МАТЕМАТИЧЕСКИЕ И ИНСТРУМЕНТАЛЬНЫЕ МЕТОДЫ ЭКОНОМИКИ

УДК: 330.42(045), 51-77(045) Научная статья
DOI: 10.35330/1991-6639-2022-4-108-96-114

Восстановление параметров процесса образования событий в экономике, заданного алгоритмической моделью

Ю. А. Кораблев

Финансовый университет при Правительстве Российской Федерации
143444, Россия, Москва, ул. Огарева, 16

Аннотация. События в экономике изучаются с точки зрения процессов, которые происходят в источниках этих событий. Процессы могут быть представлены произвольными алгоритмами. В статье представлена программная реализация на языке R метода восстановления неизвестных параметров алгоритмической модели процесса образования событий. В качестве примера рассматривается алгоритмическая модель из систем управления запасами. По выборке событий получается восстановить максимальный запас и нестационарный спрос. Показаны примеры дальнейшего использования подхода, заключающиеся в экстраполяции найденных параметров на будущее, запуск самого процесса и получение прогноза будущих событий.

Ключевые слова: редкие события, процесс образования событий, алгоритмическая модель процесса, определение параметров процесса, стационарные параметры, динамические параметры, кубический сплайн, программная реализация

Поступила 06.07.2022, одобрена после рецензирования 14.07.2022, принята к публикации 28.07.2022

Для цитирования. Кораблев Ю. А. Восстановление параметров процесса образования событий в экономике, заданного алгоритмической моделью // Известия Кабардино-Балкарского научного центра РАН. 2022. № 4 (108). С. 96–114. DOI: 10.35330/1991-6639-2022-4-108-96-114

JEL: C1, C15, C4, C5, C53 Original article

Restoration of the event formation process parameters in the economy, set by the algorithmic model

Yu.A. Korablev

Financial University under the Government of the Russian Federation
143444, Russia, Moscow, 16 Ogarev street

Annotation. Events in the economy are studied from the point of view of the processes that occur in the sources of these events. Processes can be represented by arbitrary algorithms. The article presents a software implementation in the language of R method for recovering unknown parameters of an algorithmic model of the event formation process. As an example, an algorithmic model from inventory management systems is considered. Based on a sample of events, it is possible to restore the maximum stock and nonstationary demand. An example of further use of the approach is demonstrated, which consists in extrapolating the found parameters to the future, starting the process itself and obtaining a forecast of future events.

Key words: rare events, event formation process, algorithmic model of the process, determination of process parameters, stationary parameters, dynamic parameters, cubic spline, software implementation

Submitted 06.07.2022, approved after reviewing 14.07.2022, accepted for publication 28.07.2022

For citation. Korablev Yu.A. Restoration of the event formation process parameters in the economy,set by the algorithmic model. *News of the Kabardino-Balkarian Scientific Center of RAS.* 2022. No. 4 (108). Pp. 96–114. DOI: 10.35330/1991-6639-2022-4-108-96-114

1. Введение

Редкие события могут представлять большой интерес, особенно в экономике. Различные подходы к анализу и прогнозированию редких событий можно найти в работах [1–4]. В отличие от статистических методов, которые ограничены либо изучением закона распределения интервалов времени между появлением событий (потоки событий, пуассоновские, Пальма), либо изучением вероятности появления события в зависимости от наблюдаемых признаков (методы классификации), предлагается исследовать события с точки зрения процессов, которые протекают внутри источников событий и которые приводят к образованию этих событий. После составления модели таких процессов, которая может содержать неизвестные параметры, по имеющейся выборке событий можно восстановить искомые значения параметров этого процесса. В работе [5] представлен математический метод восстановления параметров нескольких возможных процессов образования событий в виде непрерывной функции. Однако модель процесса была математической и имела определенную жесткую форму. Дальнейшее развитие этого подхода требует, чтобы модель процесса могла быть произвольной и задавалась произвольным алгоритмом (все же с некоторыми ограничениями). В работе [6] представлено краткое описание метода восстановления неизвестных параметров алгоритмической модели процесса образования редких событий, объем не позволял привести алгоритмы, реализующие данный метод. В данной работе будет сделан акцент на реализации на языке программирования R соответствующих составляющих метода с использованием параллельных вычислений на нескольких ядрах процессора. Кратко будет представлен результат работы метода. Также будет представлен пример экстраполяции найденных значений параметров на будущее, запуск самого процесса с установленными параметрами и получение прогноза будущих событий.

Метод восстановления параметров процесса образования редких событий

Как было сказано выше, модель процесса образования событий может быть задана алгоритмом. Причем у данного процесса будут внутренние переменные и неизвестные параметры. Сразу прыгнем с места в карьер и приведем пример одного такого процесса.

Время = Начальное время
$X_1 = P_1$
$X_2 = 0$
Пока (Время $\leq$ Конечное время)
$X_1 = X_1 - P_2$
Если ($X_1 \leq X_2$)
Создать событие (Время, $P_1 - X_1$)
$X_1 = P_1$
Продвинуть время и обновить параметры

Внимательный читать мог уже догадаться, что это простейшая модель из систем управления запасами, где (внутренние переменные) X_1 – текущий запас, X_2 – критический уровень запаса, берется как 0 (так как одновременное увеличение критического за-

паса и максимального приводит к одним и тем же результатам), а (неизвестные параметры) P_1 – максимальный запас и P_2 – нестационарный спрос. Внутренние переменные X_1, X_2 инициализируются либо нулем, либо значением параметра, а вот сами параметры P_1, P_2 неизвестны. Требуется определить значения неизвестных параметров, если известна исходная выборка событий $(t_1, y_1), \dots, (t_n, y_n)$, где t_i – наблюдаемый момент времени события, а y_i – известное значение (признак), которое несет это событие, n – количество наблюдений. Обратим внимание, что, конечно же, можно было использовать математическую модель процесса образования событий и восстанавливать параметры методами коллокации [7]. Но на данном этапе требуется восстановить параметры процесса, заданного алгоритмической моделью.

Задача усложняется тем, что параметры могут быть динамическими, а не статическими, то есть они изменяются со временем. В этом примере процесса параметр P_2 лучше обозначить как $P_2(t)$ – нестационарный спрос. Динамические параметры задаются в виде непрерывной функции, а именно в виде кубического сплайна $g(t)$, который задается через $g(s_1)$, $g'(s_1)$, и $g'''(s_k), k = 1, \dots, m$ в узлах сплайна $s_1 < s_2 < \dots < s_m$ (по сути как дифференциальное уравнение, условие натурального сплайна $g''(s_1) = g''(s_m) = 0$). У данного сплайна скачком изменяется третья производная в узлах сплайна, когда сама функция, первая и вторая производная непрерывны. Тогда под определением динамического параметра понимается определение значения сплайна $g(t)$ или, что то же самое, определение третьей производной в узлах сплайна и начальных условий в первом узле.

Для восстановления параметров происходит оптимизация следующего функционала:

$$S(P) = \sum_{i=2}^{n} \left(\frac{t_i - t_i'}{t_i - t_{i-1}} \right)^2 + \mu \sum_{i=2}^{n} \left(\frac{y_i - y_i'}{y_i} \right)^2 + \sum_{z=1}^{N_{\text{дин.пар.}}} \alpha_z \int_{s_1}^{s_m} \left(g_z''(t) \right)^2 dt \to \min, \qquad (1)$$

где P – искомые параметры, (t_i, y_i) и (t_i', y_i') – наблюдаемые события и события, полученные в результате моделирования (время и значения, суммирование начинается от 2, так как первое известное событие является отправной точкой), μ – весовой коэффициент для отклонений значений y_i', n – количество наблюдений, последнее слагаемое – штраф на шероховатость (нелинейность), α_z – коэффициент сглаживания (регуляризации). Штраф на шероховатость рассчитывается следующим образом:

$$\int_{s_1}^{s_m} \left(g''(t) \right)^2 dt = \sum_{k=1}^{m-1} \frac{\left(g''(s_k) + g'''(s_k) h_k \right)^3 - \left(g''(s_k) \right)^3}{3 g'''(s_k)}, \qquad (2)$$

где $g''(s_k) = g'''(s_1) h_1 + \dots + g'''(s_{k-1}) h_{k-1}$.

Если же при моделировании было получено меньше событий, чем количество наблюдений, то функционал возвращает бесконечность (Inf), однако необходимо дать возможность сформироваться заданному количеству событий, для чего немного увеличивается максимальное время моделирования.

Оптимизационная задача является нелинейной и недифференцируемой, для ее решения используется метод Нелдера – Мида [8], не требующий расчета градиента. Оптимизация происходит на сетке, диапазон и размер шага по каждому значению задаются пользователем на основе имеющихся данных. В случае если произошел выход за границы ячейки сетки, оптимизация прекращается и происходит оптимизация из другой ячейки. Однако

если значений очень много, то размерность задачи будет очень большой. Чтобы ограничить рост размерности задачи, используется особый прием, который подбирает на сетке неизвестные значения параметров не на все события, а только на несколько событий вперед. То есть одна оптимизационная задача разбивается на множество более мелких, когда по событиям двигается скользящее окно, внутри которого на сетке перебираются значения параметров (чтобы избежать эффекта бабочки, не вошедшие в окно значения участвуют в оптимизации, но не на сетке, на вход оптимизатора подаются найденные на прошлом шаге значения, трудоемкость растет линейно, а не экспоненциально).

Реализация на языке R

В начале происходит ввод исходных данных событий (t_i, y_i) и определение констант μ и α. Делается это следующим образом.

```
library(lubridate)
filename = "D:/DIR/filename.csv"
MyData = read.csv(file=filename, header=TRUE, sep=";", stringsAsFactors = FALSE,
                                        dec=",")
data_t = as.numeric(dmy(MyData[[1]]))
data_t = data_t[!is.na(data_t)]
n = length(data_t)
data_Y = MyData[[2]]

knots_number = n
Knots = data_t
mu = 0.1  #вес отклонения значений по сравнению с отклонением времени событий
alpha = 10^(-6) #коэффициент сглаживания
Window_Length = 4 #ширина скользящего окна событий
```

Задается структура параметров в виде списка, если параметр динамический, то для него задается массив значений, необходимый для идентификации сплайна (в начальном узле 4 значения, во всех последующих по одному, значение для последнего узла определять не нужно).

```
P1 = list(Dynamic=FALSE, Values = 0)
P2 = list(Dynamic=TRUE, Values =  rep(0 , knots_number + 3) )
P = list(P1, P2)
```

Так как перебор всех значений будет происходить на сетке, то задаются границы для соответствующих значений ($g(s_1)$, $g'(s_1)$, и $g'''(s_k)$). Эти границы выбираются пользователем на основе имеющихся данных, например, если объем покупки в районе 2000, то незачем искать значение максимального запаса в районе 10000. Для динамических параметров следует указать границы сетки для поиска скорости изменения параметра и третьей производной.

```
L_P1= c(1000, 2000)
U_P1= c(2000, 3000)
L_bounds_P1 = list(L_P1)
```

```
U_bounds_P1 = list(U_P1)
L_P2= c(    0, 100, 200)
U_P2= c(100, 200, 500)
L_dP2= c(-10,  -1, 1 )
U_dP2= c(   -1,   1, 10 )
L_d3P2 = c(  -0.1,   -0.01,   -0.001, -0.0001, 0.0001, 0.001,  0.01)
U_d3P2 = c(-0.01, -0.001, -0.0001,  0.0001,   0.001,   0.01,    0.1)

L_bounds_P2 = list(L_P2, L_dP2, 0, L_d3P2)
U_bounds_P2 = list(U_P2, U_dP2, 0, U_d3P2)

Bounds = list( list(L= L_bounds_P1, U= U_bounds_P1 ),
               list(L= L_bounds_P2, U= U_bounds_P2))
```

Происходит подготовка к параллельным многопоточным вычислениям (на CPU, на GPU еще не реализовано). Для этого создаются дополнительные процессы в среде R, в которых передаются переменные внутренней среды. Функции optBatch, 'Calculate_process, Decode_Parameters, Calculate_score, optNM2, runProcess, UpdateParameters будут описаны чуть ниже.

```
#================ Prepare Parallel ====================
library(doParallel)
library(foreach)
cores = detectCores(logical = TRUE)
cl = makeCluster(cores-1)
registerDoParallel(cl)
clusterExport(cl,list('alpha', 'data_t', 'data_Y', 'Knots', 'knots_number', 'Bounds', 'mu', 'n',
                      'optBatch', ‘Calculate_process', 'Decode_Parameters',
                      'Calculate_score', 'optNM2', 'runProcess', 'UpdateParameters'))
```

Подготовка к первому шагу алгоритма

```
Previous_Event =1
t_start = data_t[1]
```

Далее в цикле происходит подбор параметров P для заданного количества событий. Функция Perebor оптимизирует параметры на сетке и возвращает лучшую комбинацию параметров, после чего в функции optNM2 с большей точностью происходит уточнение этих параметров.

```
repeat #цикл по событиям
{
  Last_Event = Previous_Event+Window_Length
  t_end = data_t[Last_Event] + 10*(data_t[Last_Event]-data_t[Last_Event - 1])

  result = Perebor(Previous_Event, Last_Event, P, t_start, t_end, max_iter = 10^4,
                   abs.tol = 10^(-5),  rel.tol = 10^(-15) )

  if (result$objective==Inf) break #не получилось подобрать параметры

  # уточнение параметров с большей точностью
  result=optNM2(list(X=result$par, Lower=rep(-Inf,length(result$par)),
                     Upper=rep(Inf,length(result$par))), Calculate_process, Last_Event,
                     P, t_start, t_end, max_iter=10^4, abs.tol=10^(-7), rel.tol=10^(-15) )
```

```
    P = Decode_Parameters(P, result$par, Last_Event) #раскодирование и перенос найденных параметров
    if (Last_Event>=n) break #шаг был последним

    Previous_Event = Previous_Event+1      #Смещение скользящего окна событий
  }
```

Самая большая функция – это функция Perebor, в которой формируются целые блоки оптимизационных задач, которые в дальнейшем выполняются параллельно. Сама функция возвращает наилучший найденный результат. В самом начале определяется, какие значения попали в скользящее окно событий, далее для них происходит перебор на сетке начальных значений, которые потом подставятся в оптимизатор. В этой функции происходит многопоточный вызов функции optBatch, которая выполняет сформированный набор (пачку) оптимизационных задач.

```
Perebor=function(Previous_Event, Last_Event, P, t_start,t_end, max_iter=10000,
                                                    abs.tol=0.0001, rel.tol=0.000001)
{
 X = NULL  #вектор значений для оптимизации
 Lower = NULL
 Upper = NULL

 #какие элементы в этом векторе надо будет перебирать на сетке
 N = 0 #количество элементов для перебора
 index_X = NULL
 index_K = NULL
 index_J = NULL
 index_M = NULL

 k = 1  #номер параметра
 j = 1  #номер элемента у этого параметра, если он динамический

 repeat
 {
  thisKnot = max(1, j-3)
  if ( Knots[thisKnot]<data_t[Previous_Event] )
  {#пропуск элементов, которые не надо перебирать, и перенос их в вектор значений
   X = c(X, P[[k]]$Values[j])
   Lower = c(Lower, -Inf)
   Upper = c(Upper, Inf)
  }
  if ( Knots[thisKnot] >= data_t[Previous_Event] &&
                                    Knots[thisKnot] < data_t[Last_Event] )
  {#надо перебирать узел, попавший в интервал от последнего до следующего события
   # надо организовать счетчик, тут соберем только индексы таких переменных
   # потом по этим индексам будет перебор
   J = min(4,j)
   X = c(X,(Bounds[[k]]$L[[J]][1]+Bounds[[k]]$U[[J]][1])/2)
   Lower = c(Lower, Bounds[[k]]$L[[J]][1])
   Upper = c(Upper, Bounds[[k]]$U[[J]][1])
```

```
  N = N+1
  index_X = c(index_X, length(X))
  index_K = c(index_K, k)
  index_J = c(index_J, J)
  index_M = c(index_M, length(Bounds[[k]]$L[[J]]) )
 }
 j = j+1
 if (j == 3) #пропуск начальной второй производной (всегда равна 0)
  j = j+1
 thisKnot = max(1, j-3)
 if (j > length(P[[k]]$Values) || Knots[thisKnot] >= data_t[Last_Event])
 {
  k=k+1
  j=1
  if (k>length(P))
   break;
 }

} #repeat

Batch_size=100 #задач на ядро в блоке
Batch_count=1000 # количество блоков для подготовки
Batch_num=1
Batch_array=vector(mode = "list", length = 1)
Batch_array[[1]]=append(Batch_array[[1]], list(list(X=X, Lower=Lower,Upper=Upper)))
N_params=1
Total_Combinatios=1
Max_Combinations=prod(index_M)

min_result = list(par=0, objective=Inf, iterations=0, Total_calls=0)

# в векторе X начинаем перебирать значения и добавляем новый вектор в список
X_= rep(1,N)
repeat
{  #перебор и формирование комбинаций параметров для оптимизации
 i = N
 while ((i>0) && (X_[i]==index_M[i]) )
 {
  X_[i] = 1
  X[index_X[i]] = (       Bounds[[index_K[i]]]$L[[index_J[i]]][1] +
                          Bounds[[index_K[i]]]$U[[index_J[i]]][1]) /2
  Lower[index_X[i]] = Bounds[[index_K[i]]]$L[[index_J[i]]][1]
  Upper[index_X[i]] = Bounds[[index_K[i]]]$U[[index_J[i]]][1]
  i=i-1
 }
 if (i>0)
 {
  X_[i]=X_[i]+1
  X[index_X[i]] = (       Bounds[[index_K[i]]]$L[[index_J[i]]][X_[i]] +
                         Bounds[[index_K[i]]]$U[[index_J[i]]] [X_[i]]) /2
  Lower[index_X[i]] = Bounds[[index_K[i]]]$L[[index_J[i]]][X_[i]]
  Upper[index_X[i]] = Bounds[[index_K[i]]]$U[[index_J[i]]][X_[i]]
```

```
    Batch_array[[Batch_num]] = append(Batch_array[[Batch_num]],
                                      list(list(X = X, Lower = Lower, Upper = Upper)))
    N_params = N_params+1
    Total_Combinatios = Total_Combinatios+1
    if (N_params >= Batch_size)
    {
     if (Batch_num < Batch_count && Total_Combinatios<Max_Combinations)
     {
      Batch_num = Batch_num+1
      Batch_array = append(Batch_array,list(NULL))
      N_params = 0
     }
     else
     { # чтобы список не получился очень большим, обрабатываем его часть
       # именно здесь происходят параллельные вычисления
      results = foreach (v = 1:Batch_num , .inorder = F) %dopar%
        optBatch(Batch_array[[v]], Calculate_process, Last_Event, P, t_start, t_end,
                                                                  max_iter, abs.tol, rel.tol)
      #выбрать меньший
      for (v in 1: length(results))
      {
       min_result$iterations = min_result$iterations + results[[v]]$iterations
       min_result$Total_calls = min_result$Total_calls + results[[v]]$Total_call
       if (results[[v]]$objective < min_result$objective)
       {
        min_result$par = results[[v]]$par
        min_result$objective = results[[v]]$objective
       }
      }

      Batch_num = 1 #очистить список
      N_params = 0
      Batch_array = vector(mode = "list", length = 1)
     }
    }
   }
   else
    break
  } #end repeat

  #Оставшийся список вариантов
  if (Batch_num>0 && N_params>0)
  {
   # именно здесь происходят параллельные вычисления
   results = foreach (v = 1:Batch_num , .inorder = F) %dopar%
    optBatch(Batch_array[[v]], Calculate_process, Last_Event, P, t_start, t_end, max_iter,
                                                                  abs.tol, rel.tol)

   #выбрать меньший
   for (v in 1: length(results))
   {
```

```
    min_result$iterations = min_result$iterations + results[[v]]$iterations
    min_result$Total_calls = min_result$Total_calls + results[[v]]$Total_call
    if (results[[v]]$objective < min_result$objective)
    {
     min_result$par = results[[v]]$par
     min_result$objective = results[[v]]$objective
    }
   }

   N_params = 0 #очистить список
   Batch_num = 0
   Batch_array = vector(mode = "list", length = 0)
  }

  return(min_result)
}
```

Сама же функция optBatch, которая выполняется независимо в каждом потоке, достаточно короткая, она всего лишь в цикле минимизирует переданный ей набор оптимизационных задач, выбирая при этом лучший результат. Для решения каждой оптимизационной задачи используется метод Нелдера – Мида, функция optNM2, в котором используются дополнительные условия на прерывание исполнения, если набор параметров (точка) вышел за установленные границы (из ячейки сетки).

```
optBatch = function(Batch, Calculate_process, Last_Event, P, t_start, t_end, max_iter =
                                                    10^6, abs.tol = 10^(-5), rel.tol = 10^(-20))
{
 results = lapply(X = Batch, FUN = optNM2, Calculate_process, Last_Event, P, t_start,
                                                      t_end, max_iter, abs.tol, rel.tol  )

 min_result = list(par = NULL, objective = Inf, iterations=0, Total_calls=0)

 for (i in 1:length(results))
 {
  min_result$iterations = min_result$iterations + results[[i]]$iterations
  min_result$Total_calls = min_result$Total_calls + results[[i]]$Total_call
  if (results[[i]]$objective < min_result$objective)
  {
   min_result$par = results[[i]]$par
   min_result$objective = results[[i]]$objective
  }
 }
 return(min_result)
}
```

Функция optNM2 реализует алгоритм Нелдера – Мида с дополнительным условием остановки, если произошел выход за границы ячейки сетки. В аргумент функции Param передаются вектор значений, подлежащих оптимизации, а также нижние и верхние границы по каждому значению. Код данной функции является более оптимизированной версией кода, размещенного на GitHub пользователем под ником nicolaivicol, с дополнительными условиями остановки при выходе за границы. Значения функционала в точке оцениваются вызовом функции Calculate_process, которая передается в аргументе objective.

```
optNM2 = function(Param, objective, Last_Event, P, t_start, t_end, max_iter=100,
                                                    abs.tol=0.0001 , rel.tol=0.0001)
{
  dif = Param$Upper-Param$Lower
  dif[dif == Inf] = Param$X[dif == Inf] # сделать относительным от значения, если границы не заданы,
  dif[dif == 0] = 0.0001          # если нулевое, то брать шаг относительно минимальной границы

  n = length(Param$X)
  # строится симплекс
  Points = matrix(Param$X, nrow=n, ncol=n, byrow = TRUE) + diag(n)*0.05*dif
  Points = rbind(Param$X, Points, deparse.level =0)  #добавляется стартовая вершина
  num=n+1        #всего вершин

  iterations = 0
  Values = apply(Points, MARGIN = 1, FUN = objective, Last_Event, P, t_start, t_end)
  Total_calls = num

  while (TRUE)
  {
   ord = order(Values)      #сортировка вершин по значениям
   Points = Points[ord,]
   Values = Values[ord]

   iterations = iterations + 1
   # условия остановки, последние два условия – это выход за границы ячейки
   if ( Values[1] == Inf || iterations >= max_iter || Values[1] == 0 ||
            abs(Values[num] - Values[1]) < abs.tol ||
abs((Values[num] - Values[1]) / Values[1]) < rel.tol      ||
sum(Points[num,] - Points[1,]) < abs.tol   ||
sum(Points[1,] > Param$Upper) > 0 || sum(Points[1,] < Param$Lower) > 0  )
     break

  # Центр противолежащей плоскости, тут n = num-1
   Centr = apply(Points[1:n, ], MARGIN = 2, FUN = mean)

   Reflected = Centr + (Centr - Points[num, ])
   Reflected_Value = objective(Reflected, Last_Event, P, t_start, t_end)
   Total_calls = Total_calls+1

   if (Reflected_Value < Values[1])
   {
    Expanded = Centr + 2*(Centr - Points[num, ])
    Expanded_Value = objective(Expanded, Last_Event, P, t_start, t_end)
    Total_calls = Total_calls + 1
    if (Expanded_Value < Reflected_Value)
    {
```

```
      Points[num,] = Expanded
      Values[num] = Expanded_Value
    } else #"Expanded_Value < Reflected_Value"
    {
      Points[num,] = Reflected
      Values[num] = Reflected_Value
    }
  } else #"Reflected_Value < Values[1]"
  {
    if (Reflected_Value < Values[n])
    {
      Points[num,] = Reflected
      Values[num] = Reflected_Value
    } else #"Reflected_Value < Values[n]"
    {
      if (Reflected_Value < Values[num])
      {
        Points[num,] = Reflected
        Values[num] = Reflected_Value
      }
      Contracted = Centr + 0.5*(Centr - Points[num, ])
      Contracted_Value = objective(Contracted, Last_Event, P, t_start, t_end)
      Total_calls=Total_calls + 1
      if (Contracted_Value < Values[num])
      {
        Points[num,] = Contracted
        Values[num] = Contracted_Value
      } else #"Contracted_Value < Values[num]"
      {
        Points = sweep(0.5*sweep(Points, 2, Points[1, ], FUN = "-"), 2, Points[1, ],
                                                                    FUN="+")
        Values[2:num] = apply(Points[2:num,], MARGIN = 1, FUN = objective,
                                                          Last_Event, P, t_start, t_end)
        Total_calls=Total_calls + n  #+ num-1
      }

    }
  }
 } #while true
 return(list(par = Points[1,], objective = Values[1], iterations = iterations,
                                                              Total_calls = Total_calls))
}
```

Самое основное, что относится к анализу редких событий, начинается при вызове функции Calculate_process во время оптимизации. Эта функция принимает на вход массив значений параметров процесса образования событий, которые надо декодировать. В этой же функции рассчитывается штраф на шероховатость для динамических параметров. Наконец запускается сам процесс образования событий, который потом оценивается по описанной выше методике. Сама функция возвращает сумму этой оценки и штрафа на шероховатость.

```
Calculate_process = function(X, Last_Event, P, t_start, t_end)
{
 new_P = Decode_Parameters(P, X, Last_Event)

 #calculate penalty
 Pen = 0
 for (k in 1:length(new_P))
  if (new_P[[k]]$Dynamic)
  {
   d2g = 0
   i = 4
   while(i <= length(new_P[[k]]$Values))
   {
    thisKnot = i - 3
    if (thisKnot + 1 > knots_number || Knots[thisKnot + 1] > data_t[Last_Event])
     break
    if (new_P[[k]]$Values[i] != 0)
    {
     Pen = Pen + ((d2g+(Knots[thisKnot+1] - Knots[thisKnot]) * new_P[[k]]$Values[i])^3
                  - d2g^3)/(3*new_P[[k]]$Values[i])
     d2g = d2g + (Knots[thisKnot+1] - Knots[thisKnot]) * new_P[[k]]$Values[i]
    }
    else
     Pen = Pen + d2g^2*(Knots[thisKnot+1] - Knots[thisKnot])
    i = i + 1
   }#while
  }#"if (P[[k]]$Dynamic)"

 r = runProcess(new_P,Last_Event,t_start,t_end)
 score = Calculate_score(r, Last_Event)
 score = score + alpha*Pen
 return(score)
}
```

Декодирование параметров функцией Decode_Parameters нужно по той причине, что в методе оптимизации Нелдера – Мида координаты точки представлены массивом, куда помещаются значения всех параметров. Более того, из условия натурального сплайна $(g''(s_1) = g''(s_m) = 0)$ для динамического параметра можно выразить значение третьей производной в предпоследнем узле $g'''(s_{m-1}) = -\sum_{k=1}^{m-2} h_k g'''(s_k)/h_{m-1}$, что позволит подбирать на одно значение меньше.

```
Decode_Parameters = function(P, par, Last_Event)
{
 j = 1
 for (k in 1:length(P))
 {
  d2g = 0
  for (i in 1:length(P[[k]]$Values))
  {
```

```
      thisKnot = max(1, i - 3)
      if (thisKnot+1<=knots_number && Knots[thisKnot+1] < data_t[Last_Event])
      {
       if (P[[k]]$Dynamic && i >= 4 )
         d2g = d2g + (Knots[thisKnot+1] - Knots[thisKnot])*par[j]

       if (i != 3)
       {
        P[[k]]$Values[i] = par[j]
        j = j + 1
       }
       else
        P[[k]]$Values[i] = 0
      } #"if (Knots[thisKnot+1] < data_t[Last_Event])"
      else
      {
       if ( (Knots[thisKnot] < data_t[Last_Event]) && (i <= length(P[[k]]$Values)) &&
                                                          ((thisKnot+1)<=knots_number) )
        P[[k]]$Values[i] =  -d2g/(Knots[thisKnot + 1] - Knots[thisKnot])
       break
      }

     } #"for (i in 1:length(P[[k]]$Values))"
    } #"for (k in 1:length(P))"
    return(P)
   }
```

Наконец, сам процесс с установленными значениями параметров запускается вызовом функции runProcess, которая возвращает список образовавшихся событий. С ходом времени динамические параметры процесса необходимо обновить, для чего используется простая функция UpdateParameters.

```
runProcess = function(P, Last_Event, t_start, t_end)
{
 #системная инициализация
 T = data_t[1]
 Y = data_Y[1]
 NumEvents = 1
 t = t_start
 k = 1 # узел сплайна

 #инициализация внутренних переменных процесса
 X1 = P[[1]]$Values[1]
 X2 = 0

 #процесс образования событий
 while(t <= t_end)
 {
  X1 = X1 - P[[2]]$Values[1]
  if (X1 <= X2)
  {
```

```
      #Создать событие
      T = c(T, t)
      Y = c(Y, P[[1]]$Values[1] - X1)
      NumEvents = NumEvents + 1
      if (NumEvents >= Last_Event)
        break
      X1 = P[[1]]$Values[1]
    }
    #Продвинуть время и обновить параметры
    t = t + 1
    if (k < knots_number & t > Knots[k+1]) #переход к следующему узлу сплайна
      k=k+1
    P = UpdateParameters(P, k)
  } #"while"

  result = list(T=T,Y=Y)
  return(result)
}
```

В функции UpdateParameters происходит обновление значений динамических параметров при изменении времени на единицу с помощью формул Тейлора.

```
UpdateParameters=function(Parameters,index)
{
  for (i in length(Parameters))
    if (Parameters[[i]]$Dynamic)
    {
      V=Parameters[[i]]$Values
      Parameters[[i]]$Values[1]=V[1]+V[2]+V[3]/2+V[3+index]/6
      Parameters[[i]]$Values[2]=V[2]+V[3]+V[3+index]/2
      Parameters[[i]]$Values[3]=V[3]+V[3+index]
    }
  return(Parameters)
}
```

После запуска процесса и получения событий функцией Calculate_score рассчитывается сумма квадратов относительных отклонений времени событий и значений, которые несут эти события. Если размер исходной выборки и полученной в результате моделирования не совпадает, то возвращается бесконечность (Inf).

```
Calculate_score = function(r, Last_Event)
{
  if(length(r$T) < Last_Event || r$T[Last_Event] == r$T[Last_Event - 1])
    return(Inf)
  S1 = 0
  S2 = 0
  for (i in 2:length(r$T))
  {
    S1=S1+( (data_t[i]-r$T[i] )/ (data_t[i]-data_t[i-1]) )^2
```

```
    S2=S2+( (data_Y[i]-r$Y[i] ) / data_Y[i] )^2
  }
  return(S1 + mu*S2)
}
```

Пример использования метода

Проверим работоспособность метода на следующем искусственном примере. Рассмотрим тут же описанную в самом начале систему управления запасами, установим значения параметрам, получим выборку событий, после чего будем по этой выборке восстанавливать параметры процесса, как если бы они были неизвестными. Исходная выборка событий представлена в таблице 1 (дробные числа оставлены для большей точности).

Таблица 1.

Данные событий, по которым нужно восстановить параметры процесса образования событий

Table 1.

Event data to restore settings from event formation process

t_i	y_i	t_i	y_i	t_i	y_i	t_i	y_i
05.01.2020	549,62	26.04.2020	546,91	06.07.2020	537,02	09.10.2020	553,97
15.01.2020	538,04	05.05.2020	557,78	16.07.2020	536,29	18.10.2020	560,74
25.01.2020	505,14	13.05.2020	539,46	27.07.2020	526,83	27.10.2020	548,58
06.02.2020	541,80	21.05.2020	568,64	08.08.2020	527,37	05.11.2020	517,64
19.02.2020	501,07	28.05.2020	510,39	20.08.2020	518,21	15.11.2020	519,90
06.03.2020	527,59	04.06.2020	511,67	31.08.2020	501,60	27.11.2020	529,77
22.03.2020	516,76	11.06.2020	502,97	11.09.2020	550,25	12.12.2020	518,59
05.04.2020	530,87	19.06.2020	552,80	21.09.2020	549,75	31.12.2020	506,49

Глядя на значения событий, можно приблизительно догадаться, в каком интервале располагается значение максимального запаса. Также можно приблизительно оценить величину нестационарного спроса, поделив объемы пополнения запаса y_i на время между этими пополнениями $t_{i+1} - t_i$. Тогда в нашем алгоритме границы задаем, например, таким образом:

```
L_P1= c(470, 520)
U_P1= c(520, 600)
L_bounds_P1 = list(L_P1)
U_bounds_P1 = list(U_P1)

L_P2= c(30, 60)
U_P2= c(60, 100)
L_dP2= c(-2, -1,  1 )
U_dP2= c( -1,  1, 2 )
L_d3P2 = c( -0.1,  -0.01,  -0.001, -0.0001, 0.0001, 0.001,  0.01)
U_d3P2 = c(-0.01, -0.001, -0.0001,  0.0001,  0.001,  0.01,   0.1)
```

Запуская описанный выше алгоритм, получаем следующие результаты. Максимальный запас $P_1 = 496.381$ (закладывалось 500, значения y_i больше 500, так как компенсируется использование ненаблюдаемого страхового запаса). Начальное значение нестационарного спроса $P_2 = 54.0203$ (закладывалось 55). Значение первой производной в начальном узле и значение третьих производных в каждом узле приводят к тому, что нестационарный спрос получается, как на рисунке 1. Напоминаю, что, конечно же, все то же самое можно было проделать и чисто математическими методами: конкретно с помощью [5, 7] можно восстановить значение функции по наблюдаемым с погрешностью интегралам. Но нас интересует восстановление параметров у процесса, заданного алгоритмом. Более того, здесь в рамках одного подхода определяются оба параметра, а не только нестационарный спрос.

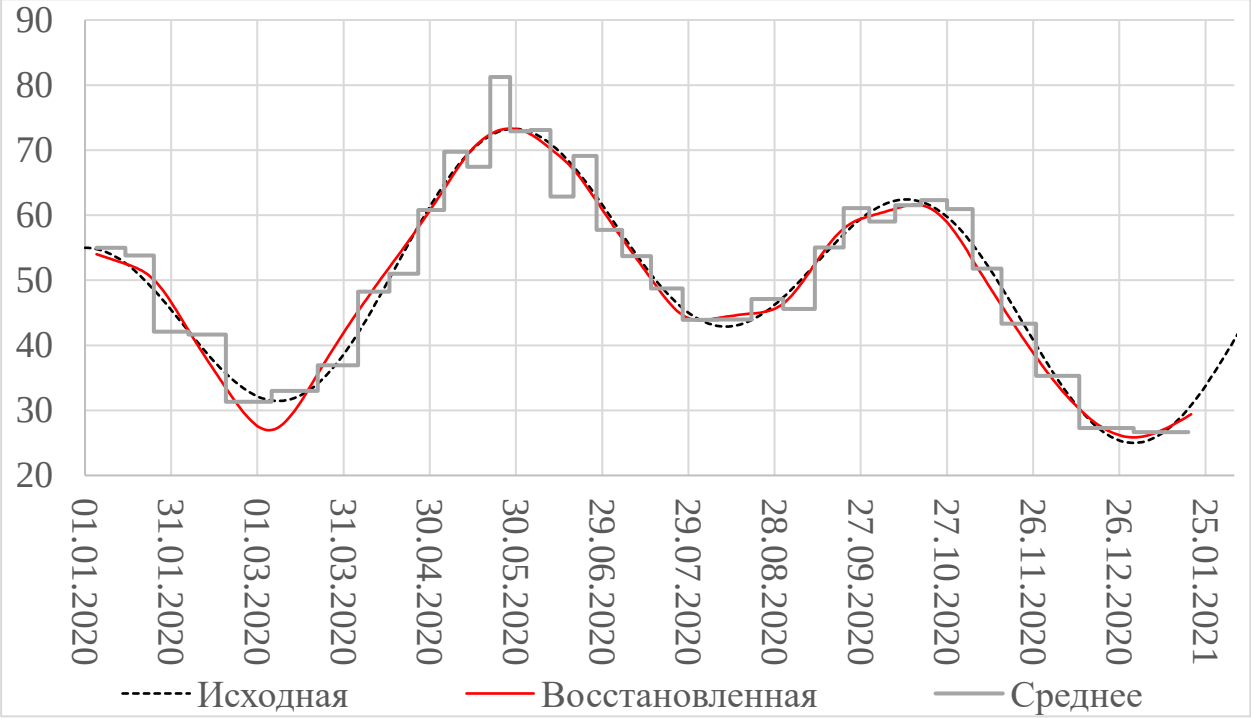

Рис. 1. *Восстановление нестационарного спроса с помощью метода определения параметров алгоритмической модели процесса. Сплошная гладкая – восстановленное значение функции, пунктирная линия – исходная функция спроса, заложенная априори (которую требовалось определить), ступенчатая линия – среднее значение* $y_i/(t_{i+1} - t_i)$

Fig. 1. *Recovery of non-stationary demand using the method of determining the parameters of the algorithmic model of the process. The solid smooth line is the restored value of the function, the dotted line is the original demand function laid down a priori (which needed to be determined), the stepped line is the average value y_i/((t_(i+1)-t_i))*

Прогнозирование событий

Дальнейшим действием является прогнозирование событий. Восстановленные значения параметров необходимо экстраполировать на будущее. Максимальный запас P_1, так как это стационарный параметр, то оставляем его значение таким, какой он есть. Для экстраполяции нестационарного спроса надо строить соответствующую модель. Так как

у нас есть определенная информация о виде зависимости (мы изначально закладывали композицию гармонических функций), будем использовать алгоритм Куинна и Фернандеса (Quinn-Fernandes algorithm) [9, 10], который раскладывает сигнал на сумму ограниченного количества гармонических функций. Проводим экстраполяцию на год от последнего события. После этого запускаем сам процесс образования редких событий с установленными значениями параметров процесса и получаем прогноз будущих событий. Результат экстраполяции восстановленной зависимости и прогноз будущих событий показаны на рисунке 2.

Хочется заметить, что для 5 первых прогнозных событий разница по времени относительно даты фактических событий (если для получении исходной выборки моделирование продолжать дольше) составляет 0 дней, для 6-го события -1 день (раньше), а для последующих из-за накопления погрешности происходит увеличение сдвига на -2, -2, -3, -4, -6, -7, -8, -8, -8, -8, -7, -7, -6 и т.д. дней (увеличение горизонта планирования дает менее точные прогнозы событий). Разница же объемов y_i прогнозных покупок и фактических составляет в среднем менее 4%.

Рис. 2. *Экстраполяция параметра процесса и прогноз событий*

Fig. 2. *Process variable extrapolation and events prediction*

Стоит сделать одно замечание: при неудачном выборе границ и шага сетки для значений параметров может получиться такая картина, когда одновременно максимальный запас и нестационарный спрос будут завышенными, при этом метод может застрять в локальном экстремуме. При этом моменты времени событий t_i будут очень хорошо соответствовать имеющимся данным. Однако значения событий y_i могут значительно отличаться, но так как весовой коэффициент μ для отклонений значений y_i небольшой, то большее внимание уделяется времени событий.

Также ничего не было сказано про коэффициент сглаживания (регуляризации) α, который в обычной сплайновой аппроксимации отвечает за степень гладкости (шероховатости) сплайна. В исследовании [11] представлены результаты подбора оптимального коэффициента сглаживания при восстановлении функции по последовательности интегралов. Однако у нас особый случай, когда модель задана алгоритмом, исследования коэффициента сглаживания в таких условиях мне не попадались.

Заключение

В статье представлена программная реализация на языке R метода восстановления параметров процесса образования редких событий в экономике, который может быть представлен произвольной алгоритмической моделью. Предполагается, что такой подход к анализу событий в экономике с точки зрения процессов, которые их порождают, будет способствовать более точному представлению действительности и природы экономических процессов. Сами по себе выявленные закономерности восстановленных параметров процесса образования событий могут иметь собственный отдельный научный интерес.

REFERENCE

1. *Kaya G.O., Sahin M., Demirel O.F.* Intermittent demand forecasting: A guideline for method selection. *Sadhana – Academy Proceedings in Engineering Sciences*. 2020. 45, 1. 45–51. DOI: 10.1007/s12046-020-1285-8

2. *Carreno A., Inza I., Lozano J.* Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. *Artificial Intelligence Review*. 2020. 53. 3575–3594. DOI:10.1007/s10462-019-09771-y

3. *Pince C., Turrini L., Meissner J.* Intermittent demand forecasting for spare parts. *A critical review. Omega*. 2021. 105. 102513. DOI:10.1016/j.omega.2021.102513

4. *Halim S.Z., Quaddus N., Pasman H.* Time-trend analysis of offshore fire incidents using nonhomogeneous Poisson process through Bayesian inference. *Process Safety and Environ-mental Protection*. 2021. 147. 421–429. DOI:10.1016/J.PSEP.2020.09.049

5. *Korablev Yu.A.* Restoration of a function from different functionals in R for predicting rare events in the economy. *Finansy: teoriya i praktika* [Finansy: Teoriya i Praktika]. Vol. 26. No. 3. Pp. 196–225. (In Russian)

6. *Korablev Yu.A.* Determination of parameters for the formation process of rare events in the economy for their subsequent forecasting. *Ekonomika i matematicheskiye metody* [Economics and Mathematical Methods]. 2022. Vol. 58. № 2. Pp. 80–91. DOI: 10.31857/S042473880020016-6

7. *Korablev Yu.A.* Restoration of function by integrals with cubic integral smoothing spline in R. *ACM Transactions on Mathematical Software*. 2022. 48(2). Pp. 1–17. DOI: 10.1145/3519384 ISSN: 0098-3500

8. *Nelder J.A., Mead R.* A Simplex Method for Function Minimization. *The Computer Journal*. 1965. 7. Pp. 308–313. https://doi.org/10.1093/comjnl/7.4.308

9. *Quinn B.G., Fernandes J.M.* A Fast Efficient Technique for the Estimation of Frequency. *Biometrika*. 1991. Vol. 78. No. 3. Pp. 489–497.

10. *Quinn B.G., Hannan E.J.* The Estimation and Tracking of Frequency. Cambridge: Cambridge University Press, 2001. 278 p.

11. *Korablev Yu.A.* Restoration of the Product Consumption Rate with Integral Cubic Smoothing Spline, Study of the Best Smoothing Parameter Choice. *Acta Appl Math* 180, 8 (2022). https://doi.org/10.1007/s10440-022-00509-7

Информация об авторе

Кораблев Юрий Александрович, к.э.н., доцент каф. «Системный анализ в экономике», Финансовый университет при Правительстве РФ;
143444, Россия, Московская область, Красногорск, мкр. Опалиха, ул. Огарева, 16;
yura-korablyov@yandex.ru, ORCID: https://orcid.org/0000-0001-5752-4866

Information about the author

Korablev Yury Aleksandrovich, Candidate of Economics, Associate Professor of the Department "System Analysis in Economics", Financial University under the Government of the Russian Federation;
143444, Russia, Moscow region, Krasnogorsk, md. Opaliha, 16 Ogarev street;
yura-korablyov@yandex.ru, ORCID: https://orcid.org/0000-0001-5752-4866