This file contains information and instructions for the code at `https://github.com/Wennink/gwreconstruction`

We build upon the SageMath program *admcycles* by Delacroix, Pixton, Schmitt, Zachhuber, and van Zelm. It can be found at `https://gitlab.com/jo314schmitt/admcycles` and is documented in [1]. We display tautological strata classes in the same way as it is done in *admcycles*, which is explained in Section 3.1 of [1].

Our program contains functions that calculate specific ingredients we use in our proofs such as for example `getL25()` from Lemma 7.4 or the following from the matrix calculations:

```
sage: calculate_discriminant_a()
(-2880) * (i^2 - 9*i - 18)
```

For a full list of these functions see the readme file of the program.

1 Computing Gromov-Witten invariants of $\mathbb{P}^2$ blown up in finitely many points

The part of the computer program that calculates these Gromov-Witten invariants is built around two functions we have written. The first is `compute_gw_formula`, which computes the expression $\mathcal{T}_\beta(L, \gamma_1, \ldots, \gamma_n)$ for any $L \in S_{g,n}$ for $g \leq 2$ and any choice of classes from the generating basis $\{pt, H, E_1, \ldots, E_r\}$. The second function is `apply_form` which evaluates this obtained formula.

To make the program calculate a Gromov-Witten invariant of degree $\beta = dH - \sum_i \alpha_i$, we write `gw_inv(g,psiprofile,[d,alpha_1,alpha_2,...])`, where the psiprofile is

- `[]` to calculate $N^{(g)}_{d,\alpha}$,
- `[(1,-1)]` to calculate $P^{(g)}_{d,\alpha}$,
- `[(1,0)]` to calculate $H^{(g)}_{d,\alpha}$, and
- `[(1,1)]` to calculate $K^{(g)}_{d,\alpha}$.

For example we can calcuate $N^{(0)}_4$ and $H^{(2)}_{4,2}$ as follows:

```
sage: gw_inv(0,[],[4])
620
sage: gw_inv(2,[(1,0)],[4,2])
-5/3
```

2 Symbols of tautological relations

We describe some of most important functions in the program related to symbols.

For $g \leq 3$, the function `get_primitive_part_fzm(g,n,r,nonsym)` gives a matrix of (symmetric) relations in $P^r_{g,n}$ modulo restriction to the primitive part. This matrix comes together with a legend that represents the columns as decorated strata classes.

```
sage: prim = get_primitive_part_fzm(2,3,2,false)
sage: prim[0]
[   1   -1 -2/3]
sage: list_tg(prim[1])
[0] : Graph :      [0, 2] [[2, 3, 5], [1, 6]] [(5, 6)]
Polynomial : 1*psi_1^1
[1] : Graph :      [0, 2] [[2, 3, 5], [1, 6]] [(5, 6)]
Polynomial : 1*psi_6^1
[2] : Graph :      [0, 0, 2] [[2, 3, 6], [1, 7, 8], [9]] [(6, 7), (8, 9)]
Polynomial : 1*
```

We can obtain a tautological class ("tautclass" in the program) from a row in this matrix.

```
sage: tcbp = partial_to_tautclass(vec=prim[0][0],taut_gens=prim[1])
sage: tcbp
Graph :      [0, 2] [[2, 3, 5], [1, 6]] [(5, 6)]
Polynomial : 1*psi_1^1

Graph :      [0, 2] [[2, 3, 5], [1, 6]] [(5, 6)]
Polynomial : (-1)*psi_6^1

Graph :      [0, 0, 2] [[2, 3, 6], [1, 7, 8], [9]] [(6, 7), (8, 9)]
Polynomial : (-2/3)*
```

Since we set `nonsym` to `false` we are working with symmetric relations and the program only stores one element in each $\mathcal{S}_n$-orbit. To obtain the full expression for the tautological relation we "unsymmetrize" it.

```
sage: unsymtc = smarter_unsym_tc(tc=tcbp,n=3)
```

We now obtain the symbol which the program displays as follows.

```
sage: sbp = symbol_from_tautclass(g=2,tc=unsymtc)
sage: sbp
-1/3 [PP*aa*bb, cc]
-1/3 [PP*aa*cc, bb]
1/3  [PP*aa, bb*cc]
```

```
-1/3 [PP*bb*cc, aa]
1/3  [PP*bb, aa*cc]
1/3  [PP*cc, aa*bb]
-2/3 [aa*bb*cc]
```

We can do substitutions of classes in symbols and do vector space operations on symbols such as multiplying by a scalar. There are also methods to apply the string, dilaton, or divisor equations from Lemmas 4.9 and 4.10.

```
sage: 3*sbp.subs([aa,W,W])
-2 [PP*aa*W, W]
1  [PP*aa, W^2]
-1 [PP*W^2, aa]
2  [PP*W, aa*W]
-2 [aa*W^2]
sage: 3*sbp.subs([aa,W,W]).applydivisor()
1  [PP*aa, W^2]
2  [PP*W, aa*W]
-5 [aa*W^2]
```

The above is the program's way of displaying the symbol

$$< \psi a, \omega^2 > +2 < \psi\omega, a\omega > -5 < a\omega^2 > .$$

3 Computing tautological relations

Memory usage is the main bottleneck when it comes to computing tautological relations. But when working with symmetric relations, the program only needs to store a single element representing an orbit of the $\mathcal{S}_n$-action. We have written a new version of the code that computes symmetric relations using improvements where we use $\mathcal{S}'_n$-actions for $n' < n$. For example one of the steps the old version takes in the process of calculating the symmetric relations of $P^r_{g,n}$ is to calculate a generating basis of (nonsymmetric) relations in $P^{r-1}_{g,n}$ and then multiply by ψ_i for $1 \le i \le n$. It would be more efficient to calculate relations that are fixed by the $\mathcal{S}_{n-1}$-action on the first $n-1$ points, and then multiply by ψ_n.

We can compare the old and new versions of computing symmetric relations.

The old version:

```
sage: m = get_memory_usage()
sage: %time a=derived_rels(3,4,6,1)
CPU times: user 2min 3s, sys: 628 ms, total: 2min 4s
Wall time: 2min 4s
sage: get_memory_usage()-m
1368.984375
```

The new version:

```
sage: m = get_memory_usage()
sage: %time a=derived_rels_SS(3,4,6,1)
CPU times: user 1min 40s, sys: 416 ms, total: 1min 41s
Wall time: 1min 41s
sage: get_memory_usage()-m
940.66796875
```

The new verion is faster but most importantly it uses less memory. The higher the number of points n, the bigger these differences become:

The old version:

```
sage: m = get_memory_usage()
sage: %time a=derived_rels(3,4,7,1)
CPU times: user 14min 15s, sys: 4.06 s, total: 14min 19s
Wall time: 14min 20s
sage: get_memory_usage()-m
9712.96484375
```

The new version:

```
sage: m = get_memory_usage()
sage: %time a=derived_rels_SS(3,4,7,1)
CPU times: user 7min 30s, sys: 2.19 s, total: 7min 32s
Wall time: 7min 32s
sage: get_memory_usage()-m
4823.75390625
```

We have now got a method to calculate partially symmetric relations. In particular we can use this to calculate nonsymmetric relations. Below we list some comparison results between our new method `pre_processed_fzm` and the method `DR.FZ_matrix` (denoted by `pre` and `dr` respectively). All results are for stable curves.

	$P^2_{3,7}$	$P^2_{2,7}$	$P^3_{2,5}$	$P^4_{2,4}$
`dr` time	2m12s	2m19s	7m23s	out of memory
`pre` time	17m50s	10m19s	1m57s	2m50s
`dr` memory usage	1242	1078	4906	more than 14500
`pre` memory usage	3109	2734	1379	1674

The `DR.FZ_matrix` method performs better at low codimension and high genus or high number of points, while `pre_processed_fzm` is more efficient for higher codimension.

Remark 3.1. An improved version of this new method has been incorporated into the `admcycles` project.

References

[1] Vincent Delecroix, Johannes Schmitt, and Jason van Zelm. "admcycles – a Sage package for calculations in the tautological ring of the moduli space of stable curves." In: *arXiv preprint arXiv:2002.01709* (2020).