In the following, one is located in a two-dimensional cartesian space.

We associate to each even integer a “global motion in the plane”, that is constituted of several moves associated to this even integer decompositions as a sum of two odd integers\(^1\). Every motion has \((0,0)\) point as origin.

We code:

- an \(n\) decomposition of the form \(p + q\) in which \(p\) and \(q\) are two primes and \(p \leq n/2\) by an increase of 1 of the current point abscissa;
- an \(n\) decomposition of the form \(p + q\) in which \(p\) is an odd compound integer and \(q\) is a prime and \(p \leq n/2\) by an increase of 1 of the current point ordinate;
- an \(n\) decomposition of the form \(p + q\) in which \(p\) is a prime and \(q\) is an odd compound integer and \(p \leq n/2\) by a decreasing by 1 of the current point ordinate;
- an \(n\) decomposition of the form \(p + q\) in which \(p\) and \(q\) are two odd compound integers and \(p \leq n/2\) by a decreasing by 1 of the current point abscissa.

**Example:** global move associated with even integer 48

48 admits 11 decompositions as a sum of two odd integers:

- \(5 + 43, 7 + 41, 11 + 37, 17 + 41, 19 + 29\) decompositions, adding two primes, are coded by 5 moves to the right;
- \(3 + 45, 13 + 35, 23 + 25\) decompositions, adding a prime and an odd compound integer are coded by 3 moves to the bottom;
- \(9 + 39, 15 + 43, 21 + 27\) decompositions, adding an odd compound integer are coded by par 3 moves to the top.

One has moved from origin point \((0,0)\) to point \((2, -3)\).

We can see that this choice allows finding easily even numbers that are of the form \(2p\) with \(p\) prime: their “global move” consists only in a unique move to the bottom or to the right.

---

\(^1\) 1 + \((n - 1)\) decomposition is omitted.
The proposal we made can be coded in C++ to verify this result concerning prime doubles:

```cpp
#include <iostream>
#include <cmath>

int prime(int atester) {
    unsigned long diviseur=2;
    bool pastrouve=true;
    unsigned long k = 2;
    if (atester == 1) return 0;
    if (atester == 2) return 1;
    if (atester == 3) return 1;
    if (atester == 5) return 1;
    if (atester == 7) return 1;
    while (pastrouve) {
        if ((k * k) > atester) return 1;
        else if ((atester % k) == 0) { return 0 ; } else k++;
    }
    return 0;
}

int main (int argc, char* argv[]) {
    int n, k, x, y, xprec, yprec;
    x = 0 ;
    y = 0 ;
    for (n=14 ; n <= 1000 ; n=n+2) {
        xprec = x ;
        yprec = y ;
        x = 0 ;
        y = 0 ;
        for (k=3 ; k <= n/2 ; k=k+2) {
            if (prime(k) && prime(n-k)) x=x+1 ;
            else if (prime(k) && (not(prime(n-k)))) y=y-1 ;
            else if (not(prime(k))) && prime(n-k)) y=y+1 ;
            else if (not(prime(k))) && (not(prime(n-k)))) x=x-1 ;
        }
        if (((x-xprec) == 1) && ((y-yprec) == 0))
            std::cout << "only one step bottom or right for integer " << n/2 << "\n" ;
    }
}