Optical system transfer matrix

December 19, 2016

1 Fried parameter measurement - optical setup

1.1 Transfer matrix calculations

```
In [1]: %matplotlib inline

        import numpy as np
        import sympy as sp
        import matplotlib.pyplot as plt
        from IPython.display import display
        from enum import Enum

        sp.init_printing()
```

```
In [2]: # transfer matrices

        def T_propagation(d):
            return sp.Matrix([[1, d],[0,1]])

        def T_thinlens(f):
            return sp.Matrix([[1,0],[-1/f, 1]])

        # offset vector (non-linear)

        def OS_wedgeprism(alpha):
            return sp.Matrix([0, alpha])
```

```
In [3]: f_1, f_2, f_3, d_coll, d_tele, d_focus, M, delta = sp.symbols("f_1, f_2, f_
```

1.1.1 Construct a transfer matrix for all optical parts

```
In [4]: def sp2np(m, params = None):
            if (params):
                m = m.subs(params)
            return np.asarray(m.tolist(), dtype=np.float64)

        OpticalElementType = Enum('OpticalElementType', ['lens', 'propagation', 'we
```

```python
class Limits:
    def __init__(self, lower, upper):
        if (upper < lower):
            raise ValueError("Upper limit is below lower limit")
        self.lower = lower
        self.upper = upper

    def is_inside(self, x):
        if (self.lower != None and self.upper != None):
            return x > self.lower and x < self.upper
        elif (self.lower != None):
            return x > self.lower
        elif (self.upper != None):
            return x < self.upper
        else:
            return true

    def is_outside(self, x):
        return not self.is_inside(x)

class OpticalElement:
    def __init__(self, el_type, el_length = sp.Integer(0), T_matrix = None,
        self.el_type = el_type
        self.T_matrix = T_matrix
        self.OS_vector = OS_vector
        self.el_limits = el_limits
        self.el_length = el_length

    # apply this optical element to a set of rays
    def apply(self, rays, parameters):
        outrays = np.zeros_like(rays)

        for i in range(rays.shape[1]):
            if (self.el_limits != None):
                # check if ray x-coordinate is outside of optical element e
                if (self.el_limits.is_outside(rays[0,i])):
                    # if yes, do nothing
                    outrays[:,i] = rays[:,i]
                    continue

            # apply the linear part of the optical element (transfer matri
            if (self.T_matrix != None):
                npT = sp2np(self.T_matrix, parameters)
                outrays[:,i] += np.dot(npT, rays[:,i])
            else:
                outrays[:,i] += rays[:,i]

            # apply the offset vector of the optical element (for prisms, 
```

```
            if (self.OS_vector != None):
                npOS = sp2np(self.OS_vector, parameters)
                outrays[:,i] += npOS.flatten()

        return outrays

class ThinLens(OpticalElement):
    def __init__(self, focal_length, extent = None):
        super().__init__(OpticalElementType.lens, sp.Integer(0), T_thinlens

class Propagation(OpticalElement):
    def __init__(self, distance, extent = None):
        super().__init__(OpticalElementType.propagation, distance, T_propag

class WedgePrism(OpticalElement):
    def __init__(self, angle, extent = None):
        super().__init__(OpticalElementType.wedge_prism, sp.Integer(0), Nor

TL_entrance = T_thinlens(f_1)
TP_telescope = T_propagation(d_tele)
TL_collimator = T_thinlens(f_2)
TP_collimated = T_propagation(d_coll)
TL_focus = T_thinlens(f_3)
TP_focussed = T_propagation(d_focus)
V_nl_prism = OS_wedgeprism(delta)

system_elements = [
    ThinLens(f_1),
    Propagation(d_tele),
    ThinLens(f_2),
    Propagation(d_coll),
    WedgePrism(delta, Limits(-1e10, 0)),
    WedgePrism(-delta, Limits(0, 1e10)),
    ThinLens(f_3),
    Propagation(d_focus)
]

# system_elements = [
#     (OpElType.lens, TL_entrance, f_1),
#     (OpElType.propagation, TP_telescope, d_tele),
#     (OpElType.lens, TL_collimator, f_2),
#     (OpElType.propagation, TP_collimated, d_coll),
#     (OpElType.double_wedge_prism, None, delta), # note: double wedge prism
#     (OpElType.lens, TL_focus, f_3),
#     (OpElType.propagation, TP_focussed, d_focus)
#]

focussed_constraints = [(d_tele, f_1 + f_2), (d_focus, f_3)]
```

```
focussed_mag_constraints = [(d_focus, f_3), (f_2, M * f_1), (f_1, d_tele /

def focussed_system(expr):
    return sp.expand(expr.subs(focussed_constraints))

def in_terms_of_mag(expr):
    return sp.simplify(expr.subs(focussed_mag_constraints))
```

1.1.2 Telescope system transfer matrix

Entrance lens -> telescope propagation -> collimator lens

```
In [5]: TS_telescope = sp.simplify(TL_collimator * TP_telescope * TL_entrance)
        TS_telescope_focussed = focussed_system(TS_telescope)
        display(TS_telescope)

        display(TS_telescope_focussed)
```

$$\begin{bmatrix} \frac{1}{f_1}\left(-d_{tele} + f_1\right) & d_{tele} \\ \frac{1}{f_1 f_2}\left(d_{tele} - f_1 - f_2\right) & \frac{1}{f_2}\left(-d_{tele} + f_2\right) \end{bmatrix}$$

$$\begin{bmatrix} -\frac{f_2}{f_1} & f_1 + f_2 \\ 0 & -\frac{f_1}{f_2} \end{bmatrix}$$

1.1.3 Focussing system transfer matrix

Focussing lens -> focus propagation

```
In [6]: TS_focussing = TP_focussed * TL_focus
        TS_focussing_focussed = focussed_system(TS_focussing)
        display(TS_focussing)
        display(TS_focussing_focussed)
```

$$\begin{bmatrix} -\frac{d_{focus}}{f_3} + 1 & d_{focus} \\ -\frac{1}{f_3} & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & f_3 \\ -\frac{1}{f_3} & 1 \end{bmatrix}$$

1.1.4 Complete *linear* system transfer matrix

Telescope system -> collimated beam propagation -> focussing system
Does not include wedge prism, as this is a non-linear element

```
In [7]: TS_system = sp.simplify(TS_focussing * TP_collimated * TS_telescope)
        TS_system_focussed = focussed_system(TS_system)
        TS_system_focussed_mag = in_terms_of_mag(TS_system_focussed)
```

```
        display(TS_system)
        display(TS_system_focussed)
        display(TS_system_focussed_mag)
```

$$\left[\begin{matrix}\frac{1}{f_{1} f_{2} f_{3}} \left(f_{2} \left(d_{focus} - f_{3}\right) \left(d_{tele} - f_{1}\right) + \left(d_{coll} \left(d_{focus} - f_{3}\right) - d_{focus} f_{3}\right) \left(- d_{tele} + f_{1} + f_{2}\right)\right) & \frac{1}{f_{2} f_{3}} \left(- d_{tele} f_{2} \left(d_{focus} - f_{3}\right)\right. \\ \frac{1}{f_{1} f_{2} f_{3}} \left(f_{2} \left(d_{tele} - f_{1}\right) + \left(d_{coll} - f_{3}\right) \left(- d_{tele} + f_{1} + f_{2}\right)\right) & \frac{1}{f_{2} f_{3}} \left(- d_{tel}\right.\end{matrix}\right.$$

$$\left[\begin{matrix}0 & - \frac{f_{1} f_{3}}{f_{2}}\\\frac{f_{2}}{f_{1} f_{3}} & \frac{d_{coll} f_{1}}{f_{2} f_{3}} - \frac{f_{1}}{f_{3}} - \frac{f_{1}}{f_{2}} - \frac{f_{2}}{f_{3}}\end{matrix}\right]$$

$$\left[\begin{matrix}0 & - \frac{f_{3}}{M}\\\frac{M}{f_{3}} & \frac{1}{M f_{3}} \left(- M d_{tele} + d_{coll} - f_{3}\right)\end{matrix}\right]$$

1.2 Ray tracing calculations

Define a ray into the system...

```
In [8]: x, theta = sp.symbols("x, theta")
        ray_in = sp.Matrix([x, theta])
        display(ray_in)
```

$$\left[\begin{matrix}x\\\theta\end{matrix}\right]$$

(some exploratory calculations here)

```
In [9]: TL_entrance * ray_in
```

```
Out[9]:
```

$$\left[\begin{matrix}x\\\theta - \frac{x}{f_{1}}\end{matrix}\right]$$

```
In [10]: T_propagation(f_1 + f_2) * TL_entrance * ray_in
```

```
Out[10]:
```

$$\left[\begin{matrix}\theta \left(f_{1} + f_{2}\right) + x \left(1 - \frac{1}{f_{1}} \left(f_{1} + f_{2}\right)\right)\\\theta - \frac{x}{f_{1}}\end{matrix}\right]$$

... let it propagate through the telescope ...

```
In [11]: ray_telescoped = sp.expand(TS_telescope_focussed * ray_in)
         ray_telescoped_mag = in_terms_of_mag(ray_telescoped)
         display(ray_telescoped)
         display(ray_telescoped_mag)
```

$$\left[\begin{matrix}f_{1} \theta + f_{2} \theta - \frac{f_{2} x}{f_{1}}\\- \frac{f_{1} \theta}{f_{2}}\end{matrix}\right]$$

$$\left[\begin{matrix}- M x + d_{tele} \theta\\- \frac{\theta}{M}\end{matrix}\right]$$

… then as a quasi-collimated beam between the telescope and the focussing system …

In [12]:
```
ray_collimated = sp.expand(TP_collimated * TS_telescope_focussed * ray_in)
ray_collimated_mag = in_terms_of_mag(ray_collimated)

display(ray_collimated)
display(ray_collimated_mag)
```

$$\left[\begin{matrix}- \frac{d_{coll} f_{1}}{f_{2}} \theta + f_{1} \theta + f_{2} \theta - \frac{f_{2} x}{f_{1}}\\- \frac{f_{1} \theta}{f_{2}}\end{matrix}\right]$$

$$\left[\begin{matrix}- M x + d_{tele} \theta - \frac{d_{coll} \theta}{M}\\- \frac{\theta}{M}\end{matrix}\right]$$

… apply the non-linear wedge prism to the ray …

In [13]:
```
ray_prismed = ray_collimated + V_nl_prism
ray_prismed_mag = in_terms_of_mag(ray_prismed)

display(ray_prismed)
display(ray_prismed_mag)
```

$$\left[\begin{matrix}- \frac{d_{coll} f_{1}}{f_{2}} \theta + f_{1} \theta + f_{2} \theta - \frac{f_{2} x}{f_{1}}\\\delta - \frac{f_{1} \theta}{f_{2}}\end{matrix}\right]$$

$$\left[\begin{matrix}- M x + d_{tele} \theta - \frac{d_{coll} \theta}{M}\\\delta - \frac{\theta}{M}\end{matrix}\right]$$

… and finally into the focussing system.

In [14]:
```
ray_focussed = sp.expand(TS_focussing * ray_prismed)
ray_focussed_mag = sp.expand(in_terms_of_mag(ray_focussed))

display(ray_focussed)
display(ray_focussed_mag)
```

$$\left[\begin{matrix}\frac{d_{coll} d_{focus} f_{1} \theta}{f_{2} f_{3}} - \frac{d_{coll} f_{1}}{f_{2}} \theta + d_{focus} \delta - \frac{d_{focus} f_{1}}{f_{3}} \theta - \frac{d_{focus} f_{1}}{f_{2}} \theta - \frac{d_{focus} f_{2}}{f_{3}} \theta + \frac{d_{focus} f_{2} x}{f_{1} f_{3}} + f_{1} \theta + f_{2} \theta - \frac{f_{2} x}{f_{1}}\\\frac{d_{coll} f_{1} \theta}{f_{2} f_{3}} + \delta - \frac{f_{1} \theta}{f_{3}} - \frac{f_{1} \theta}{f_{2}} - \frac{f_{2} \theta}{f_{3}} + \frac{f_{2} x}{f_{1} f_{3}}\end{matrix}\right]$$

$$\left[\begin{matrix}\delta f_{3} - \frac{f_{3} \theta}{M}\\\frac{M x}{f_{3}} - \frac{d_{tele} \theta}{f_{3}} + \delta + \frac{d_{coll} \theta}{M f_{3}} - \frac{\theta}{M}\end{matrix}\right]$$

1.2.1 Ray tracing plots through the system

In [15]:
```
def draw_rays(elements, parameters, inrays):
    ray_points = np.zeros((len(elements) + 2, inrays.shape[1]))
    x_points = np.zeros((len(elements) + 2))
    lens_positions = np.zeros((len([el for el in elements if el.el_type ==

    # matrix to hold the ray data as they propagate
    proprays = np.copy(inrays)

    cur_x_pos = 0
    cur_i = 0
    cur_lens = 0

    d_in = sp.symbols("d_in")
    incoming_length = 100.0
    parameters = [(d_in, incoming_length)] + parameters

    # trace the incoming rays back a little bit, to show them nicely
    TP_backtrace = T_propagation(-d_in)
    npTP_backtrace = sp2np(TP_backtrace, parameters)
    proprays = np.dot(npTP_backtrace, proprays)

    # save the first point
    x_points[cur_i] = cur_x_pos
    ray_points[cur_i, :] = proprays[0,:]

    cur_i += 1

    TP_incoming = T_propagation(d_in)

    elements = [Propagation(d_in)] + elements

    for el in elements:
        # apply transformation to the rays
        proprays = el.apply(proprays, parameters)

        cur_x_pos += float(el.el_length.subs(parameters))
        # add a lens point if this is a lens
        if (el.el_type == OpticalElementType.lens):
            lens_positions[cur_lens] = cur_x_pos
            cur_lens += 1


        x_points[cur_i] = cur_x_pos
        ray_points[cur_i, :] = proprays[0,:]

        cur_i += 1
```

```
    max_y = np.amax(np.abs(ray_points))
    lim_y = 1.2*max_y

    plt.figure(figsize=(15,8))
    plt.plot(x_points, ray_points)
    plt.ylim([-lim_y, lim_y])

    # plot the optical axis
    plt.plot((x_points[0], x_points[-1]), (0, 0), 'k--')

    # draw the lenses
    for lens_x in lens_positions:
        plt.plot((lens_x, lens_x), (-lim_y, lim_y), 'k-', linewidth=2.0)

    plt.show()

    return x_points, ray_points
```

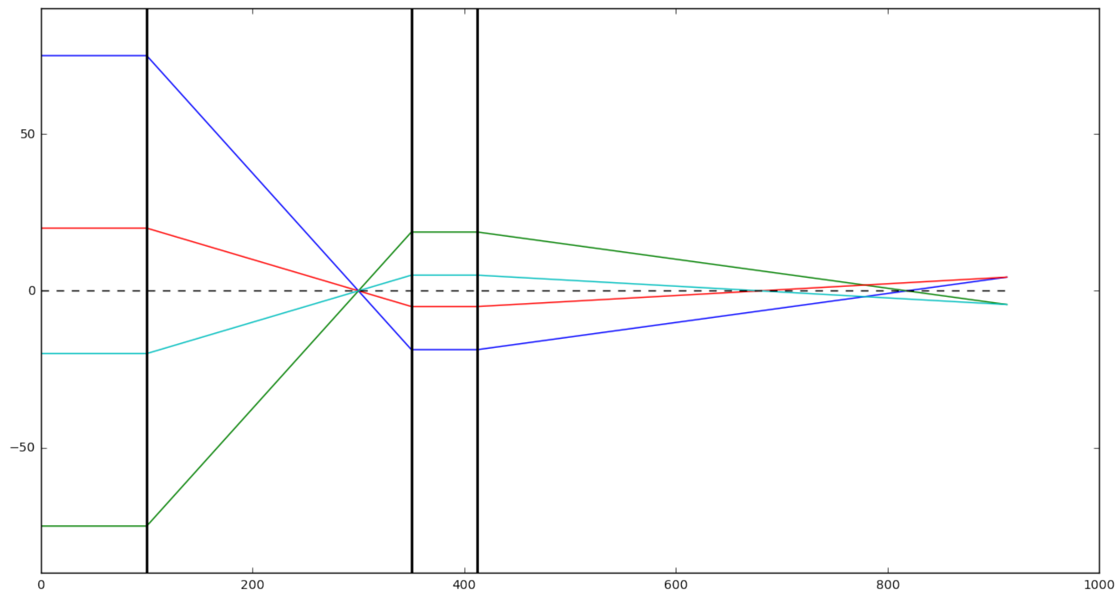
```
In [16]: system_parameters = [(f_1, 200), (f_2, 50), (d_coll, 62.5), (f_3, 500), (c

         display(ray_focussed.subs(focussed_constraints + system_parameters))

         # incoming rays x, theta
         rays = np.array([
                 [75, 0],
                 [-75, 0],
                 [20, 1e-6],
                 [-20, 1e-6],
             ]).T

         x, r = draw_rays(system_elements, focussed_constraints + system_parameters
```

$$\begin{bmatrix} -2000\theta + 4.36332312998582 \\ -4\theta + \frac{x}{2000} + 0.00872664625997165 \end{bmatrix}$$




```
In [17]: print(r.shape)

         print("spot 1 position [mm]: " + str(r[-1, 0]))
         print("spot 1 deviation [um]: " + str((r[-1, 2] - r[-1, 0]) * 1e3))

         print("spot 1 position [mm]: " + str(r[-1, 1]))
         print("spot 1 deviation [um]: " + str((r[-1, 3] - r[-1, 1]) * 1e3))
```

```
(10, 4)
spot 1 position [mm]: 4.36332312999
spot 1 deviation [um]: -1.99999999997
spot 1 position [mm]: -4.36332312999
spot 1 deviation [um]: -2.00000000001
```

```
In [ ]:
```

```
In [ ]:
```