Руководство по VisualAivika Solver

Сорокин Давид Эрнестович <davsor@mail.ru>,
Россия, Республика Марий Эл, Йошкар-Ола

16 октября 2024 г.

Оглавление

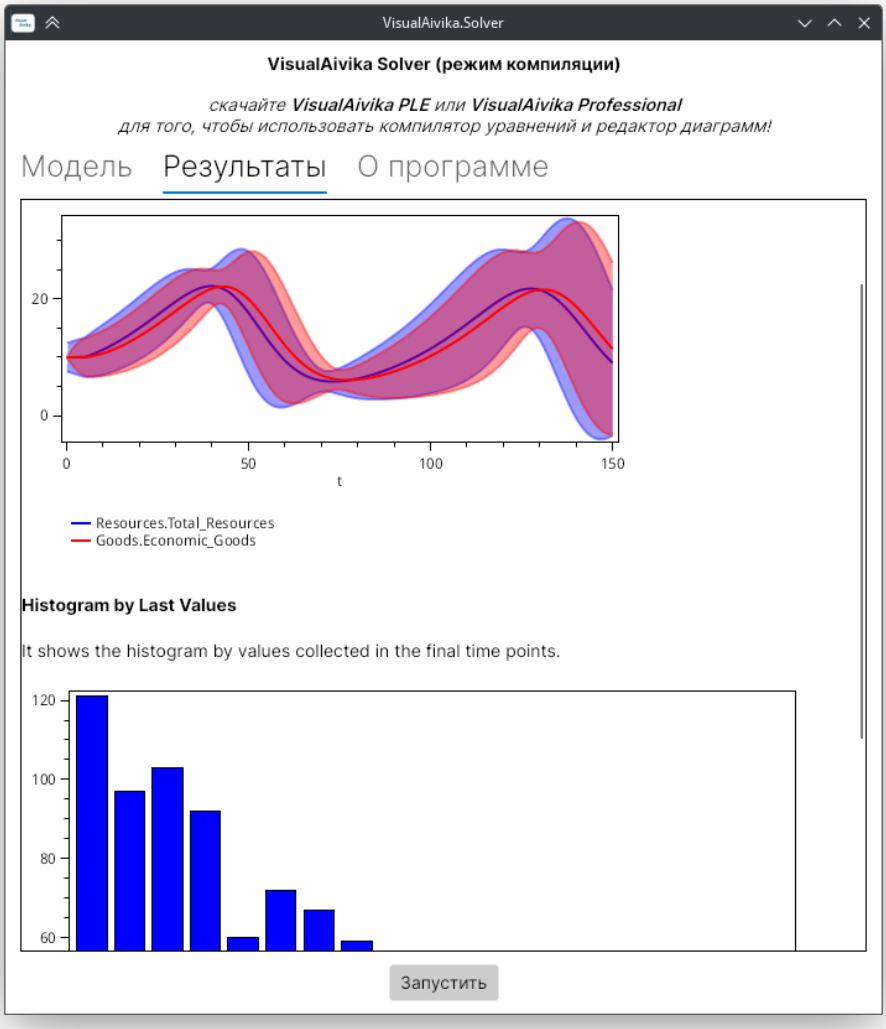

Рис. 1: Отображение результатов моделирования в VisualAivika Solver.

Глава 1

Введение

VisualAivika Solver — это бесплатный инструмент моделирования для численного интегрирования моделей системной динамики, которые по сути являются системами обыкновенных дифференциальных уравнений (ОДУ).

VisualAivika Solver имеет простой интерфейс GUI, где можно задать следующую модель и эксперимент, как демонстрирует рисунок 1.1.

```
starttime = 0;
stoptime = 10;
dt = 0.01;

ka = 1;
kb = 1;

A = integ(-ka * A, 100);
B = integ(ka * A - kb * B, 0);
C = integ(kb * B, 0);

experiment {
    timeSeries {
        series = [A, B, C];
        width = 500;
        height = 300;
    }

    csvTable {
        series = [time, A, B, C];
        width = 500;
        height = 300;
    }
}
```

Эти уравнения соответствуют следующей математической системе

уравнений, описанных в 5-Minute Tutorial из Berkeley-Madonna[1].

$$\begin{aligned}
\dot{a} &= -ka \times a, \qquad a(t_0) = 100, \\
\dot{b} &= ka \times a - kb \times b, \qquad b(t_0) = 0, \\
\dot{c} &= kb \times b, \qquad c(t_0) = 0, \\
ka &= 1, \\
kb &= 1.
\end{aligned}$$

Мы задаем параметры моделирования, которые включают в себе предопределенные переменные: `starttime`, `stoptime` и `dt`. Они задают начальное время, конечной время и временной шаг интегрирования, соответственно.

Затем мы определяем параметры уравнений `ka` и `kb`, после чего следуют три дифференциальных уравнения. Они используют функцию `integ`, которая определяет интеграл по заданной производной и начальному значению.

В конце мы определяем вычислительный эксперимент, который содержит "представления". Каждое представление задает то, как мы желаем видеть результаты моделирования. Здесь мы определяем два представления. Первое предназначено для отображения графика временного ряда (*англ.* Time Series). Второе предназначено для отображения данных CSV, которые затем легко могут быть скопированы и экспортированы в другое приложение или сохранены в файл.

После нажатия на кнопку *Запустить* (*англ.* Run) активируется вкладка *Результаты*, где мы можем просматривать результаты моделирования с помощью тех представлений, которые мы определили ранее во время задания вычислительного эксперимента во вкладке *Модель*. В случае нашей модели мы получим результаты, которые показаны на рисунке 1.2.

Здесь мы запустили только один вычислений прогон, хотя мы можем запустить множество имитационных прогонов в рамках одного и того же эксперимента.

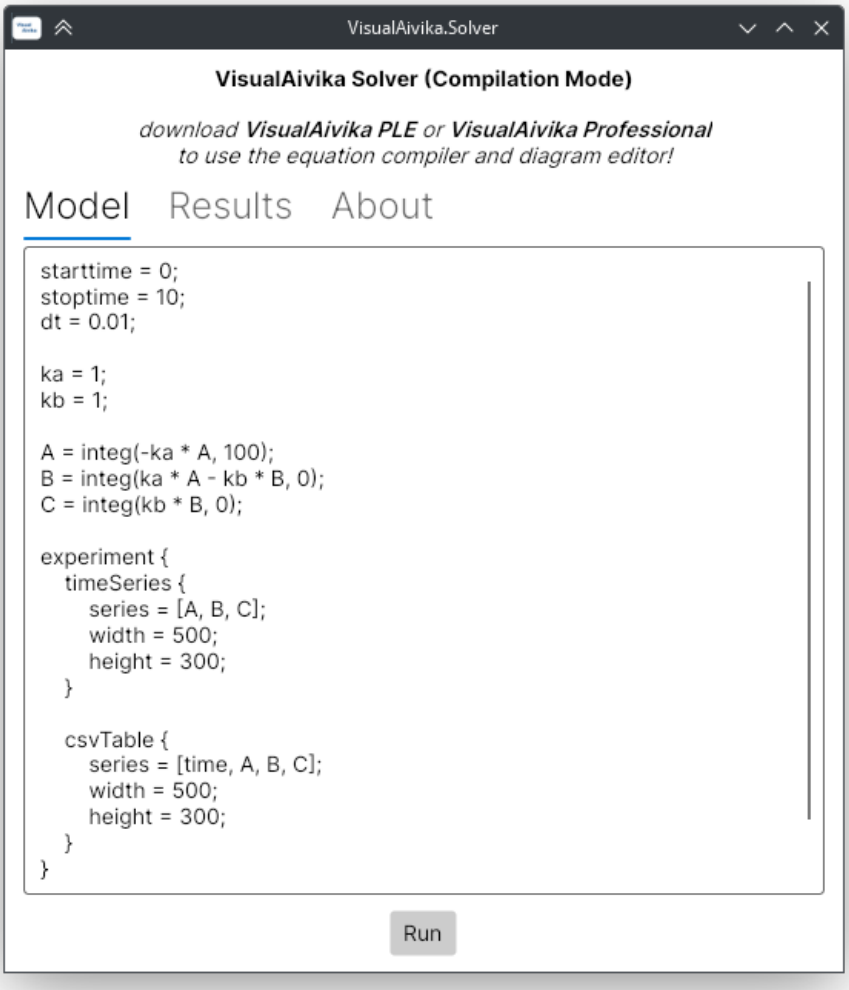

Рис. 1.1: Простейшая система дифференциальных уравнений.

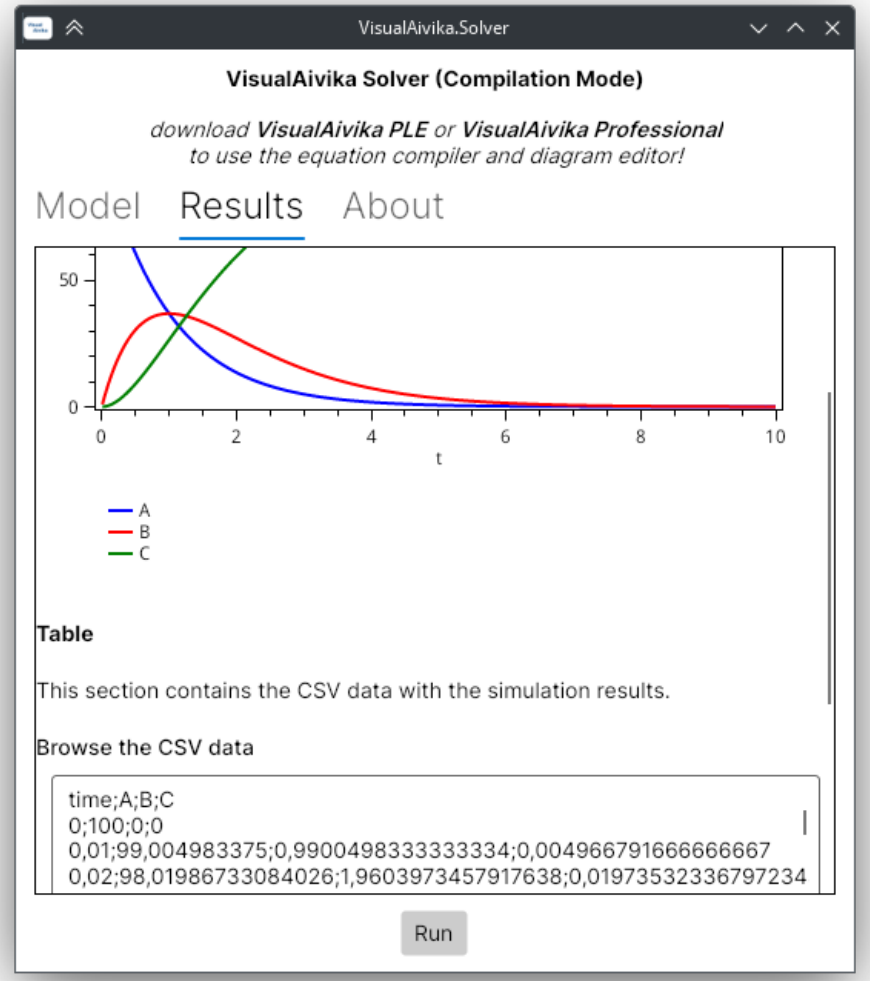

Рис. 1.2: Результаты, соответствующие системе обыкновенных дифференциальных уравнений.

Глава 2

Эксперимент по методу Монте-Карло

VisualAivika Solver поддерживает проведение вычислительных экспериментов по методу Монте-Карло.

Давайте возьмем те же уравнения из предыдущей главы и добавим другую предопределенную переменную runCount, как показано на рисунке 2.1.

```
starttime = 0;
stoptime = 10;
dt = 0.01;

runCount = 3;

ka = 1;
kb = 1;

A = integ(-ka * A, 100);
B = integ(ka * A - kb * B, 0);
C = integ(kb * B, 0);

experiment {
    timeSeries {
        series = [A, B, C];
        width = 500;
        height = 300;
    }

    csvTable {
        series = [time, A, B, C];
        width = 500;
        height = 300;
```

```
    }
}
```

Здесь мы сказали, что эксперимент Монте-Карло будет иметь три имитационных прогона (запуска). Действительно, после нажатия на кнпоку *Запустить* мы увидим, что вкадка *Результаты* изменилась, что демонстрирует рисунок 2.2. Сейчас мы видим тройные графики и тройные таблицы с данными CSV, по одному на каждый запуск.

Не очень полезно иметь идентичиные графики и данные CSV. Мы можем изменить это, сделав параметры `ka` и `kb` случайными, которые бы менялись случайно для каждого имитационного запуска. Параметры были бы постоянными в рамках запуска, но каждый запуск мог бы иметь разные значения этих параметров.

Чтобы добавить случайности, давайте заменим уравнения для этих параметров:

```
ka = 1 + randomParam(-0.3, 0.3);
kb = 1 + randomParam(-0.2, 0.4);
```

Сейчас `ka` имеет равномерное распределение в интервале [0.7; 1.3], тогда как `kb` распределено равномерно в интервале [0.8, 1.4]. Мы могли бы задать те же самые интервалы непосредственно без использования оператора сложения.

Если мы перезапустим вычислительный эксперимент, то мы можем заметить, что графики и таблицы CSV изменятся, и они будут разными для каждого запуска.

Однако если мы захотим запустить 10000 вычислительных прогонов, то тогда будет не очень продуктивным иметь такие представления, которые отображают временные графики и данные CSV. К счастью, существует представление для графика отклонения (*англ.* Deviation Chart), которое является идеальным для вычислительного эксперимента по методу Монте-Карло. График отклонения показывает тренд и доверительные интервалы по правилу трех сигм (на основе неравенства Чебышёва).

Давайте запустим 10000 имитационных прогонов и отобразим график отклонения. Для этого нам придется поменять определение эксперимента, как показано на рисунке 2.3.

```
starttime = 0;
stoptime = 10;
dt = 0.01;

runCount = 10000;
```

```
ka = 1 + randomParam(-0.3, 0.3);
kb = 1 + randomParam(-0.2, 0.4);

A = integ(-ka * A, 100);
B = integ(ka * A - kb * B, 0);
C = integ(kb * B, 0);

experiment {
    deviationChart {
        series = [A, B, C];
        width = 500;
        height = 300;
    }
}
```

Результаты отображены на рисунке 2.4. Мы можем видеть, что система уравнения довольно стабильна. Она сходится независимо от небольших возмущений параметров.

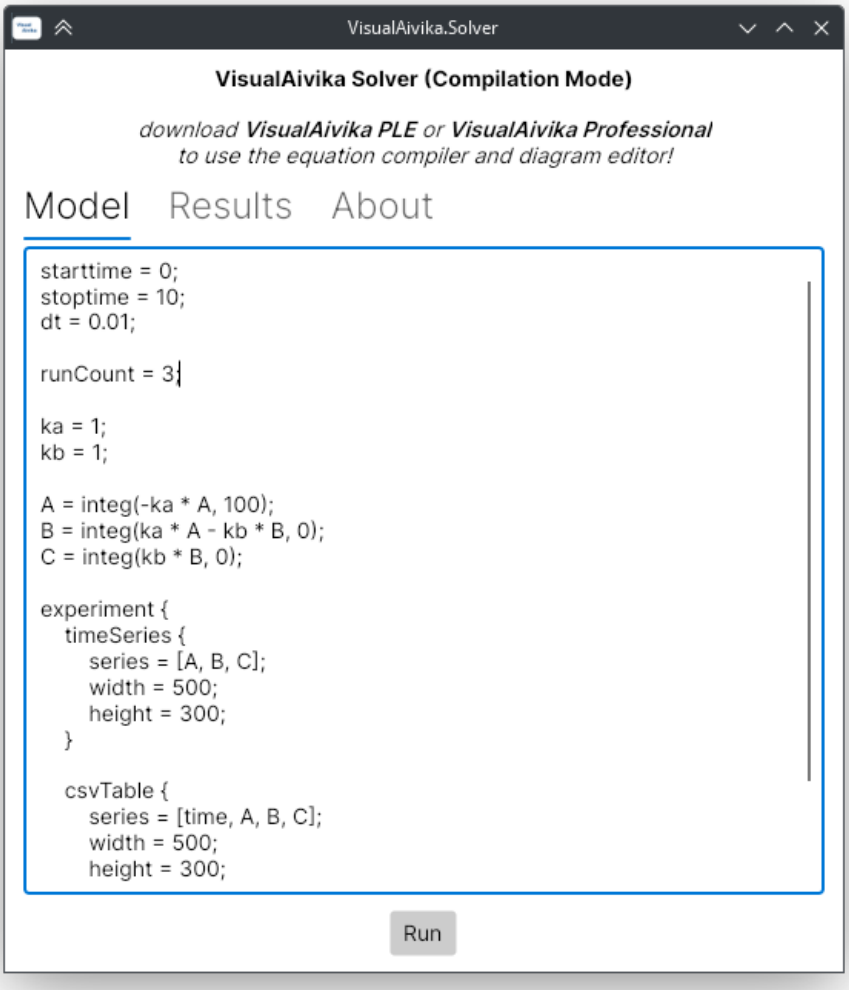

Рис. 2.1: Простейший эксперимент по методу Монте-Карло на основе системы обыкновенных дифференциальных уравнений.

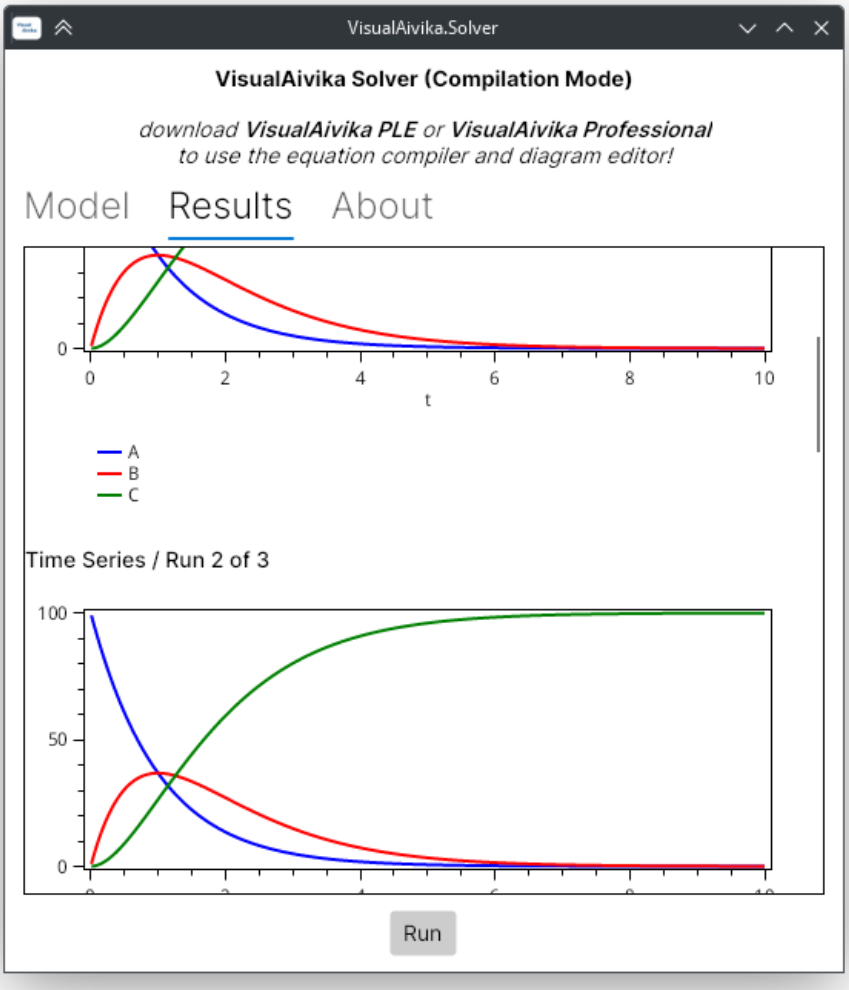

Рис. 2.2: Результаты простейшего эксперимента по методу Монте-Карло.

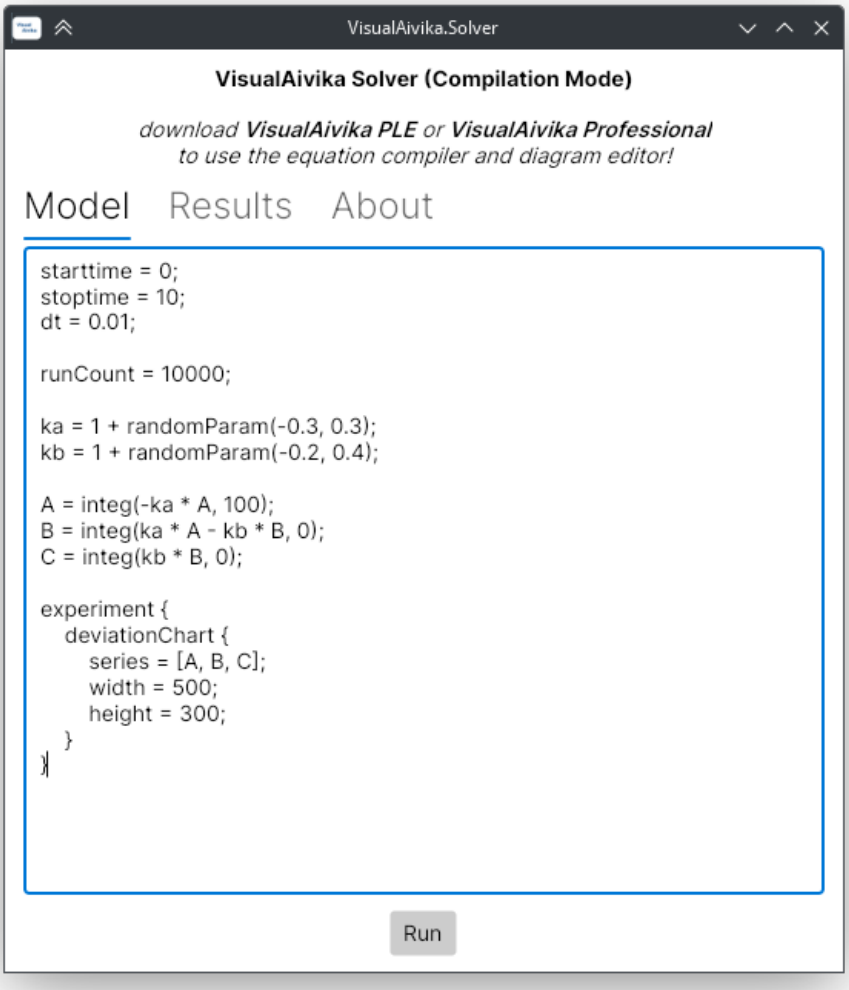

Рис. 2.3: Мы задаем эксперимент по методу Монте-Карло с 10000 имитационными запусками.

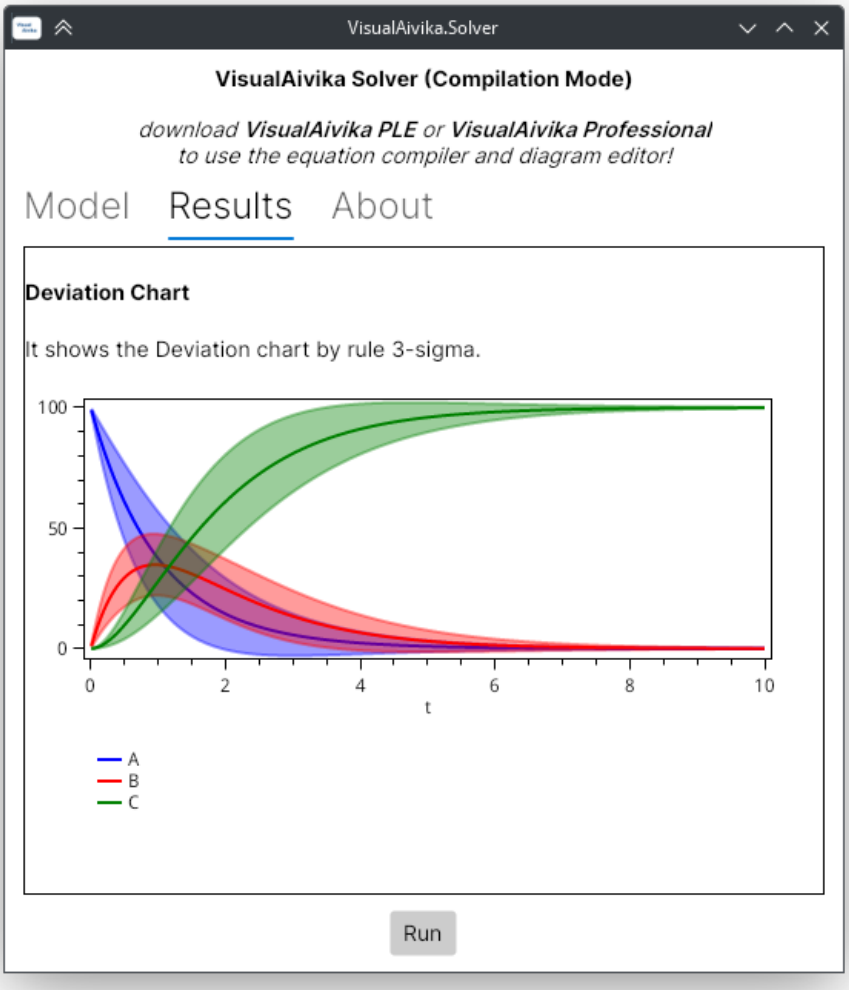

Рис. 2.4: Тренды и доверительные интервалы, полученные с помощью эксперимента по методу Монте-Карло.

Глава 3

Язык моделирования

Язык моделирования VisualAivika позволяет вам задавать динамические системы. Модель может состоять из накопителей и дополнительных переменных. Переменная накопителя может быть интегралом (резервуаром). Дополнительная переменная — это функция от других переменных.

3.1 Параметры моделирования

Заданы предопределенные переменные, которые существуют в каждой имитации:

- `starttime` определяет начальное время;
- `stoptime` задает конечное время;
- `dt` определяет шаг интегрирования по модельному времени;
- `time` возвращает текущее модельное время.

Первые три переменные могут заданы явно для каждой модели.

Текущая версия VisualAivika Solver поддерживает метод Рунге-Кутта четвертого порядка для численного интегрирования уравнений.

Более того, вы можете определить число запусков для вычислительного эксперимента по методу Монте-Карло, который отражается на следующих предопределенных переменных:

- `runIndex` возвращает индекс текущего запуска имитации, начиная с 0;

- `runCount` возвращает общее количество имитационных запусков в рамках заданного вычислительного эксперимента.

Эти две встроенные переменные полезны для планирования вычислительного эксперимента при проведении анализа чувствительности, как показано ниже в разделе 3.9.

Переменная `runCount` может быть задана явна.

3.2 Переменные

Следующий список определяет типы переменных в VisualAivika.

- *Дополнительная переменная*. Это любая переменная, которая определена как функция от других переменных, или постоянная.
- *Накопитель*. Это динамическая переменная в модели. В текущей версии такая переменная может быть только резервуаром (интегралом).
- *Поток*. Это переменная, которая влияет на накопитель. Поток может быть либо входящим, либо исходящим. Также он может быть либо двунаправленным, либо однонаправленным.
- *Таблица*. Это табличная функция со значениями для координат x и y.
- *Диапазон*. Это диапазон индексов для массивов.
- *Массив*. Массив из других переменных.

Название переменной в VisualAivika чувствительно к регистру. Первый символ должен быть буквой или подчеркиванием. Следующие символы могут содержать буквы, цифры и подчеркивания в любой последовательности.

Пример

```
Total_Resources, Scope, x, y
```

3.3 Уравнения

Язык моделирования VisualAivika позволяет вам определять модели, записывая множество математических уравнений и выражений. Уравнения могут быть записаны в любом порядке. Порядок вычислений определяется, исходя из зависимостей между переменными.

VisualAivika использует стандартные алгебраические выражения с теми же правилами приоритетов, которые используются в Java, C/C++ и других популярных языках программирования.

Также VisualAivika поддерживает стандартные математические функции, а также имеет дополнительные функции, которые делают процесс написание уравнений более быстрым и простым.

Пример

```
y = integ (1 + 0.2 * y * sin (t) - 1.5 * t ^ 2, 0);
z = integ ((y - z)^2, 0);
```

Здесь функция `integ` возвращает интеграл по заданной производной и начальному значению.

Пример

```
Adequacy_of_Control_Resources =
  Control_Resources / Desired_Control_Resources;
```

Каждое уравнение должно заканчиваться символом точки с запятой.

3.4 Операторы

VisualAivika поддерживает стандартные двоичные и унарные операторы, которые следуют обычным правилам приоритетов. Операторы приведены в таблицах ниже.

Таблица 3.1: Унарные операторы

`not`	Логическое отрицание
`+ -`	Знак плюса и минуса

Таблица 3.2: Двоичные операторы

`^`	Возведение в степень
`* / + -`	Арифметические операторы
`< <= > >=`	Сравнение
`== !=`	Сравнение на равенство и неравенство
`and or`	Логическая конъюнкция и дизъюнкция с правилом короткого вычисления

3.5 Постоянные

VisualAivika определяет следующие постоянные.

Таблица 3.3: Встроенные постоянные

`true`	Логическая ”истина”
`false`	Логическая ”ложь”
`pi`	Значение числа π
`infinity`	Представляет положительную бесконечность
`nan`	Представляет значение, которое не является числом

3.6 Функции

Ниже в таблице приведен список основных предопределенных функций.

Таблица 3.4: Основные функции

`abs (x)`	Возвращает абсолютное значение для `x`
`sqrt (x)`	Возвращает квадратный корень от `x`
`int (x)`	Возвращает наибольше целое число, которое меньше или равно, чем `x`
`round (x)`	Возвращает ближайшее число к `x`
`power (x, y)`	Возвращает то же самое, что и x^y
`mod (x, y)`	Возвращает остаток от деления `x/y`

`min (x1, x2, ...)`	Вовращает минимальное значение из `x1`, `x2`, ...
`max (x1, x2, ...)`	Возвращает максимальное значение из `x1`, `x2`, ...
`sum (x1, x2, ...)`	Возвращает сумму значений `x1`, `x2`, ...
`mean (x1, x2, ...)`	Возвращает среднее значение для `x1`, `x2`, ...
`sin (x)`	Возвращает значение синуса для `x`
`cos (x)`	Возвращает косинус для `x`
`tan (x)`	Возвращает тангенс для `x`
`arcsin (x)`	Возвращает арксинус для `x`
`arccos (x)`	Возвращает арккосинус для `x`
`arctan (x)`	Возвращает арктангенс для `x`
`arctan (y, x)`	Возвращает арктангенс для отношения `(y/x)`
`sinh (x)`	Возвращает гиперболический синус для `x`
`cosh (x)`	Возвращает гиперболический косинус для `x`
`tanh (x)`	Возвращает гиперболический тангенс для `x`
`sinwave (a, t)`	Возвращает синусоидальную волну с амплитудой `a` и периодом `t`
`coswave (a, t)`	Возвращает косинусоидальную волну с амплитудой `a` и периодом `t`
`exp (x)`	Возвращает e в степени `x`
`log (x)`	Возвращает натуральный логарифм от `x` (по основанию e)
`log (x, y)`	Возвращает логарифм от `x` по основанию `y`

Функции `sinwave` и `coswave` требуют некоторого пояснения. Они определены следующим образом.

Определение

```
sinwave(a, t) = a * sin(2 * pi * time / t);
coswave(a, t) = a * cos(2 * pi * time / t);
```

Таблица 3.5: Генераторы случайных чисел

`random (a, b)`	Возвращает равномерно распределенное случайное число между `a` и `b`
`normal (m, n)`	Возвращает нормально распределенное случайное число со средним `m` и отклонением `n`
`binomial (p, n)`	Возвращает биномиальное случайное число при количестве попыток `n` с вероятностью `p`
`poisson (m)`	Возвращает случайное число по распределению Пуассона со средним `m`

Таблица 3.6: Условное выражение

`if x then y else z`	Возвращает `y`, если `x` истинно; иначе возвращает `z`

Таблица 3.7: Различные функции

`step (h, t)`	Возвращает 0 до времени `t`, а затем возвращает `h`
`pulse (t, d)`	Возвращает импульс высоты 1, начиная со времени `t` и длительностью `d`
`pulse (t, d, p)`	Возвращает импульс высоты 1, начиная со времени `t`, длительностью `d` и периодом `p`
`ramp (s, t1, t2)`	Возвращает 0 до времени `t1`, а затем наклоняется ко времени `t2` с коэффициентом наклона `s`

Эти функции имеют следующее определение. Они используют описанный далее оператор `discrete`. Если кратко, этот оператор возвращает такое значение, которое не меняется за исключением интервалов интегрирования по dt независимо от того, каким бы ни был используемый метод интегрирования.

Определение

```
step(h, t) = discrete(if time + dt/2 > t then h else 0);

pulse(t, d) = discrete(if (time + dt/2 > t)
                         and (time + dt/2 < t + d)
                       then 1 else 0);

pulse(t, d, p) = pulse(t + (if (p > 0) and (time > t)
                            then round((time - t) / p)*p
                            else 0), d);

ramp(s, t1, t2) = discrete(if (time + dt/2 > t1)
                           then (if (time - dt/2 < t2)
                                 then s*(time - t1)
                                 else s*(t2 - t1))
                           else 0);
```

Таблица 3.8: Интерполяция

`table ((x1, y1), (x2, y2), ...)`	Создает таблицу, состоящую из точек `(x1, y1)`, `(x2, y2)`, ..., где таблица потом может быть использована как функция
`table ((x1, y1), (x2, y2), ...) (x)`	Оценивает значение `y` для заданного `x` по списку точек `(x1, y1)`, `(x2, y2)`, ..., используя линейную интерполяцию
`step (t)`	Создает функцию дискретной ступенчатой интерполяции на основе заданной таблицы `t`
`step (table ((x1, y1), (x2, y2), ...)) (x)`	Оценивает значение `y` для заданного `x` по списку точек `(x1, y1)`, `(x2, y2)`, ..., используя дискретную ступенчатую интерполяцию

Табличные функции могут быть определены непосредственно в уравнениях.

Пример

```
Effect_of_Scope_Stability_of_Rules =
  table ((0, 0.5), (0.2, 0.535), (0.4, 0.585), (0.6, 0.675),
         (0.8, 0.82), (1, 1), (1.2, 1.21), (1.4, 1.35),
         (1.6, 1.42), (1.8, 1.47), (2, 1.5))
        (Scope);
```

Таблица 3.9: Функции интегралов

`integ (d, i)`	Возвращат интеграл с производной `d` и начальным значением `i`
`delay (x, t)`	Возвращает экспоненциальную задержку первого порядка `x` на время `t` с сохранением `x`
`delay (x, t, i)`	Возвращает экспоненциальную задержку первого порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`delay3 (x, t)`	Возвращает экспоненциальную задержку третьего порядка `x` на время `t` с сохранением `x`
`delay3 (x, t, i)`	Возвращает экспоненциальную задержку третьего порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`delayN (x, t, n)`	Возвращает экспоненциальную задержку `n`-го порядка `x` на время `t` с сохранением `x`
`delayN (x, t, n, i)`	Возвращает экспоненциальную задержку `n`-го порядка `x`, начиная с `i` и на время `t` с сохранением `x`
`smooth (x, t)`	Возвращает экспоненциальное сглаживание первого порядка `x` за время `t`
`smooth (x, t, i)`	Возвращает экспоненциальное сглаживание первого порядка `x` за время `t`, начиная с `i`
`smooth3 (x, t, i)`	Возвращает экспоненциальное сглаживание третьего порядка `x` за время `t`
`smooth3 (x, t, i)`	Возвращает экспоненциальное сглаживание третьего порядка `x` за время `t`, начиная с `i`

`smoothN (x, t, n)`	Возвращает экспоненциальное сглаживание `n`-го порядка `x` за время `t`
`smoothN (x, t, n, i)`	Возвращает экспоненциальное сглаживание `n`-го порядка `x` за время `t`, начиная с `i`
`forecast (x, t, h)`	Прогноз для `x` на временном горизонте `h` с использованием среднего значения времени `t`
`trend (x, t, i)`	Возвращает дробную скорость изменения `x`, используя среднее время `t` и начиная с `i`

Здесь функции из семейства `delay` используют следующую идею, где функции `delayN` являются обобщением правила. Если вы будете сравнивать эти функции с другими программными инструментами моделирования, то имейте в виду, что функции `delayN` не имеют эффекта оператора `discrete`, который неявно может присутствовать в других инструментах моделирования. В случае необходимости этот оператор может быть вручную добавлен в уравнения.

Определение

```
D1=delay(x, t) то же самое, что и
  D1=1/t * integ(x - D1, x*t);

D1I=delay(x, t, i) то же самое, что и
  D1I=1/t * integ(x - D1I, i*t);

D3_3=delay3(x, t) то же самое, что и
  D3_3=1/(t/3) * integ(D3_2 - D3_3, x*t/3);
  D3_2=1/(t/3) * integ(D3_1 - D3_2, x*t/3);
  D3_1=1/(t/3) * integ(x - D3_1, x*t/3);

D3I_3=delay3(x, t, i) то же самое, что и
  D3I_3=1/(t/3) * integ(D3I_2 - D3I_3, i*t/3);
  D3I_2=1/(t/3) * integ(D3I_1 - D3I_2, i*t/3);
  D3I_1=1/(t/3) * integ(x - D3I_1, i*t/3);
```

Например, рисунок 3.1 показывает функции задержки первого и третьего порядка для следующего входа и временного интервала.

Пример

```
x=sin(time);
t=40*dt;
dt=0.025;
```

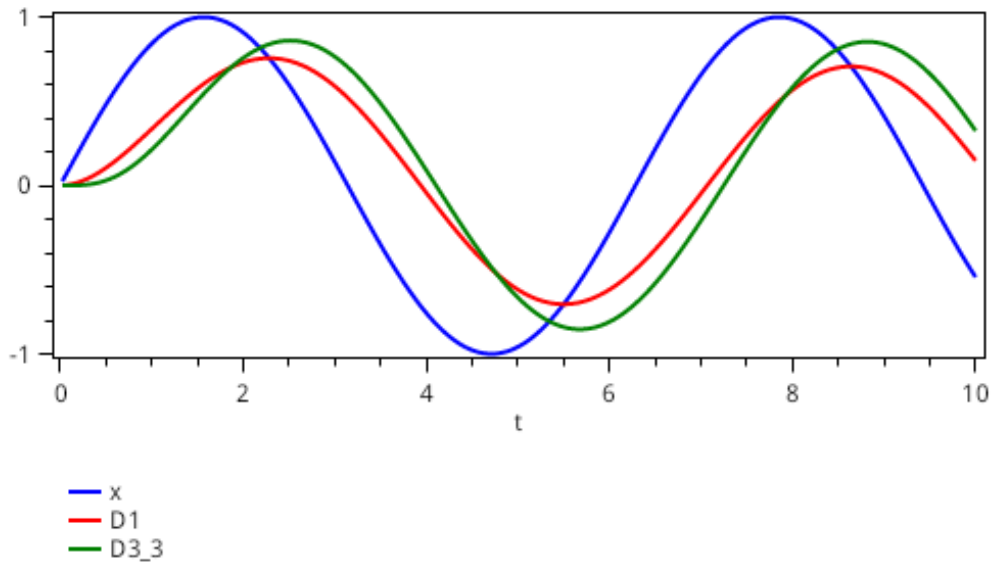

Рис. 3.1: Задержка первого порядка, D1, и задержка третьего порядка, D3_3, для заданного входа, x, и периода времени.

Функции из семейства smooth используют похожую идею, где функции smoothN являются обобщением правила. Если вы будете сравнивать эти функции с другими программными инструментами моделирования, то имейте в виду, что функции smoothN не имеют эффекта оператора discrete, который неявно может присутствовать в других инструментах моделирования. В случае необходимости этот оператор может быть вручную добавлен в уравнения.

Определение

```
S1=smooth(x, t) то же самое, что и
  S1=integ((x - S1)/t, x);

S1I=smooth(x, t, i) то же самое, что и
  S1I=integ((x - S1I)/t, i);
```

```
S3_3=smooth3(x, t) то же самое, что и
  S3_3=integ((S3_2 - S3_3)/(t/3), x);
  S3_2=integ((S3_1 - S3_2)/(t/3), x);
  S3_1=integ((x - S3_1)/(t/3), x);

S3I_3=smooth3(x, t, i) то же самое, что и
  S3I_3=integ((S3I_2 - S3I_3)/(t/3), i);
  S3I_2=integ((S3I_1 - S3I_2)/(t/3), i);
  S3I_1=integ((x - S3I_1)/(t/3), i);
```

Функции `delay` и `smooth` ведут себя по-разному, когда период времени начинает меняться.

Функции `forecast` и `trend` имеют следующее определение, где оператор `init` возвращает значение выражения в начальной точке моделирования.

Определение

```
forecast(x, t, h) = x * (1 + (x / smooth(x, t) - 1) / t * h);

trend(x, t, i) = (x / smooth(x, t,
                   init(x) / (1 + i * init(t))) - 1) / t;
```

Таблица 3.10: Финансовые функции

`npv (s, r, i, f)`	Возвращает чистую приведенную стоимость (NPV) потока `s`, вычисленную по заданной ставке дисконта `r`, начальному значению `i` и некоторому коэффициенту `f` (обычно 1)
`npve (s, r, i, f)`	Возвращает конечную чистую приведенную стоимость (NPVE) потока `s`, вычисленную по заданной ставке дисконта `r`, начальному значению `i` и некоторому коэффициенту `f`

В VisualAivika финансовые функции имеют следующее определение.

Определение

```
npv=npv(s, r, i, f) то же самое, что и

  npv = (accum + dt * s * df) * f;
```

```
  df = integ(- df * r, 1);
  accum = integ(s * df, i);

npve=npve(s, r, i, f) то же самое, что и

  npve = (accum + dt * s * df) * f;
  df = integ(- df * r / beta, 1 / beta);
  accum = integ(s * df, i);
  beta = 1 + r * dt;
```

Таблица 3.11: Операторы моделирования

`discrete (x)`	Возвращает значение `x`, которое не меняется за исключением интервалов интегрирования `dt` независимо от того, каким бы ни был используемый метод интегрирования
`init (x)`	Возвращает значение выражения `x` в начальной точке моделирования

В отличие от других программных инструментов моделирования VisualAivika имеет довольно нестандартные операторы `discrete` и `init`. Они связаны с самим движком моделирования в VisualAivika. Любое числовое выражение может иметь начальное значение. Также мы можем трактовать любое числовое выражение как нечто, что менялось бы, как если бы использовался метод Эйлера. Обычно вам не следует применять эти операторы в ваших моделях. Более того, последние два оператора могут быть специфичными для VisualAivika, и они могут быть плохо переносимыми между разными программными инструментами моделирования.

3.7 Диапазоны

Диапазон задается двумя целочисленными выражениями: нижний индекс диапазона и его верхний индекс. Выражения должны быть ограничены двумя точками.

Пример

```
N = 10;

I_Range = 1..N;
J_Range = 1..5;
```

Диапазоны могут быть заданы подобно другим переменным, а затем использованы в уравнениях. Они нужны для массивов.

3.8 Массивы

Для определения одномерного массива вы можете использовать особый синтаксис:

[*Element* | *Index* <- *Range*]

Здесь выражение элемента может зависеть от индекса, чьи значения будут получены из заданного диапазона.

Синтаксис обобщен также для других размерностей. Например, двумерный массив может быть определен после добавления другого индекса:

[*Element* | *Index1* <- *Range1* , *Index2* <- *Range2*]

VisualAivika поддерживает до 5 размерностей.

Чтобы сослаться на элемент одномерного массива по заданному индексу, вы можете использовать следующую конструкцию:

Array[*Index*]

Похожим образом можно ссылаться на элементы двумерного массива по двум индексам:

Array2D[*Index1* , *Index2*]

Это правило также обобщается на случай других размерностей.

Пример

```
n = 51;

C = [ if (i == 0) or (i == n + 1) then 0 else M[i] / v |
      i <- 0..n+1 ];

M = [ integ (q + k*(C[i-1] - C[i]) + k*(C[i + 1] - C[i]), 0) |
      i <- 1..n ];

q = 1;
k = 2;
v = 0.75;
```

В этом примере C и M являются одномерными массивами с разными индексами. Массив C имеет значения `C[0]`, ..., `C[n+1]`, тогда как массив M имеет значения `M[1]`, ..., `M[n]`.

Между прочим, диапазоны могли быть определены как отдельные переменные. Для простоты здесь диапазоны определены внутри массивов. Применимы оба метода.

3.8.1 Функции для массивов и диапазонов

Для массивов и диапазонов определены следующие функции.

Таблица 3.12: Функции для массивов и диапазонов

`length (a)`	Возвращает общее количество элементов для заданного диапазона или массива `a`
`length (a, d)`	Возвращает количество элементов для заданного массива `a` и размерности `d`, начиная с 0
`low (a)`	Возвращает нижний индекс для заданного диапазона или одномерного массива `a`
`low (a, d)`	Возвращает нижний индекс для заданного массива `a` и размерности `d`, начиная с 0
`high (a)`	Возвращает верхний индекс для заданного диапазона или одномерного массива `a`
`high (a, d)`	Возвращает верхний индекс для заданного массива `a` и размерности `d`, начиная с 0
`min (a)`	Возвращает минимальный элемент заданного массива `a`
`max (a)`	Возвращает максимальный элемент заданного массива `a`
`sum (a)`	Возвращает сумму элементов для заданного массива `a`
`prod (a)`	Возвращает произведение элементов для заданного массива `a`
`mean (a)`	Возвращает среднее значение для элементов заданного массива `a`

Последние функции являются агрегирующими. Полезно, что мы можем передать таким функциям временные массивы, которые сами являются результатом выражений.

Следующий пример эквивалентен примеру из [Vensim 5 Reference

Manual, страница 30, пример 3] документации к Vensim[2].

Пример

```
efficiency = prod(factor_efficiency);
US_population = sum(population);
revenue = [ sum([ sales[c, p] * price[p, b] | p <- product ]) |
              c <- country, b <- brand ];
```

Пожалуйста, обратите внимание на то, как создается временный массив для суммирования элементов в массиве-переменной revenue.

Следующий пример относится к теории игр. Он вычисляет значения максимина и минимакса по заданной матрице размерности $n \times n$, соответственно.

Пример

```
max_min = max([ min([ A[i,j] | j <- 1..n ]) | i <- 1..n ])
min_max = min([ max([ A[i,j] | j <- 1..n ]) | i <- 1..n ])
```

3.8.2 Инициализация массива

Некоторые массивы могут быть заданы в табличной форме без явного указания диапазонов и индексов. Существует упрощенный синтаксис для этого.

Пример

```
A1 = [0, 1, 2, 3, 4, 5];
A2 = [[0, 1], [2, 3], [4, 5]];
A3 = [[[0, 1], [2, 3]], [[4, 5], [6, 7]]];
```

Только такие массивы всегда имеют индексы, начинающиеся с 0.

3.9 Анализ чувствительности

Как было замечено выше в тексте, существуют встроенные переменные `runIndex` и `runCount`, которые возвращают индекс текущего запуска, начиная с 0, и общее количество запусков в рамках одного вычислительного эксперимента по методу Монте-Карло, соответственно.

Важно, что переменная `runIndex` является постоянной в рамках текущего имитационного запуска, а затем обновляется для другого запуска. Существуют похожие случайные параметры с тем же свойством. Они постоянны в рамках текущего запуска, а затем они переопределяются для другого запуска.

Таблица 3.13: Случайные параметры

`randomParam (a, b)`	Возвращает равномерно распределенный параметр из интервала между `a` и `b`
`normalParam (m, n)`	Возвращает нормально распределенный параметр со средним `m` и отклонением `n`
`binomialParam (p, n)`	Возвращает биномиально распределенный параметр с количеством попыток `n` и вероятностью `p`
`poissonParam (m)`	Возвращает случайный параметр по закону Пуассона со средним `m`

Такие параметры полезны для проведения анализа чувствительности. Они могут быть использованы в уравнениях.

Переопределяя некоторые постоянные как случайные внешние параметры, мы можем проверить модель на устойчивость. Для этого VisualAivika поддерживает метод Монте-Карло. Результаты такого анализа могут быть отображены на графике подобно рисунку 3.2, где используется представление для графика отклонения (*англ.* Deviation Chart).

Более того, сочетая инициализацию массивов со встроенными переменными `runIndex` и `runCount`, мы можем задать выборку для внешних параметров.

Идея довольно проста. Мы определяем массив со значениями, а затем ссылаемся на него по индексу запуска, вероятно, ограниченному размером массива.

Пример

```
A = [[1, 2, 3],
     [2, 3, 1],
     [3, 1, 2]];

Sample = mod(runIndex, length(A, 0));

X1 = A[Sample, 0];
X2 = A[Sample, 1];
X3 = A[Sample, 2];
```

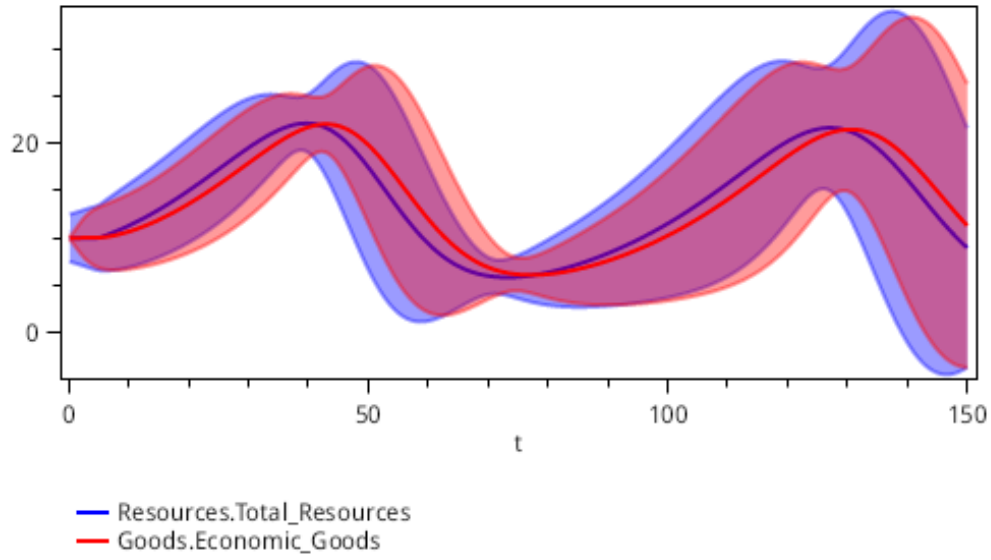

Рис. 3.2: График показывает тренды и доверительные интервалы, т.е. насколько устойчива модель относительно изменений внешних случайных араметров.

3.10 Вложенные модули

Модель может содержать модули, которые могут быть вложенными. Каждый модуль определяет свое собственное пространство имен для переменных.

VisualAivika использует модули для хранения переменных из одной и той же диаграммы в отдельном модуле.

Пример

```
// глобальная переменная
A = 1;

module M1 {

    // переменная M1.B
    B = A + 1;

    module M2 {

        // переменная M1.M2.C
        C = 2 * B;
```

```
        // использовать полное название для M1.B
        D = C - M1.B;
    }
}
```

Глава 4

Представления эксперимента

Представления эксперимента позволяют вам видеть результаты моделирования в предпочтительном виде. Это может быть график, или может быть таблица с данными CSV. Представление задает в точности то, как результаты будут представлены. Эксперимент может содержать произвольное количество представлений, перечисленных друг за другом.

4.1 График отклонения

График отклонения (*англ.* Deviation Chart) — это наиболее универсальный способ представления, который применим как к единичному запуску, так и ко множеству запусков. Если запуск единичный, то тогда график отклонения становится похожим на график временного ряда из раздела 4.2. Однако если используется вычислительный эксперимент по методу Монте-Карло, то тогда график отклонения показывает тренды и доверительные интервалы по правилу 3-х сигм.

Следующая таблица содержит опциональные свойства, которые могут быть использованы при построении графика отклонения.

Таблица 4.1: График отклонения

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при построении графика
`width =` *ширина*`;` `height =` *высота*`;`	Задает ширину графика Задает высоту графика

Пример

```
experiment {
    deviationChart {
        series = [
            Finance.net_cash_flow,
            Finance.npv_cash_flow
        ];
    }
}
```

Соответствующий график представлен на рисунке 4.1.

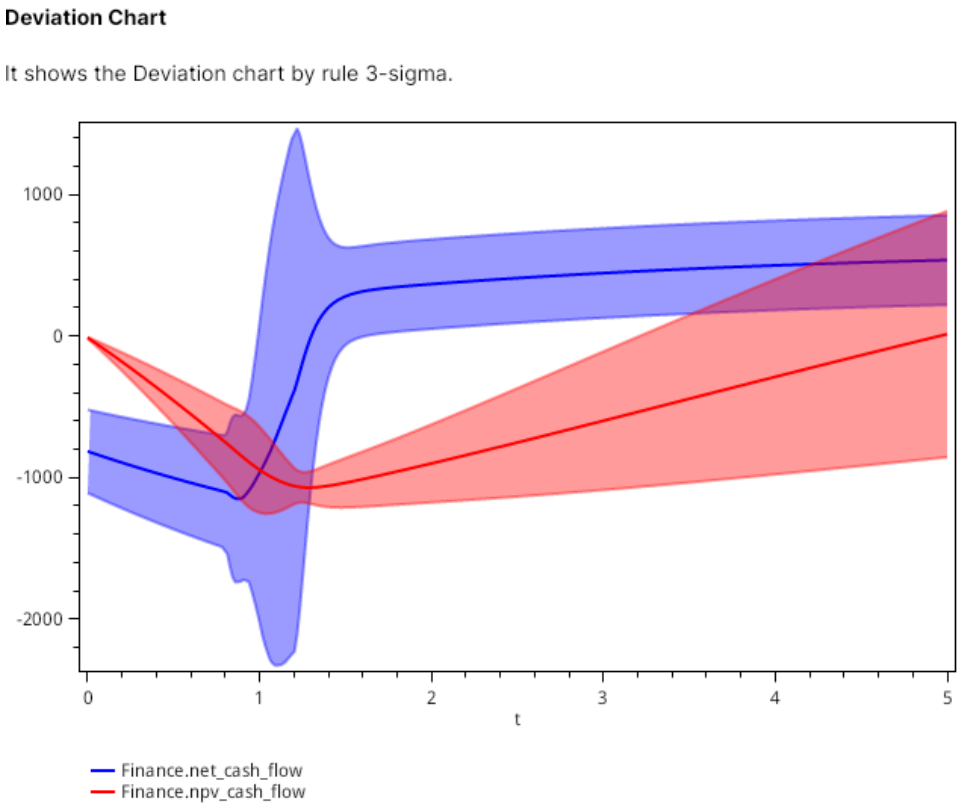

Рис. 4.1: Представление графика отклонения.

4.2 Временной ряд

График временного ряда (*англ.* Time Series) отображает обычный график, где временной ряд зависит от модельного времени. Если исполь-

зуется метод Монте-Карло, то тогда будет нарисовано соответствующее количество графиков, по одному на каждый запуск. Поэтому, пожалуйста, используйте представление временного ряда только для единичного запуска или для метода Монте-Карло с малым количеством запусков!

Следующая таблица содержит опциональные свойства, которые могут быть использованы при построении графика временного ряда.

Таблица 4.2: График временного ряда

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при построении графика
`width =` *ширина*`;` `height =` *высота*`;`	Задает ширину графика Задает высоту графика

Пример

```
x = sin(time);
y = cos(time);

experiment {
    timeSeries {
        series = [x, y];
    }
}
```

Соответствующий график представлен на рисунке 4.2.

4.3 График XY

График XY (*англ.* XY Chart) похож на график временного ряда, описанный в предыдущем разделе 4.2. Только мы задаем ряды, которые зависят от некоторого произвольного ряда. Если используется метод Монте-Карло, то тогда будет нарисовано соответствующее количество графиков, по одному на каждый запуск. Поэтому, пожалуйста, используйте представление графика XY только для единичного запуска или для метода Монте-Карло с малым количеством запусков!

Следующая таблица содержит свойства, которые могут быть использованы при отображении графика XY. Свойство `seriesX` является обязательным.

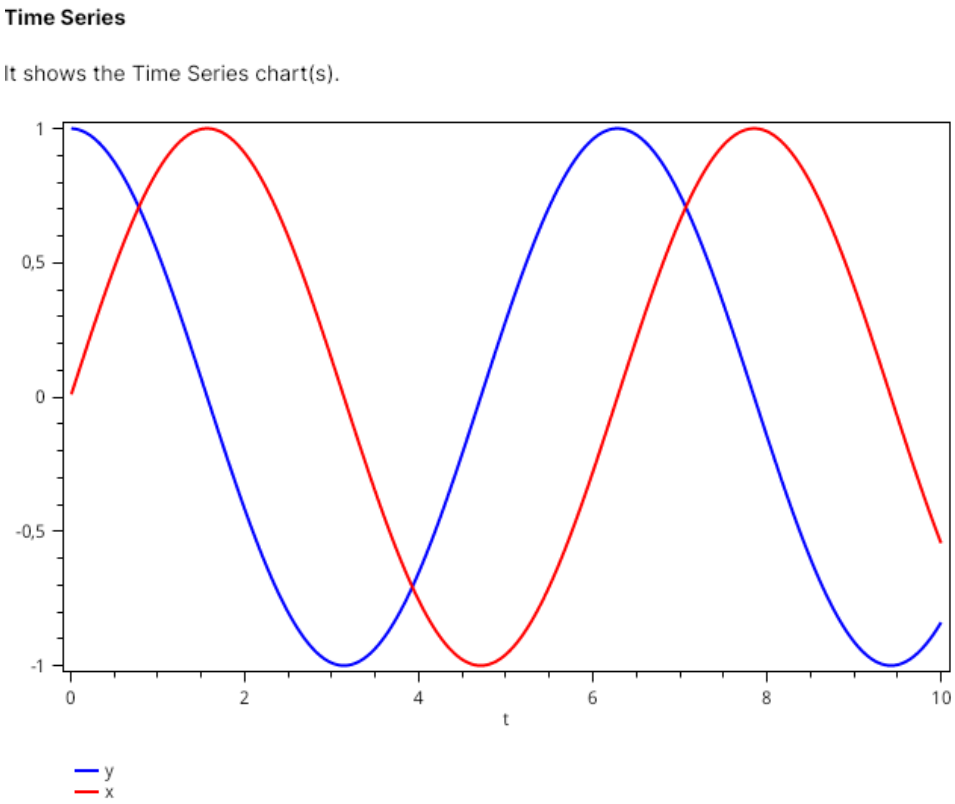

Рис. 4.2: Представление графика временного ряда.

Таблица 4.3: График XY

`seriesX =` *seriesX*`;`	Задает ряд, который используется как X
`seriesY = [`*seriesY1*, *seriesY2*, `...];`	Перечисляет названия переменных *seriesY1*, *seriesY2*, ..., которые будут использованы как Y при построении графика
`width =` *ширина*`;` `height =` *высота*`;`	Задает ширину графика Задает высоту графика

Пример

```
x = sin(time);
y = cos(time);
z = x + y;

experiment {
    xyChart {
        seriesX = x;
        seriesY = [y, z];
    }
}
```

Соответствующий график представлен на рисунке 4.3.

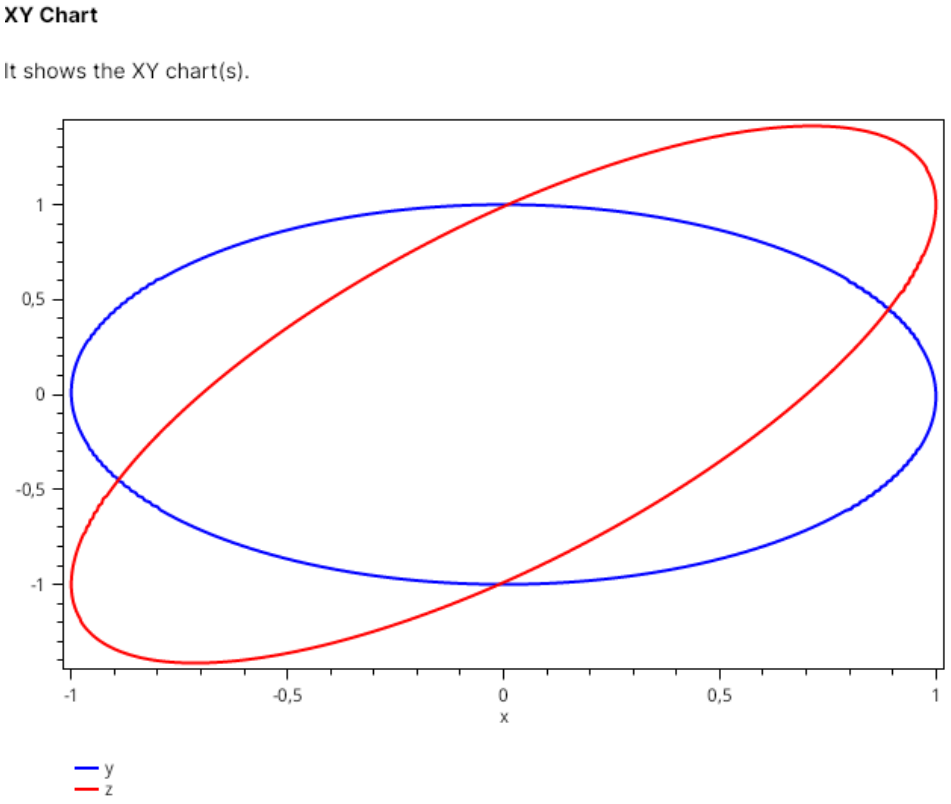

Рис. 4.3: Представление графика XY.

4.4 Таблица CSV

Таблица CSV (*англ.* CSV Table) предназначена для экспорта данных CSV в другие приложения или для сохранения данных в файл. Отображается компонент текстового блока, откуда вы можете скопировать соответствующие данные CSV. Если используется метод Монте-Карло, то тогда будет создано соответствующее количество компонентов, по одному на каждый запуск. Поэтому, пожалуйста, используйте представление таблицы CSV только для единичного запуска или для метода Монте-Карло с малым количеством запусков!

Следующая таблица содержит опциональные свойства, которые могут быть использованы при отображении таблицы CSV.

Таблица 4.4: Таблица CSV

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при отображении таблицы
`width =` *ширина*`;`	Задает ширину компонента таблицы
`height =` *высота*`;`	Задает высоту компонента таблицы

Пример

```
x = sin(time);
y = cos(time);

experiment {
    timeSeries {
        series = [x, y];
    }

    csvTable {
        series = [time, x, y];
    }
}
```

Соответствующая таблица представлена на рисунке 4.4.

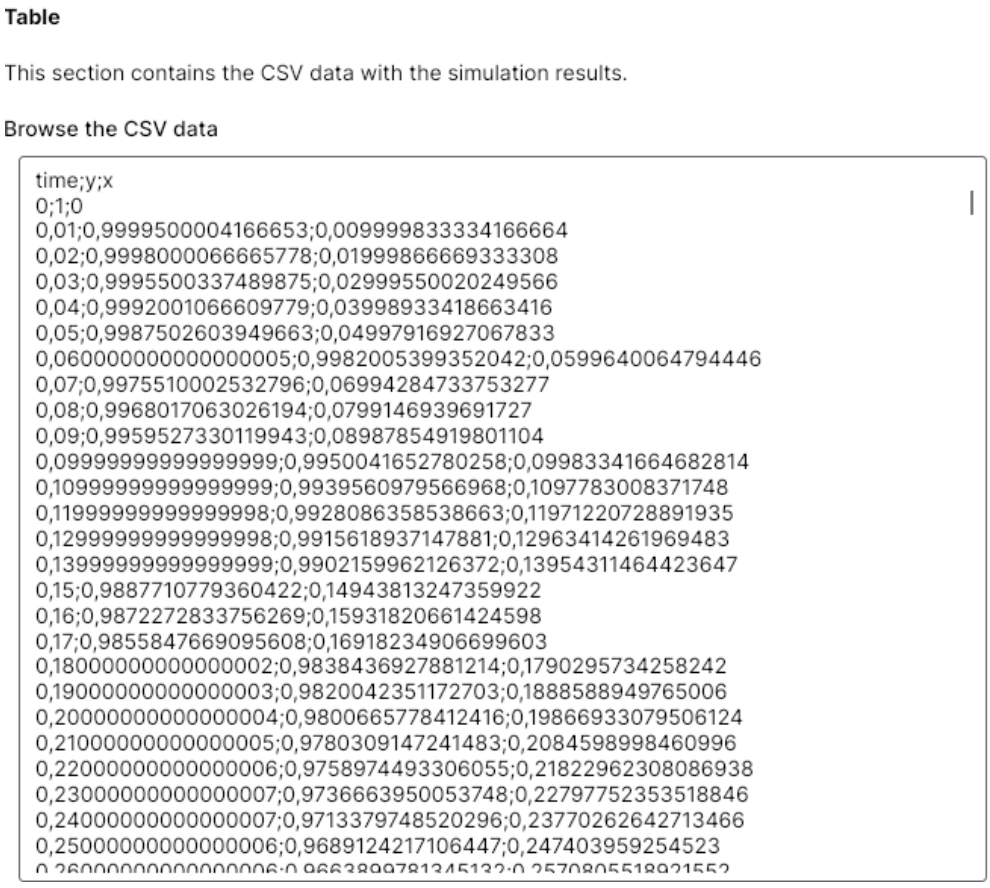

Рис. 4.4: Представление таблицы CSV.

4.5 Последние значения

Представление последних значений (*англ.* Last Values) предназначено для отображения значений рядов в конечной точке имитации. Если используется метод Монте-Карло, то тогда будет создано соответствующее количество компонентов, по одному на каждый запуск. Поэтому, пожалуйста, используйте представление последних значений только для единичного запуска или для метода Монте-Карло с малым количеством запусков!

Следующая таблица описывает свойство, которое может быть использовано при отображении последних значений.

Таблица 4.5: Последние значения

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при отображении последних значений

Пример

```
dt = 0.01;

ka = 1;
kb = 1;

A = integ(-ka * A, 100);
B = integ(ka * A - kb * B, 0);
C = integ(kb * B, 0);

experiment {
    lastValues {
        series = [A, B, C];
    }

    timeSeries {
        series = [A, B, C];
    }
}
```

Соответствующий вывод представлен на рисунке 4.5.

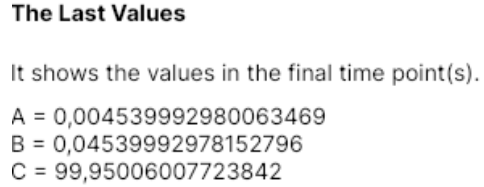

Рис. 4.5: Представление последних значений.

4.6 Гистограмма последних значений

Представление гистограммы последних значений (*англ.* Last Value Histogram) предназначено для отображения гистограммы по значе-

ниям рядов в конечной модельной точке при использовании метода Монте-Карло с множеством запусков.

Следующая таблица содержит опциональные свойства, которые могут быть использованы при отображении гистограммы последних значений.

Таблица 4.6: Гистограмма последних значений

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при построении гистограммы
`width =` *ширина*`;` `height =` *высота*`;`	Задает ширину графика Задает высоту графика

Пример

```
experiment {
    lastValueHistogram {
        series = [
            Resources.Total_Resources,
            Goods.Economic_Goods
        ];
    }
}
```

Соответствующий график представлен на рисунке 4.6.

4.7 Таблица CSV последних значений

При использовании метода Монте-Карло представление таблицы CSV последних значений (*англ.* Last Value CSV Table) собирает данные в конечных точках моделирования. Это представление предназначено для экспорта данных в другие приложения или для записи данных в файл. Отображается компонент текстового блока, откуда вы можете скопировать соответствующие данные CSV.

Следующая таблица содержит опциональные свойства, которые могут быть использованы при отображении таблицы CSV последних значений.

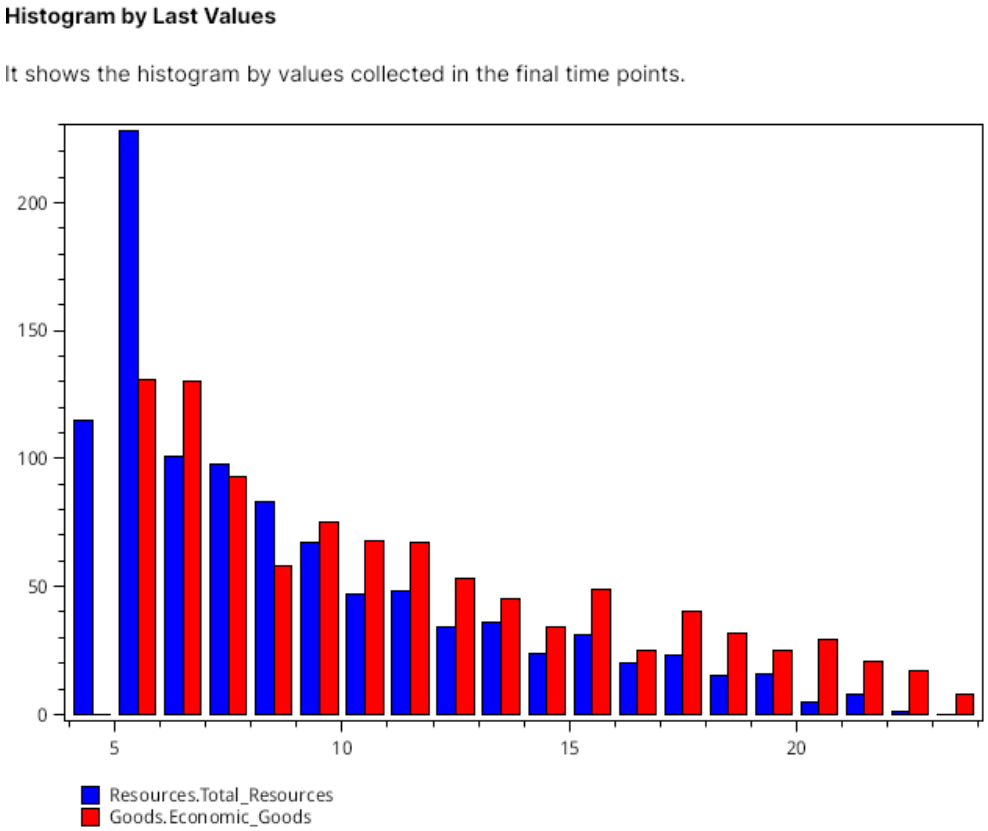

Рис. 4.6: Представление гистограммы последних значений.

Таблица 4.7: Таблица CSV последних значений

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при сборе данных в конечных точках моделирования

`width =` *ширина*`;`
 	
`height =` *высота*`;`
	Задает ширину компонента таблицы
	Задает высоту компонента таблицы

Пример

```
experiment {
    lastValueCsvTable {
        series = [
            Resources.Total_Resources,
            Goods.Economic_Goods
        ];
    }
}
```

Соответствующая таблица представлена на рисунке 4.7.

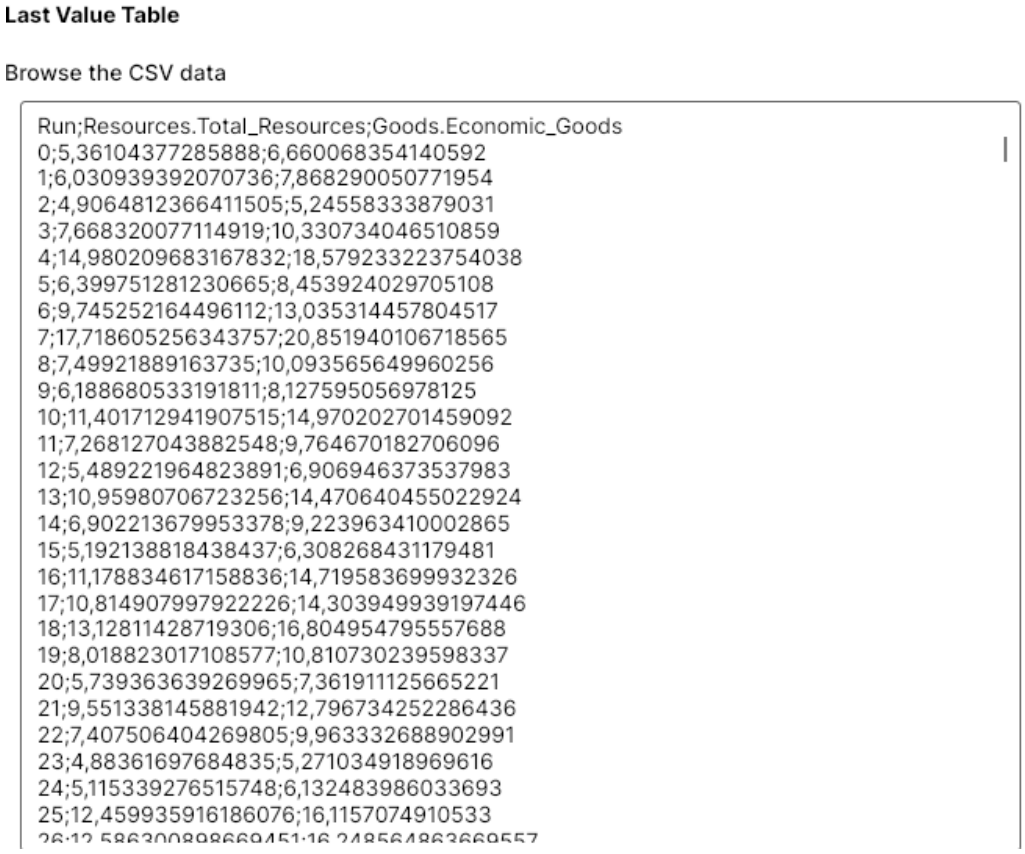

Рис. 4.7: Представление таблицы CSV последних значений.

4.8 Сводная статистика по последним значениям

При использовании метода Монте-Карло представление сводной статистики по последним значениям (*англ.* Last Value Statistics Summary) собирает данные в конечных точках моделирования, а затем представляет результаты на соответствующем компоненте.

Следующая таблица содержит опциональные свойства, которые могут быть использованы при отображении сводной статистики последних значений.

Таблица 4.8: Сводная статистика последних значений

`series = [`*series1*, *series2*, ...`];`	Перечисляет названия переменных *series1*, *series2*, ..., которые будут использованы при сборе данных в конечных точках моделирования
`width =` *ширина*`;`	Задает ширину компонента

Пример

```
experiment {
    lastValueStats {
        series = [
            Resources.Total_Resources,
            Goods.Economic_Goods
        ];
    }
}
```

Соответствующий компонент представлен на рисунке 4.8.

Last Value Statistics

This section displays the statistics summary collected in final time points.

Resources.Total_Resources	
mean	9,145071704459967
deviation	4,325187364978447
minimum	4,865129475228269
maximum	22,447110921857973
count	1000

Goods.Economic_Goods	
mean	11,485351584722222
deviation	5,102404247087857
minimum	5,125828743683213
maximum	23,535121045723905
count	1000

Рис. 4.8: Представление сводной статистики последних значений.

Литература

[1] Berkeley Madonna. http://www.berkeleymadonna.com, 2024. Accessed: 20-July-2024.

[2] Vensim Software. http://vensim.com, 2024. Accessed: 20-July-2024.