README file for PFMCal.

A. Butykai, P. Domínguez-García, F. M. Mor, R. Gaál, L. Forró, S. Jeney

1. ORIGINAL `PFMCAL.M`

1.1 Software requirements

MatLab 2011a or later, Statistics toolbox. Tested in MatLab 2016a.

1.2 Installation of the software

Unzip `CalibrationSoftware.zip` archive to the destination path. No further installation steps are needed.

1.3 Executing the software

The calibration software can be executed from MatLab (MathWorks Inc.) by running the `PFMCal.m` file. Select either 'Add to path' or 'Change current directory' options.

1.4 Using the software

Please refer to Section 7 in the original manuscript: **Computer Physics Communications, 196 (2015) 599**

1.5 Test data

Test data are located in the `Testdata` folder. `Si_Spheres` archive contains experimental data measured on silica spheres. `Si_splinters_non_spherical` folder contains experimental data measured on fragmented silica particles with random shape. `Synthetic_non_spherical` folder contains 100 generated datasets with random error for the validation of the non-spherical model. `Synthetic_spherical` folder contains 100 generated datasets with random error for the validation of the spherical model.

2. EXTENDED VERSION

2.1 Software requirements

The extended code can be executed without modification using MatLab or GNU Octave. The code has been tested in Linux and Windows operating systems.

2.2 Installation of the software

Unzip `CalibrationSoftware.zip` archive to the destination path. The extended files are inside the `AUTO` folder. For running the files an installation of MatLab 2011 (or later) or GNU Octave 4.0 (or later) is needed.

Regarding Octave, the code uses `Forge`* packages `strut` and `optim`. Issues may appear in relation with those packages in Octave 4.0 (not expected in later versions). To avoid these errors, update the packages in Octave command window by using the following commands:

```
pkg install -forge struct
pkg install -forge optim
```

If Linux is used as operating system, the following packages have to be installed in the system: `liboctave-dev`, `epstool` and `transfig`.

*`https://octave.sourceforge.io/`

2.3 Executing the software

The extended calibration software can be executed from MatLab command window or GNU Octave by running the files `PFMCal_auto.m` and `PFMCal_histo.m` located inside folder `AUTO` (Linux-style complete path: `./CalibrationSoftware/AUTO`). Therefore, for the automatic process of calibration, enter in the MatLab/Octave command window:

>PFMCal_auto

For running the thermal noise statistics method, simply write:

>PFMCal_histo

There is no need to enter a path to the data files, because it is included in those scripts. The test data are loaded from the folder `Testdata` (Path: `./CalibrationSoftware/AUTO/Testdata`). The loading of the files have been written to be compatible with the different path separators depending on the operative system (tested in Windows and Linux). By default, the water data is selected, but it can be changed to acetone data by editing directly the main files `PFMCal_auto.m` or `PFMCal_histo.m`. Inside these two files, the input data needed for each running are defined. This input data are defined in the following sections.

2.4 Using the software

2.4.1 PFMCal_auto

The analytical solution of Langevin equations for an optically trapped spherical particle with no-slip boundaries in a Newtonian fluid with hydrodynamic effects only depends numerically on the following timescales:

$$\tau_f \equiv \frac{\rho_f a^2}{\eta};\ \tau_p^* \equiv \frac{m^*}{6\pi\eta a};\ \tau_k \equiv \frac{6\pi\eta a}{k} \tag{1}$$

where a is the bead radius and ρ_p its density, ρ_f is the density of the fluid, while$m^* \equiv m_p + m_f/2$ is an effective mass which modifies the mass of the particle, m_p, because of the influence of hydrodynamics, m_f being the mass of the displaced fluid. By knowing the characteristics of the beads and the fluid (a, ρ_f and ρ_p), the calibration method allows to calculate these timescales and, consequently, to obtain the viscosity of the fluid, η, and the trap stiffness, k.[†]

Additionally to these input values, the value of the MSD plateau which is going to be used in the computation of the timescales has to be obtained. The value of the MSD plateau is often difficult to obtain as the standard deviation of the data points increases due to the increasing correlations in the MSD for long time values (see Fig. 1). Errors in the estimation of the plateau value results in the imprecise determination of η.

In order to extract a reliable plateau value, we use the following procedure. First, we establish the location, where the relative noise of the MSD signal becomes large. For that aim, we calculate the second derivative of the MSD(t) in log-log scale. In a typical MSD signal, the first derivative varies gradually from 1.5 at lowest times (temporal exponent previous to the ballistic regime) to 0 at higher times (plateau in the MSD) with the second derivative being close zero for all times values. The onset of the noise characterizing the MSD plateau is indicated by a significant deviation of the second derivative from its zero value. The function `getMSDPlateau`

[†]For further details see: M. Grimm, T. Franosch, S. Jeney, High-resolution detection of Brownian motion for quantitative optical tweezers experiments, Phys. Rev. E 86 (2012) 021912.

automatically calculates this second derivative. In Fig. 1 (left), this data is plotted showing the deviation from zero at higher values of $\log t$. For detecting the point where the noise is affecting the plateau, we use a modulation function defined by:

$$M(f) \equiv \frac{\max(f) - \min(f)}{\max(f) + \min(f)}$$

We apply this M function iteratively to the second derivative of the MSD, beginning with the array composed by the first element of that data, $i = 1$ ($M = 0$), then by the array composed by its two next elements $i = 1, 2$, then $i = 1, 2, 3$, until $i = 1, 2, 3, ..n$. The resulting data is plotted in Fig. 1 (right), where an abrupt increase in the function is observed. We attribute the first peak to the beginning of noise, where the MSD already takes its plateau value and, therefore, the MSD plateau is selected as the value of the MSD in that point (temporal value).[‡]

This value of the MSD plateau together with the input data a, ρ, ρ_p and T allows to calculate the calibration curves for the MSD and VAF, and to obtain β_{MSD}, β_{VAF}, k and η.

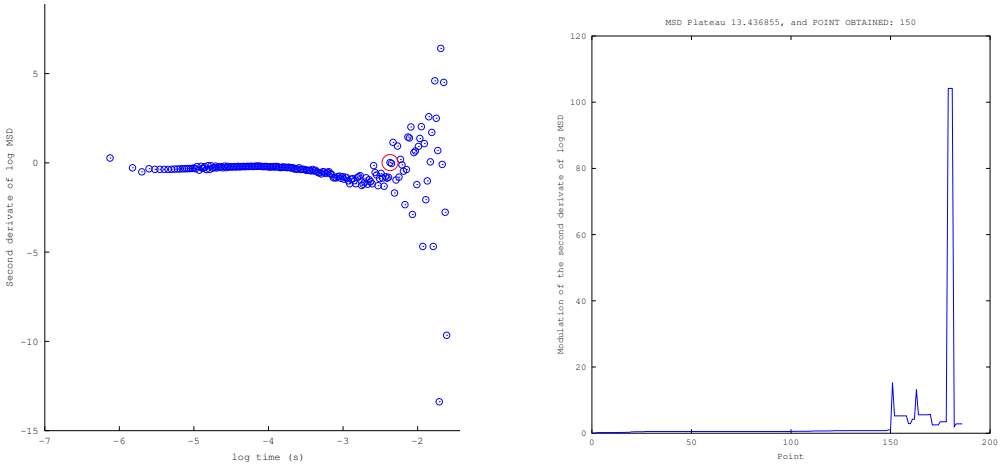

Figure 1. Left: Second derivative of the mean-squared displacement in log-log scale for acetone data, showing the noise affecting the calculation of the plateau of the MSD. The red circle marks the approximate point chosen by the user where the noise begins to be relevant. Right: Modulation function: the beginning of noise is the first peak in the figure.

Regarding the PSD, we use the classic methodology for Newtonian fluids, by fitting a Lorentzian curve to the measured data to obtain its corner frequency, f_c, from where the corresponding calibration factor β_{PSD} and k can be obtained. The PSD of the bead in the fluid has a Lorentzian form PSD $(f) = S_0 f_c^2/(f_c^2 + f^2)$, where S_0 is a constant. This quantity can be approximated as $\mathrm{PSD}(f) \simeq S_0 f_c^2 f^{-2}$ in the limit of high frequencies. Then, the calibration factor is retrieved using $\beta_{\mathrm{PSD}} = k_B T/(6\pi^3 \eta a S_0 f_c^2)$. Note here that η and a have to be previously known to obtain this β_{PSD}, and therefore we use the η value obtained from the calibration of the MSD and VAF.

The execution of `PFMCal_auto.m` finally provides Fig. 2, which plots measured MSD, VAF and PSD together with the fitted model functions. The characteristic timescales obtained by the fitting are displayed in the plots with vertical lines. This figure is automatically saved in a folder created by the software named `output`. In this folder we save the calibration figure in 'pdf' and 'eps' formats and a text filed named `output_data.txt` which contains all the input and output data.

[‡]This calculation is not shown during the execution of the code but these figures can be shown and saved by changing a boolean parameter in `getMSDPlateau` function.

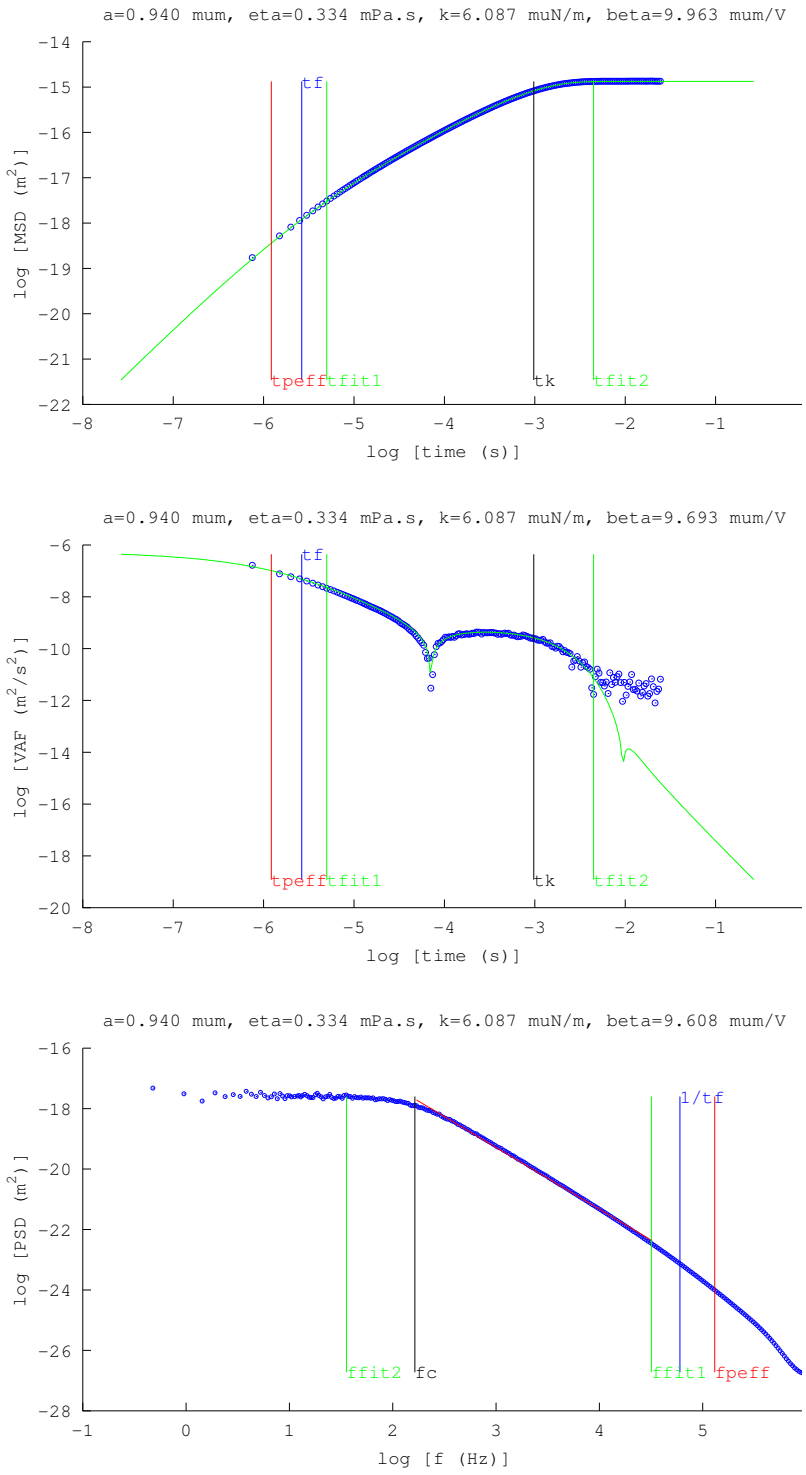

Figure 2. Experimental data measured in acetone. MSD and the VAF for with their theoretical curves. The experimental PSD is indicated by a fitting to the Lorentzian curve. Vertical lines: tf corresponds to τ_f, tpeff is τ_p^*, tk is τ_k. The time window in which the described methods can be applied are delimited by tfit1 and tfit2.

2.4.2 PFMCal_histo.m

Additionally to the methods explained above, we also include a calibration method by using the experimental histogram of particle positions. By calculating the probability density $p(x)$, the potential can be deduced as $E(x) = -k_B\,T\,\ln p(x) + C$, where C is a constant related to the potential offset and can be neglected. In the case of a spherical bead in an ideal optical trap, $E(x) = \frac{1}{2}k_E x^2$ in a harmonic approximation, only depending on the trap stiffness, k. Here, $p(x)$ is a Gaussian distribution and, through the equipartition theorem: $\frac{1}{2}k\left\langle x^2\right\rangle = \frac{1}{2}k_B T$, we have $k_{\sigma^2} \equiv k_B\,T/\sigma^2$, where $\sigma^2 = \left\langle x^2\right\rangle$ is the variance of the probability density.

If we use some additional method to obtain a value for the trap stiffness, i.e., by calculating averaged values from the calibration of MSDs for independent water measurements, we can calculate two additional calibration factors, β_E, from the harmonic potential curve and β_{σ^2}, from the Gaussian distributions.

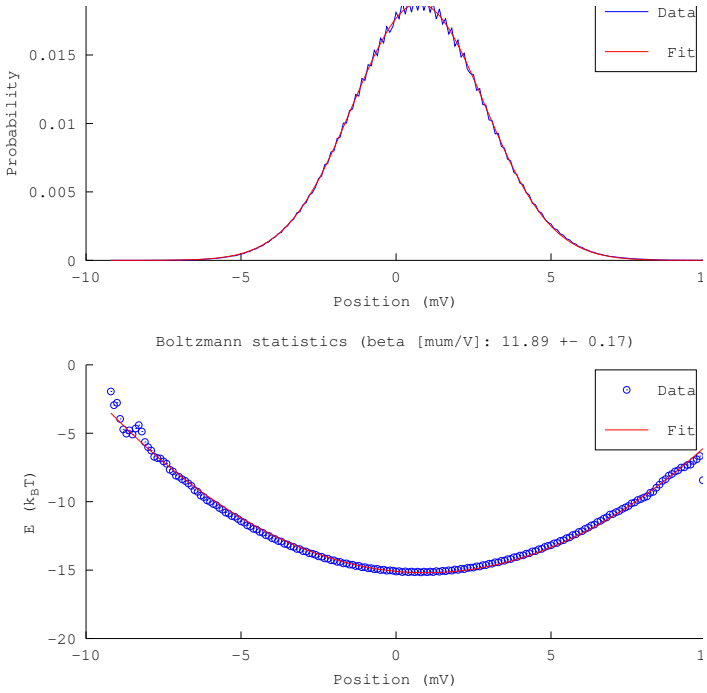

Figure 3. Calibration factors obtained from the fitting to a harmonic potential curve, β_E, and from the Gaussian distribution, β_{σ^2}. The fluid is water.

The fittings and calculation of these calibration factors is done by using the script `PFMCal_histo.m`. The histograms data are read from the `AUTO/Testdata` folder and the k value is defined inside `PFMCal_histo.m`. This code fits the data to a Gaussian function and to a harmonic potential, obtaining calibration values very similar to the ones determined with the calibration methodology of `PFMCal.m` and `PFMCal_auto.m` (see Table in the new paper submission). The fitting plot is shown in Fig. 3, which is saved in the `output` folder. The input and output data regarding this calibration code are saved in a file named `output_data_histo.txt`.

All the fittings results (texts and figures for water and acetone data) are shown in the folder `./AUTO/results_auto`.

APPENDIX A. LIST OF PROGRAM SUBROUTINES

Program subroutines inside `AUTO` folder are briefly explained in this section. Subroutines for the original submission have not been changed and had been described in the first CPC paper. Here, we comment the two main files and then all the subroutines sorted alphabetically:

- `PFMCal_auto.m`: Main subroutine for calibration. Contains input data to be modified if needed.
- `PFMCal_histo.m`: Main subroutine for calibration using histogram of positions. Contains input data to be modified if needed.
- `calculateCalibration.m`: Calibration of the optical tweezers set-up via the Velocity-Autocorrelation.
- `calculateCalibrationHisto.m`: Calibration via histogram of positions. Contains code for fitting to quadratic polynomial and Gaussian functions.
- `calcV.m`: Calculates array and positions of modulation function (getV) for an array x which is read element by element as explained in this text.
- `datawithoutheader.m`: Reads text file without the header.
- `faddeeva.m`: This function computes the complex scaled complementary error function also known as the FADDEEVA function
- `findMinVAF.m`: Read out the minimum of the theoretical VAF.
- `findMSDHalf.m`: This function finds the time, where the MSD reaches half of its plateau via linear interpolation between the last point, which is below and the first one, which is above.
- `findZero.m`: This function finds the zero-crossing of the VACF via linear interpolation between the last point, which is positive and the first negative one.
- `getCalPSD.m`: Calculation of the calibration by the classic method of the corner frequency on the PSD.
- `getHalfMSD.m`: This function calculates half of the MSD plateau value for given parameters tps/tf and tk/tf. Time is given out in units of tf.
- `getHalfMSDbyt.m`: Calculates half of the MSD for a given time t.
- `getMSDPlateau.m`: Calculates the MSD Plateau before noise begins.
- `getTheoMSDandVAF.m`: Theoretical curves for MSD and VAF.
- `get_tktf.m`: Calculates the ratio tk/tf.
- `get_V.m`: Modulation function (getV) for a array x.
- `get_VAFvalue.m`: Calculates the value of the VAF at a given time t.
- `getZeroCrossingVAF.m`: This function calculates the zero-crossing of a VAF with given parameters tps/tf and tk/tf.
- `isOctave.m`: Returns true if the environment is Octave.
- `plotLinesTimes.m`: Code of lines with temporal scales for plots.
- `theoreticalMSDandVAF.m`: This function calculates the VAF and MSD for a given set of parameters tf,tps and tk for a time t by Clercx and Schram's formula.