**TITLE:**
Experimental and Data Analysis Workflow for Soft Matter Nanoindentation

**AUTHORS AND AFFILIATIONS:**
Giuseppe Ciccone[1], Mariana Azevedo Gonzalez Oliva[1], Nelda Antonovaite[2], Ines Lüchtefeld[3], Manuel Salmeron-Sanchez[1], Massimo Vassalli[1]

[1]Centre for the Cellular Microenvironment, James Watt School of Engineering, University of Glasgow, Glasgow, UK
[2]Optics 11 life, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
[3]Laboratory of Biosensors and Bioelectronics, ETH Zürich, Gloriastrasse 35, 8092, Zürich, Switzerland

Email addresses of co-authors:
Nelda Antonovaite (nelda@optics11.com)
Ines Lüchtefeld (luechtefeld@biomed.ee.ethz.ch)
Manuel Salmeron-Sanchez (manuel.salmeron-sanchez@glasgow.ac.uk)
Massimo Vassalli (massimo.vassalli@glasgow.ac.uk)

Corresponding authors:
Giuseppe Ciccone (g.ciccone.1@research.gla.ac.uk)
Mariana Azevedo Gonzalez Oliva (m.azevedo-gonzalez-oliva.1@research.gla.ac.uk)

**SUPPLEMENTARY NOTE 1: Adding custom contact points algorithms and filters to NanoAnalysis**

Custom contact point (CP) algorithms and filters can be easily added to the graphical user interface (GUI). CP finding routines and filters are in separate modules called panels.py (CP algorithms) and filter_panel.py (filters), respectively.

Adding a new CP algorithm

1. Using an editor of choice, such as Visual Studio Code (https://code.visualstudio.com/) open **panels.py**.

2. Scroll down to the end of the module where a template for a new algorithm is given. The rationale behind the template is explained with an example in the following steps.

3. Create a new Class that inherits the ContactPoint class. The Class will contain the new CP finding algorithm. For example, for the RoV algorithm, a new class is created called ThRoV which inherits the ContactPoint class:

```
1.  class ThRoV(ContactPoint):
2.      """Ratio of Variances (RoV) algorithm. doi: https://doi.org/10.1038/srep21267."""
```

4. Add the parameters that will be used in the CP algorithm. Those can be Floats, Integers or Combo (both). For example, the RoV algorithm is based on a 'Safe Threshold' parameter,

a ‘X Range’ parameter and a ‘Window RoV’ parameter, as explained in the main text of the protocol. To do so, use the `create` method. After adding the parameters, the code looks like:

```
1.  class ThRov(ContactPoint):
2.      """Ratio of Variances (RoV) algorithm. doi: https://doi.org/10.1038/srep21267."""
3.      def create(self):
4.          self.Fthreshold = CPPFloat('Safe Threshold [nN]')
5.          self.Fthreshold.setValue(10.0)
6.          self.Xrange = CPPFloat('X Range [nm]')
7.          self.Xrange.setValue(1000.0)
8.          self.windowr = CPPInt('Window RoV [nm]')
9.          self.windowr.setValue(200)
10.         self.addParameter(self.Fthreshold)
11.         self.addParameter(self.Xrange)
12.         self.addParameter(self.windowr)
```

To understand the above lines of code, consider the ‘Safe Threshold’ parameter as an example. Line 4 `self.Fthreshold = CPPFloat('Safe Threshold [nN]')` does the following: creates the variable Fthreshold (as per referenced in the code), which is a float (CPPFloat) that will be displayed in the GUI as ‘Safe Threshold [nN]’. Line 5 sets its default value to 10.0 nN. To add the parameter to the GUI, the addParameter() method is used (line 10). The same rationale applies to the other parameters.

5. In principle, after setting the required parameters (step 4), the CP algorithm per se can be written in the `calculate` method, which takes the curve object (c) and returns a list with the *x* and *y* coordinates of the CP. In practice, intermediate steps are often required, such as selecting a region of interest (ROI) where to look for the CP and calculating the weight on which the algorithm is based (in this case, the ratio of variances). To select the ROI, we often use the same block of code in the `getRange` method:

```
1.  class ThRov(ContactPoint):
2.      """Ratio of Variances (RoV) algorithm. doi: https://doi.org/10.1038/srep21267."""
3.      def create(self):
4.          self.Fthreshold = CPPFloat('Safe Threshold [nN]')
5.          self.Fthreshold.setValue(10.0)
6.          self.Xrange = CPPFloat('X Range [nm]')
7.          self.Xrange.setValue(1000.0)
8.          self.windowr = CPPInt('Window RoV [nm]')
9.          self.windowr.setValue(200)
10.         self.addParameter(self.Fthreshold)
11.         self.addParameter(self.Xrange)
12.         self.addParameter(self.windowr)
13.
14.     def getRange(self, c):
15.         x = c._z
16.         y = c._f
17.         try:
18.             jmax = np.argmin((y - self.Fthreshold.getValue()) ** 2)
19.             jmin = np.argmin((x - (x[jmax] - self.Xrange.getValue())) ** 2)
20.         except ValueError:
21.             return False
22.         return jmin, jmax
```

Lines 14-22 define the ROI where the CP will be searched for. Specifically, lines 15 and 16 extract the displacement (`c._z`) and force (`c._f`) from the curve (`c`) object. Line 18 takes the index of the point closest to the ‘Safe Threshold [nN]’ (`jmax`), and line 19 takes the

index of the point which is closest to the displacement *z* corresponding to the safe threshold, shifted left by a value defined by 'X Range [nm]' (`jmin`). Thus, `jmin` and `jmax` define the ROI where the CP will be searched for, and should be returned from `getRange`.

6. Calculate and store the weight in the ROI. This is done with the `getWeight` method. For example, in the case of the RoV algorithm, one wants to calculate the ratio between the variances of the force signal to the right and to the left of the trial CP in a given window ('Window RoV'), for all trial points in the ROI. Therefore, the code looks like:

```
1.  class ThRov(ContactPoint):
2.      """Ratio of Variances (RoV) algorithm. doi: https://doi.org/10.1038/srep21267."""
3.      def create(self):
4.          self.Fthreshold = CPPFloat('Safe Threshold [nN]')
5.          self.Fthreshold.setValue(10.0)
6.          self.Xrange = CPPFloat('X Range [nm]')
7.          self.Xrange.setValue(1000.0)
8.          self.windowr = CPPInt('Window RoV [nm]')
9.          self.windowr.setValue(200)
10.         self.addParameter(self.Fthreshold)
11.         self.addParameter(self.Xrange)
12.         self.addParameter(self.windowr)
13.
14.     def getRange(self, c):
15.         x = c._z
16.         y = c._f
17.         try:
18.             jmax = np.argmin((y - self.Fthreshold.getValue()) ** 2)
19.             jmin = np.argmin((x - (x[jmax] - self.Xrange.getValue())) ** 2)
20.         except ValueError:
21.             return False
22.         return jmin, jmax
23.
24.     def getWeight(self, c):
25.         jmin, jmax = self.getRange(c)
26.         winr = self.windowr.getValue()
27.         x = c._z
28.         y = c._f
29.         if (winr + jmax) > len(y):  # check1
30.             return False
31.         if jmin < winr:  # check2
32.             return False
33.         rov = []
34.         for j in range(jmin, jmax):
35.             rov.append((np.var(y[j+1: j+winr])) / (np.var(y[j-winr: j-1])))
36.         return x[jmin:jmax], rov
```

7. Finally, the point where the weight is optimised (i.e., where it is either maximum or minimum depending on the algorithm) corresponds to the CP. It is required to return a list containing the *x* and *y* coordinates of the CP, which can be done with the `calculate` method. For example, the complete RoV algorithm looks like:

```
1.  class ThRov(ContactPoint):
2.      """Ratio of Variances (RoV) algorithm. doi: https://doi.org/10.1038/srep21267."""
3.      def create(self):
4.          self.Fthreshold = CPPFloat('Safe Threshold [nN]')
5.          self.Fthreshold.setValue(10.0)
6.          self.Xrange = CPPFloat('X Range [nm]')
7.          self.Xrange.setValue(1000.0)
8.          self.windowr = CPPInt('Window RoV [nm]')
9.          self.windowr.setValue(200)
10.         self.addParameter(self.Fthreshold)
11.         self.addParameter(self.Xrange)
```

```
12.         self.addParameter(self.windowr)
13.
14.     def getRange(self, c):
15.         x = c._z
16.         y = c._f
17.         try:
18.             jmax = np.argmin((y - self.Fthreshold.getValue()) ** 2)
19.             jmin = np.argmin((x - (x[jmax] - self.Xrange.getValue())) ** 2)
20.         except ValueError:
21.             return False
22.         return jmin, jmax
23.
24.     def getWeight(self, c):
25.         jmin, jmax = self.getRange(c)
26.         winr = self.windowr.getValue()
27.         x = c._z
28.         y = c._f
29.         if (winr + jmax) > len(y):  # check1
30.             return False
31.         if jmin < winr:  # check2
32.             return False
33.         rov = []
34.         for j in range(jmin, jmax):
35.             rov.append((np.var(y[j+1: j+winr])) / (np.var(y[j-winr: j-1])))
36.         return x[jmin:jmax], rov
37.
38.     def calculate(self, c):
39.         z = c._z
40.         f = c._f
41.         try:
42.             zz_x, rov = self.getWeight(c)
43.         except TypeError:
44.             return
45.         rov_best_ind = np.argmax(rov)
46.         j_rov = np.argmin((z-zz_x[rov_best_ind])**2)
47.         return [z[j_rov], f[j_rov]]
```

8. For the algorithm to be added to the GUI, it must be appended to the ALL list as a dictionary containing the name to be displayed on the GUI and the Class corresponding to the given algorithm. For example, for the RoV algorithm, the following line of code should be added to the end of the module:

```
1.  ALL.append({'label': 'Ratio of Variances', 'method': ThRov})
```

The algorithm can now be used as part of the GUI.

NOTE: Return `False` when the algorithm fails to complete a step (e.g., line 20). This will ensure that curves where the algorithm fails will be moved to the 'Included' set, as explained in the main protocol.

Adding a new Filter

The procedure of adding a custom filter to the GUI is akin to that of adding a new CP algorithm.

1. Using an editor of choice, such as Visual Studio Code (https://code.visualstudio.com/) open filter_panel.py.

2. Scroll down to the end of the module where a template for a new algorithm is given. The rationale behind the template is explained with an example in the following steps.

3. Create a new Class that inherits the Filter class. The Class will contain the new filter algorithm. For example, for the SAVGOL filter (see main text) the code looks like:

```
class SavGolFilter(Filter):
    """
    Filters data with the Savitzky-Golay filter from the Scipy Library.
    doi: https://doi.org/10.1038/s41592-019-0686-2
    """
```

4. Add the filter parameters that the user can change in the GUI. For the savgol filter, those are the polynomial order and the window length. The code looks like:

```
class SavGolFilter(Filter):
    """
    Filters data with the Savitzky-Golay filter from the Scipy Library.
    doi: https://doi.org/10.1038/s41592-019-0686-2
    """
    def create(self):
        self.win = FilterInt('Window Length [nm]')
        self.win.setValue(25)
        self.polyorder = FilterInt('Polynomical Order (Int)')
        self.polyorder.setValue(3)
        self.addParameter(self.win)
        self.addParameter(self.polyorder)
```

5. Finally, calculate the filtered data with the `calculate` method, which returns the filtered force data:

```
class SavGolFilter(Filter):
    """
    Filters data with the Savitzky-Golay filter from the Scipy Library.
    doi: https://doi.org/10.1038/s41592-019-0686-2
    """
    def create(self):
        self.win = FilterInt('Window Length [nm]')
        self.win.setValue(25)
        self.polyorder = FilterInt('Polynomical Order (Int)')
        self.polyorder.setValue(3)
        self.addParameter(self.win)
        self.addParameter(self.polyorder)

    def calculate(self, c):
        y = c._f
        win = self.win.getValue()
        polyorder = self.polyorder.getValue()
        if win % 2 == 0:
            win += 1
        if polyorder > win:
            return False
        y_smooth = savgol_filter(y, win, polyorder)
        return y_smooth
```

6. If additional methods are needed to filter the data, those can be added within the algorithm. For the new filter to be added to the GUI it must be appended to the `ALL_FILTERS` list as a dictionary containing the name to be displayed on the GUI and the Class corresponding to the given algorithm. For example, for the savgol algorithm, the following line of code should be added to the end of the module:

```
ALL_FILTERS.append({'label': 'Savitzky Golay', 'method': SavGolFilter})
```

NOTE: Return `False` in case of any exceptions. For example, the SAVGOL filter from the SciPy library requires the polynomial order to be less than the window length. Lines 20-21 handle this.