Στην εργασία αυτή θα προγραμματίσουμε την αποσύνθεση πολυωνύμων και συγκεκριμμένα τον αλγόριθμο Poly_decomp_2(u, x) του βιβλίου. Ο αριθμός 2 στο όνομα του αλγορίθμου σημαίνει πως είναι μόνο για 2 πολυώνυμα.

1 Εισαγωγή

Στο αρχείο polydecomp_Trivial_Nontrivial.pdf που βρίσκεται στα έγγραφα, θα βρείτε τις συναρτήσεις compose & polydecomp του συστήματος maxima καθώς και παραδείγματα από τα οποία συμπεραίνουμε πως η συνάτηση polydecomp δεν δουλεύει "κανονικά". Δηλαδή η αποσύνθεση του πολυωνύμου

$$x^8+4x^7+10x^6+16x^5+29x^4+36x^3+40x^2+24x+39$$

περιέχει και το γραμμικό πολυώνυμο

$$2x+1.$$

```
Maxima]
(%i1) f : x^8 + 4*x^7 +10*x^6 + 16*x^5 + 29*x^4 + 36*x^3 +
      40*x^2 + 24*x + 39
```

(%o1) $x^8+4\,x^7+10\,x^6+16\,x^5+29\,x^4+36\,x^3+40\,x^2+24\,x+39$

```
(%i2) polydecomp(f, x)
```

(%o2) $\left[x^2+3, x^2+5, \frac{x^2+3}{4}, 2\,x+1\right]$

Οπως έχουμε αναφέρει το maxima είναι από τα πιο παλιά συστήματα υπολογιστικής άλγεβρας και σήμερα τα πράγματα έχουν αλλάξει, όπως μπορείτε να το επαληθεύσετε στο wolframalpha και στο sympy.

```
SymPy 1.6.2 under Python 3.7.3
>>> from sympy import S, compose, decompose, expand, quo,
    rem
>>> x = S('x')
>>> f = x**8 + 4*x**7 +10*x**6 + 16*x**5 + 29*x**4+ 36*x**3
    + 40*x**2 + 24*x + 39
```

```
>>> decompose(f)

   [x**2 + 12*x + 39, x**2 + 2*x, x**2 + x]
```

Το αξιοσημείωτο είναι πως από την σύνθεση των πολυωνύμων και στα δύο συστήματα προκύπτει το αρχικό πολυώνυμο. Επειδή το maxima δεν έχει την συνάρτηση compose την δημιουργούμε ευκολότατα

```
(%i3) compose(L , x):= block([r : x],
          for e in L do
              r:subst(e, x, r),
          r)
```

(%o3) compose$(L, x) :=$ **block** $([r\colon x],$ **for** e **in** L **do** $r\colon \mathrm{subst}(e, x, r),$ $r)$

και έχουμε

```
(%i4) expand(compose(polydecomp(f, x), x))
```

(%o4) $x^8 + 4\,x^7 + 10\,x^6 + 16\,x^5 + 29\,x^4 + 36\,x^3 + 40\,x^2 + 24\,x + 39$

Το ίδιο συμβαίνει και με το sympy, με την διαφορά ότι η συνάρτηση compose που έχει το σύστημα δουλεύει μόνο για 2 πολυώνυμα. Ετσι γράφουμε την δική μας συνάρτηση

```
>>> def our_compose(L):
        r = x
        for poly in L:
            r = r.subs({x : poly})
        return expand(r)
>>> fg = decompose(f)
>>> fg

   [x**2 + 12*x + 39, x**2 + 2*x, x**2 + x]

>>> our_compose(fg)
```

$$x^8 + 4\,x^7 + 10\,x^6 + 16\,x^5 + 29\,x^4 + 36\,x^3 + 40\,x^2 + 24\,x + 39$$

Σε αντίθεση η συνάρτηση compose του sympy μας επιστρέφει

```
>>> compose(*fg)
```

$$x^2 + 12\,x + 39$$

Ενώ για το πολυώνυμο

```
>>> f = x**4 - 6*x**3 + 21*x**2 - 36*x + 36
```

που είναι σύνθεση 2 πολυωνύμων έχουμε

```
>>> fg = decompose(f)
>>> fg
    [x**2 + 12*x + 36, x**2 - 3*x]
>>> compose(*fg)
```

$$x^4 - 6\,x^3 + 21\,x^2 - 36\,x + 36$$

2 Προγραμματισμός της Poly_decomp_2(u, x)

Οπως βλέπουμε από τον αλγόριθμο της Poly_decomp_2(u, x), πρέπει να προγραμματίσουμε και να λάβουμε υπόψη μας τα εξής.

2.1 Polynomial_divisors(u, x)

Για να βρούμε **όλους** τους διαιρέτες ενός πολυωνύμου χρησιμοποιούμε τις συναρτήσεις του **sympy** factor_list(u) και itermonomials την οποία φορτώνουμε από το sympy.polys.monomials, δηλαδή from sympy.polys.monomials import itermonomials.

2.2 Polynomial_expansion(u, w, x, t)

Ο προγραμματισμός της συνάρτησης αυτής είναι απλός.

2.3 Free_of

Για να βεβαιωθούμε πως ένα πολυώνυμο p δεν περιέχει και δεύτερη μεταβλητή, ένας τρόπος είναι να βεβαιωθούμε πως ο αριθμός των μεταβλητών του είναι 1. Η εντολή Poly(p).free_symbols μας επιστρέφει την λίστα των μεταβλητών του.