# Berechnung einer Schattenprojektion

Karsten Goebbels, Cornelia Krome,
Vera Loeser, Andreas Mergl

**Vortrag**

Fachhochschule Aachen, Standort Jülich
Studiengang: Technomathematik

28. Juni 2016

# INHALT

# MOTIVATION

# MOTIVATION

→ Entwicklung einer neuen Messtechnologie

→ Schattenprojektion

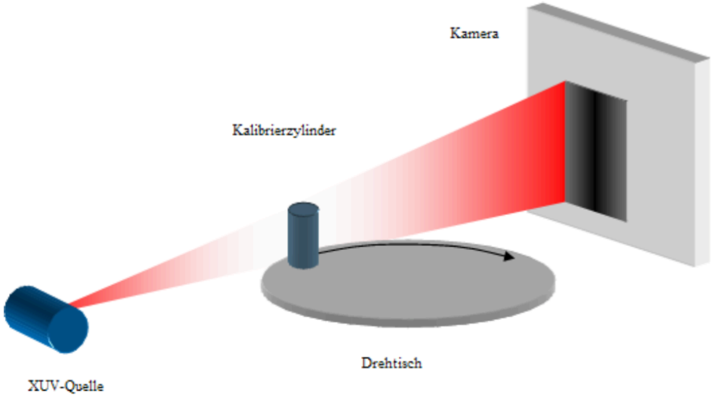
→ XUV-Strahlung

→ Genauigkeit taktiler und der Schnelligkeit optischer Messgeräte.

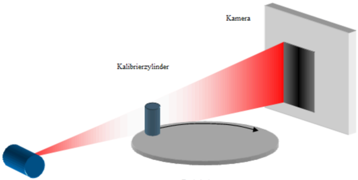
# PROBLEMSTELLUNG

- → Kalibrierung: Aktivitäten zur Ermittlung des Zusammenhanges zwischen einem Messwert und der zugehörigen Messgröße
- → kein Eingriff in die Messeinrichtung
- → mehrere Komponenten:
  - → XUV-Quelle (eXtrem UtraViolette-Strahlung)
  - → Detektor
  - → Drehteller
  - → Zylinder

# AUFBAU



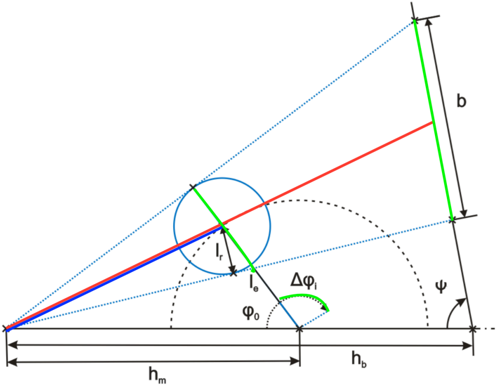
# GROBE WERTE




- → Radius vom Zylinder: 3 mm
- → Radius vom Drehteller: 10 cm
- → Abstand XUV-Quelle–Drehteller: 30 cm
- → Abstand Drehteller–Wand: 60 cm
- → Winkel Wand: 90°

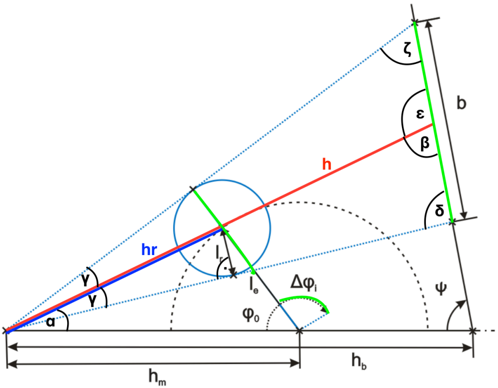
# MODELLIERUNG

# GEOMETRIE




- bekant:
  - b
  - $l_r$
  - $\Delta\phi_i$
- unbekannt: x
  - $\phi_0$
  - $l_e$
  - $h_m$
  - $h_b$
  - $\psi$

# GEOMETRIE




- → bekannt:
  - → b
  - → $l_r$
  - → $\Delta\phi_i$
- → unbekannt: x
  - → $\phi_0$
  - → $l_e$
  - → $h_m$
  - → $h_b$
  - → $\psi$

# BERECHNUNG

# SINUS- UND KOSINUSSATZ

Sinussatz:

$$\frac{a}{sin\alpha} = \frac{b}{sin\beta} = \frac{c}{sin\gamma}$$

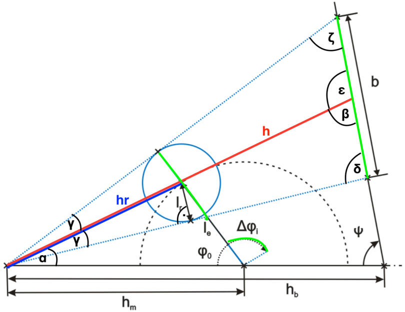
Kosinussatz (b, c analog):

$$a^2 = b^2 + c^2 - 2bc \cdot cos\alpha$$

# EXPLIZITE FORMEL

$$
\begin{aligned}
b(l_r, l_e, h_m, h_b, \phi, \psi) = \\
\Bigg[ & \frac{1}{-\sqrt{1-\left(\frac{l_r}{\sqrt{l_e^2+h_m^2-2 l_e h_m \cdot cos(\phi)}}\right)^2} + cos\left(2 * arcsin\left(\frac{l_e * sin(\phi)}{\sqrt{l_e^2+h_m^2-2 l_e h_m \cdot cos(\phi)}}\right) + 2 * \psi - arcsin\left(\frac{l_r}{\sqrt{l_e^2+h_m^2-2 l_e h_m \cdot cos(\phi)}}\right)\right)} \\
+ & \frac{1}{\sqrt{1-\left(\frac{l_r}{\sqrt{l_e^2+h_m^2-2 * l_e * h_m * cos(\phi)}}\right)^2} - cos\left(2 * arcsin\left(\frac{l_e * sin(\phi)}{\sqrt{l_e^2+h_m^2-2 * l_e * h_m * cos(\phi)}}\right) + 2 * \psi - arcsin\left(\frac{l_r}{\sqrt{l_e^2+h_m^2-2 * l_e * h_m * cos(\phi)}}\right)\right)} \Bigg] \\
& \cdot \left(2 * \frac{l_r}{\sqrt{l_e^2 + h_m^2 - 2 * l_e * h_m * cos(\phi)}} * sin(\phi) * h_b\right)
\end{aligned}
$$

# HERLEITUNG




$$h_r = \sqrt{l_e^2 + h_m^2 - 2 l_e h_m \cdot cos(\phi)}$$

$$\alpha = arcsin\left(l_e \cdot \frac{sin(\phi)}{h_r}\right)$$

$$\beta = \pi - \alpha - \psi$$

$$h = sin(\phi) \cdot \frac{h_b}{sin(\beta)}$$

$$\gamma = arcsin(\frac{l_r}{h_r})$$

$$\delta = \pi - \beta - \gamma$$

$$b_{bottom} = \frac{sin(\gamma)}{sin(\delta)} \cdot h$$

$$\epsilon = \pi - \beta$$

$$\zeta = \pi - \epsilon - \gamma$$

$$b_{top} = \frac{sin(\gamma)}{sin(\zeta)} \cdot h$$

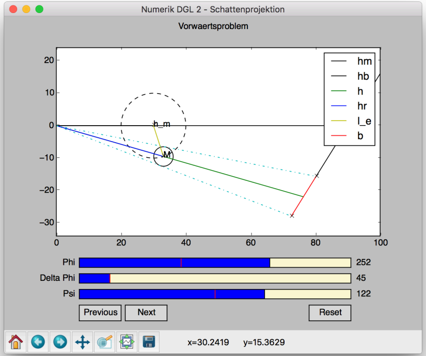
$$b = b_{top} + b_{bottom}$$

# IMPLEMENTIERUNG

# VORWÄRTSPROBLEM

→ alles bekannt, berechne Schattenbreite
→ direkte Implementierung der Herleitung
→ Plots und Prints zum Verifizieren

Live-Vorführung



# RÜCKWÄRTSPROBLEM

→ bekannt: Schattenbreite, Radius vom Zylinder und Drehwinkel

→ berechne: x

→ Gauß-Newton-Verfahren

$$x_{k+1} = x_k - F'(x_k)^{+} F(x_k)$$

# REGULARISIERUNG

→ Regularisierung mit Tikhonov

$$R_\alpha = (A^T A + \alpha I)^{-1} A^T \tag{1}$$

$$x_\alpha = R_\alpha y \Rightarrow (A^T A + \alpha I) x_\alpha = A^T y \tag{2}$$

→ adaptive Steuerung vom $\alpha$ zur Konvergenzbeschleunigung

# LIVE-DEMO

```python
def newton(F,x0,eps=1e-5,alpha=1.05e-4):
    it=int(1e2)
    x=np.copy(x0)
    Fx=F(x)

    residuum2=0
    residuum=np.linalg.norm(Fx,np.inf)
    while residuum > eps and it > 0:
        dFx = jacobi(F, x)

        # Tikhonov regularisation
        Fx = F(x)
        A = dFx
        R = np.dot(np.linalg.inv(np.dot(A.T, A) + alpha * np.eye(A.shape[1])), A.T)
        xalpha = np.dot(R, Fx)
        x -= xalpha

        residuum2=np.linalg.norm(Fx)
        rel_error=(residuum-residuum2)/residuum
        if residuum2 > residuum:
            alpha*=rel_error
        elif residuum > residuum2:
            alpha/=rel_error
        if abs(residuum2-residuum) < eps:
            break

        residuum=residuum2
        it-=1

    return x, residuum

def rueckwaerts(n, z, l_r):
    inp = np.array([[np.radians(float(i)/n * 360), vorwaerts(np.radians(float(i)/n * 360), x=z, l_r=l_r)] for i in range(n+1)], dtype=np.float64)
    F = lambda x: np.array([abs(vorwaerts(i[0], x, l_r)-i[1]) for i in inp], dtype=np.float64)
    x_0 = np.array([10, 30, np.radians(90), np.radians(90), 90], dtype=np.float64)

    for method in [ "CG", "BFGS", "COBYLA",]:
        res = minimize(lambda x: np.linalg.norm(F(x), np.inf), x_0, method=method)
        print("Minimize ({method}): \tResiduum = {residuum}, \tx = {x}".format(method=method, residuum=np.linalg.norm(F(res.x), np.inf), x=res.x))
    res = fmin(lambda x: np.linalg.norm(F(x), np.inf), x_0, disp=0)
    print("Fmin: \t\tResiduum = {residuum}, \tx = {x}".format(residuum=np.linalg.norm(F(res), np.inf), x=res))
    x, residuum = newton(F, x_0, alpha=5e-4)
    return x, residuum
```

ENDE

# ENDE

Vielen Dank für Eure Aufmerksamkeit.