T- and Y-systems from periodic quivers through Python

Joe Pallister

May 6, 2024

Abstract

Quivers that are periodic under cluster mutation give rise to T- and Y-systems, but verifying that a quiver is periodic and writing down these systems is cumbersome and accident prone. We provide and explain Python code to automate these calculations. Our code is also able to search for mutation sequences between quivers and we give guidance on how this can be used to search for new periodic quivers and their systems.

The code that is the focus of this work is found in the Jupyter notebook, MutationAndTYSystems.ipynb, hosted at

`https://github.com/JoePallister/QuiverMutation`

Contents

1 Introduction

Quiver mutation [1] is an operation that transforms one quiver into another. It is central to the theory of cluster algebras, where we also attach a pair of variables to each node of our quiver (called cluster and coefficient variables). Quiver mutation is extended to also affect these variables, and we call this cluster mutation (or we'll just abbreviate it to "mutation").

In [4] the authors began the search for "periodic" quivers, that are fixed (up to permutation) by a sequence of mutations. These quivers are of interest since the cluster and coefficient variables we obtain satisfy dynamical systems, which are known as T- and Y-systems. In the period 1 case the authors there also were able to classify such quivers if the permutation is cyclic. One family of such quivers gives rise [3] to the T-system:

$$x_{n+N}x_n = x_{n+p}x_{n+q} + 1, \quad p+q=N \tag{1}$$

There it was shown that the cluster variables for this system satisfy linear relations, despite the relation (1) being nonlinear.

A classification for period 2 quivers is still elusive, though in our work [11] we were able to identify all of them with a low number of vertices. A particular interesting period 2 example is Figure 15 of [9] which has the Y-system

$$y_{n+3}z_n = \frac{(1+z_{n+1})(1+y_{n+2})}{(1+y_{n+1}^{-1})(1+z_{n+2}^{-1})} \qquad z_{n+3}y_n = \frac{(1+y_{n+1})(1+z_{n+2})}{(1+z_{n+1}^{-1})(1+y_{n+2}^{-1})}$$

and is shown to be reducible to the q-Painlevé III equation.

Searching for periodic quivers so far has mainly relied on intuition. Here we provide Python code for searching for these: given a quiver it will perform all possible mutation sequences (up to a given length) to see if any of these sequences fix the quiver (up to permutation), hence proving periodicity. As for the T- and Y-systems, the general form was written down in [8] but, while impressive, calculating specific systems is still laborious. Our code automates this, we can simply input the quiver and the mutation sequence we have found (either by hand or using our search function) and the output is the system.

In summary our code can:

- Apply mutations and sequences of mutations to quivers.
- Given two quivers, search for mutation sequences that can be applied to the first to give the second (allowing for possible relabelling of these quivers).
- Search for mutation sequences that fix a quiver (up to permutation), i.e. search for periodic quivers.
- Given a mutation sequence that fixes a quiver (not up to permutation), generate T- and Y-systems.
- Given a mutate sequence that fixes a quiver (up to permutation), generate reduced T- and Y-systems.

This document is organised as follows: in Section 2 we define quiver mutation and explain our Python implementation. In Section 3 we define "equivalence of quivers" and explain our function to search for mutation sequences between quivers. In Section 4 we define and give examples of periodic quivers. We explain how our mutation sequences function can also search for periodic quivers. In Section 5 we formally define cluster variables and cluster mutation. We explain how T- and Y-systems arise from periodic quivers. We then give an example of how our code can print out these systems with no hand-calculations needed. In Section 6 we discuss the extension of our work to quivers with frozen vertices. In Section 7 we show how our code can be used to obtain known systems and suggest how it can be used to search for potential new systems. Finally in Section 8 we give sundry examples of systems and how we can obtain them. In the Appendix we give a short introduction to Jupyter notebooks, which should hopefully be enough for a beginner to get started.

Acknowledgements

Sage users might find the package for cluster algebras useful [7], though to our knowledge this cannot identify mutation sequences or produce T and Y-systems. Also of note is Keller's useful app for viewing quiver mutation found at

`https://webusers.imj-prg.fr/~bernhard.keller/quivermutation/`

2 Quiver mutation

In this section we define quiver mutation and demonstrate how we have implemented it in our notebook.

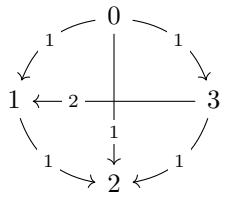
Quiver mutation was defined in [1] and is an operation for transforming one quiver Q into another, provided we do not have any loops or 2-cycles. The mutation is denoted μ_k and can be performed at any vertex k. It is defined in 3 steps:

1. For each length two path $i \to k \to j$ add a new arrow $i \to j$.

2. Reverse the direction of all arrows entering or exiting k.

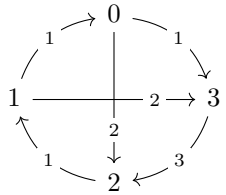
3. Delete all 2-cycles that have appeared.

For example in the quiver



(2)

if we mutate at vertex 1 we obtain



(3)

We can also present quivers as matrices, which we focus on here. We write $b_{i,j}$ and $b'_{i,j}$ for the number of arrows from i to j in Q and $\mu_k(Q)$ respectively then the mutation definition is equivalent to

$$b'_{i,j} = \begin{cases} -b_{i,j} & i = k \text{ or } j = k, \\ b_{i,j} + \frac{1}{2}(|b_{i,k}|b_{k,j} + b_{i,k}|b_{k,j}|) & \text{otherwise.} \end{cases} \tag{4}$$

We express the information given by the $b_{i,j}$ as a skew-symmetric matrix $B = (b_{i,j})$ called the exchange matrix (or B matrix) for Q. Either presentation of quiver mutation can be used to show that mutating at the same vertex twice consecutively has no effect, i.e. $\mu_k^2 = 1$ for any vertex k. It is probably easier to see that this is true by looking at the quiver definition.

Calculating a few quiver mutations by hand might be instructive but and more than that becomes a burden. Luckily we can use the notebook for this, where we implement mutation as

```
mutate(quiver, vertex)
```

Some examples can be found in the notebook's "mutation and mutation sequences" section. There we implement our matrices as numpy arrays (numpy is a mathematical Python package, which we usually import as `np`). For example the above quiver (2) as a numpy array is

```
quiver1 = np.array([
[0, 1, 1, 1],
[-1, 0, 1, -2],
```

```
[-1, -1, 0, -1],
[-1, 2, 1, 0]
])
```

Here the `np.array` part is an example of dot notation. It means that we are using the array object that belongs to the numpy (np) package.

We can apply the same mutation as above (at vertex 1) to this quiver by doing

```
mutate(quiver1, 1)
```

```
>>> array([[ 0., -1.,  2.,  1.],
[ 1.,  0., -1.,  2.],
[-2.,  1.,  0., -3.],
[-1., -2.,  3.,  0.]])
```

Here the `>>>` denotes the cell output, which we see is the matrix representation of (3).

We can compose mutations, meaning we apply each mutation in sequence. For example once we have applied μ_1 to (2) we can apply μ_3 then μ_0 then μ_1 again. Our function for doing this is:

```
mutation_sequence(quiver, sequence)
```

where `sequence` is a list of integers and we will perform the mutations in this list from left to right. For example we can do

```
mutation_sequence(quiver1, [1, 3, 0, 1])
```

```
>>> array([[  0.,  -1.,   5.,   1.],
[  1.,  -0., -10.,   2.],
[ -5.,  10.,   0., -22.],
[ -1.,  -2.,  22.,   0.]])
```

3 Equivalence of quivers

In Section 3.1 we discuss relabelling of quivers and how, in the matrix picture, this manifests as conjugation by a permutation matrix. In Section 3.2 we define equivalence of quivers, meaning there is a sequence of mutations we can apply to get from one quiver to the other. We also show how our code can be used to search for such sequences.

3.1 Relabelling of quivers

We note that the labelling of the vertices of our quivers is arbitrary, we are free to relabel as we want. This should be reflected in the matrix representation. The effect of a quiver relabelling on the B matrix is given by conjugating it with

a permutation matrix. In Python we can generate all of the $n \times n$ permutation matrices by

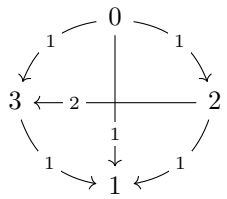
```
[np.array(mat) for mat in list(itertools.permutations(np.eye(n)))]
```

We use this list as part of our functions, and it may be useful if the user wants to see all possible permutation matrices or wants to select one without manually typing it.

Once we select a permutation (here we call it `sigma`) we can perform the conjugation $\sigma^{-1}Q\sigma$ in Python as:

```
np.linalg.inv(sigma).dot(quiver).dot(sigma)
```

As an example the permutation $(1, 3, 2)$ applied to (2) is



(5)

This can easily be done by hand by just rewriting the vertex labels, but, as mentioned above, calculating the matrix form of this is more involved. We proceed as follows: the permutation $\sigma := (1, 3, 2)$ as a matrix is

```
sigma = np.array([
[1, 0, 0, 0],
[0, 0, 0, 1],
[0, 1, 0, 0],
[0, 0, 1, 0]
])
```

where we stress that we label our rows and columns starting from 0, so the first row here represents that 0 is fixed by the permutation $(1, 3, 2)$. We can calculate the permuted matrix by:

```
np.linalg.inv(sigma).dot(quiver1).dot(sigma)
```

```
>>> array([[ 0.,  1.,  1.,  1.],
[-1.,  0., -1., -1.],
[-1.,  1.,  0.,  2.],
[-1.,  1., -2.,  0.]])
```

which agrees with quiver (5), as expected.

3.2 Equivalence of quivers

Examples for the following section can be found in the "Mutation sequences between pairs of quivers" section of the notebook.

We say that two quivers are equivalent if we can obtain one by applying a sequence of mutations to the other, or if we can do this by including a permutation (relabelling). To show that two quivers are equivalent we only need to find a single mutation sequence that works. To show the converse we would have to check all (infinitely many) mutation sequences. Of course the second check isn't possible, but we can check all mutation sequences up to a given length (also known as the depth). We have written an algorithm for this check. This function is called as follows:

```
mutation_sequences_between_quivers(quiver1, quiver2, depth, early_stop=False)
```

Here the first two arguments are the quivers we are looking at, the `depth` argument determines the maximum number of mutations we are allowed to apply. If we just want to see whether two quivers are equivalent `early_stop` should be set to `True`, since this function will then stop as soon as it finds a viable mutation sequence. If we are interested in finding (potentially) many possible sequences then `early_stop` should be set to `False`.

For example we take the following two quivers

```
quiver2 = np.array([
[0, 0, -1, 1, 1],
[0, 0, 1, -1, -1],
[1, -1, 0, 0, -1],
[-1, 1, 0, 0, 1],
[-1, 1, 1, -1, 0]
])

quiver3 = np.array([
[0, 0, 0, 1, -1],
[0, 0, 0, 1, -1],
[0, 0, 0, 1, -1],
[-1, -1, -1, 0, 0],
[1, 1, 1, 0, 0]
])
```

and the function applied to these two quivers gives

```
mutation_sequences_between_quivers(quiver2, quiver3, depth=3, early_stop=False)

>>> [(0, 3, 2), (2, 1, 0)]
```

This means that applying either of the mutation sequences $[0, 3, 2]$ or $[2, 1, 0]$ to `quiver2` will give a permutation of `quiver3`. The `depth` argument means we

don't search the sequences of length greater than 3 (actually there are plenty of valid sequences of length 4).

Note that this function doesn't record which permutation we use, this can be found separately with

```
which_permutation(quiver1, quiver2)
```

which will output a list with all possible permutations which we can apply to the first quiver to give the second. In our example we actually obtain many permutations: we define `mutated_quiver_2` to be the quiver given by applying the mutation sequence $(2, 1, 0)$ to `quiver2`, then `which_permuation` will give us all possible permutations we can apply to `mutated_quiver_2` to give `quiver_2`:

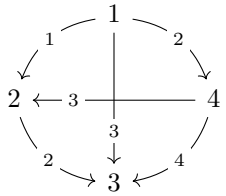
```
mutated_quiver2 = mutation_sequence(quiver2, [2, 1, 0])
which_permutation(mutated_quiver2, quiver3)
```

```
>>> [array([[1., 0., 0., 0., 0.],
[0., 0., 1., 0., 0.],
[0., 0., 0., 1., 0.],
[0., 0., 0., 0., 1.],
[0., 1., 0., 0., 0.]]),
array([[1., 0., 0., 0., 0.],
[0., 0., 0., 1., 0.],
[0., 0., 1., 0., 0.],
[0., 0., 0., 0., 1.],
[0., 1., 0., 0., 0.]]),
...
```

4 Periodic quivers

In this section we define periodic quivers and show how our code can identify them. The corresponding piece of the notebook for this section is also called "periodic quivers"

In general we say that a quiver is periodic, with period p, if it is fixed by a mutation sequence of length p, up to permutation. As an example the quiver



(6)

is period 2, since it satisfies $\sigma\mu_4\mu_1(Q) = Q$ with $\sigma = (1, 2)(3, 4)$.

Luckily we don't need to adapt our code to deal with periodic quivers, since a quiver is periodic if it is equivalent to itself. Due to this if we want to search

for periodic quivers we can use the same function as above but with the same quiver as both arguments:

```
mutation_sequences_between_quivers(quiver, quiver, depth=3, early_stop=False)
```

which searches for the mutation sequences of length less than 4 that fix the quiver, up to permutation.

As an example we can do

```
quiver4 = np.array([
[0, 1, 0, 0],
[-1, 0, 1, 0],
[0, -1, 0, 1],
[0, 0, -1, 0]
])

mutation_sequences_between_quivers(quiver4, quiver4, depth=3, early_stop=False)

>>> [(0, 2, 3),
(1, 0, 3),
(1, 2, 3),
(1, 3, 0),
(2, 0, 3),
(2, 1, 0),
(2, 3, 0),
(3, 1, 0)]
```

which means that any of the above length 3 mutation sequences fix `quiver4` (up to permutation), so the quiver has period 3, but not 1 or 2. It may also have higher periods that we haven't found.

5 T- and Y-systems

In Section 5.1 we explain briefly what cluster and coefficient variables are and how quiver mutation is extended to also mutate these. In Section 5.2 we then show how T- and Y-systems arise from periodic quivers. In Sections 5.3 and 5.4 we give examples of using our code to print out a T- and a Y-system, avoiding any hand calculations. Finally in Section 5.5 we show how any T- or Y-system can be reduced to a system of p equations, where p is the period of the quiver.

5.1 Cluster and coefficient variables

As mentioned above, we can extend quiver mutation to an operation defined in [1] called cluster mutation. We first attach variables x_i to each vertex i of Q, each is called an initial cluster variable and the set $\mathbf{x}_0 := \{x_1, \ldots, x_N\}$ is called

the initial cluster. For any vertex k the mutation μ_k, in addition to giving a new quiver as defined above, gives a new variable $x'_k := \mu_k(x_k)$ defined by

$$x'_k := \frac{1}{x_k}\left(\prod_{i\to k} x_i + \prod_{i\leftarrow k} x_i\right) \tag{7}$$

where the products are over the arrows into (and out of) k in Q. This gives a new cluster $\mu_k(\mathbf{x}_0) := \{x_1, \ldots, x'_k, \ldots, x_N\}$. We write the exchange matrix for Q as B_0 and call the pair $(B_0, \mathbf{x}_0)$ the initial seed. We can consider mutation at k as an operation giving a new seed

$$\mu_k(B_0, \mathbf{x}_0) = (\mu_k(B_0), \mu_k(\mathbf{x}_0))$$

In addition to the cluster variables x_i, there are "coefficient" variables y_i attached to each vertex. When we mutate, μ_k gives new y variables defined by

$$\mu_k(y_j) = \begin{cases} y_k^{-1} & j = k, \\ y_j\left(1 + y_k^{\mathrm{sgn}(b_{jk})}\right)^{b_{jk}} & j \neq k, \end{cases} \tag{8}$$

where sgn denotes the sign function. We can include this information in the seeds defined above: the initial seed is now $(B_0, \mathbf{x}_0, \mathbf{y}_0)$ and mutation gives a new seed

$$\mu_k(B_0, \mathbf{x}_0, \mathbf{y}_0) = (\mu_k(B_0), \mu_k(\mathbf{x}_0), \mu_k(\mathbf{y}_0))$$

We noted above that quiver mutation satisfies $\mu_k^2 = 1$, and this also holds for cluster mutation.

We remark that there is a more general definition of cluster mutation where the coefficient variables can also appear in the cluster mutation formula (7), but we do not deal with that here.

5.2 T- and Y-systems for periodic quivers

Here we briefly show how T- and Y-systems arise from periodic quivers. The construction of the two types of system is written very succinctly in [8], which is used as the inspiration for our code to generate these systems.

Our definition of a periodic quiver Q is equivalent to

$$\mu_{\mathbf{i}}(Q) = \sigma Q \tag{9}$$

where $\mu_{\mathbf{i}}$ is a sequence of mutations and σ is a permutation. We can apply $\mu_{\sigma(\mathbf{i})}$ to this equation

$$\mu_{\sigma(\mathbf{i})}\mu_{\mathbf{i}}(Q) = \mu_{\sigma(\mathbf{i})}\sigma Q = \sigma(\mu_{\mathbf{i}}Q) = \sigma(\sigma Q) = \sigma^2 Q$$

Next by applying $\mu_{\sigma^2(\mathbf{i})}$ we get

$$\mu_{\sigma^2(\mathbf{i})}\mu_{\sigma(\mathbf{i})}\mu_{\mathbf{i}}(Q) = \sigma^3(Q)$$

We can continue similarly to get

$$\mu_{\sigma^{m-1}(\mathbf{i})} \cdots \mu_{\sigma(\mathbf{i})} \mu_{\mathbf{i}}(Q) = \sigma^m Q = Q \tag{10}$$

where we have taken m to be the order of σ. Here this sequence of mutations really does fix the quiver (not up to mutation). For convenience we write this mutation as

$$\mu_{\mathbf{I}} := \mu_{\sigma^{n-1}(\mathbf{i})} \cdots \mu_{\sigma(\mathbf{i})} \mu_{\mathbf{i}}$$

The T- and Y-systems we are interested in arise from the behaviour of the cluster and coefficient variables under applying $\mu_{\mathbf{I}}^l$ for $l \in \mathbb{Z}$.

We write the nth cluster variable we obtain at vertex k as $x_k(n)$. Since our quiver is period p, $\mu_{\mathbf{i}}$ is made of p mutations. We label the sequence of vertices we mutate at in $\mu_{\mathbf{I}}$ by k and let n_k be the label of the kth vertex we mutate at. We also let $s_k(j)$ be the number of times we have mutated at vertex j, at the time of mutation k.

For example in a five vertex quiver the (entirely arbitrary) sequence

$$(0, 4, 2, 0, 2, 3, 0, \ldots)$$

$n_0 = 0$, $n_1 = 4$, etc. The vector $s_0(j)$ is the number of times we have mutated at each vertex when we do the first mutation at 0, so it is

$$s_0(j) = (1, 0, 0, 0, 0)$$

The next vectors $s_1(j)$ and $s_2(j)$ are

$$s_1(j) = (1, 0, 0, 0, 1), \quad s_2(j) = (1, 0, 1, 0, 1)$$

we next mutate at 0 for the second time giving

$$s_3(j) = (2, 0, 1, 0, 1)$$

Since we are applying the same mutation sequence $\mu_{\mathbf{I}}$ over and over we have that

$$n_k = n_{\bar{k}} \tag{11}$$

where the bar denotes reduction mod mp. There should also be some periodicity property for $s_k(j)$. To see this we write $k = \lfloor k/mp \rfloor + \bar{k}$, or equivalently we take $\bar{k}$ to be k reduced mod mp. By the time we do mutation k we have done $\lfloor k/mp \rfloor$ lots of $\mu_{\mathbf{I}}$ and $\tilde{k}$ additional mutations. We let the number of mutations at vertex j in the sequence be α_j, so the contribution to $s_k(j)$ from $\mu_{\mathbf{I}}^{\lfloor k/mp \rfloor}$ is $\lfloor k/mp \rfloor \alpha_j$ and the contribution from the additional $\bar{k}$ mutations is $s_{\bar{k}}(j)$, hence we can write

$$s_k(j) = \lfloor k/mp \rfloor \alpha_j + s_{\bar{k}}(j) \tag{12}$$

We can now look at how our mutation sequence affects our cluster variables. When we do the kth mutation at vertex n_k, this should be the $s_k(k)$th cluster variable we obtain there, by definition of $s_k(j)$. Comparing with the cluster mutation definition (7) we see that $x_{n_k}(s_k(n_k))$ corresponds to x'_k there. Similarly

x_k in (7) is the last variable we obtained at vertex k, so we can simply subtract 1 from $s_k(n_k)$ to get $x_{n_k}(s_k(n_k)-1)$. The other x_i in (7) are the last cluster variables we obtained at vertex i, so are $x_i(s_k(i))$. We can compare this with the cluster mutation formula (7) in the following table.

Equation (7) notation	New notation
x'_k	$x_{n_k}(s_k(n_k))$
x_k	$x_{nk}(s_k(n_k)-1)$
x_i	$x_i(s_k(i))$

With this in mind, (7) becomes

$$x_{n_k}(s_k(n_k)) \times x_{n_k}(s_k(n_k)-1) = \\ f_k(x_0(s_k(0)), x_1(s_k(1)), \ldots, x_N(s_k(N))) \tag{13}$$

where f_k is a function that determines the product on the RHS of (7), so it only depends on the shape of the quiver.

Since $\mu_{\mathbf{I}}$ fixes the quiver we have

$$f_k = f_{\bar{k}}$$

We can combine this with the periodic properties (11) and (12) to reduce (13) to

$$x_{n_{\bar{k}}}(\tilde{k}\alpha_{n_k} + s_{\bar{k}}(n_k)) \times x_{n_{\bar{k}}}(\tilde{k}\alpha_{n_k} + s_{\bar{k}}(n_k)-1) = \\ f_{\bar{k}}(x_0(\tilde{k}\alpha_0 + s_{\bar{k}}(0)), \ldots, x_N(\tilde{k}\alpha_N + s_{\bar{N}}(0))) \tag{14}$$

where we have written $\tilde{k} := \lfloor k/mp \rfloor$. This is a system of relations parametrized by $(\tilde{k}, \bar{k}) \in \mathbb{Z} \times \mathbb{Z}/mp\mathbb{Z}$ and is the most general form of the T-system in this case. While this looks complicated, each of the ingredients involved ($\bar{k}$ and $\tilde{k}$, the α_i and the $s_k(i)$) are not too difficult to calculate, except for $f_{\bar{k}}$ which requires knowledge of the mutated quivers).

One notable special case is if every vertex appears in the mutation sequence the same number of times, then we have that all the α_i are the same, so we call it $\alpha := \frac{mp}{N}$ and the T-system becomes

$$x_{n_{\bar{k}}}(\tilde{k}\alpha + s_{\bar{k}}(n_k)) \times x_{n_{\bar{k}}}(\tilde{k}\alpha + s_{\bar{k}}(n_k)-1) = \\ f_{\bar{k}}(x_0(\tilde{k}\alpha + s_{\bar{k}}(0)), \ldots, x_N(\tilde{k}\alpha + s_{\bar{N}}(0))) \tag{15}$$

We note that so far we have only used the fact that the full mutation sequence fixes the quiver, we have not yet used the periodicity $\mu_{\mathbf{i}}(Q) = \sigma Q$. This will be discussed in Section 5.5, where we show that we can always write the T-system as a system of p equations.

For the Y-system we might get new coefficient variables at every vertex at each mutation, so there we use slightly different notation and call $Y_i(j)$ the coefficient variable at vertex i after j mutations. Apart from this the Y-system is constructed similarly and we don't go in to details.

5.3 T-system example

Here we give an example of our code calculating a T-system, so we don't have to go through the difficulty of doing it by hand. As mentioned above the "full" T-system contains mp equations, where m is the order of our permutation and p is the period of the quiver. We will later see a way to reduce this to a set of only p equations, which we recommend using most of the time.

Since we need a mutation sequence that completely fixes a quiver, we need to extend our mutation sequence that fixes the quiver only up to permutation. Luckily the sequence and the permutation are enough to give this.

We saw above that `quiver4` is fixed by the sequence $[0, 2, 3]$ up to permutation. We can obtain one (of many possible) permutations by doing:

```
sigma = which_permutation(mutation_sequence(quiver4, [0, 2, 3]), quiver4)[0]
```

where the `[0]` at the end just selects the first possible permutation. The full sequence that fixes the quiver will be given by

$$(0, 2, 3, \sigma(0), \sigma(2), \sigma(3), \sigma^2(0), \ldots, \sigma^{m-1}(0), \sigma^{m-1}(2), \sigma^{m-1}(3))$$

where m is the order of σ. We can obtain this full sequence by

```
permuted_mutation_sequence(sigma, [0, 2, 3])
```

```
>>> [0, 2, 3, 2, 1, 0, 1, 3, 2, 3, 0, 1]
```

The initial quiver and this sequence are enough to describe the T-system, which we can obtain by doing:

```
sequence = [0, 2, 3, 2, 1, 0, 1, 3, 2, 3, 0, 1]
T_system(quiver3, sequence)
```

The output of this is a nicely printed set of 12 equations

$$\begin{aligned}
x_0(1)x_0(0) &= 1 + x_1(0), \quad x_2(1)x_2(0) = x_1(0) + x_3(0) \\
x_3(1)x_3(0) &= x_1(0) + x_2(1), \quad x_2(2)x_2(1) = 1 + x_3(1) \\
x_1(1)x_1(0) &= x_3(1) + x_0(1), \quad x_0(2)x_0(1) = x_3(1) + x_1(1) \\
x_1(2)x_1(1) &= 1 + x_0(2), \quad x_3(2)x_3(1) = x_0(2) + x_2(2) \\
x_2(3)x_2(2) &= x_0(2) + x_3(2), \quad x_3(3)x_3(2) = 1 + x_2(3) \\
x_0(3)x_0(2) &= x_2(3) + x_1(2), \quad x_1(3)x_1(2) = x_2(3) + x_0(3)
\end{aligned} \tag{16}$$

By replacing the $x_k(j) \to x_k(j + 3l)$ in (16) we obtain the full system of equations parametrized by $(l, k) \in \mathbb{Z} \times \mathbb{Z}/3$. In this case each of the vertices in the sequence appears the same number of times so our system (16) is a case of (15).

5.4 Y-system example

We can get the corresponding Y-system for the same quiver and permutation as in the previous section by doing

```
Y_system(quiver3, sequence)
```

which prints

$$y_0(5)y_0(0) = \frac{1}{(1+y_1(4)^{-1})}, \quad y_2(3)y_2(1) = \frac{1}{(1+y_3(2)^{-1})}$$

$$y_3(7)y_3(2) = \frac{(1+y_1(4))(1+y_0(5))}{(1+y_2(3)^{-1})}, \quad y_2(8)y_2(3) = \frac{1}{(1+y_3(7)^{-1})}$$

$$y_1(6)y_1(4) = \frac{1}{(1+y_0(5)^{-1})}, \quad y_0(10)y_0(5) = \frac{(1+y_3(7))(1+y_2(8))}{(1+y_1(6)^{-1})}$$

$$y_1(11)y_1(6) = \frac{1}{(1+y_0(10)^{-1})}, \quad y_3(9)y_3(7) = \frac{1}{(1+y_2(8)^{-1})}$$

$$y_2(13)y_2(8) = \frac{(1+y_0(10))(1+y_1(11))}{(1+y_3(9)^{-1})}, \quad y_3(14)y_3(9) = \frac{1}{(1+y_2(13)^{-1})}$$

$$y_0(12)y_0(10) = \frac{1}{(1+y_1(11)^{-1})}, \quad y_1(16)y_1(11) = \frac{(1+y_2(13))(1+y_3(14))}{(1+y_0(12)^{-1})}$$

Due to the way we count the y variables the full system here is given by replacing $y_k(j) \to y_k(j+12l)$ for $l \in \mathbb{Z}$, giving a system parametrized by $j + 12l \in \mathbb{Z}$.

5.5 Reductions of T- and Y-systems

As mentioned above we do not recommend using the long forms of the T- and Y-systems, since we have a way of reducing them to a system of only p equations. This is because so far we have only used that our (full) mutation sequence fixes the quiver. Of course we should include the fact that we have a shorter mutation sequence that fixes the quiver, up to permutation.

We can observe from our examples that the T- and Y-systems have some periodicity, in that equation $k+p$ has the same form as equation k, where p in the period. It seems intuitive that we should be able to take a reduction of these systems, giving a system of p equations. There are options for doing this, depending on the system, but here we show that the most obvious way, which always works.

For the T-system we map the kth cluster variable we obtain, which in our above notation is $x_{n_k}(s_k(n_k))$, to $w_{\bar{k}\lfloor k/p \rfloor}$, where the bar denotes reduction mod p.

For example in (16) we map

$$(x_0(1), x_2(0), x_3(0), x_2(1), x_1(0), x_0(1), x_1(1), x_3(1), x_2(2), \ldots) \to$$

$$(w_0(0), w_1(0), w_2(0), w_0(1), w_1(1), w_2(1), w_0(2), w_1(2), w_0(3), \ldots)$$

where the first list is the cluster variables we obtain, in order.

Applying this reduction to (16) will give 4 copies of the same 3 equations, up to a shift in index. We can either do this by hand, or more conveniently with our function

```
T_system_reduced(quiver, sequence, sigma)
```

which takes a sequence that fixes the quiver up to permutation, and the permutation itself (called `sigma`).

For our example in Section 5.3 we took a permutation by doing

```
sigma = which_permutation(mutation_sequence(quiver4, [0, 2, 3]), quiver4)[0]
```

So we can obtain the reduced T-system by

```
T_system_reduced(quiver4, [0, 2, 3], sigma)
```

which prints

$$w_2(1)w_0(0) = 1+w_1(1), \quad w_0(1)w_1(0) = w_1(1)+w_2(0), \quad w_1(2)w_2(0) = w_1(1)+w_0(1)$$

which is the reduced T-system of (16). This is a system of only $p = 3$ equations, not the $mp = 12$ equations that we had before.

The Y-system reduction works almost the same. In our notation $y_i(j)$ actually is the jth coefficient variable that we obtain, so the reduction here is

$$y_i(j) \to v_{\bar{n}}\lfloor n/p \rfloor$$

Our code for getting the reduced Y-system is almost identical to that for the T-system, we simply do:

```
Y_system_reduced(quiver4, [0, 2, 3], sigma)
```

which prints

$$v_2(1)v_0(0) = \frac{1}{(1+v_1(1)^{-1})}, \quad v_0(1)v_1(0) = \frac{1}{(1+v_2(0)^{-1})}$$

$$v_1(2)v_2(0) = \frac{(1+v_1(1))(1+v_2(1))}{(1+v_0(1)^{-1})}$$

6 Frozen vertices

Here we define frozen vertices and show how our work so far can be extended to include them.

A frozen vertex is a vertex where we are not allowed to mutate. Our algorithm so far is free to mutate at any vertex it wants, though it may decide not to use some. For frozen vertices we want to ensure that some vertices can never be mutated at.

To do this we just need to encode our quiver as a rectangular matrix. Our algorithm for finding mutation sequences will not be allowed to mutate at the frozen vertices. For example a quiver with frozen vertices is (2.21) of [6]:

```
quiver221 = np.array([
[0, 1, -1, 0, -1, 1],
[-1, 0, 2, -1, 1, -1],
[1, -2, 0, 1, 1, -1],
[0, 1, -1, 0, -1, 1],
[1, -1, -1, 1, 0, 0],
[-1, 1, 1, -1, 0, 0],
[1, 0, 0, -1, 1, -1],
[-1, -1, 1, 1, 0, 0]
])
```

As before we search for mutation sequences by doing

```
mutation_sequences_between_quivers(quiver221, quiver221, depth=3, early_stop=False)
```

```
>>> [(0, 1),
(0, 3),
(0, 4),
(3, 0),
(3, 2),
(3, 5),
(4, 5),
(5, 4),
(1, 0, 1),
...
```

where we note that the frozen vertices (6 and 7) do not appear in these sequences. We can find the associated systems by doing

```
sigma = which_permutation(mutation_sequence(quiver221, [0, 1]), quiver221)[0]
T_system_reduced(quiver221, [0, 1], sigma)
```

which prints

$$w_0(3)w_0(0) = w_0(2)w_0(1)c_6 + w_1(0)w_1(1)c_7, \quad w_1(3)w_1(0) = w_1(2)w_1(1)c_6 + w_0(3)w_0(2)c_7$$

Since we are not allowed to mutate at 6 or 7 the cluster variables $x_6(0)$ and $x_7(0)$ are constant, so we write them as c_6 and c_7.

7 Identifying old and new systems

In Section 7.1 we give an example to show how our code can be used to verify known T- and Y-systems. In Section 7.2 we then demonstrate how it can be used to search for potentially new systems.

For this section we focus on Okubo's quiver, Figure 15 of [9], which is given by the following matrix

```
okubo_III = np.array([
[0, 0, -1, 1, 1, -1],
[0, 0, 1, -1, -1, 1],
[1, -1, 0, 0, -1, 1],
[-1, 1, 0, 0, 1, -1],
[-1, 1, 1, -1, 0, 0],
[1, -1, -1, 1, 0, 0]
])
```

and it is shown there that an associated Y-system is equivalent to the q-Painlevé III equation.

7.1 Known systems

Okubo takes the mutation sequence $[0, 1]$ and a suitable permutation to obtain the T and Y-systems (equations (4.19) and (4.23)). To avoid identifying the correct permutation by ourselves we can print all the systems at once by doing

```
for sigma in which_permutation(mutation_sequence(okubo_III, [0, 1]), okubo_III):
    T_system_reduced(okubo_III, [0, 1], sigma)
```

which will print out the systems we get from each of the possible permutations for the sequence $[0, 1]$. We won't print them all here, but we observe that the third permutation gives the correct system, i.e.

```
sigma = which_permutation(mutation_sequence(okubo_III, [0, 1]), okubo_III)[2]
T_system_reduced(okubo_III, [0, 1], sigma)
```

prints the system

$$w_1(3)w_0(0) = w_0(1)w_1(2)+w_1(1)w_0(2), \quad w_0(3)w_1(0) = w_1(1)w_0(2)+w_0(1)w_1(2)$$

which is the T-system we are looking for, (4.19) of [9], by mapping $w_0(n) \to x_n$ and $w_1(n) \to w_n$.

Since we have identified the mutation sequence and the permutation we can get the corresponding Y-system as

```
sigma = which_permutation(mutation_sequence(okubo_III, [0, 1]), okubo_III)[2]
Y_system_reduced(okubo_III, [0, 1], sigma)
```

which prints

$$v_1(3)v_0(0) = \frac{(1 + v_0(1))(1 + v_1(2))}{(1 + v_1(1)^{-1})(1 + v_0(2)^{-1})}$$
$$v_0(3)v_1(0) = \frac{(1 + v_1(1))(1 + v_0(2))}{(1 + v_0(1)^{-1})(1 + v_1(2)^{-1})}$$

which gives the desired Y-system, equation (4.24) of [9], by taking $v_0(n) \to z_n$ and $v_1(n) \to y_n$. Note that this isn't yet the q-Painlevé III equation, Okubo does more work to bring it to the final form by identifying periodic quantities for this system.

7.2 New systems

While doing Okubo's sequence $[0, 1]$ we only mentioned the permutation that gave us the systems we were looking for. When looking for new systems it is worth checking the other permutations for possibilities. The command

```
for sigma in which_permutation(mutation_sequence(okubo_III, [0, 1]), okubo_III):
    T_system_reduced(okubo_III, [0, 1], sigma)
```

prints the systems for each permutation. For example the first permutation gives

$$w_0(1)w_0(0) = c_2c_5 + c_3c_4, \quad w_1(1)w_1(0) = c_3c_4 + c_2c_5$$

Where the c_i represent frozen cluster variables. Of course, this sequence is trivial so we can discount it from our search. The other permutations give systems that are the same as this, or probably equivalent to Okubo's system.

Since we have exhausted the possibilities of the sequence $[0, 1]$ we search for more usable sequences:

```
mutation_sequences_between_quivers(okubo_III, okubo_III, depth=4, early_stop=False)
```

```
>>> [(0, 1),
(1, 0),
(2, 3),
(3, 2),
(4, 5),
(5, 4),
(0, 1, 0, 1),
(0, 1, 2, 3),
(0, 1, 3, 2),
(0, 1, 4, 5),
(0, 1, 5, 4),
(0, 2, 4, 0),
...
```

where we haven't displayed all of the many length 4 possibilities. This gives many leads for potentially new systems. For example we can look at the sequence $[0, 2, 4, 0]$. To avoid picking a permutation we just look at all of them, so we do

```
for sigma in which_permutation(mutation_sequence(okubo_III, [0, 2, 4, 0]), okubo_III):
    T_system_reduced(okubo_III, [0, 2, 4, 0], sigma)
```

which gives 6 systems. It seems likely that many are equivalent and that we only have 2 distinct systems:

$$w_3(0)w_0(0) = w_1(0)c_5 + c_3w_2(0), \quad w_2(1)w_1(0) = w_3(0)c_1 + c_3c_5$$

$$w_1(1)w_2(0) = c_3c_5 + w_3(0)c_1, \quad w_0(1)w_3(0) = w_2(1)c_3 + w_1(1)c_5$$

and

$$w_3(0)w_0(0) = w_1(0)w_2(1)+w_1(1)w_2(0), \quad w_2(2)w_1(0) = w_3(0)w_0(1)+w_1(1)w_2(1)$$

$$w_1(2)w_2(0) = w_1(1)w_2(1)+w_3(0)w_0(1), \quad w_0(2)w_3(0) = w_2(2)w_1(1)+w_1(2)w_2(1)$$

Cursory observations show that these systems are much simpler than they appear, but the question is are they so simple to be trivial. It is worth exploring the rest of the mutation sequences to find interesting systems, though the high level of symmetry of Okubo's quiver means many of these will be equivalent.

8 Examples

This section is a (non-exhaustive) collection of other known T- and Y-systems and shows how we can obtain them using our code.

8.1 $\tilde{A}_{N,1}$ type quiver

One of the foundational results of cluster algebras [2] is that a cluster algebra has finitely many cluster variables (called finite type) if and only if its quiver is mutation equivalent to a Dynkin ADE quiver. This means that any system from one of these quivers is necessarily periodic. The Dynkin quivers appear in the classification of semisimple Lie algebras and their natural extension is to "affine type" quivers, which classify affine Lie algebras. This suggests looking at the significance of affine quivers in cluster algebras. Affine A type T-systems were studied in [4, 3] where it was shown that the cluster variables satisfy linear relations, despite the nonlinearity of the system. This was extended to affine D and E types in [10].

A particular orientation of the affine A type quivers, written $\tilde{A}_{N,1}$, has a T-system that appears as equation (3.13) in [3]. For example the $\tilde{A}_{3,1}$ quiver is

```
affineA = np.array([
[0, -1, 0, -1],
[1, 0, -1, 0],
[0, 1, 0, -1],
[1, 0, 1, 0]
])
```

The authors of [3] show that this quiver is fixed by mutation at 0, and use that to generate their T-system. Before doing this let's check if other mutation sequences exists:

```
mutation_sequences_between_quivers(affineA, affineA, depth=2, early_stop=False)
```

```
>>> [(0,), (3,), (0, 1), (3, 2)]
```

Here there is another length one sequence (3), but the T-system we obtain from that will be the same as running the sequence (0) T-system backwards in time. The length two sequences here are just extensions of the length one sequences, so the is really only one sequence of interest here, (0).

We run our code to print the T-system for each possible permutation:

```
sequence = [0]
for sigma in which_permutation(mutation_sequence(affineA, sequence), affineA):
    T_system_reduced(affineA, sequence, sigma)
```

but here there is actually only one permutation so this only prints one system:

$$w_0(4)w_0(0) = 1 + w_0(1)w_0(3)$$

which is the $\tilde{A}_{3,1}$ T-system we want. The corresponding Y-system, given by

```
sequence = [0]
for sigma in which_permutation(mutation_sequence(affineA_4, sequence), affineA_4):
Y_system_reduced(affineA_4, sequence, sigma)
```

is

$$v_0(4)v_0(0) = (1 + v_0(1))(1 + v_0(3))$$

which is mentioned in [5] equation (5.2).

8.2 Somos-4

The connection between the Somos-4 sequence and cluster algebras was shown in [4]. The T-system appears as equation (1) there and the corresponding quiver is given in Figure 1. As a matrix it is

```
somos_4 = np.array([
[0, -1, 2, -1],
[1, 0, -3, 2],
[-2, 3, 0, -1],
[1, -2, 1, 0]
])
```

To identify the T-system we search for mutation sequences

```
mutation_sequences_between_quivers(somos_4, somos_4, depth=2, early_stop=False)
```

```
>>> [(0,), (3,), (0, 1), (3, 2)]
```

Either of the length one sequences gives the system we want. We do

```
sequence = [0]
for sigma in which_permutation(mutation_sequence(affine_4, sequence), affine_4):
    T_system_reduced(somos_4, sequence, sigma)
```

to give

$$w_0(4)w_0(0) = w_0(1)w_0(3) + w_0(2)^2$$

which is [4], equation (1). The corresponding Y-system can be obtained by doing

```
sequence = [0]
for sigma in which_permutation(mutation_sequence(affine_4, sequence), affine_4):
    Y_system_reduced(somos_4, sequence, sigma)
```

which gives

$$v_0(4)v_0(0) = \frac{(1 + v_0(1))(1 + v_0(3))}{((1 + v_0(2)^{-1}))^2}$$

This is [5], equation (2.12), and is shown there to be reducible to the q-Painlevé I equation. Somos-5, 6 and 7 are also treated in that paper.

8.3 Okubo's quivers

We have already mentioned Okubo's quiver that reduces to q-Painlevé III. He also gives many other reduced other Painlevé equations. We don't give all the details here but we can look at one more, Figure 11 of [9]. This is given by

```
okubo_I = np.array([
[0, -1, 2, -1],
[1, 0, -3, 2],
[-2, 3, 0, -1],
[1, -2, 1, 0]
])
```

With the T-system code:

```
for sigma in which_permutation(mutation_sequence(okubo_I, [0]), quiver):
    T_system_reduced(okubo_I, [0], sigma)
```

giving

$$w_0(4)w_0(0) = w_0(1)w_0(3) + w_0(2)^2$$

which is [9] equation (4.2), and the Y-system code:

```
for sigma in which_permutation(mutation_sequence(okubo_I, [0]), quiver):
    Y_system_reduced(okubo_I, [0], sigma)
```

giving

$$v_0(4)v_0(0) = \frac{(1 + v_0(1))(1 + v_0(3))}{(1 + v_0(2)^{-1})^2}$$

which is [9], equation (4.7).

A Jupyter Notebooks

As mentioned above the code documented here can be founded in the notebook MutationAndTYSystems.ipynb at

```
https://github.com/JoePallister/QuiverMutation
```

Clicking on this file in the GitHub repo will open it and a download link can be found in the options box (the ellipses ...) in the top right corner.

Once you have a copy of the notebook you can open it in Jupyter and are free to make changes, however anything in the "functions" section is necessary for the code to run so should be only altered with care. Before getting started experimenting with these functions the cells containing them need to be run. This is easiest done with the double arrows button, which will run each cell in the notebook in turn.

Cells can be markdown type for text or maths (we can use latex there) or code (here Python). By selecting a cell we can change its type with Y (code) or M (markdown). We can also do this in the "Cell" menu under "Cell Type". We can create a new cell above or below the currently selected cell with A and B or in the insert menu. Finally the selected cell can be run with Ctrl+Enter.

References

[1] Sergey Fomin and Andrei Zelevinsky. Cluster algebras I: Foundations. *Journal of the American Mathematical Society*, 15(2):497–529, 2002.

[2] Sergey Fomin and Andrei Zelevinsky. Cluster algebras II: Finite type classification. *Inventiones Mathematicae*, 154(1):63–121, 2003.

[3] Allan P Fordy and Andrew Hone. Discrete integrable systems and Poisson algebras from cluster maps. *Communications in Mathematical Physics*, 325(2):527–584, 2014.

[4] Allan P Fordy and Bethany R Marsh. Cluster mutation-periodic quivers and associated Laurent sequences. *Journal of Algebraic Combinatorics*, 34(1):19–66, 2011.

[5] Andrew NW Hone and Rei Inoue. Discrete Painlevé equations from Y-systems. *Journal of Physics A: Mathematical and Theoretical*, 47(47):474007, 2014.

[6] Andrew NW Hone, Wookyung Kim, and Takafumi Mase. New cluster algebras from old: integrability beyond zamolodchikov periodicity. *arXiv preprint arXiv:2403.00721*, 2024.

[7] Gregg Musiker and Christian Stump. A compendium on the cluster algebra and quiver package in sage. *arXiv preprint arXiv:1102.4844*, 2011.

[8] Tomoki Nakanishi. Periodicities in cluster algebras and dilogarithm identities. *Representations of algebras and related topics*, 5:407, 2011.

[9] Naoto Okubo. Bilinear equations and q-discrete Painlevé equations satisfied by variables and coefficients in cluster algebras. *Journal of Physics A: Mathematical and Theoretical*, 48(35):355201, 2015.

[10] Joe Pallister. Linear relations and integrability for cluster algebras from affine quivers. *Glasgow Mathematical Journal*, page 1–38, 2020.

[11] Joe Pallister. Period 2 quivers and their t-and y-systems. *arXiv preprint arXiv:2109.11107*, 2021.