Nonlinear Optimization Ex.3

Ariel Vishne 204149371

30 05 2022

Ex. 1

Our function Y_k is defined by:

$$Y_k = \max(0, 10 - (1 + 0.007\sum_{i=1}^{k}\xi_i)\cdot s) = max(0, 10 - s - 0.007\sum_{i=1}^{k}\xi_i \cdot s)$$

For $0 \leq k \leq 10000$ and $\xi_1, \ldots, \xi_{10000}$ takes probability ± 1 with equal probability 0.5 where Y_0 is defined by:

$$Y_0 = \max(0, 10 - s)$$

We would like to find the optimal stopping value

$$V(s) = max_\tau(\mathbb{E}(Y_\tau))$$

We define recursively from end to start (for $n = 10000$):

$$V_n = Y_n = \max(0, 10 - s - 0.007\sum_{i=1}^{n}\xi_i \cdot s)$$

For general k we have:

$$V_k = \max(Y_k, \mathbb{E}[V_{k+1}|\xi_1, \ldots, \xi_k]) = max(0, 10 - s - 0.007\sum_{i=1}^{k}\xi_i \cdot s, \mathbb{E}[V_{k+1}|\xi_1, \ldots, \xi_k])$$

We use backward recursion starting with V_n to calculate the values of all V_k. This ostensibly requires $2^n = 2^{10000}$ different values at all leaves of the tree, but we notice that the values for some number of iterations k is actually bounded between $[-k, k]$ where more specifically the values for a given k are $[-k, -k+2, \ldots, k-2, k]$ for a total of $2k+1$ values possible. We can thus create a matrix going backward in which the rows will represent possible events of $\sum_{i=1}^{k}\xi_i$ and the columns will represent k (the current time in the series). Starting with filling values at the last columns, we will propagate the values towards the start by averaging on the adjacent values in the columns to the right using the following function:

```
dynamic.table <- function(n, Y0, s){
  d <- matrix(-1, nrow = ((2*n) +1), ncol = (n+1))
  final.vals.indices <- seq(1, ((2*n) + 1), 2)
  events <- seq(-n,n)
  for (i in final.vals.indices){
    d[i,n+1] <- max(0, Y0 - (0.007 * events[i] * s))
  }
  for (i in n:1){
    if (i %% 500 == 0)
    {print(paste("reached iteration number", i))}
    vals.layer.i <- seq(n - i + 2, n + i, 2)
    for(j in vals.layer.i){
      d[j,i] <- mean(c(
        d[j-1,i+1],
        d[j+1,i+1]
      ))
    }
  }
  return(d)
}
```

For example, for $k = 3, s = 9.9$ we will have the following values:

```
k = 3
s <- 9.9
d <- dynamic.table(k, 10 - s, s)
d <- round(d, 4)
events <- seq(-k,k)
d[d==-1] <- ""
k.cols <- paste("k", seq(0,k), sep = "")
d <- as_tibble(cbind(events, d))
colnames(d) <- c("events", k.cols)
print(d)
```

```
## # A tibble: 7 x 5
##   events k0       k1       k2       k3
##   <chr>  <chr>    <chr>    <chr>    <chr>
## 1 -3     ""       ""       ""       "0.3079"
## 2 -2     ""       ""       "0.2386" ""
## 3 -1     ""       "0.1693" ""       "0.1693"
## 4 0      "0.1135" ""       "0.1"    ""
## 5 1      ""       "0.0577" ""       "0.0307"
## 6 2      ""       ""       "0.0153" ""
## 7 3      ""       ""       ""       "0"
```

We note that the size of the matrix we create is therefore of size $2n + 1 \times n + 1$ (we include k_0), while the number of cells that are actually filled is actually $\frac{n(n+1)}{2}$ since we only fill $k + 1$ possible values per time period k. We could have chosen a somewhat better representation creating a matrix of size $k \times k + 1$ where the columns still represented the times periods but the rows represented the number of positive ξ_i so far up to the point k. This would have meant that for some cell $[i, j]$ we would take the sum to be of $i - 1$ positive ξ_i's and of $j - i$

negative ξ_i's, thus representing the event that after k steps the total sum is $i-1-j+i=2i-j-1$. In total this alternative representation would result in an upper triangular matrix where the diagonal represents events in which all ξ_i's are positive and the lower triangle represents events that are impossible and will be assigned some default value (like we did in our representation). This alternative representation will be somewhat more compact but will not change the number of cells we actually have to go over during the forward process (i.e. it would somewhat improve memory usage but not computational time). For visualization and simplicity our representation is still valid and we will continue with it for clarity (given a much larger n this might be of slight significance)

Next, we create the following helper functions to help us calculate Y_k and V_k:

```
yk <- function(s, k, xi, inital.value = 10, with.max = TRUE){
  a <- 0
  if(k != 0){
   coef <- 1 + ((0.007) * sum(xi[1:(k+1)]))
  }
  else{
   coef <- 1
  }
  b <- inital.value - (coef * s)
  if (with.max == TRUE){
    return(max(a,b))
  }
  else{
    return(b)
  }
}

compute.yk <- function(s, n, xi, initial.value, with.max = TRUE){
  yk.array <- rep(0,n+1)
  yk.array[1] <- yk(s=s, k=0, xi=xi, inital.value = initial.value, with.max = with.max)
  for (i in 1:n){
    yk.array[i+1] <- yk(s = s, k = i, xi = xi, inital.value = initial.value, with.max = with.ma
x)
  }
  return(yk.array)
}

compute.vk <- function(n, s, xi, initial.value = 10){

  d <- dynamic.table(n, initial.value - s, s)
  events <- seq(-n,n)

  coefs <- cumsum(xi)
  vk.array <- rep(0,n+1)
  for (i in 1:(n+1)){
    vk.array[i] <- d[which(events == coefs[i]),i]
  }
  return(vk.array)
}
```

Having built a dynamic table which relies on our value s and the total number of iterations n, and with the helper functions provided, we can look at some sequence $Y_1, \ldots, Y_n$ and figure out the corresponding $V_1, \ldots, V_n$. We will then conclude by:

$$\tau^* = min_k : Y_k = V_k$$

We finally create the required function:

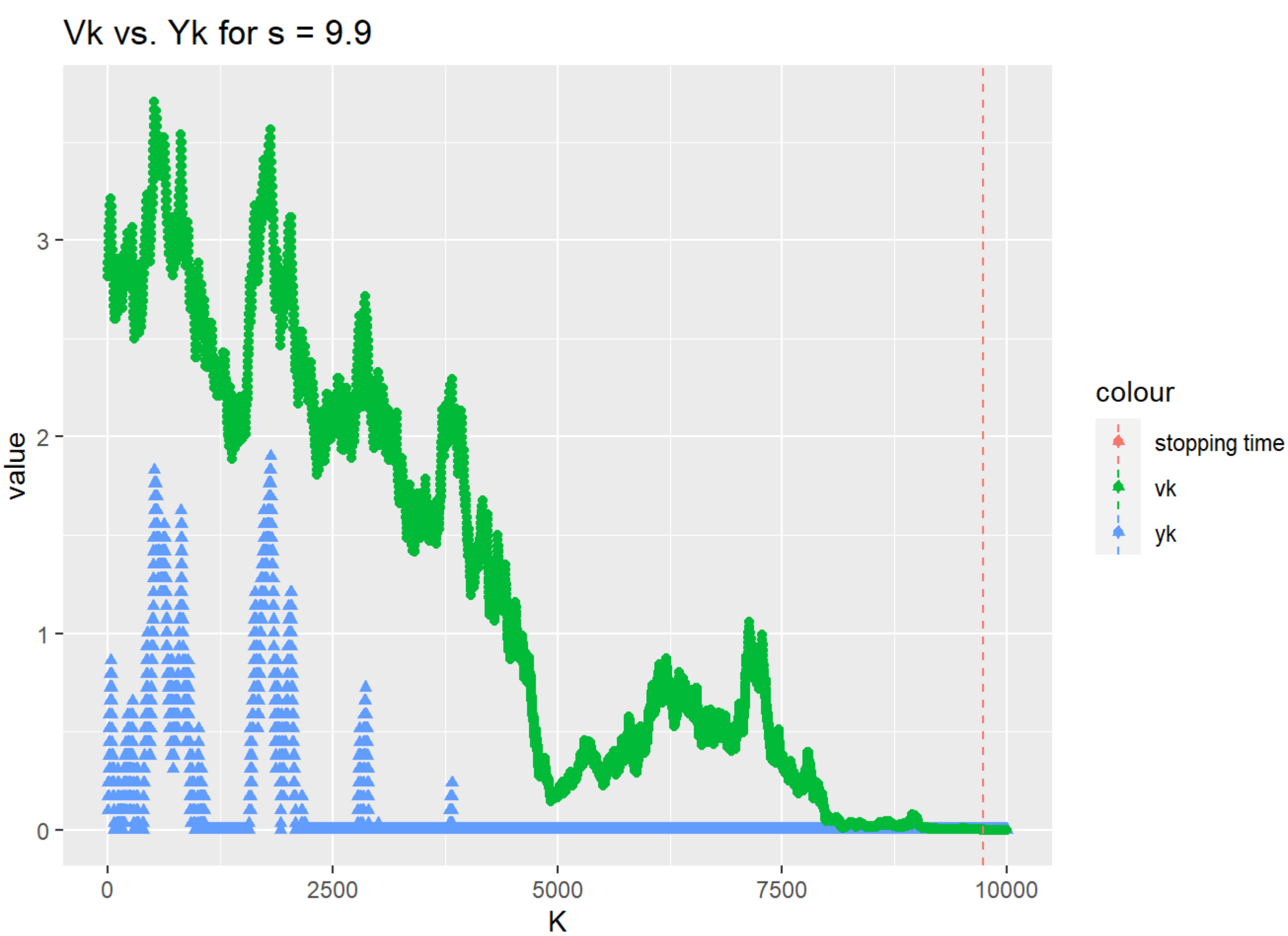
```
vmax1 <- function(s, n = 10000, initial = 10, print.plot = TRUE, print.data = FALSE){
  k <- seq(0,n)
  xi <- sample(x = c(-1,1), size = n, replace = TRUE)
  xi <- c(0, xi)
  sum.xi <- cumsum(xi)
  yk.array <- compute.yk(s,n,xi, initial.value = initial)
  vk.array <- compute.vk(n, s, xi, initial.value = initial)
  data <- as.tibble(cbind(k,yk.array, vk.array, xi, sum.xi))
  if (
    sum(
      round(vk.array, 4) == round(yk.array,4)
      ) > 0
    ){
    tau <- which(round(data$yk.array, 4) == round(data$vk.array, 4))[[1]] - 1
  }
  else{
    tau <- n
  }
  plot.title <- paste("Vk vs. Yk for s =",s)
  pl <- ggplot(data = data, mapping = aes(x = k)) +
    geom_point(mapping = aes(y = yk.array, color = "yk"),shape = 17) +
    geom_point(mapping = aes(y = vk.array, color = "vk")) +
    geom_vline(mapping = aes(color = "stopping time", xintercept = tau), linetype = "dashed") +
    xlab("K") + ylab("value") + ggtitle(plot.title)

  if (print.plot == TRUE){
    print(pl)
  }
  if (print.data == TRUE){
  print(data)
  }
  return(tau)
}
```

We emphasize that the essence of the function is that of creating the dynamic table, which is itself the purpose of the exercise. We randomize ξ_i for the sake of visualization only.

We initialize for the following results

```
set.seed(1)
tau1 <- vmax1(s = 9.9)
```

```
## [1] "reached iteration number 10000"
## [1] "reached iteration number 9500"
## [1] "reached iteration number 9000"
## [1] "reached iteration number 8500"
## [1] "reached iteration number 8000"
## [1] "reached iteration number 7500"
## [1] "reached iteration number 7000"
## [1] "reached iteration number 6500"
## [1] "reached iteration number 6000"
## [1] "reached iteration number 5500"
## [1] "reached iteration number 5000"
## [1] "reached iteration number 4500"
## [1] "reached iteration number 4000"
## [1] "reached iteration number 3500"
## [1] "reached iteration number 3000"
## [1] "reached iteration number 2500"
## [1] "reached iteration number 2000"
## [1] "reached iteration number 1500"
## [1] "reached iteration number 1000"
## [1] "reached iteration number 500"
```

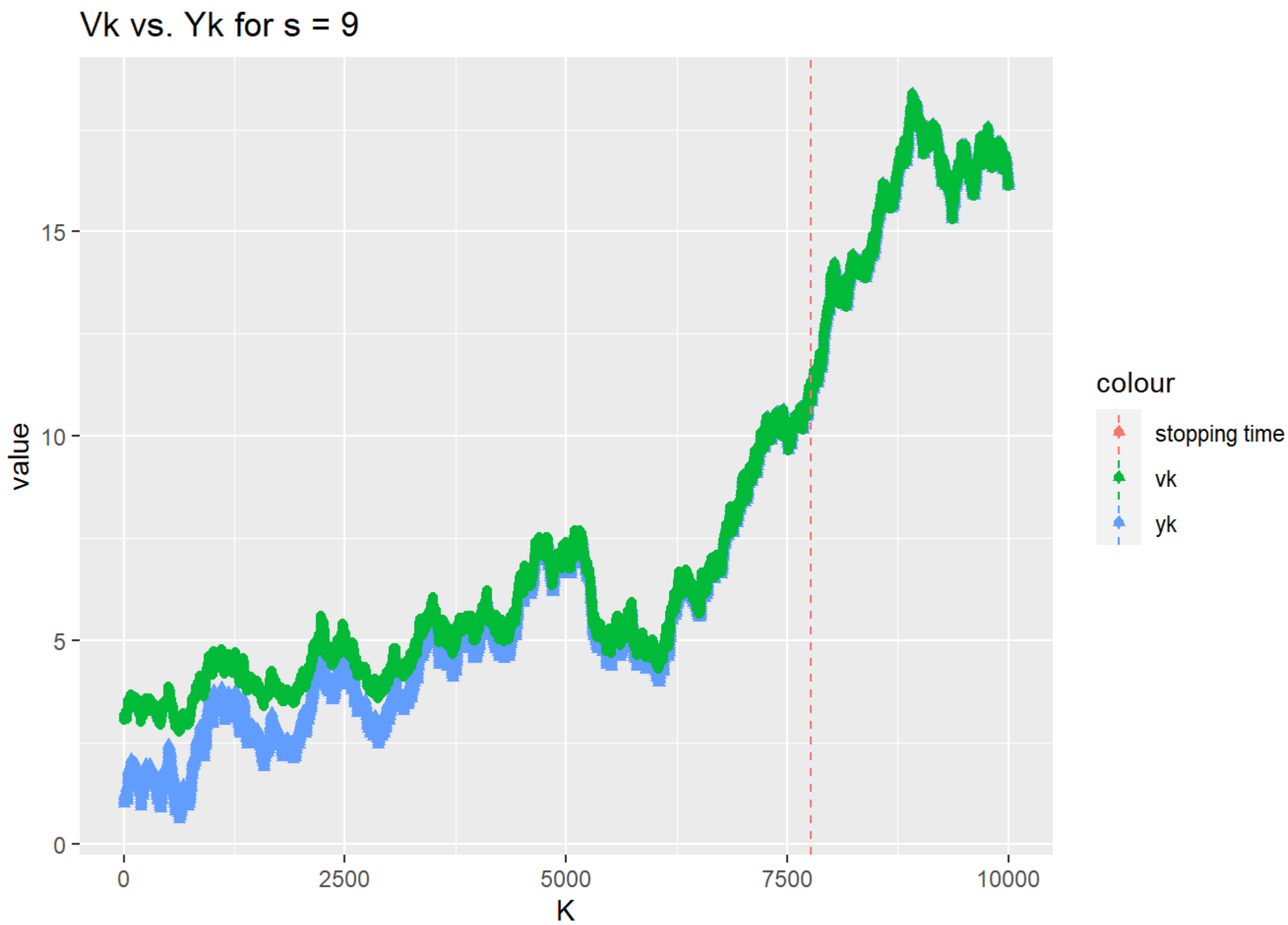



```
print(paste("optimal stopping for s = 9.9 is", tau1))
```

```
## [1] "optimal stopping for s = 9.9 is 9739"
```

```
tau2 <- vmax1(s = 9)
```

```
## [1] "reached iteration number 10000"
## [1] "reached iteration number 9500"
## [1] "reached iteration number 9000"
## [1] "reached iteration number 8500"
## [1] "reached iteration number 8000"
## [1] "reached iteration number 7500"
## [1] "reached iteration number 7000"
## [1] "reached iteration number 6500"
## [1] "reached iteration number 6000"
## [1] "reached iteration number 5500"
## [1] "reached iteration number 5000"
## [1] "reached iteration number 4500"
## [1] "reached iteration number 4000"
## [1] "reached iteration number 3500"
## [1] "reached iteration number 3000"
## [1] "reached iteration number 2500"
## [1] "reached iteration number 2000"
## [1] "reached iteration number 1500"
## [1] "reached iteration number 1000"
## [1] "reached iteration number 500"
```

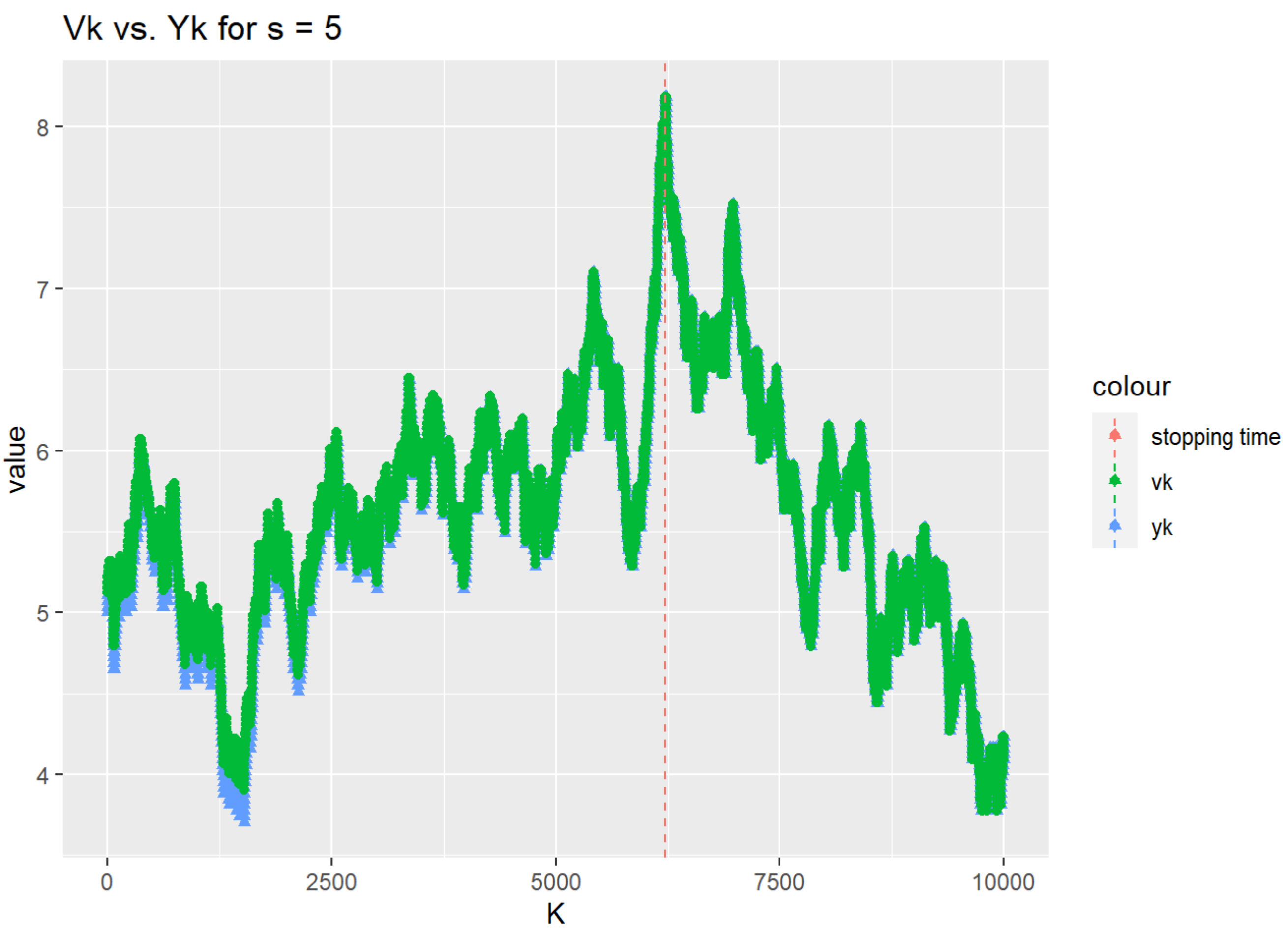



```
print(paste("optimal stopping for s = 9.9 is", tau2))
```

```
## [1] "optimal stopping for s = 9.9 is 7764"
```

```
tau2 <- vmax1(s = 5)
```

```
## [1] "reached iteration number 10000"
## [1] "reached iteration number 9500"
## [1] "reached iteration number 9000"
## [1] "reached iteration number 8500"
## [1] "reached iteration number 8000"
## [1] "reached iteration number 7500"
## [1] "reached iteration number 7000"
## [1] "reached iteration number 6500"
## [1] "reached iteration number 6000"
## [1] "reached iteration number 5500"
## [1] "reached iteration number 5000"
## [1] "reached iteration number 4500"
## [1] "reached iteration number 4000"
## [1] "reached iteration number 3500"
## [1] "reached iteration number 3000"
## [1] "reached iteration number 2500"
## [1] "reached iteration number 2000"
## [1] "reached iteration number 1500"
## [1] "reached iteration number 1000"
## [1] "reached iteration number 500"
```




```
print(paste("optimal stopping for s = 5 is", tau2))
```

```
## [1] "optimal stopping for s = 5 is 6221"
```

We can see that as s is smaller, which means that the inital value of the asset is larger, then the stopping time tends to be earlier. We note that the stopping time is the earliest value k at which all future leaves of branches will have a non-zero value. This is because the probability is 0.5 for 1 and -1 and therefore if all future values are non-zero then the average will be same for V_k and Y_k.

Ex. 2

Having our dynamic.table enables us to quickly also compute what values it is necessary to stop at that moment k by comparing the possible V_k values we computed with the possible Y_k outcomes.

```
vmax2 <- function(k, s, n = 10000, initial = 10){
  d <- dynamic.table(n, initial - s, s)
  our.k <- k + 1
  vals.layer.k <- seq(n - our.k + 2, n + our.k, 2)
  events <- seq(-n,n)
  sums.to.stop <- c()
  events.to.stop <- rep(-1, 2 * n +1)
  for (val.index in vals.layer.k){
  if (
    round(d[val.index,our.k],4) ==
    round((initial - s) - (0.007 * events[val.index] * s),4)
    )
    {
    events.to.stop[val.index] = 1
    sums.to.stop <- c(sums.to.stop, events[val.index])
    }
  }
values.to.stop <- d[,our.k][events.to.stop == 1]
return(sums.to.stop)
}
```

```
k.stop.values <- vmax2(k = 3000, s = 9.9, n = 10000)
```

```
## [1] "reached iteration number 10000"
## [1] "reached iteration number 9500"
## [1] "reached iteration number 9000"
## [1] "reached iteration number 8500"
## [1] "reached iteration number 8000"
## [1] "reached iteration number 7500"
## [1] "reached iteration number 7000"
## [1] "reached iteration number 6500"
## [1] "reached iteration number 6000"
## [1] "reached iteration number 5500"
## [1] "reached iteration number 5000"
## [1] "reached iteration number 4500"
## [1] "reached iteration number 4000"
## [1] "reached iteration number 3500"
## [1] "reached iteration number 3000"
## [1] "reached iteration number 2500"
## [1] "reached iteration number 2000"
## [1] "reached iteration number 1500"
## [1] "reached iteration number 1000"
## [1] "reached iteration number 500"
```

```
print("For k = 3000 and s = 9.9 the sums of xi_s to this point for which this is an optimal stop
ping time are:")
```

```
## [1] "For k = 3000 and s = 9.9 the sums of xi_s to this point for which this is an optimal sto
pping time are:"
```

```
print(k.stop.values)
```

```
##    [1] -3000 -2998 -2996 -2994 -2992 -2990 -2988 -2986 -2984 -2982 -2980 -2978
##   [13] -2976 -2974 -2972 -2970 -2968 -2966 -2964 -2962 -2960 -2958 -2956 -2954
##   [25] -2952 -2950 -2948 -2946 -2944 -2942 -2940 -2938 -2936 -2934 -2932 -2930
##   [37] -2928 -2926 -2924 -2922 -2920 -2918 -2916 -2914 -2912 -2910 -2908 -2906
##   [49] -2904 -2902 -2900 -2898 -2896 -2894 -2892 -2890 -2888 -2886 -2884 -2882
##   [61] -2880 -2878 -2876 -2874 -2872 -2870 -2868 -2866 -2864 -2862 -2860 -2858
##   [73] -2856 -2854 -2852 -2850 -2848 -2846 -2844 -2842 -2840 -2838 -2836 -2834
##   [85] -2832 -2830 -2828 -2826 -2824 -2822 -2820 -2818 -2816 -2814 -2812 -2810
##   [97] -2808 -2806 -2804 -2802 -2800 -2798 -2796 -2794 -2792 -2790 -2788 -2786
##  [109] -2784 -2782 -2780 -2778 -2776 -2774 -2772 -2770 -2768 -2766 -2764 -2762
##  [121] -2760 -2758 -2756 -2754 -2752 -2750 -2748 -2746 -2744 -2742 -2740 -2738
##  [133] -2736 -2734 -2732 -2730 -2728 -2726 -2724 -2722 -2720 -2718 -2716 -2714
##  [145] -2712 -2710 -2708 -2706 -2704 -2702 -2700 -2698 -2696 -2694 -2692 -2690
##  [157] -2688 -2686 -2684 -2682 -2680 -2678 -2676 -2674 -2672 -2670 -2668 -2666
##  [169] -2664 -2662 -2660 -2658 -2656 -2654 -2652 -2650 -2648 -2646 -2644 -2642
##  [181] -2640 -2638 -2636 -2634 -2632 -2630 -2628 -2626 -2624 -2622 -2620 -2618
##  [193] -2616 -2614 -2612 -2610 -2608 -2606 -2604 -2602 -2600 -2598 -2596 -2594
##  [205] -2592 -2590 -2588 -2586 -2584 -2582 -2580 -2578 -2576 -2574 -2572 -2570
##  [217] -2568 -2566 -2564 -2562 -2560 -2558 -2556 -2554 -2552 -2550 -2548 -2546
##  [229] -2544 -2542 -2540 -2538 -2536 -2534 -2532 -2530 -2528 -2526 -2524 -2522
##  [241] -2520 -2518 -2516 -2514 -2512 -2510 -2508 -2506 -2504 -2502 -2500 -2498
##  [253] -2496 -2494 -2492 -2490 -2488 -2486 -2484 -2482 -2480 -2478 -2476 -2474
##  [265] -2472 -2470 -2468 -2466 -2464 -2462 -2460 -2458 -2456 -2454 -2452 -2450
##  [277] -2448 -2446 -2444 -2442 -2440 -2438 -2436 -2434 -2432 -2430 -2428 -2426
##  [289] -2424 -2422 -2420 -2418 -2416 -2414 -2412 -2410 -2408 -2406 -2404 -2402
##  [301] -2400 -2398 -2396 -2394 -2392 -2390 -2388 -2386 -2384 -2382 -2380 -2378
##  [313] -2376 -2374 -2372 -2370 -2368 -2366 -2364 -2362 -2360 -2358 -2356 -2354
##  [325] -2352 -2350 -2348 -2346 -2344 -2342 -2340 -2338 -2336 -2334 -2332 -2330
##  [337] -2328 -2326 -2324 -2322 -2320 -2318 -2316 -2314 -2312 -2310 -2308 -2306
##  [349] -2304 -2302 -2300 -2298 -2296 -2294 -2292 -2290 -2288 -2286 -2284 -2282
##  [361] -2280 -2278 -2276 -2274 -2272 -2270 -2268 -2266 -2264 -2262 -2260 -2258
##  [373] -2256 -2254 -2252 -2250 -2248 -2246 -2244 -2242 -2240 -2238 -2236 -2234
##  [385] -2232 -2230 -2228 -2226 -2224 -2222 -2220 -2218 -2216 -2214 -2212 -2210
##  [397] -2208 -2206 -2204 -2202 -2200 -2198 -2196 -2194 -2192 -2190 -2188 -2186
##  [409] -2184 -2182 -2180 -2178 -2176 -2174 -2172 -2170 -2168 -2166 -2164 -2162
##  [421] -2160 -2158 -2156 -2154 -2152 -2150 -2148 -2146 -2144 -2142 -2140 -2138
##  [433] -2136 -2134 -2132 -2130 -2128 -2126 -2124 -2122 -2120 -2118 -2116 -2114
##  [445] -2112 -2110 -2108 -2106 -2104 -2102 -2100 -2098 -2096 -2094 -2092 -2090
##  [457] -2088 -2086 -2084 -2082 -2080 -2078 -2076 -2074 -2072 -2070 -2068 -2066
##  [469] -2064 -2062 -2060 -2058 -2056 -2054 -2052 -2050 -2048 -2046 -2044 -2042
##  [481] -2040 -2038 -2036 -2034 -2032 -2030 -2028 -2026 -2024 -2022 -2020 -2018
##  [493] -2016 -2014 -2012 -2010 -2008 -2006 -2004 -2002 -2000 -1998 -1996 -1994
##  [505] -1992 -1990 -1988 -1986 -1984 -1982 -1980 -1978 -1976 -1974 -1972 -1970
##  [517] -1968 -1966 -1964 -1962 -1960 -1958 -1956 -1954 -1952 -1950 -1948 -1946
##  [529] -1944 -1942 -1940 -1938 -1936 -1934 -1932 -1930 -1928 -1926 -1924 -1922
##  [541] -1920 -1918 -1916 -1914 -1912 -1910 -1908 -1906 -1904 -1902 -1900 -1898
##  [553] -1896 -1894 -1892 -1890 -1888 -1886 -1884 -1882 -1880 -1878 -1876 -1874
##  [565] -1872 -1870 -1868 -1866 -1864 -1862 -1860 -1858 -1856 -1854 -1852 -1850
##  [577] -1848 -1846 -1844 -1842 -1840 -1838 -1836 -1834 -1832 -1830 -1828 -1826
##  [589] -1824 -1822 -1820 -1818 -1816 -1814 -1812 -1810 -1808 -1806 -1804 -1802
##  [601] -1800 -1798 -1796 -1794 -1792 -1790 -1788 -1786 -1784 -1782 -1780 -1778
##  [613] -1776 -1774 -1772 -1770 -1768 -1766 -1764 -1762 -1760 -1758 -1756 -1754
```

```
##  [625] -1752 -1750 -1748 -1746 -1744 -1742 -1740 -1738 -1736 -1734 -1732 -1730
##  [637] -1728 -1726 -1724 -1722 -1720 -1718 -1716 -1714 -1712 -1710 -1708 -1706
##  [649] -1704 -1702 -1700 -1698 -1696 -1694 -1692 -1690 -1688 -1686 -1684 -1682
##  [661] -1680 -1678 -1676 -1674 -1672 -1670 -1668 -1666 -1664 -1662 -1660 -1658
##  [673] -1656 -1654 -1652 -1650 -1648 -1646 -1644 -1642 -1640 -1638 -1636 -1634
##  [685] -1632 -1630 -1628 -1626 -1624 -1622 -1620 -1618 -1616 -1614 -1612 -1610
##  [697] -1608 -1606 -1604 -1602 -1600 -1598 -1596 -1594 -1592 -1590 -1588 -1586
##  [709] -1584 -1582 -1580 -1578 -1576 -1574 -1572 -1570 -1568 -1566 -1564 -1562
##  [721] -1560 -1558 -1556 -1554 -1552 -1550 -1548 -1546 -1544 -1542 -1540 -1538
##  [733] -1536 -1534 -1532 -1530 -1528 -1526 -1524 -1522 -1520 -1518 -1516 -1514
##  [745] -1512 -1510 -1508 -1506 -1504 -1502 -1500 -1498 -1496 -1494 -1492 -1490
##  [757] -1488 -1486 -1484 -1482 -1480 -1478 -1476 -1474 -1472 -1470 -1468 -1466
##  [769] -1464 -1462 -1460 -1458 -1456 -1454 -1452 -1450 -1448 -1446 -1444 -1442
##  [781] -1440 -1438 -1436 -1434 -1432 -1430 -1428 -1426 -1424 -1422 -1420 -1418
##  [793] -1416 -1414 -1412 -1410 -1408 -1406 -1404 -1402 -1400 -1398 -1396 -1394
##  [805] -1392 -1390 -1388 -1386 -1384 -1382 -1380 -1378 -1376 -1374 -1372 -1370
##  [817] -1368 -1366 -1364 -1362 -1360 -1358 -1356 -1354 -1352 -1350 -1348 -1346
##  [829] -1344 -1342 -1340 -1338 -1336 -1334 -1332 -1330 -1328 -1326 -1324 -1322
##  [841] -1320 -1318 -1316 -1314 -1312 -1310 -1308 -1306 -1304 -1302 -1300 -1298
##  [853] -1296 -1294 -1292 -1290 -1288 -1286 -1284 -1282 -1280 -1278 -1276 -1274
##  [865] -1272 -1270 -1268 -1266 -1264 -1262 -1260 -1258 -1256 -1254 -1252 -1250
##  [877] -1248 -1246 -1244 -1242 -1240 -1238 -1236 -1234 -1232 -1230 -1228 -1226
##  [889] -1224 -1222 -1220 -1218 -1216 -1214 -1212 -1210 -1208 -1206 -1204 -1202
##  [901] -1200 -1198 -1196 -1194 -1192 -1190 -1188 -1186 -1184 -1182 -1180 -1178
##  [913] -1176 -1174 -1172 -1170 -1168 -1166 -1164 -1162 -1160 -1158 -1156 -1154
##  [925] -1152 -1150 -1148 -1146 -1144 -1142 -1140 -1138 -1136 -1134 -1132 -1130
##  [937] -1128 -1126 -1124 -1122 -1120 -1118 -1116 -1114 -1112 -1110 -1108 -1106
##  [949] -1104 -1102 -1100 -1098 -1096 -1094 -1092 -1090 -1088 -1086 -1084 -1082
##  [961] -1080 -1078 -1076 -1074 -1072 -1070 -1068 -1066 -1064 -1062 -1060 -1058
##  [973] -1056 -1054 -1052 -1050 -1048 -1046 -1044 -1042 -1040 -1038 -1036 -1034
##  [985] -1032 -1030 -1028 -1026 -1024 -1022 -1020 -1018 -1016 -1014 -1012 -1010
##  [997] -1008 -1006 -1004 -1002 -1000  -998  -996  -994  -992  -990  -988  -986
## [1009]  -984  -982  -980  -978  -976  -974  -972  -970  -968  -966  -964  -962
## [1021]  -960  -958  -956  -954  -952  -950  -948  -946  -944  -942  -940  -938
## [1033]  -936  -934  -932  -930  -928  -926  -924  -922  -920  -918  -916  -914
## [1045]  -912  -910  -908  -906  -904  -902  -900  -898  -896  -894  -892  -890
## [1057]  -888  -886  -884  -882  -880  -878  -876  -874  -872  -870  -868  -866
## [1069]  -864  -862  -860  -858  -856  -854  -852  -850  -848  -846  -844  -842
## [1081]  -840  -838  -836  -834  -832  -830  -828  -826  -824  -822  -820  -818
## [1093]  -816  -814  -812  -810  -808  -806  -804  -802  -800  -798  -796  -794
## [1105]  -792  -790  -788  -786  -784  -782  -780  -778  -776  -774  -772  -770
## [1117]  -768  -766  -764  -762  -760  -758  -756  -754  -752  -750  -748  -746
## [1129]  -744  -742  -740  -738  -736  -734  -732  -730  -728  -726  -724  -722
## [1141]  -720  -718  -716  -714  -712  -710  -708  -706  -704  -702  -700  -698
## [1153]  -696  -694  -692  -690  -688  -686  -684  -682  -680  -678  -676  -674
## [1165]  -672  -670  -668  -666  -664  -662  -660  -658  -656  -654  -652  -650
## [1177]  -648  -646  -644  -642  -640  -638  -636  -634  -632  -630  -628  -626
## [1189]  -624  -622  -620  -618  -616  -614  -612  -610  -608  -606  -604  -602
## [1201]  -600  -598  -596  -594  -592  -590  -588  -586  -584  -582  -580  -578
## [1213]  -576  -574  -572  -570  -568  -566  -564  -562  -560  -558  -556  -554
## [1225]  -552  -550  -548  -546  -544  -542  -540  -538  -536  -534  -532  -530
## [1237]  -528  -526  -524  -522  -520  -518  -516  -514  -512  -510  -508  -506
```

```
## [1249]  -504  -502  -500  -498  -496  -494  -492  -490  -488  -486  -484  -482
## [1261]  -480  -478  -476  -474  -472  -470  -468  -466  -464  -462  -460  -458
## [1273]  -456  -454  -452  -450  -448  -446  -444  -442  -440  -438  -436  -434
## [1285]  -432  -430  -428  -426  -424  -422  -420  -418  -416  -414  -412  -410
## [1297]  -408  -406  -404  -402  -400  -398  -396  -394  -392  -390  -388  -386
## [1309]  -384  -382  -380  -378  -376  -374  -372  -370  -368  -366  -364  -362
## [1321]  -360  -358  -356  -354  -352  -350  -348  -346  -344  -342  -340  -338
## [1333]  -336  -334  -332  -330
```