ggf Documentation

Release 0.2.0

Paul Müller

Jul 18, 2018

Contents:

ggf is a Python library for computing global geometric factors and corresponding stresses acting on dielectric objects in the optical stretcher. This is the documentation of ggf version 0.2.0.

CHAPTER 1

Installation

ggf is written in pure Python and supports Python version 3.5 and higher.

To install ggf, use one of the following methods (package dependencies will be installed automatically):

- **from PyPI:** `pip install ggf`
- **from sources:** `pip install .` or `python setup.py install`

CHAPTER 2

Introduction

2.1 What is the package “ggf” used for?

It is a Python implementation of two Matlab scripts by Lars Boyde, *StretcherNStress.m* and *GGF.m*, which are used in the Guck lab to compute optical stress distributions and resulting global geometric factors for spherical and spheroidal objects in the optical stretcher.

2.2 What is an optical stretcher?

The optical stretcher consists of a dual beam laser trap, in its original configuration built from two opposing optical fibers *[GAM+01]*. When increasing the trapping power, compliant objects such as cells are stretched along the axis of the trap. Using video analysis, the measured shape change can be translated into physical properties of the cell.

2.3 What is the global geometric factor?

The global geometric factor (GGF) connects (the unknown variable) compliance J (how easy it is to deform a body consisting of a certain material) and (the measured variable) strain ϵ (how much this body is deformed). Thus, the GGF is a measure of stress (force acting on the surface of the body).

$$J = \frac{\epsilon}{\mathrm{GGF}}$$

In an optical stretcher (OS) experiment, the strain ϵ of a cell can be measured by analyzing its deformation (e.g. via a contour in the intensity image). Using cell size and the measured change in eccentricity, as well as several parameters of the OS setup itself, `ggf` can be used to compute the optical stress σ from which the GGF is computed.

2.4 How should I migrate my Matlab pipeline to Python?

2.4.1 To reproduce data

You can access the computations performed in *StretcherNStress.m* via `ggf.core.stress.boyde2009()`.

```
from ggf.stress.boyde2009 import stress
theta, sigma, coeff = stress(object_index=1.41,
                             medium_index=1.3465,
                             radius=2.8466e-6,        # [m]
                             poisson_ratio=0.45,
                             stretch_ratio=0.1,
                             wavelength=780e-9,       # [m]
                             beam_waist=3.08,         # [wavelengths]
                             power_left=.65,          # [W]
                             power_right=.65,         # [W]
                             dist=180e-6,             # [m]
                             field_approx="davis",
                             ret_legendre_decomp=True)
```

The GGF can be computed from the coefficients `coeff` via `ggf.globgeomfact.coeff2ggf()`.

```
from ggf import legendre2ggf
GGF = legendre2ggf(coeff, poisson_ratio=.45)
```

These methods produce the same output as the original Matlab scripts with an accuracy that is below the standard tolerance of `numpy.allclose()`.

2.4.2 For a new project

In general, the method *`ggf.get_ggf()`* is recommended. The difference to the above method is:

- It makes use of precomputed look-up tables (LUTs) which avoids long computation times. The error made by using LUTs is below 10^{-5} Pa.
- It comes with user-convenient keyword arguments.
- The GGF is computed from 120 Legendre coefficients by default, a number that was previously determined automatically and could have potentially been too low.
- TODO: Poisson's ratio ambiguity

Please note that due to these points, the resulting GGF might vary from the GGF computed with the original Matlab script.

```
import ggf
GGF = ggf.get_ggf(model="boyde2009",
                  semi_major=3.2788e-6,             # [m]
                  semi_minor=2.8466e-6,             # [m]
                  object_index=1.41,
                  medium_index=1.3465,
                  effective_fiber_distance=180e-6,  # [m]
                  mode_field_diameter=4.8e-06,      # [m]
                  power_per_fiber=.65,              # [W]
                  wavelength=780e-9,                # [m]
                  poisson_ratio=0.45)
```

CHAPTER 3

Concept and theory

3.1 Summary

The computation of the compliance J for elastic spheres in the OS can be divided into three main tasks: measuring the deformation w, modeling the optical stress σ_r, and computing the GGF from the stress. Several approaches to these problems have been presented in the related literature and are discussed in the following.

3.2 Experimentally quantifying deformation

3.2.1 Semimajor and -minor axes of an ellipse fit

todo

3.2.2 Boundary function fitted to the contour

tbd

3.3 Optical stress profile acting on a prolate spheroid

The optical stress $\sigma(\theta)$ in dependence of the angle θ is a result of the optical forces acting on the surface of the spheroid. The angle θ is defined in the imaging plane in a typical OS experiment, with $\theta = 0$ pointing to the right hand fiber.

3.3.1 $\cos^2\theta$ approximation

Ray optics is used to compute the optical stress acting on a prolate spheroid and a $\sigma_0 cos^2\theta$ model is fitted to the resulting stress profile with the peak stress σ_0 *[GAM+01]*. The $\sigma_0 cos^2\theta$ approximation simplifies subsequent computations.

Note that a more general model $\sigma_0 cos^2 n\theta$ with larger exponents (e.g. n = 2,3,4,...) can also be applied, e.g. for different fibroblast cell lines *[AGW+06]*.

3.3.2 Semi-analytical perturbation approach (Boyde et al. 2009)

- gaussian laser beam
- $a > \lambda$: higher order perturbation theory
- *[BCG09]*

3.3.3 Generalized Lorentz-Mie theory (Boyde et al. 2012)

tbd

3.4 Computation of the GGF

The following derivations are based on the theoretical considerations of Lur'e *[Lure64]* for a rotationally symmetric deformation of a sphere and their application to the OS by Ananthakrishnan et al. *[AGW+06]*. Note that a corrigendum has been published for this article in 2008 *[AGW+08]*.

3.4.1 General approach

The GGF connects the measured deformation to the shear modulus G which, in OS literature, is usually written in the form

$$\frac{w}{r_0} = \frac{\text{GGF}}{G}$$

where w is the change in radius of the stretched sphere along the stretcher axis and r_0 is the radius of the unstretched sphere. Note that the quantity w/r_0 resembles a measure of strain along the stretcher axis.

The GGF can be computed from the radial stress $\sigma_r(\theta)$ via the radial displacement $u_r(r,\theta)$. These quantities can be connected via a Legendre decomposition according to (*[Lure64]*, chapter 6)

$$u_r(r,\theta) = \sum_n \left[A_n r^{n+1}(n+1)(n-2+4\nu) + B_n r^{n-1} n\right] P_n(\cos\theta)$$

$$\frac{\sigma_r(r,\theta)}{2G} = \sum_n \left[A_n r^n (n+1)(n^2-n-2-2\nu) + B_n r^{n-2} n(n-1)\right] P_n(\cos\theta)$$

with the Legendre polynomials P_n and the Poisson's ratio ν. The coefficients A_n and B_n have to be determined from boundary conditions. For the case of normal loading, which is given by the electromagnetic boundary conditions in the OS ($\sigma_\theta = \tau_{r,\theta} = 0$), these coefficients compute to:

$$A_0 = -\frac{s_0}{4G(1+\nu)}$$

$$B_0 = A_1 = B_1 = 0$$

and for $n >= 2$:

$$A_n = -\frac{s_n}{4Gr_0^n\Delta}$$

$$B_n = \frac{s_n}{4Gr_0^{n-2}\Delta} \cdot \frac{n^2+2n-1+2\nu}{n-1}$$

$$\text{with } \Delta = n(n-1) + (2n+1)(\nu+1)$$

Where s_n is the nth component of the Legendre decomposition of σ_r

$$\sigma_r(\theta) = \sum_n s_n P_n(\cos\theta).$$

The radial displacement then takes the form

$$u_r(r,\theta) = \frac{r_0}{G}\left[\frac{(1-2\nu)s_0}{2(1+\nu)} + \sum_{n=2}^{\infty}\frac{2s_n}{2n+1}\left(L_n\left(\frac{r}{r_0}\right)^n + M_n\left(\frac{r}{r_0}\right)^{n-2}\right)P_n(\cos\theta)\right]$$

with the coefficients L_n and M_n given in *[Lure64]*, chapter 6.6. We measure the displacement at the outer perimeter of the stretched object and on the stretcher axis only; Thus, we set $r = r_0$ and $\theta = 0$ with $w = u_r(r_0, 0)$.

To obtain the GGF, we finally compute

$$\begin{aligned}\mathrm{GGF} &= \frac{G}{r_0}u_r(r_0,0)\\ &= \left[\frac{(1-2\nu)s_0}{2(1+\nu)} + \sum_{n=2}^{\infty}\frac{2s_n}{2n+1}\left(L_n + M_n\right)P_n(\cos\theta)\right].\end{aligned}$$

Notes:

- Due to the fact hat the stress profile in the OS is rotationally symmetric w.r.t. the stretcher axis, all odd coefficients s_n are zero.
- The polar displacement u_θ has been omitted here, because it does not represent a quantity measurable in an OS experiment.

3.4.2 Special case: $\cos^2\theta$ approximation

Following the above approach, the stress profile

$$\sigma_r(\theta) = \sigma_0\cos^2\theta$$

with the peak stress σ_0 can be decomposed into two Legendre polynomials

$$\begin{aligned}\sigma_r(\theta) &= s_0P_0(\cos\theta) + s_2P_2(\cos\theta)\\ s_0 &= \frac{1}{3}\sigma_0\\ s_2 &= \frac{2}{3}\sigma_0\end{aligned}$$

Inserting these Legendre coefficients in the above equation for the GGF yields

$$\mathrm{GGF} = \frac{\sigma_0}{2(1+\nu)}\left[\frac{1}{3}\left((1-2\nu) + \frac{(-7+4\nu)(1+\nu)}{7+5\nu}\right) + \frac{(7-4\nu)(1+\nu)}{7+5\nu}\cos^2\theta\right].$$

Historically, the relation between strain, stress, and shear modulus was written in the form

$$\frac{w}{r_0} = \frac{\sigma_0 F_\mathrm{G}}{G}$$

with the geometrical factor F_G that does not include the peak stress σ_0. Hence the term "global geometrical factor" $\mathrm{GGF} = \sigma_0 F_\mathrm{G}$.

3.5 Computation of compliance

A typical OS experiment records the deformation $w(t)$ over time t. The quantity of interest is the (creep) compliance $J(t)$. With $J = 1/G$, it computes to

$$J(t) = \frac{w(t)}{r_0} \cdot \frac{1}{\mathrm{GGF}(t)}.$$

Note that the GGF is now time-dependent, because the optical stress profile σ_r, from which the GGF is computed, also depends on the deformation.

CHAPTER 4

Code examples

4.1 Applications

4.1.1 Creep compliance analysis

This example uses the contour data of an cell in the OS to compute its GGF and creep compliance. The contour data were determined from this phase-contrast video (prior to video compression). During stretching, the total laser power was increased from 0.2W to 1.3W (reflexes due to second harmonic effects appear as white spots).

creep_compliance.py

```
import ggf
import h5py
import lmfit
import matplotlib.pylab as plt
import numpy as np
import percache

mycache = percache.Cache("creep_compliance.cache", livesync=True)


def ellipse_fit(radius, theta):
    '''Fit a centered ellipse to data in polar coordinates

    Parameters
    ----------
    radius: 1d ndarray
        radial coordinates
    theta: 1d ndarray
        angular coordinates [rad]

    Returns
    -------
    a, b: floats
```

(continues on next page)

(continued from previous page)

```
        semi-axes of the ellipse; `a` is aligned with theta=0.
    '''
    def residuals(params, radius, theta):
        a = params["a"].value
        b = params["b"].value
        r = a*b / np.sqrt(a**2 * np.sin(theta)**2 + b**2 * np.cos(theta)**2)
        return r - radius

    parms = lmfit.Parameters()
    parms.add(name="a", value=radius.mean())
    parms.add(name="b", value=radius.mean())

    res = lmfit.minimize(residuals, parms, args=(radius, theta))

    return res.params["a"].value, res.params["b"].value

@mycache
def get_ggf(**kw):
    f = ggf.get_ggf(use_lut=True, **kw)
    return f

# load the contour data (stored in polar coordinates)
with h5py.File("data/creep_compliance_data.h5", "r") as h5:
    radius = h5["radius"].value * 1e-6  # [µm] to [m]
    theta = h5["theta"].value
    time = h5["time"].value
    meta = dict(h5.attrs)


factors = np.zeros(len(radius), dtype=float)
semimaj = np.zeros(len(radius), dtype=float)
semimin = np.zeros(len(radius), dtype=float)
strains = np.zeros(len(radius), dtype=float)
complnc = np.zeros(len(radius), dtype=float)

for ii in range(len(radius)):
    # determine semi-major and semi-minor axes
    smaj, smin = ellipse_fit(radius[ii], theta[ii])
    semimaj[ii] = smaj
    semimin[ii] = smin
    # compute GGF
    print("compute ggf smaj={:.3e}, smin={:.3e}".format(smaj, smin))
    if (time[ii] > meta["time_stretch_begin [s]"]
            and time[ii] < meta["time_stretch_end [s]"]):
        power_per_fiber=meta["power_per_fiber_stretch [W]"]
    else:
        power_per_fiber=meta["power_per_fiber_trap [W]"]
    f = get_ggf(model="boyde2009",
                semi_major=smaj,
                semi_minor=smin,
                object_index=meta["object_index"],
                medium_index=meta["medium_index"],
                effective_fiber_distance=meta["effective_fiber_distance [m]"],
                mode_field_diameter=meta["mode_field_diameter [m]"],
                power_per_fiber=power_per_fiber,
                wavelength=meta["wavelength [m]"],
                poisson_ratio=.5)
```

(continues on next page)

(continued from previous page)

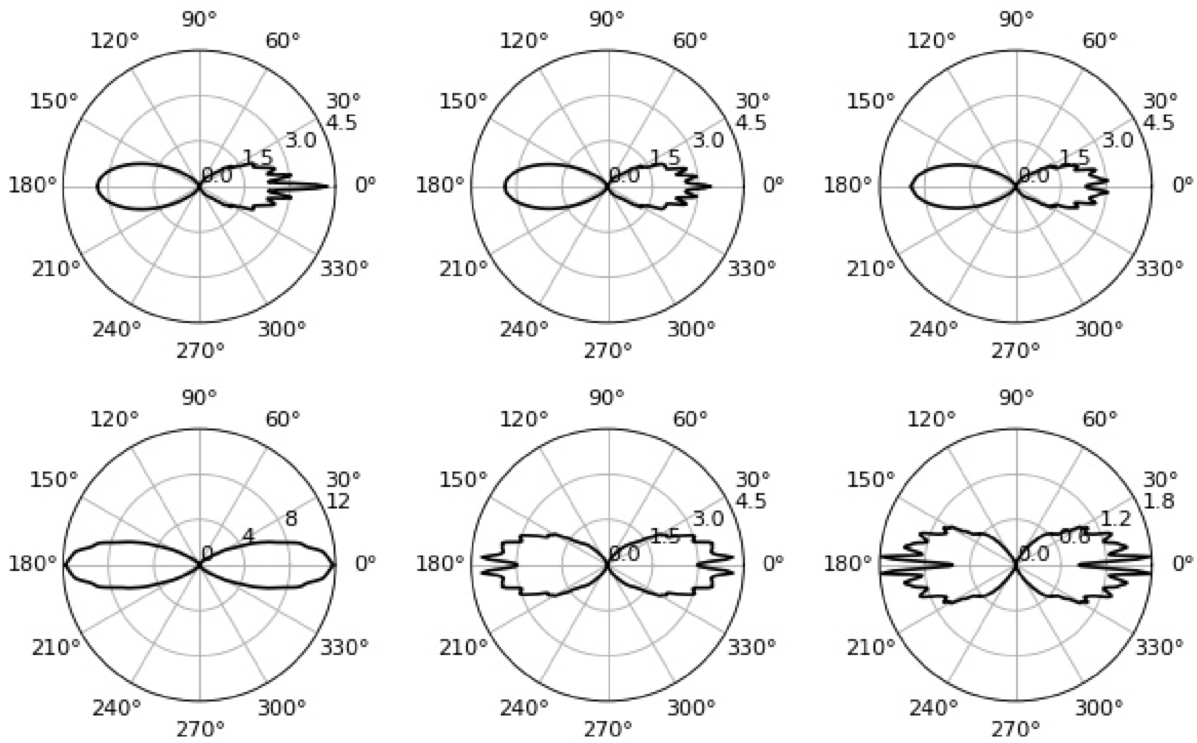
```
    print("... ", ii, f)
    factors[ii] = f

# compute compliance
strains = (semimaj-semimaj[0]) / semimaj[0]
complnc = strains / factors
compl_ival = (time>meta["time_stretch_begin [s]"]) * (time<meta["time_stretch_end [s]"])
stretch_index = np.where(compl_ival)[0][0]
complnc_1 = strains/factors[stretch_index]


# plots
plt.figure(figsize=(8, 7))

ax1 = plt.subplot(221, title="ellipse fit semi-axes")
ax1.plot(time, semimaj*1e6, label="semi-major axis")
ax1.plot(time, semimin*1e6, label="semi-minor axis")
ax1.legend()
ax1.set_xlabel("time [s]")
ax1.set_ylabel("axis radius [µm]")

ax2 = plt.subplot(222, title="GGF")
ax2.plot(time, factors)
ax2.set_xlabel("time [s]")
ax2.set_ylabel("global geometric factor [Pa]")

ax3 = plt.subplot(223, title="strain")
ax3.plot(time, (strains)*100)
ax3.set_xlabel("time [s]")
ax3.set_ylabel("deformation $w(t)/r_0$ [%]")

ax4 = plt.subplot(224, title="compliance")
ax4.plot(time[compl_ival], complnc_1[compl_ival], label="single GGF")
ax4.plot(time[compl_ival], complnc[compl_ival], label="series GGF")
ax4.legend()
ax4.set_xlabel("time [s]")
ax4.set_ylabel("creep compliance $J(t)$ [Pa¹]")

for ax in [ax1, ax2, ax3, ax4]:
    ax.set_xlim(0, np.round(time.max()))
    ax.axvline(x=meta["time_stretch_begin [s]"], c="r")
    ax.axvline(x=meta["time_stretch_end [s]"], c="r")

plt.tight_layout()
plt.show()
```

4.2 Reproduction tests

4.2.1 Radial stresses of a prolate spheroid

This examples computes radial stress profiles for spheroidal objects in the optical stretcher, reproducing figures (9) and (10) of *[BCG09]*.

`stress_reproduced.py`




```
import matplotlib.pylab as plt
import numpy as np
import percache

from ggf.stress.boyde2009.core import stress


@percache.Cache("stress_reproduced.cache", livesync=True)
def compute(**kwargs):
    "Locally cached version of ggf.core.stress"
    return stress(**kwargs)


# variables from the publication
alpha = 47
wavelength = 1064e-9
radius = alpha * wavelength / (2 * np.pi)

kwargs = {"stretch_ratio": .1,
          "object_index": 1.375,
          "medium_index": 1.335,
          "wavelength": wavelength,
          "beam_waist": 3,
          "radius": radius,
          "power_left": 1,
          "power_right": 1,
          "poisson_ratio": 0,
          "n_points": 200,
          }
```

(continues on next page)

(continued from previous page)

```

kwargs1 = kwargs.copy()
kwargs1["power_right"] = 0
kwargs1["stretch_ratio"] = 0
kwargs1["dist"] = 90e-6

kwargs2 = kwargs.copy()
kwargs2["power_right"] = 0
kwargs2["stretch_ratio"] = .05
kwargs2["dist"] = 90e-6

kwargs3 = kwargs.copy()
kwargs3["power_right"] = 0
kwargs3["stretch_ratio"] = .1
kwargs3["dist"] = 90e-6

kwargs4 = kwargs.copy()
kwargs4["dist"] = 60e-6

kwargs5 = kwargs.copy()
kwargs5["dist"] = 120e-6

kwargs6 = kwargs.copy()
kwargs6["dist"] = 200e-6


# polar plots
plt.figure(figsize=(8, 5))

th1, sigma1 = compute(**kwargs1)
ax1 = plt.subplot(231, projection='polar')
ax1.plot(th1, sigma1, "k")
ax1.plot(th1 + np.pi, sigma1[::-1], "k")

th2, sigma2 = compute(**kwargs2)
ax2 = plt.subplot(232, projection='polar')
ax2.plot(th2, sigma2, "k")
ax2.plot(th2 + np.pi, sigma2[::-1], "k")

th3, sigma3 = compute(**kwargs3)
ax3 = plt.subplot(233, projection='polar')
ax3.plot(th3, sigma3, "k")
ax3.plot(th3 + np.pi, sigma3[::-1], "k")

for ax in [ax1, ax2, ax3]:
    ax.set_rticks([0, 1.5, 3, 4.5])
    ax.set_rlim(0, 4.5)

th4, sigma4 = compute(**kwargs4)
ax4 = plt.subplot(234, projection='polar')
ax4.plot(th4, sigma4, "k")
ax4.plot(th4 + np.pi, sigma4[::-1], "k")
ax4.set_rticks([0, 4, 8, 12])
ax4.set_rlim(0, 12)

th5, sigma5 = compute(**kwargs5)
ax5 = plt.subplot(235, projection='polar')
```

(continues on next page)

(continued from previous page)

```
ax5.plot(th5, sigma5, "k")
ax5.plot(th5 + np.pi, sigma5[::-1], "k")
ax5.set_rticks([0, 1.5, 3, 4.5])
ax5.set_rlim(0, 4.5)

th6, sigma6 = compute(**kwargs6)
ax6 = plt.subplot(236, projection='polar')
ax6.plot(th6, sigma6, "k")
ax6.plot(th6 + np.pi, sigma6[::-1], "k")
ax6.set_rticks([0, 0.6, 1.2, 1.8])
ax6.set_rlim(0, 1.8)

for ax in [ax1, ax2, ax3, ax4, ax5, ax6]:
    ax.set_thetagrids(np.linspace(0, 360, 12, endpoint=False))

plt.tight_layout()
plt.show()
```

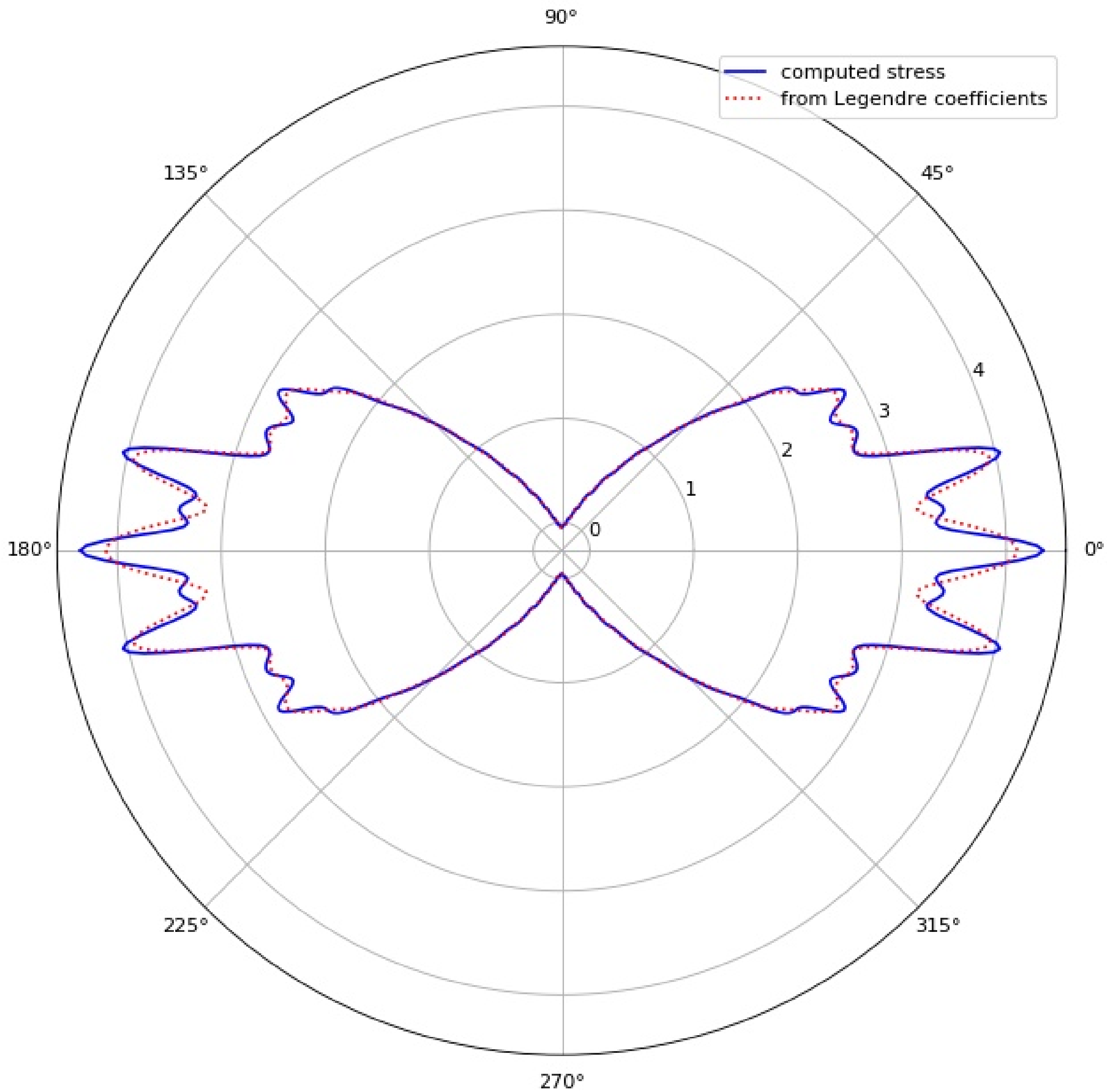
4.2.2 Decomposition of stress in Legendre polynomials

To compute the GGF, `ggf.globgeomfact.coeff2ggf()` uses the coefficients of the decomposition of the stress into Legendre polynomials $P_n(\cos(\theta))$. This example visualizes the small differences between the original stress and the stress computed from the Legendre coefficients. This plot is automatically produced by the original Matlab script *StretcherNStress.m*.

stress_decomposition.py

```
import matplotlib.pylab as plt
import numpy as np
import percache

from ggf.sci_funcs import legendrePlm
from ggf.stress.boyde2009.core import stress


@percache.Cache("stress_decomposition.cache", livesync=True)
def compute(**kwargs):
    "Locally cached version of ggf.core.stress"
    return stress(**kwargs)


# compute default stress
theta, sigmarr, coeff = compute(ret_legendre_decomp=True,
                                numpoints=300)

# compute stress from coefficients
numpoints = theta.size
sigmarr_c = np.zeros((numpoints, 1), dtype=float)
for ii in range(numpoints):
    for jj, cc in enumerate(coeff):
        sigmarr_c[ii] += coeff[jj] * \
            np.real_if_close(legendrePlm(0, jj, np.cos(theta[ii])))

# polar plot
plt.figure(figsize=(8, 8))
```

(continues on next page)

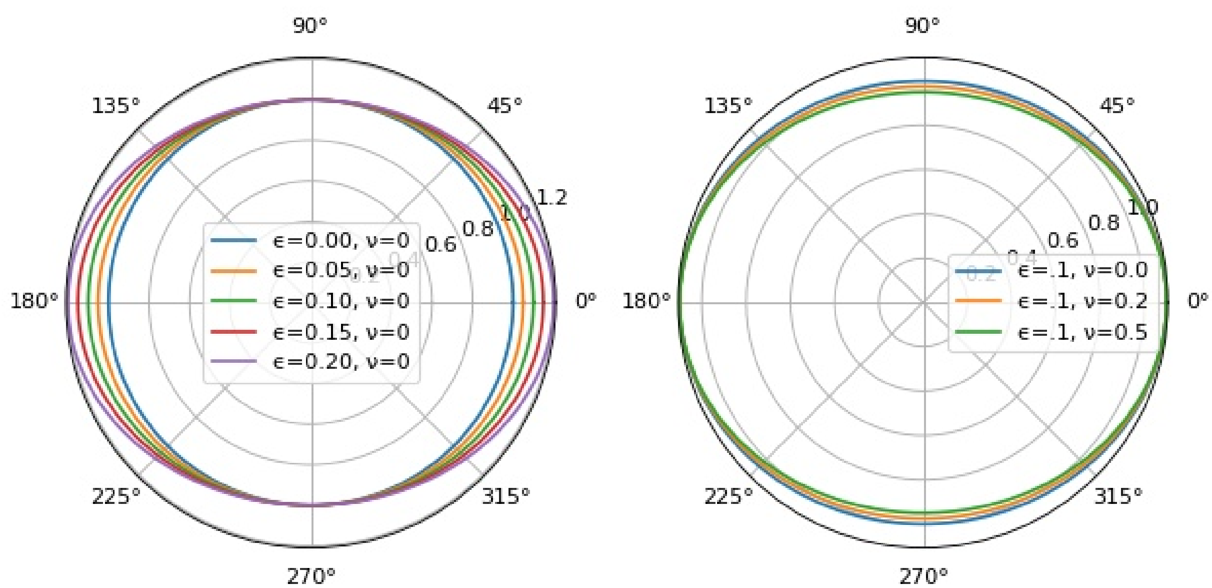

90°
135°
45°
180°
0°
225°
315°
270°
0
1
2
3
4
computed stress
from Legendre coefficients

(continued from previous page)

```
ax = plt.subplot(111, projection="polar")
plt.plot(theta, sigmarr, '-b', label="computed stress")
plt.plot(theta + np.pi, sigmarr[::-1], '-b')
plt.plot(theta, sigmarr_c, ':r', label="from Legendre coefficients")
plt.plot(theta + np.pi, sigmarr_c[::-1], ':r')
plt.legend()

plt.tight_layout()
plt.show()
```

4.2.3 Object boundary: stretching and Poisson's ratio

This example illustrates how the parameters Poisson's ratio ν and stretch ratio ϵ influence the object boundary used in `ggf.core.stress()` and defined in `ggf.core.boundary()`.



boundary.py

```
import numpy as np
import matplotlib.pylab as plt

from ggf.stress.boyde2009 import boundary

theta = np.linspace(0, 2*np.pi, 300)
costheta = np.cos(theta)

# change epsilon
eps = [.0, .05, .10, .15, .20]
b1s = []
for ep in eps:
    b1s.append(boundary(costheta=costheta,
                        epsilon=ep,
                        nu=.0))
```

(continues on next page)

(continued from previous page)

```

# change Poisson's ratio
nus = [.0, .25, .5]
b2s = []
for nu in nus:
    b2s.append(boundary(costheta=costheta,
                        epsilon=.1,
                        nu=nu))

# plot
plt.figure(figsize=(8, 4))

ax1 = plt.subplot(121, projection="polar")
for ep, bi in zip(eps, b1s):
    ax1.plot(theta, bi, label="={:.2f}, ν=0".format(ep))
ax1.legend()

ax2 = plt.subplot(122, projection="polar")
for nu, bi in zip(nus, b2s):
    ax2.plot(theta, bi, label="=.1, ν={:.1f}".format(nu))
ax2.legend()

plt.tight_layout()
plt.show()
```

CHAPTER 5

Code reference

5.1 module-level

ggf.**fiber_distance_capillary**(*gel_thickness=2e-06*, *glass_thickness=4e-05*, *channel_width=4e-05*, *gel_index=1.449*, *glass_index=1.474*, *medium_index=1.335*)

Effective distance between the two optical fibers

When the optical stretcher is combined with a microfluidic channel (closed setup), then the effective distance between the two optical fibers (with the the stretched object at the channel center) is defined by the refractive indices of the optical components: index matching gel between fiber and channel wall, microfluidic glass channel wall, and medium inside the channel.

Parameters

- **gel_thickness** (*float*) – Thickness of index matching gel (distance between fiber and glass wall) [m]
- **glass_thickness** (*float*) – Thickness of glass wall [m]
- **channel_width** (*float*) – Width of the microfluidic channed [m]
- **gel_index** (*float*) – Refractive index of index matching gel
- **glass_index** (*float*) – Refractive index of channel glass wall
- **medium_index** (*float*) – Refractive index of index medium inside channel

Returns **eff_dist** – Effective distance between the fibers

Return type float

Notes

The effective distance is computed relative to the medium, i.e. if *gel_index* == *glass_index* == *medium_index*, then *eff_dist* = 2*‘gel_dist‘ + 2*‘glass_dist‘ + *channel_width*.

`ggf.get_ggf`(*model*, *semi_major*, *semi_minor*, *object_index*, *medium_index*, *effective_fiber_distance=0.0001*, *mode_field_diameter=3e-06*, *power_per_fiber=0.6*, *wavelength=1.064e-06*, *poisson_ratio=0.5*, *n_poly=120*, *use_lut=None*, *verbose=False*)

Model the global geometric factor

Parameters

- **model** (*str*) – Model to use, one of: *boyde2009*
- **semi_major** (*float*) – Semi-major axis of an ellipse fit to the object perimeter [m]
- **semi_minor** (*float*) – Semi-minor axis of an ellipse fit to the object perimeter [m]
- **object_index** (*float*) – Refractive index of the object
- **medium_index** (*float*) – Refractive index of the surrounding medium
- **effective_fiber_distance** (*float*) – Effective distance between the two trapping fibers relative to the medium refractive index [m]. For an open setup, this is the physical distance between the fibers. For a closed setup (capillary), this distance takes into account the refractive indices and thicknesses of the glass capillary and index matching gel. For the closed setup, the convenience function `ggf.fiber_distance_capillary()` can be used.
- **mode_field_diameter** (*float*) – The mode field diameter MFD of the fiber used [m]. Note that the MFD is dependent on the wavelength used. If the manufacturer did not provide a value for the MFD, the MFD can be approximated as `3*wavelenth` for a single-mode fiber.
- **power_per_fiber** (*float*) – The laser power coupled into each of the fibers [W]
- **wavelength** (*float*) – The laser wavelength used for the trap [m]
- **poisson_ratio** (*float*) – The Poisson's ratio of the stretched material. Set this to 0.5 for volume conservation.
- **n_poly** (*int*) – Number of Legendre polynomials to use for computing the GGF. Note that only even Legendre polynomials are used and thus, this number is effectively halved. To reproduce the GGF as computed with the Boyde2009 Matlab script, set this value to *None*.
- **use_lut** (*None, str, pathlib.Path or bool*) – Use look-up tables to compute the GGF. If set to *None*, the internal LUTs will be used or the GGF is computed if it cannot be found in a LUT. If *True*, the internal LUTs will be used and a *NotInLUTError* will be raised if the GGF cannot be found there. Alternatively, a path to a LUT hdf5 file can be passed.
- **verbose** (*int*) – Increases verbosity

Returns **ggf** – global geometric factor

Return type float

`ggf.legendre2ggf`(*coeff*, *poisson_ratio*)

Compute the global geometric factor from Legendre coefficients

The definition of the Legendre coefficients is given in *the theory section*.

Parameters

- **coeff** (*1d ndarray*) – Legendre coefficients as defined in *[Lure64]*
- **poisson_ratio** (*float*) – Poisson's ratio of the stretched material. Set this to 0.5 for volume conservation.

Returns **ggf** – Global geometric factor

Return type float

Notes

All odd Legendre coefficients are assumed to be zero, because the stress profile is symmetric with respect to the stretcher axis.

`ggf.`**`stress2ggf`**(*stress*, *theta*, *poisson_ratio*, *n_poly=120*)

Compute the GGf from radial stress using Legendre decomposition

Parameters

- **stress** (*1d ndarray*) – Radial stress profile (in imaging plane)
- **theta** (*1d ndarray*) – Polar angles corresponding to *stress*
- **poisson_ratio** (*float*) – Poisson's ratio of the stretched material. Set this to 0.5 for volume conservation.
- **n_poly** (*int*) – Number of Legendre polynomials to use

Returns **ggf** – Global geometric factor

Return type float

Notes

All odd Legendre coefficients are assumed to be zero, because the stress profile is symmetric with respect to the stretcher axis. Therefore, only *n_poly/2* polynomials are considered.

`ggf.`**`stress2legendre`**(*stress*, *theta*, *n_poly*)

Decompose stress into even Legendre Polynomials

The definition of the Legendre decomposition is given in *the theory section*.

Parameters

- **stress** (*1d ndarray*) – Radial stress profile (in imaging plane)
- **theta** (*1d ndarray*) – Polar angles corresponding to *stress*
- **n_poly** (*int*) – Number of Legendre polynomials to use

Returns **coeff** – Legendre coefficients as defined in *[Lure64]*

Return type 1d ndarray

Notes

All odd Legendre coefficients are assumed to be zero, because the stress profile is symmetric with respect to the stretcher axis. Therefore, only *n_poly/2* polynomials are considered.

5.2 matlab_funcs

Special functions translated from Matlab to Python `ggf.matlab_funcs.besselh(n, z)`

Hankel function with k = 1

Parameters

- **n** (*int*) – real order
- **z** (*float*) – complex argument

Notes

https://de.mathworks.com/help/matlab/ref/besselh.html

`ggf.matlab_funcs.besselj(n, z)`

Bessel function of first kind

Parameters

- **n** (*int*) – real order
- **z** (*float*) – complex argument

Notes

https://de.mathworks.com/help/matlab/ref/besselj.html

`ggf.matlab_funcs.gammaln(x)`

Logarithm of the absolute value of the Gamma function

Notes

https://de.mathworks.com/help/matlab/ref/gammaln.html

See also:

`scipy.special.gammaln()`

`ggf.matlab_funcs.legendre(n, x)`

Associated Legendre functions

Parameters

- **n** (*int*) – degree
- **x** (*ndarray of floats*) – argument

Notes

x is treated always as a row vector

The statement legendre(2,0:0.1:0.2) returns the matrix

/	x = 0	x = 0.1	x = 0.2
m = 0	-0.5000	-0.4850	-0.4400
m = 1	0	-0.2985	-0.5879
m = 2	3.0000	2.9700	2.8800

Notes

https://de.mathworks.com/help/matlab/ref/legendre.html

ggf.matlab_funcs.**lscov**(*A*, *B*, *w=None*)

Least-squares solution in presence of known covariance

$A \cdot x = B$, that is, x minimizes $(B - A \cdot x)^T \cdot \text{diag}(w) \cdot (B - A \cdot x)$. The matrix w typically contains either counts or inverse variances.

Parameters

- **A** (*matrix or 2d ndarray*) – input matrix
- **B** (*vector or 1d ndarray*) – input vector

Notes

https://de.mathworks.com/help/matlab/ref/lscov.html

ggf.matlab_funcs.**quadl**(*func*, *a*, *b*)

Numerically evaluate integral with scipy QUADPACK quadrature

Parameters

- **func** (*callable*) – function to integrate
- **a** (*float*) – interval start
- **b** (*float*) – interval end

Notes

https://de.mathworks.com/help/matlab/ref/quadl.html

5.3 sci_funcs

Other scientific functions

ggf.sci_funcs.**legendrePlm**(*m*, *l*, *x*)

5.4 stress

ggf.stress.**get_stress**(*model*, *semi_major*, *semi_minor*, *object_index*, *medium_index*, *effective_fiber_distance=0.0001*, *mode_field_diameter=3e-06*, *power_per_fiber=0.6*, *wavelength=1.064e-06*, *n_points=100*, *verbose=False*)

Compute the optical stress profile in the optical stretcher

Parameters

- **model** (*str*) – Model to use, one of: *boyde2009*
- **semi_major** (*float*) – Semi-major axis of an ellipse fit to the object perimeter [m]
- **semi_minor** (*float*) – Semi-minor axis of an ellipse fit to the object perimeter [m]

- **object_index** (*float*) – Refractive index of the object
- **medium_index** (*float*) – Refractive index of the surrounding medium
- **effective_fiber_distance** (*float*) – Effective distance between the two trapping fibers relative to the medium refractive index [m]. For an open setup, this is the physical distance between the fibers. For a closed setup (capillary), this distance takes into account the refractive indices and thicknesses of the glass capillary and index matching gel. For the closed setup, the convenience function *ggf.fiber_distance_capillary()* can be used.
- **mode_field_diameter** (*float*) – The mode field diameter MFD of the fiber used [m]. Note that the MFD is dependent on the wavelength used. If the manufacturer did not provide a value for the MFD, the MFD can be approximated as `3*wavelenth` for a single-mode fiber.
- **power_per_fiber** (*float*) – The laser power coupled into each of the fibers [W]
- **wavelength** (*float*) – The laser wavelength used for the trap [m]
- **n_points** (*int*) – Number of points to compute.
- **verbose** (*int*) – Increases verbosity

Returns

- **theta** (1d ndarray of length *n_points*) – Polar angles [rad]
- **sigma** (1d ndarray of length *n_points*) – Radial stress profile along *theta* [Pa]

5.4.1 stress.boyde2009

5.4.1.1 stress.boyde2009.core

ggf.stress.boyde2009.core.**boundary**(*costheta*, *a=1*, *epsilon=0.1*, *nu=0*)

Projected boundary of a prolate spheroid

Compute the boundary according to equation (4) in *[BCG09]* with the addition of the Poisson's ratio of the object.

$$B(\theta) = a(1+\epsilon)(1-\nu\epsilon)\left[(1+\epsilon)^2 - \epsilon(1+\nu)(2+\epsilon(1-\nu))\cos^2\theta\right]^{-1/2}$$

Parameters

- **costheta** (*float or np.ndarray*) – Cosine of polar coordinates θ at which to compute the boundary.
- **a** (*float*) – Equatorial radii of prolate spheroid (semi-minor axis) $b' = c' \equiv a$.
- **epsilon** (*float*) – Stretch ratio; defines size of semi-major axis: $a' = (1+\epsilon)a$. Note that this is not the eccentricity of the prolate spheroid.
- **nu** (*float*) – Poisson's ratio ν of the material.

Returns **B** – Radial object boundary in dependence of theta $B(\theta)$.

Return type 1d ndarray

Notes

For $\nu = 0$, the above equation becomes equation (4) in *[BCG09]*.

ggf.stress.boyde2009.core.**get_hgc**

Load hypergeometric coefficients from *hypergeomdata2.dat*.

These coefficients were computed by Lars Boyde using Wolfram Mathematica.

ggf.stress.boyde2009.core.**stress**(*object_index=1.41*, *medium_index=1.3465*, *poisson_ratio=0.45*, *radius=2.8466e-06*, *stretch_ratio=0.1*, *wavelength=7.8e-07*, *beam_waist=3*, *power_left=0.6*, *power_right=0.6*, *dist=0.0001*, *n_points=100*, *theta_max=<Mock name='mock.pi' id='140228861086856'>*, *field_approx='davis'*, *ret_legendre_decomp=False*, *verbose=False*)

Compute the stress acting on a prolate spheroid

Parameters

- **object_index** (*float*) – Refractive index of the spheroid
- **medium_index** (*float*) – Refractive index of the surrounding medium
- **poisson_ratio** (*float*) – Poisson's ratio of the spheroid material
- **radius** (*float*) – TODO
- **stretch_ratio** (*float*) – TODO
- **wavelength** (*float*) – Wavelenth of the gaussian beam [m]
- **beam_waist** (*float*) – Beam waist radius of the gaussian beam [wavelengths]
- **power_left** (*float*) – Laser power of the left beam [W]
- **power_right** (*float*) – Laser power of the right beam [W]
- **dist** (*float*) – Distance between beam waist and object center [m]
- **n_points** (*int*) – Number of points to compute stresses for
- **theta_max** (*float*) – Maximum angle to compute stressed for
- **field_approx** (*str*) – TODO
- **ret_legendre_decomp** (*bool*) – If True, return coefficients of decomposition of stress into Legendre polynomials
- **verbose** (*int*) – Increase verbosity

Returns

- **theta** (*1d ndarray*) – Angles for which stresses are computed
- **sigma_rr** (*1d ndarray*) – Radial stress corresponding to angles
- **coeff** (*1d ndarray*) – If *ret_legendre_decomp* is True, return compositions of decomposition of stress into Legendre polynomials.

Notes

- The angles *theta* are computed on a grid that does not include zero and *theta_max*.
- This implementation was first presented in *[BCG09]*.

5.4.1.2 stress.boyde2009.globgeomfact

Computation of the global geometric factor

`ggf.stress.boyde2009.globgeomfact.`**`coeff2ggf`**(*coeff*, *poisson_ratio=0.45*)

Compute the global geometric factor from stress coefficients

The radial displacements of an elastic sphere can be expressed in terms of Legendre polynomials (see *[Lure64]* equation 6.2.9) whose coefficients are computed from the Legendre decomposition of the radial stress.

Notes

- For a $\sigma_0 \cos^n(\theta)$ stress profile, the GGF already includes the peak stress according to:

$$\mathrm{GGF} = \sigma_0 F_{\mathrm{G}}.$$

- This is a conversion of the Matlab script GGF.m to Python. The code solves a linear system of equations to determine all Legendre coefficients. The new implementation in `ggf.legendre2ggf()` uses the direct solution and thus should be preferred.

CHAPTER 6

Changelog

List of changes in-between ggf releases.

6.1 version 0.2.0

- BREAKING CHANGES:
  - ref: changed submodule imports, please revise your scripts
  - ref: move computation of stress to stress.boyde2009 submodule
- feat: added support for GGF look-up tables Note that an experimental look-up table is already included in the release on PyPI, but not yet in the source tree.
- feat: analytical computation of GGF from Legendre polynomials
- globgeomfact.coeff2ggf returned complex instead of float
- docs: add preliminary theoretical part

6.2 version 0.1.0

- initial version

CHAPTER 7

Bilbliography

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[AGW+06] Revathi Ananthakrishnan, Jochen Guck, Falk Wottawah, Stefan Schinkinger, Bryan Lincoln, Maren Romeyke, Tess Moon, and Josef Käs. Quantifying the contribution of actin networks to the elastic strength of fibroblasts. *Journal of Theoretical Biology*, 242(2):502–516, sep 2006. doi:10.1016/j.jtbi.2006.03.021.

[AGW+08] Revathi Ananthakrishnan, Jochen Guck, Falk Wottawah, Stefan Schinkinger, Bryan Lincoln, Maren Romeyke, Tess Moon, and Josef Käs. Corrrigendum to "quantifying the contribution of actin networks to the elastic strength of fibroblasts". *Journal of Theoretical Biology*, 255(1):162, nov 2008. doi:10.1016/j.jtbi.2008.08.004.

[BCG09] Lars Boyde, Kevin J. Chalut, and Jochen Guck. Interaction of gaussian beam with near-spherical particle: an analytic-numerical approach for assessing scattering and stresses. *Journal of the Optical Society of America A*, 26(8):1814, jul 2009. doi:10.1364/josaa.26.001814.

[GAM+01] Jochen Guck, Revathi Ananthakrishnan, Hamid Mahmood, Tess J. Moon, C. Casey Cunningham, and Josef Käs. The optical stretcher: a novel laser tool to micromanipulate cells. *Biophysical Journal*, 81(2):767–784, aug 2001. doi:10.1016/s0006-3495(01)75740-2.

[Lure64] A.I Lur'e. *Three dimensional problems of the theory of elasticity*. Wiley, 1964.

Python Module Index

g

Index