Simulation modeling

Altaisoft

29 марта 2011 г.

1 Введение

Программа *Simulation Modeling* предназначена для выполнения лабораторных работ по курсу "Имитационное моделирование экономических процессов" в Томском университете систем управления и радиоэлектроники.

Просто добавь воды

2 Модели

2.1 Склад

2.1.1 Постановка задачи

В нашем распоряжении находится склад некого товара (пусть это будет топливо для АЭС). Время от времени (в среднем каждые `demand` дней) к нам приходят покупатели, которые желают купить по урановому блочку каждый по определённой цене `demand_price`. Мы имеем перед ними договорные обязательства, – поэтому, если в данный момент у нас топлива нет, то мы платим неудовлетворённому покупателю неустойку в размере `fine`.

Чтобы пополнять свои запасы, мы можем обращаться к поставщикам – заводам-переработчикам сырья, заказывая у них блочки партиями. Каждый блочок вместе с доставкой стоит `supply_price`. Задача усложняется тем, что время доставки каждой партии – случайная величина. Положим, что в среднем партия доставляется за `supply` дней.

Мы бы могли заказать сразу заведомо слишком большое количество блочков и таким образом наверняка выполнить свои обязательства, но хранение каждого блочка требует ресурсов и обходится в `storage_price` в расчёте на один блочок в сутки. Размер склада мы в текущей версии модели, следуя учебно-методическому пособию, считаем бесконечным.

Нам необходимо определить, когда заказывать партии блочков и каков должен быть размер каждой партии, чтобы мы удовлетворили как можно больше покупателей и наша прибыль была максимальна.

Наша модель должна помочь в решении этой задачи.

2.1.2 Основные параметры

Подбор оптимального решения мы будем проводить, подстраивая два параметра модели:

`lot_size` – размер каждой партии;

`limit` – минимальный остаток на складе. Если запас блочков опускается величины `limit`, то наша автоматическая система управления складом отправляет поставщику заказ на новую партию.

Текущее количество блочков на складе мы будем хранить в переменной `amount`. Итак, вот наши входные переменные:

`supply` **и** `demand` – потоки заказов и заявок. Каждый из них является генератором положительных чисел с нормальным (гауссовым) распределением.

`amount` **и** `limit` – текущее количество блочков на складе и минимально допустимое их количество.

`supply_price`, `demand_price`, `storage_price` **и** `fine` – цена покупки, цена продажи, цена хранения и неустойка в пользу покупателя.

`lot_size` – размер каждой партии.

`total_time` – общая продолжительность моделирования.

2.1.3 Алгоритм

В жизни нашего виртуального склада может произойти лишь два события: прибытие покупателя или новой партии топлива. Поскольку первое происходит чаще, будем ориентироваться на него.

Как мы помним, `demand` – это последовательность пауз между прибытием покупателей. Иными словами, если он вернул нам поочерёдно 1 и 3, это значит, что первый покупатель придёт на следующий день после начала работы склада, а второй пожалует ещё через три дня (на четвёртый день), и так далее.

Мы обозначим паузу как `delta` и будем проходить этой переменной по всей последовательности `demand`. На каждом шаге цикла переменная `time` будет обозначать текущее время.

```
time = 0
for delta in demand:
    time += delta

    # <...>
```

Вместо многоточия должна стоять обработка вновь совершённой покупки.

Однако мы забыли о том, что время от времени к нам приходят новые партии топлива, и если такая партия пришла перед вновь совершаемой покупкой, нам необходимо её учесть на складе, прежде чем эту покупку проводить. Пусть время доставки партии хранится в переменной `delivery`. Тогда, если `delivery <= time`, сначала следует обработать прибытие партии, а затем уже покупку.

```
never = total_time + 1

time = 0
for delta in demand:
    time += delta

    if delivery <= time:
        amount += lot_size
        delivery = never

    # <...>
```

Значение `never` означает, что на данный момент у поставщика обязательств перед нами нет – все наши заказы уже удовлетворены.

Теперь рассмотрим, как будет обрабатываться покупка (то, что сейчас обозначено как двоеточие). Имеются следующие три случая.

На складе нет продукции. Мы разводим руками – придётся платить неустойку.

На складе есть продукция. Мы отдаём один блочок покупателю и он уходит довольным.

Оставшийся запас не превышает лимита. Отправляем заводу-поставщику заказ на новую партию, если только мы сейчас не ожидаем прибытия уже сделанного заказа.

Это легко записывается в виде следующего кода.

```
if amount:
    amount -= 1

    if amount <= limit and delivery > total_time:
        delivery = time + supply.next()
```

Описанный нами цикл является бесконечным. Необходимо гарантировать, что время `time` никогда не превысит `total_time` – тогда алгоритм будет в любом случае завершаться. И последнее – необходимо добавить в модель сбор статистики и вывод результатов моделирования.

Результат можно посмотреть в файле `models/warehouse.py`.

Однако это ещё не всё. Обратим внимание на тот факт, что прибытие партии будет обработано лишь в том случае, если после него следует хотя бы одна покупка. В обратном случае партия не будет учтена – как будто её и не было, что может в определённых условиях исказить результаты работы. Поэтому ещё один блок обработки партии мы добавим после завершения цикла.

```
if delivery <= total_time:
    amount += lot_size
    supplies += 1
    average += (total_time - delivery) * lot_size
    delivery = never
```

2.1.4 Тесты

Алгоритм достаточно простой и представляется очевидным. Но его необходимо протестировать. Тесты содержатся в файле `tests/warehouse_tests.py`.