — С. 20.

[14] Иванов А.В., Хилков С.А. Бета–аппроксимация двухчастичной функции распределения при описании цепочек фазовых осцилляторов // Препринты ИПМ им. М.В. Келдыша. — 2017. — № 87. — С. 19.

[15] Моделирование фазовых диаграмм переключения для термоассистированных наноприборов MRAM / И.М. Искандарова, А.В. Иванов, А.А. Книжник и др. // Российские нанотехнологии. — 2015. — Т. 10, № 11–12. — С. 112–117.

[16] Simulation of switching maps for thermally assisted MRAM nanodevices / I.M. Iskandarova, A.V. Ivanov, A.A. Knizhnik et al. // Nanotechnologies in Russia. — 2016. — Vol. 11, no. 3-4. — P. 208–214.

[17] A software package for computer–aided design of spintronic nanodevices / A.A. Knizhnik, I.A. Goryachev, G.D. Demin et al. // Nanotechnologies in Russia. — 2017. — Vol. 12, no. 3-4. — P. 208–217.

[18] Модель анизотропии на скомпенсированном интерфейсе кубический ферромагнетик-антиферромагнетик со структурой Cu_3Au ($L1_2$) / А.В. Иванов, Е.В. Зипунова, А.А. Книжник, А.Ф. Попков // Препринты ИПМ им. М.В. Келдыша. — 2018. — № 63. — С. 31.

[19] Khilkov S.A., Ivanov A.V. Numerical simulation of the magnetic moment distribution evolution for superparamagnetic materials // Keldysh Institute Preprints. — 2014. — no. 29. — P. 16.

[20] Хилков С.А., Иванов А.В. Резонансные свойства суперпарамагнетиков при малых амплитудах внешнего периодического поля // Математическое моделирование. — 2015. — Т. 27, № 8. — С. 96–110.

[21] Formation of the preheated zone ahead of a propagating flame and the mechanism underlying the deflagration-to-detonation transition / M.A. Liberman, M. Kuznetsov, A. Ivanov, I. Matsukov // Physics Letters A. — 2009. — Vol. 373, no. 5. — P. 501–510.

[22] Стохастическое моделирование флуктуационной стадии высокотемпературного блистеринга в слоистой среде / Г.И. Змиевская, А.Л. Бондарева, А.В. Иванов, П.А. Покаташкин // Препринты ИПМ им. М.В. Келдыша. — 2008. — № 103. — С. 28.

[23] Бондарева А.Л., Змиевская Г.И., Иванов А.В. Блистеринг на тонкой пленке и в слоистой среде // Прикладная физика. — 2009. — № 6. — С. 91–97.

[24] Zmievskaya G.I., Bondareva A.L. Kinetics of the formation of pores and a change in the properties of materials in numerical model // Journal of Surface Investigation: X-Ray, Synchrotron and Neutron Techniques. — 2016. — Vol. 10, no. 4. — P. 802–808.

[25] Иванов А.В., Савенков Е.Б. Моделирование и визуальное представление динамики поверхности с подвижным краем на стационарной неструктурированной сетке // Научная визуализация. — 2017. — Т. 9, № 2. — С. 64–81.

[26] Benzi R., Sutera A., Vulpiani A. Mechanism of stochastic resonance // J.Phys.A. — 1981. — Vol. 14. — P. 453–457.

[27] Климонтович Ю.Л. Что такое стохастическая фильтрация и стохастический резонанс? // Успехи физических наук. — 1999. — Т. 169, № 1. — С. 39–47.

[28] Landa P.S., Neimark Yu.I., McClintock P.V.E. Changes in the Effective Parameters of Averaged Motion in Nonlinear Systems Subject to Noise // Journal of Statistical Physics. — 2006. — Vol. 125, no. 3. — P. 593–620.

[29] Ланда П.С., Трубецков Д.И., Гусев В.А. Заблуждения и реальность в некоторых задачах физики (теория и эксперимент) // Успехи физических наук. — 2009. — Т. 179, № 3. — С. 255–277. —
Landa P.S., Trubetskov D.I., Gusev V.A. Delusions versus reality in some physics problems: theory and experiment. // Phys. Usp. — 2009. — V. 52., no. 3. — P. 235–255.

[30] Stocks N.G., Stein N.D., McClintock P.V.E. Stochastic resonance in monostable systems // Journal of Physics A: Mathematical and General. — 1993. — Vol. 26, no. 7. — P. L385.

[31] Климонтович Ю.Л. Статистическая теория открытых систем. — Москва : ТОО ‘‘Янус’’, 1995.

[32] Змиевская Г.И. Численные стохастические модели неравновесных процессов // Математическое моделирование. — 1996. — Т. 8, № 11. — С. 3–40.

[33] Иванов А.В., Хилков С.А. Библиотека aiwlib. — URL: https://github.com/aivn/aiwlib/blob/master/doc/aiwlib.pdf.

[34] Imanbayeva A.K., Tokmyrzaeva Zh.B. The signal-to-noise ratio in bistable stochastic resonators // Recent Contributions to Physics. — 2017. — Vol. 61, no. 2. — P. 101–107. — URL: http://bph.kaznu.kz/index.php/zhuzhu/article/view/544.

[35] Lai Zhi-Hui, Leng Yong-Gang. Generalized Parameter-Adjusted Stochastic Resonance of Duffing Oscillator and Its Application to Weak-Signal Detection // Sensors. — 2015. — Vol. 15, no. 9. — P. 21327–21349. — URL: http://www.mdpi.com/1424-8220/15/9/21327.

[36] Stroescu Ion, Hume David B., Oberthaler Markus K. Dissipative Double-Well Potential for Cold Atoms: Kramers Rate and Stochastic Resonance // Phys. Rev. Lett. — 2016. — Dec. — Vol. 117. — P. 243005. — URL: https://link.aps.org/doi/10.1103/PhysRevLett.117.243005.