1 Reverse Curve

For the problem of constructing the space curve from the control fields, we can work with two different methods.

2 General considerations

The procedure must generate a SpaceCurve class object that functionally behaves the same as an instance created using functional definitions. The `frenet_dict` is constructed through the private method `_frenet_dict_fun(x_values, self.params)`. The proposed implementation must be consistent with the output of

```
def frenet_dict_fun(x, params):

        deriv_array = deriv_array_fun(x, params)

        frenet_dict = frametools.calculate_frenet_dict(deriv_array)

        frenet_dict['x_values'] = x
        frenet_dict['params'] = params

        frenet_dict['curve'] = curve_fun(x, params)

        return frenet_dict
```

The function must be modified appropriately so that it connects correctly with the `evaluate_frenet_dict(self, n_points=None)` method. It would be preferable if the numerical data are interpolated when the number of points is higher than the given samples otherwise it will be an acceptable solution if an exception can be thrown.

In the initial stage, the elements of the dictionary are

```
frenet_dict['frame'] = jnp.array([tangent, normal, binormal])
frenet_dict['speed'] = speed
frenet_dict['curvature'] = curvature
frenet_dict['torsion'] = torsion
frenet_dict['deriv_array'] = deriv_array
```

We want the new feature to redefine the private method appropriately so that numerical data are used instead. We will assume the fields are provided in a control dictionary. From SCQC, the field equations are

$$\Omega(t) = \kappa(t) \tag{1}$$

$$\dot{\Phi}(t) - \Delta(t) = \tau(t) \tag{2}$$

Numerical differentiation may be required to extract the geometric quantities. If fields are given for each axis, we can then use the polar coordinate transformation equations as

$$\Omega(t) = f(t)\sqrt{\Omega_x^2 + \Omega_y^2}, \tag{3}$$

$$\tan\Phi = \Omega_y/\Omega_x. \tag{4}$$

Note: We will always treat the phase field as a continuous quantity. In that case, the radius variable of the polar coordinates can be considered negative. It is important for the code to track the global sign of Ω. For instance, for two neighboring points t_i, t_{i+1}, we need to check whether it holds $\Omega_x(t_i)\Omega_x(t_{i+1}) < 0$. The sign will be handled by the sign function $f(t)$.

For the extraction of the moving frame vectors, we suggest the usage of the adjoint representation. The following code snippet can calculate the adjoint representation of the unitary U.

```
from qurveros.qubit_bench import simulator, quantumtools
qu_evol = simulator._single_qubit_sim(control_dict)
adj_evol = np.array([quantumtools.calculate_adj_rep(unitary)
                        for unitary in qu_evol])
```

The adjoint representation is a matrix that contains the vectors as

$$R_U = R_Z(\Phi(t)) \begin{bmatrix} -\vec{B}(t) \\ \vec{N}(t) \\ \vec{T}(t) \end{bmatrix} R_{\text{init}}, \tag{5}$$

where R_{init} is an arbitrary rotation matrix and the vectors enter as rows. By calculating the phase field, the moving frame vectors can be readily extracted.

In the reverse curve problem, we will make the association $\vec{T}(t) = \dot{\vec{r}}(t)$. In this regard, the speed of the curve is always unit throughout the evolution.

```
frenet_dict['speed'] = ones (# samples)
```

The interval will be set as

```
interval = [0, times[-1]]
```

from the control dictionary. The parameters and the order can be set to None. The function signature must provide the optional argument as

```
def space_curve_from_pulse(pulse, initial_rotation=None):
curve = _curve_from_pulse(pulse)
return SpaceCurve(curve=curve, order=1, interval=[0,  times[-1]])
```

The following subsections will describe the two cases to fill in the remaining entries.

2.1 The direct method: Constructing the `frenet_dict` manually

In general, the aforementioned private method must be overridden so that the numerical data is returned. Since we now have access to the set of vectors $\vec{T}, \vec{N}, \vec{B}$ along with the fields, we need to note one implementation detail. The global sign of Ω indicates the existence of singular points. Since qurveros corrects the singular point at the control dictionary stage, the curvature is assumed always non-negative while the vectors $\vec{N}$, $\vec{B}$ can be discontinuous. In that case, the implementation must assign:

```
frenet_dict['curvature'] = abs(\Omega)
frenet_dict['torsion'] = (The one extracted from the fields)
```

For the frame, we will have for the normal and binormal vectors respectively as

$$\vec{N}_{\text{frame}} = f(t)\vec{N} \tag{6}$$

$$\vec{B}_{\text{frame}} = f(t)\vec{B} \tag{7}$$

The frame entry must be of the form [samples,3,3] where the vectors are arranged as rows.

In this case, the `deriv_array` can be set to None.

2.2 The indirect method: Using the `deriv_array` function

The first step is basically the same for recovering the moving frame vectors. An alternative approach is to utilize the Frenet-Serret equations.

$$\dot{\vec{T}} = \kappa\vec{N}, \tag{8}$$

$$\dot{\vec{N}} = \tau\vec{B} - \kappa\vec{T}, \tag{9}$$

$$\dot{\vec{B}} = -\tau\vec{N}, \tag{10}$$

and starting from the fact that $\vec{T} = \dot{\vec{r}}(t)$, find all other derivatives through differentiation. This method will provide the correct discontinuous vectors with

$$\kappa(t) = |\Omega(t)|. \tag{11}$$

Once the curve derivatives are found, the deriv array function will return the derivatives' matrix for each sample point, so that it is consistent with the `x_values` in the `evaluate_frenet_dict(self, n_points=None)` method.

3 Test cases

There are some simple examples that can be made to ensure the consistency of the implementation. A simple case is the usage of a constant curvature curve as seen in the first example of qurveros. The implementation must correctly reproduce the curve and the control fields. The initial rotation argument might be required so that the tangent vector is appropriately aligned.

The second example is the usage of a helix. Using a control dictionary with constant Ω, Δ, the resulting curve will be a helix. Once either methods are used to reconstruct the curve, the fields must be correctly reproduced.

The final example will test the correct tracking of the sign. Run the

```
examples/bessel_curve_robust_pulse.ipynb
```

and save the control fields in a csv file. The envelope field will contain a global sign that needs to be tracked. The frenet dictionary must contain the same vectors as the Bessel curve to ensure that the test is correct. An initial rotation must be multiplied from the right to ensure that the vectors are correctly aligned.