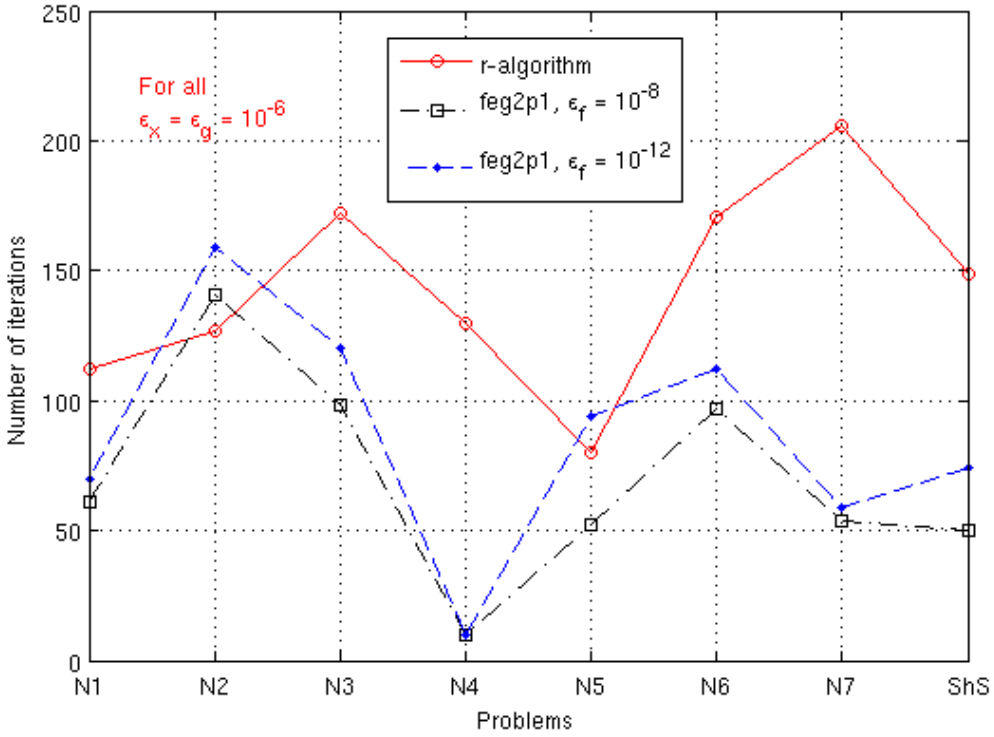

Рис. 6. Сравнение r-алгоритма и метода $Feg2p1(x_0, \varepsilon, x_\varepsilon^*)$.

Рассматривались 7 известных тестовых задач безусловной мимимизации гладких и негладких выпуклых функций [32] (стр. 279–282) и кусочно-квадратичная функция [27] (стр. 176). Количество переменных в тестовых примерах было от 5 до 50. Все примеры решались r-алгоритмом с рекомендуемыми параметрами адаптивной регулировки шага, критериями останова $\varepsilon_x = 10^{-6}$ и $\varepsilon_g = 10^{-6}$ и методом $Feg2p1(x_0, \varepsilon, x_\varepsilon^*)$ при достаточно малых $\varepsilon = 10^{-8}$ и $\varepsilon = 10^{-12}$. Число затраченных методами итераций дано на рис. 6. Из него видно, что метод $Feg2p1(x_0, \varepsilon, x_\varepsilon^*)$ только в одном случае уступил r-алгоритму (пример №2).

Приведенные выше алгоритмы можно использовать при решении негладких задач, которые возникают в различных областях приложений. Отметим, что матрично-векторные вычисления для обоих семейств алгоритмов легко поддаются параллельной обработке, что может быть полезным при их реализации на параллельных ЭВМ. Для решения непрерывной задачи оптимальной загрузки мощностей энергоблоков при ограничениях на переходы по мощностям энергоблоков (раздел 3) ограничимся использованием конкретного варианта r-алгоритмов, который использует постоянный коэффициент растяжения пространства и адаптивный способ регулировки шага для приближенного поиска точки минимума в направлении антисубградиента в преобразованном пространстве переменных. Этот вариант r-алгоритмов будет предметом обсуждения в следующем подразделе.

3.2 $r(\alpha)$-алгоритм с адаптивной регулировкой шага

Пусть $f(x)$ – выпуклая функция, определенная на евклидовом пространстве E^n, обладающая свойством

$$\lim_{\|x\| \to \infty} f(x) = +\infty. \tag{3.2}$$

Пусть X^* – множество минимумов, $x^* \in X^*$ – точка минимума; $\inf f(x) = f^*$; $g_f(x)$ – субградиент (произвольный) функции $f(x)$ в точке x. Функция $f(x)$ может быть как гладкой (дифференцируемой), так и негладкой (недифференцируемой).

Для нахождения $x_r \approx x^*$ и $f_r \approx f^*$ можно использовать конкретные варианты из семейства r-алгоритмов, вычислительная схема которых описана в подразделе 3.1. Практическая эффективность r-алгоритмов зависит от выбора коэфициентов растяжения пространства α_k и шаговых множителей h_k, $k = 0, 1, \ldots$. В практических вариантах r-алгоритмов h_k выбирается из условия приближенного поиска минимума $f(x)$ по направлению, при этом при минимизации выпуклых функций должно соблюдаться условие $h_k \geq h_k^*$, (h_k^* – значение шагового множителя, соответствующего минимуму по направлению). В общем случае необходимо, чтобы направление субградиента в точке x_{k+1} образовывало нетупой угол с направлением спуска из точки x_k.

Для полноты изложения приведем описание наиболее удачных вариантов r-алгоритмов из монографии [28] (один из них будет далее использоваться в программе ralgb5). *"При минимизации негладких выпуклых функций, определенных на E^n, наиболее удачными оказались следующие варианты r-алгоритма при проведении экспериментальных и практических расчетов. Коэффициенты растяжения пространства α_k выбираются в пределах 2–3, для шагового множителя h_k применяется адаптивный способ регулировки. Задается некоторое натуральное число m, постоянные $q > 1$ и $t_k^0 > 0$. После k шагов получаем постоянную t_k^0. Двигаемся из точки x_k в направлении спуска с шагом t_k^0 до тех пор, пока не будет выполнено условие завершения спуска по направлению, либо число шагов не станет равным m. Условие завершения спуска может состоять в том, что значение функции в очередной точке не меньше, чем значение функции в предыдущей точке; другой вариант такого условия – производная по направлению спуска в данной точке неотрицательна. Если прошло m шагов, а условие завершения спуска не выполнено, то вместо t_k^0 запоминаем $t_k^1 = qt_k^0$, где $q > 1$, и продолжаем спуск в том же направлении с большим шагом. Если после очередных m шагов условие завершения спуска не выполнено, то вместо t_k^1 берем $t_k^2 = qt_k^1$ и т. д. Так как предполагаем, что $\lim\limits_{\|x\| \to \infty} f(x) = +\infty$, то после конечного числа шагов в определенном направлении обязательно выполнится условие завершения спуска. Постоянная шага $t_k^{p_k} = q^{p_k} t_k^0 (p \in \{0, 1, 2, \ldots\})$, которая использовалась на последнем шаге, принимается в качестве начальной при спуске в новом направлении из точки x_{k+1}, т.е. $t_{k+1}^0 = t_k^{p_k}$.*

Приведенный выше способ регулировки шагового множителя основан на следующих соображениях. Допустим, что мы применили r-алгоритм с постоянным коэффициентом растяжения пространства α. Тогда за n итераций произвольное направление растянется в "среднем" в α раз, т. е. если идти в растянутом пространстве с постоянным шагом, то в первоначальном пространстве за n итераций шаг уменьшится примерно в α раз. Если расстояние до точки минимума при этом будет уменьшаться с такой же средней скоростью, т. е. примерно в α раз за n итераций, то число шагов по направлению будет ограниченным. При более медленном темпе сходимости шаг в первоначальном пространстве "в среднем" будет уменьшаться быстрее, чем расстояние до точки минимума и число шагов по направлению может неограниченно возрастать. Для предотвращения этого вводится адаптивная процедура увеличения шагового множителя h, стабилизирующая число шагов по направлению от одного до нескольких единиц. Эта процедура позволяет быстрее выявить направления, в которых функция неограниченно убывает (такие случаи часто встречаются на практике, когда задачу линейного программирования решают в пространстве двойственных переменных с использованием схемы декомпозиции по ограничениям, а задача оказывается несовместной). В этом слу-

чае шаговый множитель резко возрастает, что служит сигналом к остановке процесса спуска.

Как показали многочисленные вычислительные эксперименты и практические расчеты, в большинстве случаев при $\alpha \in [2,3]$ *и* $m = 3$ *и указанном выше способе регулировки* h *число шагов по направлению в среднем редко превосходит 2, при этом за* n *шагов* r*-алгоритма точность по функционалу, как правило, улучшается в 3-5 раз.*

В случае минимизации гладкой функции для ускорения сходимости можно применять более тонкие способы поиска минимума по направлению, например квадратичную аппроксимацию по трем точкам, процесс "золотого сечения" и др. В гладком случае хорошо зарекомендовал себя адаптивный способ регулировки шага по направлению, подобный приведенному выше, с небольшим изменением: если на данной итерации функция приняла уже после первого шага большее значение, то шаговый множитель умножается на заданное число, меньшее единицы (порядка 0,8-0,95). Это связано с тем, что в гладком случае скорость сходимости может оказаться более быстрой при более точном нахождении минимума по направлению, а дополнительное измельчение шага способствует увеличению точности поиска минимума по направлению."

Указанные выше соображения положены в основу алгоритмической схемы (условимся ее называть программой ralgb5), которая реализует вариант $r(\alpha)$-алгоритма при $\alpha_k = \alpha$, т. е. используется постоянный коэффициент растяжения пространства. Для нее точка x_{k+1} получена с помощью адаптивного наискорейшего спуска, который есть довольно грубой реализацией наискорейшего спуска. Он означает регулировку шагового множителя в направлении нормированного антисубградиента с помощью параметров n_h, $q_1 \leq 1$, $q_2 > 1$. Так, например, на 0 итерации шаг равен h_0, и если мы проделали n_h шагов в направлении антисубградиента (в преобразованном пространстве) и не нашли нашли субградиента с неострым углом к субградиенту в точке x_0, то шаг h_0^0 увеличиваем в q_2 раз. Т. е. дальше двигаемся уже с шагом $h_1^0 = q_2 \times h_0$, если снова через n_h шагов нет неострого угла, то еще раз умножаем, т. е. $h_2^0 = q_2 \times h_1^0$, и т. д. пока не найдем неострый угол. Последнее полученное значение из h_0^p которое получилось используем для следующего шага. Условие спуска по направлению (т. е. найти неострый угол) для выпуклых функций корректно, так как его выполнение обеспечивается в силу предположения (3.2). Если условие спуска по направлению выполнилось на первом же шаге, то для гладких функций можно использовать уменьшение шага, т.е. $h_1 = q_1 h_0$. И так повторяем на каждой следующей итерации...

Ниже приведено краткое описание алгоритмической схемы для программы ralgb5, которую можно изложить на любом из языков программирования.

```
  Программа ralgb5(x,alp,h0,nh,q1,q2,maxitn,epsx,epsg,intp, #вход
                   fr,xr,itn,istop)                          #выход
#
#  Входные параметры:
#    x(1:n) -- начальная (стартовая) точка (на выходе портится)
#    alp -- коэффициент растяжения пространства переменных (аргументов)
#           (рекомендуется использовать от 2.0 до 4.0)
#    h0,nh,q1,q2 -- параметры адаптивной регулировки шагового множителя
#    h0 -- величина начального шага для нормированного субградиента
#          (h0 = 1.0, либо порядка dnorm(x_0-x^*), если можно оценить)
#    q2 -- во сколько увеличивать шаговый множитель через каждые
#          nh-спусков по направлению, т.е. до тех пор пока не выполнено
#          условие завершения спуска по направлению.
#          (рекомендуются nh=2 либо 3, а q2 от 1.1 до 1.2)
```

```
#    q1 -- во сколько уменьшить шаговый множитель, если сделан
#          всего один шаг до завершения спуска по направлению
#          (рекомендуется для дифференцируемых функций использовать
#           q1 от 0.85 до 0.95, для недифференцируемых -- q1=1.0)
#    maxitn, epsx, epsg  -- критерии останова
#        maxitn -- по превышению количества итераций
#                  (рекомендуется max{100,20n})
#        epsx, epsg  -- остановы по аргументу и норме градиента
#               (рекомендуется порядка 10^{-6}).
#    intp -- печать о ходе процесса через каждые intp-итераций
#              если intp<0 -- печати не будет
#              если intp=0 -- то будет только информация об 0-ой
#                             и последней итерациях

#    Выходные параметры:
#      itn -- количество затраченных итераций
#      istop -- код останова r-алгоритма (2--по epsg, 3--по epsx,
#               4--по превышению maxitn,
#               5--не найден минимум по направлению за 500 шагов)
#      xr(1:n) -- найденная (рекордная) точка минимума функции
#      fr  -- значение функции в точке xr

#    Используются рабочие массивы:
#      b(1:n,1:n) -- матрица обратного преобразования пространства
#      g1(1:n),g2(1:n) -- используются для хранения промежуточных векторов

#    Вычисление функции f=f(x) и субградиента  g=g(x) в точке x(1:n)
#    реализуется с помощью функции calcfg(n,f,g,x)

#      Собственно сам алгоритм для программы ralgb5
#
#    Инициализация (установка) всего что нужно для нулевой итерации
#                               и того что дальше будет использовано
     w=1.d0/alp-1.d0      # далее используется взамен коэффициента alpha
     itn=0; lp=itn+intp   # установить нулевой итерации и счетчика для печати
     hs=h0;               # установить начальный шаговый множитель для
                          # дальнейшей адаптации его при одномерных спусках
     B:=I                 # присвоить единичную матрицу
     call calcfg(n,f,g,x) # вычислить f=f(x)=f(x_0) и субградиент g=g(x)=g(x_0)
     fr=f; xr:=x;         # запомнить как рекордную точку x_0
     nls=0; nlsa=0        # счетчики для количества одномерных спусков
                          # (nls--полное,nlsa--между печатаемымы итерациями)
     if(intp>=0) { write(title) # печать заголовка о ходе процесса
                   write(itn,f,fr,nlsa,nls) }  # печать о нулевой итерации
     dg=dnorm(g);         # вычислить норму субградиента
     if(dg<=epsg) { istop=2; goto 100}  # останов по норме (суб)градиента
#
#    Основной цикл реализующий r-алгоритм с адаптивной регулировкой шага
#
```

```
 10: itn=itn+1          # перейти к k-ой итерации
      g2:=B(transp)*g  # вычислить субградиент в преобразованном пространстве
      g2:=g2/dnorm(g2) # нормировать это субградиент
      g1:=B*g2         # вычислить направление движения
      dg1=dnorm(g1)    # запомнить норму единичного сдвига по направлению
      ls=0; ls1=0; dx=0.d0   # начальные параметры для одномерного спуска
  20: ls=ls+1; ls1=ls1+1       # начинаем адаптивный спуск по направлению g1
      x:=x-hs*g1; dx=dx+hs*dg1 # переходим в точку и запоминаем насколько
      call calcfg(n,f,g2,x)    # вычислить f=f(x) и g2=g(x)
      if (f<fr) { fr=f; xr:=x} # запомнить рекордную точку
      if(dnorm(g2)<epsg) {istop=2; goto 100} # останов по epsg
      if(ls1==nh) {hs=hs*q2; ls1=0}    #увеличить шаговый множитель в q2
      if(ls>500) {istop=5; goto 100}   #за 500 шагов не завершен спуск
      d=ddot(g1,g2);     # ddot -- скалярное произведение векторов
      if(d>=0) goto 20   # т.е. не выполнено условие завершения спуска по g1
                         # конец цикла по метке 20
      if(ls==1) hs=hs*q1 # уменьшить шаговый множитель (для дифф. функц)
      nls=nls+ls; nlsa=nlsa+ls # донакопить количества одномерных спусков
      if(itn==lp) {write(itn,f,fr,nlsa,nls)  # печать инфо о итерации
                   nlsa=0; lp=lp+intp }       # подготовка next print-итерации
      if(dabs(dx)<=epsx) {istop=3; goto 100} # останов по аргументу
#  Подготовка к очередной итерации (матрицы B_{k+1} и g_{k+1})
      g1:=g2-g                  # вычислить разность субградиентов
      g:=B(traspose)*g1         # разность в преобразованном пространстве
      g:=g/dnorm(g)             # нормировать эту разность
      B:=B*(I+w*g*g(transpose))  # подготовить матрицу B_{k+1}
      g:=g2;                    # запомнить g_{k+1}
     if(itn<maxitn) go to 10;     # перейти к очередной итерации
     istop=4;  # если нет то останов по превышению maxitn
#  Закончить итерации ....
  100: if(intp>=0) write(itn,f,fr,nlsa,nls) # печать info о последней итерации
      return  # выход из программы
```

Алгоритмическая схема программы ralgb5 на языке Ratfor использована для минимизации негладких функций, связанных с решением непрерывной оптимизационной задачи C для учета меневренности режимами загрузки энергоблоков в разделе 4. В следующем разделе дадим ее код на языке Фортран, который может быть использован автономно для безусловной минизации произвольной выпуклой функции, для которой требуется на том же Фортране написать подпрограмму вычисления значения функции и ее субградиента.

3.3 Фортрановский код программы ralgb5.

```
      subroutine ralgb5(n,x,calcfg,
     a                  alp,h0,nh,q1,q2,maxitn,epsx,epsg,intp,
     b                  fr,xr,itn,istop,
     c                  b,g,g1,g2)
C
C  Входные параметры:
C    n -- размерность пространства переменных
```

```
C     alp -- коэффициент растяжения пространства
C     h0,nh,q1,q2 -- параметры адаптивной регулировки шагового множителя
C     maxitn, epsx, epsg  -- критерии останова
C     intp -- печать о ходе процесса через каждые intp-итераций
C             (если intp<0 -- печати не будет)
C     x(n) -- начальная точка (на выходе портится)
C     calcfg -- имя внешней (external) подпрограммы
C                calcfg(n,f,g,x) -- подпрограмма для вычисления f и g
C     Выходные параметры:
C       itn -- число затраченных итераций
C       istop -- код останова (2-по epsg, 3-по epsx, 4-по числу итераций,
C                5--не найден минимум по направлению)
C       xr(n) -- найденная точка минимума функции xr(n)
C       fr  -- значение функции в точке минимума
C     Рабочие массивы:
C       b(n,n) -- матрица обратного преобразования пространства
C       g1(n),g2(n) -- используются для хранения промежуточных векторов
C
C     Используется функция dsqrt().
C
C     Последняя модификация 03.11.2007 /Стецюк П.И./
C     Тел. (044)526-21-68, e-mail: stetsyuk@d120.icyb.kiev.ua
C
      implicit real*8(a-h,o-z),integer*4(i-n)
      external calcfg
      dimension g(n),x(n),b(n,n),xr(n),g1(n),g2(n)
C     Установка нуля и начальных параметров
      iprint=6
      w=1./alp-1.
      hs=h0
      itn=0
      lp=itn+intp
      do i=1,n
         do j=1,n
            b(j,i)=0.
         enddo
         b(i,i)=1.
      enddo
      call calcfg(n,f,g,x)
      fr=f
      do i=1,n
         xr(i)=x(i)
      enddo
      dg=0.d0
      do i=1,n
         dg=dg+g(i)*g(i)
      enddo
      istop=2
      if(dsqrt(dg).le.epsg) goto 100
      nls=0
      nlsa=0
```

```
      if(intp.ge.0) write(iprint,3000)
      if(intp.ge.0) write(iprint,3100) itn,f,fr,nlsa,nls
C     Основное тело программы, реализующей алгоритм
      do itn=1,maxitn
C     Вычислить субградиент в преобразованном пространстве
         dg2=0.d0
         do i=1,n
            d=0.
            do j=1,n
               d=d+b(j,i)*g(j)
            enddo
            g2(i)=d
            dg2=dg2+d*d
         enddo
      dg2=1.d0/dsqrt(dg2)
      do i=1,n
         g2(i)=dg2*g2(i)
      enddo
C     Вычислить направление движения
         dg1=0.d0
         do i=1,n
            d=0.
            do j=1,n
               d=d+b(i,j)*g2(j)
            enddo
            g1(i)=d
            dg1=dg1+d*d
         enddo
         dg1=dsqrt(dg1)
C      Одномерный спуск по направлению
         ls=0
         ls1=0
         lsa=0
         dx=0.d0
   20    continue
           ls=ls+1
           ls1=ls1+1
           dx=dx+hs*dg1
           do i=1,n
              x(i)=x(i)-hs*g1(i)
           enddo
           call calcfg(n,f,g2,x)
           if(f.lt.fr) then
             fr=f
             do i=1,n
                xr(i)=x(i)
             enddo
           endif
C     Критерий останова по норме градиента
         dg2=0.d0
         do i=1,n
```

```
            dg2=dg2+g2(i)*g2(i)
          enddo
          istop=2
          if(dsqrt(dg2).le.epsg) goto 100
          d=0.
          do i=1,n
            d=d+g1(i)*g2(i)
          enddo
          if(ls1.gt.nh) then
            hs=hs*q2
            ls1=0
          endif
          if(d.gt.0.d0) go to 20
        if(ls.eq.1) hs=hs*q1
        nls=nls+ls
        nlsa=nlsa+ls
        if(itn.eq.lp) then
          write(iprint,3100) itn,f,fr,nlsa,nls
          nlsa=0
          lp=lp+intp
        endif
        istop=3
        if(dabs(dx).lt.epsx) goto 100
C       Вычислить нормированную разность субградиентов
        do i=1,n
          g1(i)=g2(i)-g(i)
        enddo
        dg=0.d0
        do i=1,n
          d=0.
          do j=1,n
            d=d+b(j,i)*g1(j)
          enddo
          g(i)=d
          dg=dg+d*d
        enddo
      dg=1.d0/dsqrt(dg)
      do i=1,n
        g(i)=dg*g(i)
      enddo
C     Пересчитать матрицу B
        do i=1,n
          d=0.
          do j=1,n
            d=d+b(i,j)*g(j)
          enddo
          d=w*d
          do j=1,n
            b(i,j)=b(i,j)+d*g(j)
          enddo
        enddo
```

```
      do i=1,n
        g(i)=g2(i)
      enddo
    enddo
    itn=itn-1
    istop=4
100 continue
    if(intp.ge.0) write(iprint,3100) itn,f,fr,nlsa,nls
    return
3000 format(/10x,'␣Протокол␣процесса␣негладкой␣оптимизации␣'/2x,
   a       '␣␣Itn.',7x,'..f(x)..',13x,'...f(x_r)...',
   b       5x,'LS',4x,'LSa')
3100 format(2x,i5,2(2x,1pd18.10),3(2x,i5))
    end
```

3.4 Метод негладких штрафных функций

Метод штрафных функций с негладкой функцией штрафа используется для учета ограничений, заданных в форме равенств или неравенств, в условных задачах математического программирования. Он позволяет свести условную задачу математического программирования на минимум к задаче безусловной минимизации негладкой функции. Рассмотрим метод негладких штрафных функций согласно монографии [27], стр. 188–189.

Пусть рассматриваетсяя задача выпуклого программирования

$$\min f_0(x) \quad \text{при} \quad f_i(x) \le 0, \quad i = 1, \ldots, m. \tag{3.3}$$

Определим функцию

$$S(x) = f_0(x) + \sum_{i=1}^{m} p_i \left[f_i(x) \right], \tag{3.4}$$

где $p_i(t)$, $i = 1, \ldots, m$, удовлетворяют условиям выпуклости, $p_i(t) = 0$ для $t < 0$, $p_i(t) \ge 0$ для $t > 0$. Допустим, что существует x^* — оптимальное решение задачи $\inf_x S(x)$.

Теорема 3 *Для того чтобы x^* было оптимальным решением исходной задачи, необходимо, чтобы*

$$\lim_{t \to +0} \frac{p_i(t)}{t} \ge \overline{y_i}, \quad i = 1, \ldots, m,$$

где $\overline{y} = (\overline{y_1}, \ldots, \overline{y_m})$ – некоторые множители Лагранжа задачи (3.3).

Для того, чтобы (3.3) и (3.4) имели одинаковое множество оптимальных решений, достаточно, чтобы $\lim_{t \to 0} \frac{p_i(t)}{t} > \overline{y_i}$.

Простейший вариант негладкой функции штрафа имеет вид

$$p(t) = \begin{cases} 0, & t \le 0, \\ ct, & t > 0. \end{cases} \tag{3.5}$$

Из теоремы 3 вытекает, что для того чтобы задача минимизации $S(x)$ была эквивалентна задаче (3.3) при

$$p_i(t) = \begin{cases} 0, & t \le 0, \\ c_i t, & t > 0. \end{cases} \qquad i = 1, \ldots, m, \tag{3.6}$$

достаточно брать $c_i > \overline{y_i}$.