Test PolaRam

Es wird überprüft, ob Simulationen mit PolaRam für partiell polarisiertes und unpolarisiertes Licht funktionieren. Es wird der Ramantensor α mit PolaRam in die entsprechende Müllermatrix M überführt.

$$\alpha = \begin{pmatrix} -0.2 & 0.4 & -0.9 \\ 0.4 & -0.4 & 0.6 \\ -0.9 & 0.6 & -0.7 \end{pmatrix}$$

$$M = \begin{pmatrix} 0.839481643 & -0.000334108129 & -0.000475955055 & 0 \\ -0.000334108129 & 0.281161455 & -7.1360143e-05 & 0 \\ -0.000475955055 & -7.1360143e-05 & 0.281248617 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Mit der Müllermatrix M werden zwei Experimente simuliert:

- M wird auf diverse totalpolarisierte lineare Stokesvektoren angewendet und berechnet wie groß das theoretisch messbare Detektorsignal ist
- Die Detektorantwort wird für Detektoren berechnet, der entweder empfindlich oder unempfindlich gegenüber der Polarisation des Lichtes ist
- M wird auf diverse verschieden stark polarisierte Stokesvektoren mit der gleichen Polarisationsebene angewendet
- Die Detektorantwort wird für Detektoren berechnet, der entweder empfindlich oder unempfindlich gegenüber der Polarisation des Lichtes ist

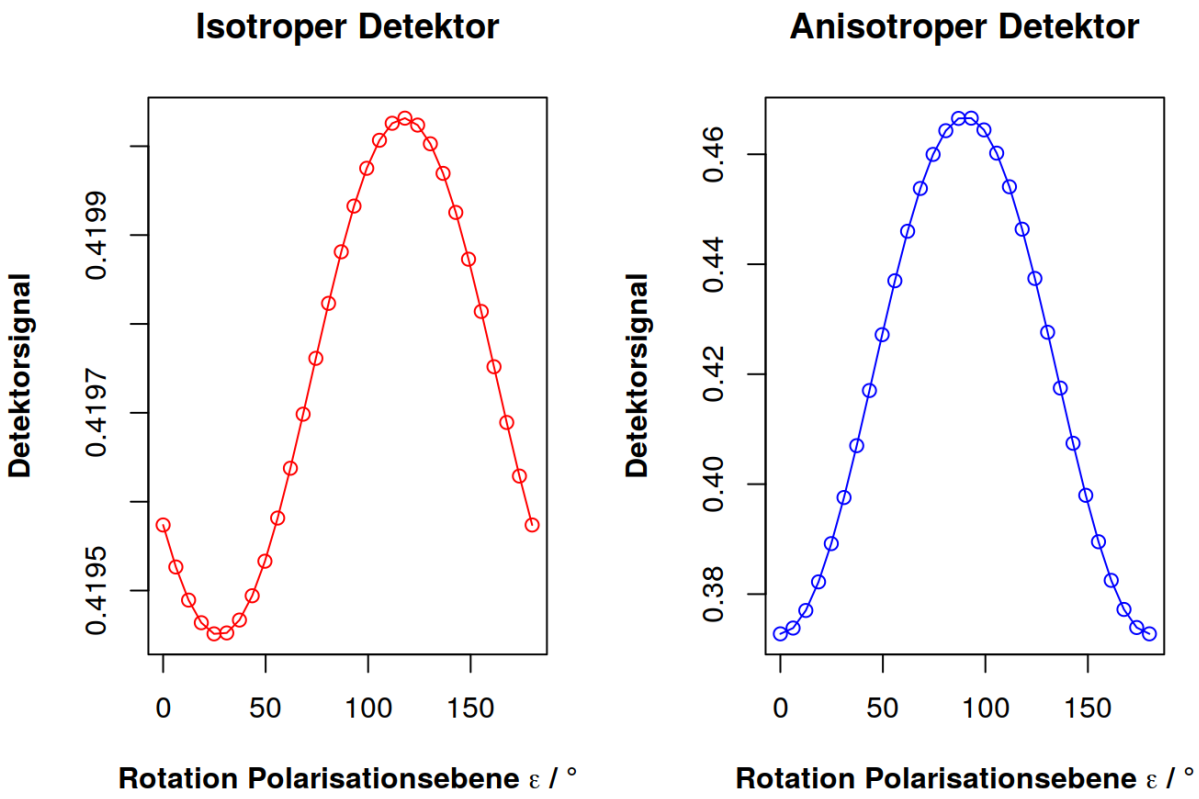
Variation der Polarisationsebene

- Es werden 30 total linearpolarisierte Stokesvektoren generiert
- Die Vektoren sind uniform auf dem Einheitskreis verteilt
- Die Müllermatrix M wird auf die Stokesvektoren angewendet
- Aus den resultierenden Stokesvektoren werden die Intensität entlang der x- und y-Achse berechnet
- Das Detektorsignal wird als Summe der Intensitäten entlang der x- und y-Achse berechnet
- Das Detektorsignal wird für gleichgewichtete Summanden und für ungleichgewichteten Summanden berechnet ($I_x : I_y = 1 : 2$)

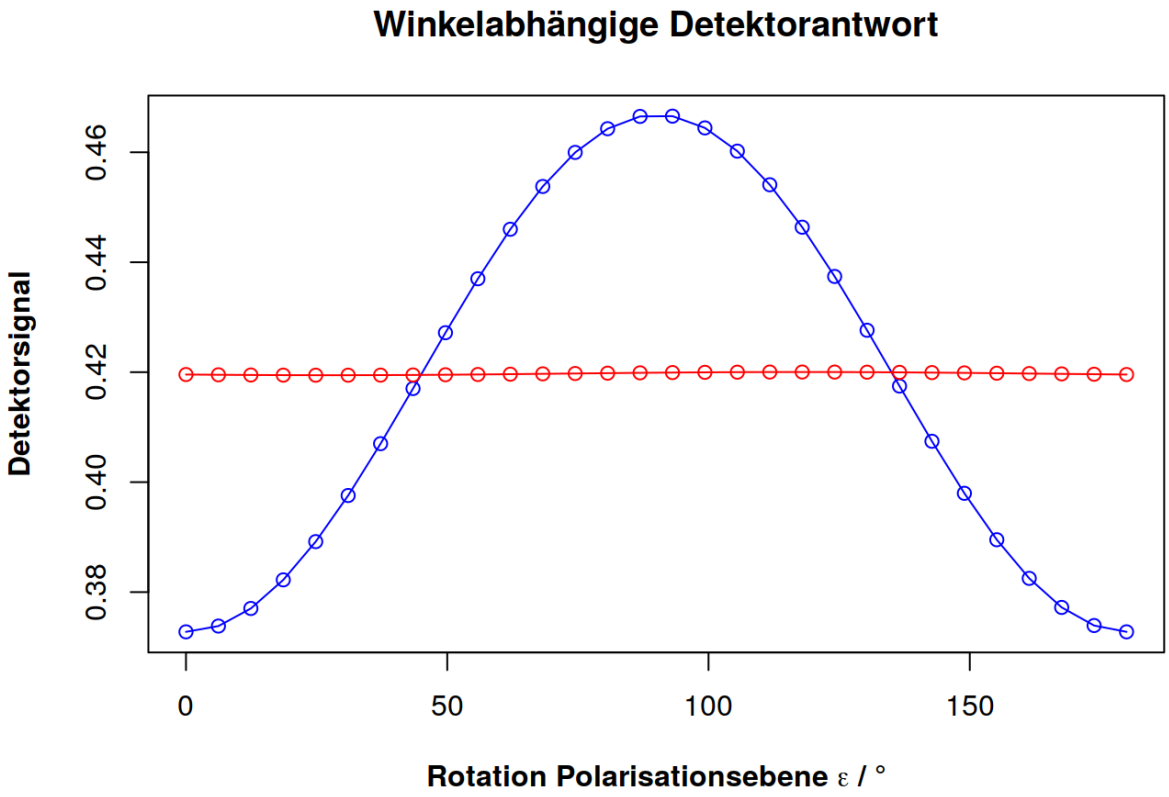
```
#
# CREATE STOKES VECTORS
#
# Create a matrix with totaly polarised stokes vectors
# Every stokes vector describes a different plane of polarisation
# All vectors combined describe a whole circle and therefore every possible linear polaris
ation
polaramTest.angle <- seq(from=0, to=2*pi, length.out=30)
polaramTest.laserAngle <- sapply( polaramTest.angle, function(angle) {
  # Totally linear polarised stokes vector
  c(1,
    cos(angle),
    sin(angle),
    0)
  }) %>%
  # Translate vectors into a matrix
  matrix(., ncol = 4, byrow = T)

#
# RUN POLARAM
#
# Construct command line call for PolaRam
polaramTest.simulate <- function(stokesvec, muellermatrix=polaramTest.matrixFile) {
  # If polaram simulate fails, it won't generate any output
  # If this file is not removed, an exception in simulate may raise
  # no error and will stay undetected, because the results of an old
  # simulation will be readble from that file
  file.remove(polaramTest.outputFile)
  # Create cli call
  polaramTest.cli <- paste(c(
      # Call the programm 'polaram simulate'
      Sys.getenv("POLARAM"), "simulate",
      # Pass instruction file
      polaramTest.instructionFile,
      # Pass output file
      paste("--output", polaramTest.outputFile),
      # Pass matrix file
      paste("--matrix", muellermatrix),
      # Pass laser polarisations
      # The formatting is important, because PolaRam has a bug:
      # PolaRam can't handle negative numbers in scientific notation
      apply(stokesvec, 1, function(vec) paste(c("--laser", format(vec, scientific=F)), col
lapse = " ")),
      # Pass some arguments for formatting and other behaviour
      "--unpolarised-scattering", "--verbose",
      "--raw-output", "--silent"
    ), collapse = " ")
  # Call PolaRam
  system(polaramTest.cli)
}
# Run PolaRam
polaramTest.simulate(polaramTest.laserAngle)
```

```
## Warning in file.remove(polaramTest.outputFile): kann Datei './
## test_simulationResult.txt' nicht löschen. Grund 'Datei oder Verzeichnis nicht
## gefunden'
```

```
#
# READ AND FORMAT SIMULATION RESULTS
#
# Read results
polaramTest.rotationSim <- read.table(file = polaramTest.outputFile, comment.char = "#")[,
c(1,6:9)]
# Replace descriptive first column by the initial angle of the plane of polarisation
polaramTest.rotationSim[,1] <- polaramTest.angle
# Give the columns descriptive names
colnames(polaramTest.rotationSim) <- c("sigma", "S0", "S1", "S2", "S3")

#
# COMPUTE DETECTOR RESPONSE
#
polaramTest.detectorResponse <- function(stokes, bias) {
  scaleX <- 1    / (1+bias)
  scaleY <- bias / (1+bias)
  scaleX*(stokes$S0 + stokes$S1)/2 + scaleY*(stokes$S0 - stokes$S1)/2
}
polaramTest.rotationSim$fairResponse <- polaramTest.detectorResponse(polaramTest.rotationS
im, bias=1)
polaramTest.rotationSim$biasResponse <- polaramTest.detectorResponse(polaramTest.rotationS
im, bias=2)

#
# PLOTS
#
# Plot fair and biased detector response in two plots
# next to each other
par(mfrow=c(1,2))
# Fair Response
plot( x=polaramTest.rotationSim$sigma/pi*180/2,
      y=polaramTest.rotationSim$fairResponse,
      main="Isotroper Detektor",
      xlab=expression(bold("Rotation Polarisationsebene "*epsilon*" / °")),
      ylab=expression(bold("Detektorsignal")),
      type="o", col="red")
# Biased Response
plot( x=polaramTest.rotationSim$sigma/pi*180/2,
      y=polaramTest.rotationSim$biasResponse,
      main="Anisotroper Detektor",
      xlab=expression(bold("Rotation Polarisationsebene "*epsilon*" / °")),
      ylab=expression(bold("Detektorsignal")),
      type="o", col="blue")
```




```
# Plot Fair and Biased Response together
par(mfrow=c(1,1))
plot( x=polaramTest.rotationSim$sigma/pi*180/2,
      y=polaramTest.rotationSim$biasResponse,
      main="Winkelabhängige Detektorantwort",
      xlab=expression(bold("Rotation Polarisationsebene "*epsilon*" / °")),
      ylab=expression(bold("Detektorsignal")),
      type="o", col="blue")
lines( x=polaramTest.rotationSim$sigma/pi*180/2,
       y=polaramTest.rotationSim$fairResponse,
       type="o", col="red")
legend( x=290, y=0.46, fill=c("red", "blue"), legend=c("isotrop", "anisotrop"))
```



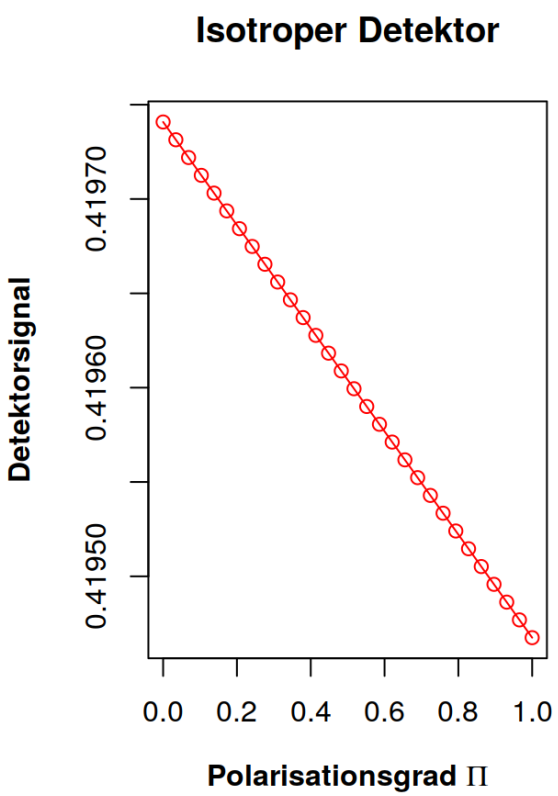
- Es wurde eine konstante Detektorantwort für den isotropen Detektor erwartet
- Der Fehler ist sehr klein ($0.4197353 \pm 2.0784698 \times 10^{-4}$)
- Der Fehler ist systematisch
- Wird die Müllermatrix durch eine weitere Monte-Carlo-Simulation neu berechnet, zeigt sich ein anderer aber ähnlicher Fehler (siehe letzter letzter Abschnitt)
- Die Monte-Carlo-Simulation muss anscheinend länger iterieren
- Bisher wurden die Müllermatrizen mit 1.000.000 Iterationen berechnet
- Der anisotrope Detektor zeigt den erwarteten Verlauf
- Das Detektorsignal wird minimal, wenn die Polarisationsebene entlang der benachteiligten Achse orientiert ist (0°/180°)
- Das Detektorsignal wird maximal, wenn die Polarisationsebene entlang der bevorzugten Achse orientiert ist (90°)

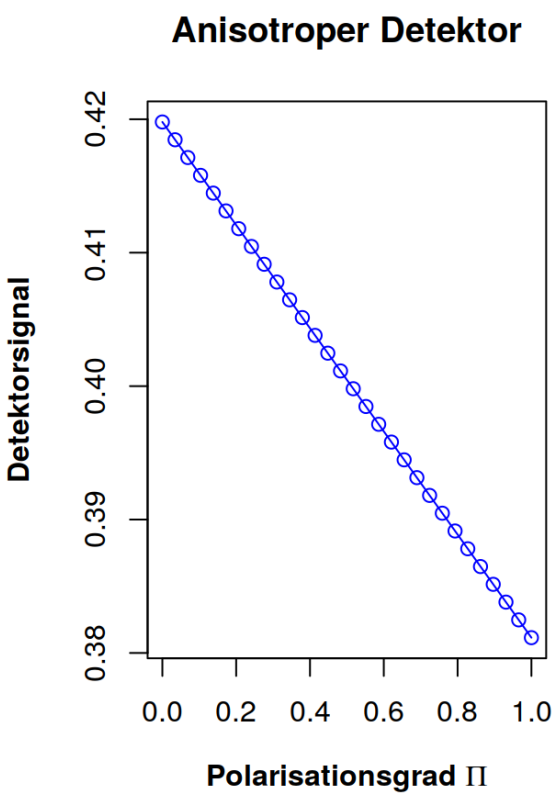
Variation des Polarisationsgrades

- Es werden 30 linearpolarisierte Stokesvektoren mit unterschiedlichem Polarisationsgrad generiert
- Der polare Stokesparameter σ aller Vektoren ist $\sigma = 35°$
- Die Müllermatrix M wird auf die Stokesvektoren angewendet
- Aus den resultierenden Stokesvektoren werden die Intensität entlang der x- und y-Achse berechnet
- Das Detektorsignal wird als Summe der Intensitäten entlang der x- und y-Achse berechnet
- Das Detektorsignal wird für gleichgewichtete Summanden und für ungleichgewichteten Summanden berechnet ($I_x : I_y = 1 : 2$)

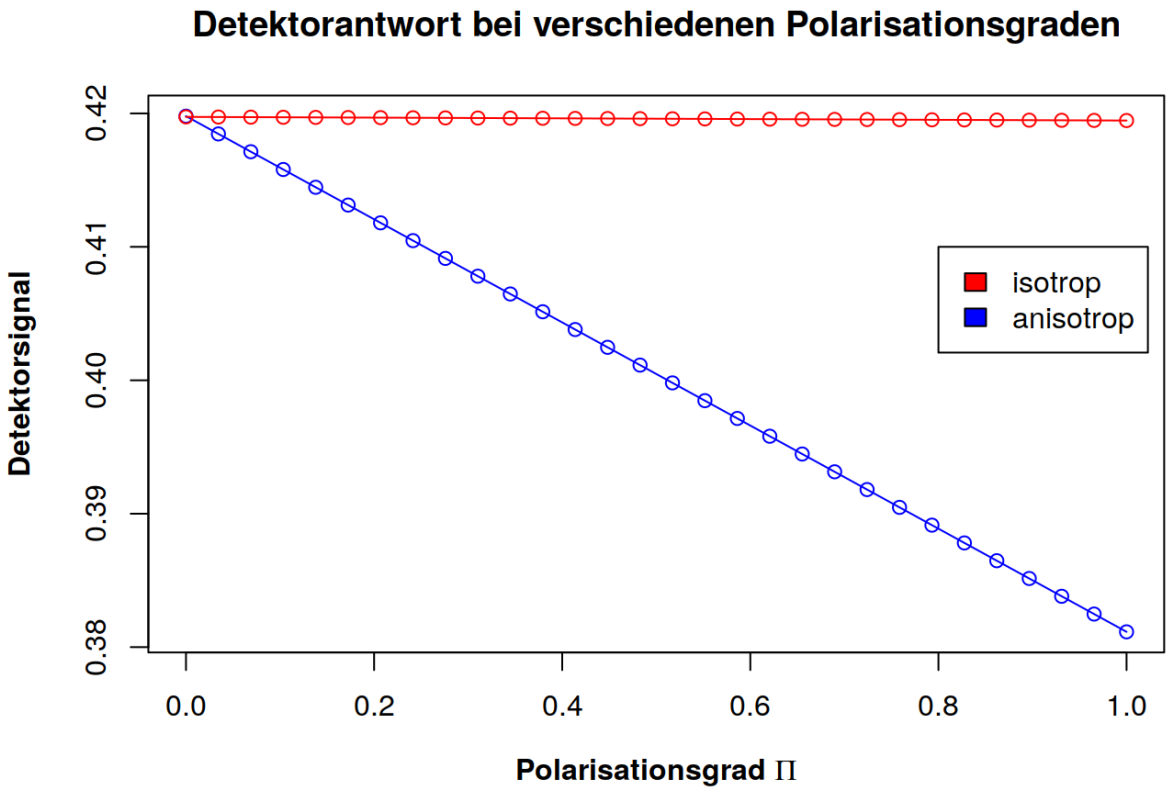
```
#
# CREATE STOKES VECTORS
#
# Create a matrix with totaly polarised stokes vectors
# Every stokes vector describes the same plane of polarisation,
# but each one of them has a different degree of polarisation
polaramTest.degree <- seq(from=0, to=1, length.out=30)
polaramTest.laserDegree <- sapply(polaramTest.degree, function(Pi) {
  angle <- 35 *pi/180
  c(1,
    Pi*cos(angle),
    Pi*sin(angle),
    0)
  }) %>%
  # Translate vectors into a matrix
  matrix(., ncol = 4, byrow = T)

#
# RUN POLARAM
#
polaramTest.simulate(polaramTest.laserDegree)
#
# READ AND FORMAT SIMULATION RESULTS
#
# Read results
polaramTest.polarisationSim <- read.table(file = polaramTest.outputFile, comment.char = "
#")[,c(1,6:9)]
# Replace descriptive first column by the initial degree of polarisation
polaramTest.polarisationSim[,1] <- polaramTest.degree
# Give the columns descriptive names
colnames(polaramTest.polarisationSim) <- c("Pi", "S0", "S1", "S2", "S3")

#
# COMPUTE DETECTOR RESPONSE
#
polaramTest.polarisationSim$fairResponse <- polaramTest.detectorResponse(polaramTest.polar
isationSim, bias=1)
polaramTest.polarisationSim$biasResponse <- polaramTest.detectorResponse(polaramTest.polar
isationSim, bias=2)

#
# PLOTS
#
# Plot fair and biased detector response in two plots
# next to each other
par(mfrow=c(1,2))
# Fair Response
plot( x=polaramTest.polarisationSim$Pi,
      y=polaramTest.polarisationSim$fairResponse,
      main="Isotroper Detektor",
      xlab=expression(bold("Polarisationsgrad "*Pi)),
      ylab=expression(bold("Detektorsignal")),
      type="o", col="red")
# Biased Response
plot( x=polaramTest.polarisationSim$Pi,
      y=polaramTest.polarisationSim$biasResponse,
      main="Anisotroper Detektor",
      xlab=expression(bold("Polarisationsgrad "*Pi)),
      ylab=expression(bold("Detektorsignal")),
```

```
      type="o", col="blue")
```






```
# Plot Fair and Biased Response together
par(mfrow=c(1,1))
plot( x=polaramTest.polarisationSim$Pi,
      y=polaramTest.polarisationSim$biasResponse,
      main="Detektorantwort bei verschiedenen Polarisationsgraden",
      xlab=expression(bold("Polarisationsgrad "*Pi)),
      ylab=expression(bold("Detektorsignal")),
      type="o", col="blue")
lines( x=polaramTest.polarisationSim$Pi,
       y=polaramTest.polarisationSim$fairResponse,
       type="o", col="red")
legend( x=0.8, y=0.41, fill=c("red", "blue"), legend=c("isotrop", "anisotrop"))
```



- Es wurde eine konstante Detektorantwort für den isotropen Detektor erwartet
- Der Fehler ist sehr klein ($0.4196042 \pm 8.2977116 \times 10^{-5}$)
- Der Fehler ist systematisch
- Wird die Müllermatrix durch eine weitere Monte-Carlo-Simulation neu berechnet, zeigt sich ein anderer aber ähnlicher Fehler (siehe letzter letzter Abschnitt)
- Die Monte-Carlo-Simulation muss anscheinend länger iterieren
- Bisher wurden die Müllermatrizen mit 1.000.000 Iterationen berechnet
- Der anisotrope Detektor zeigt den erwarteten Verlauf
- Die Polarisationsebene liegt mit einem Rotationswinkel von $\epsilon = 17.5°$ nahe bei der benachteiligten Achse (x-Achse)
- Damit wird die Detektorantwort für totalpolarisiertes Licht minimal

Fehler der Monte-Carlo-Simulation

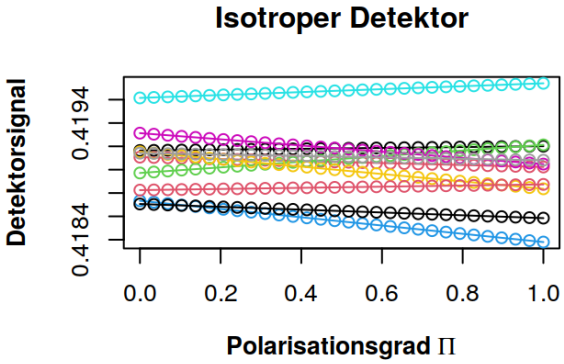
- Die Berechnungen für den isotropen und anisotropen Detektor werden wiederholt
- Jede Neuberechnung der Detektorantwort erfolgt mit einer neu berechneten Müllermatrix
- Die Müllermatrix wird mit 1.000.000 Iterationen aus dem Ramantensor α gerneriert
- Es werden 10 verschiedene Müllermatrizen generiert

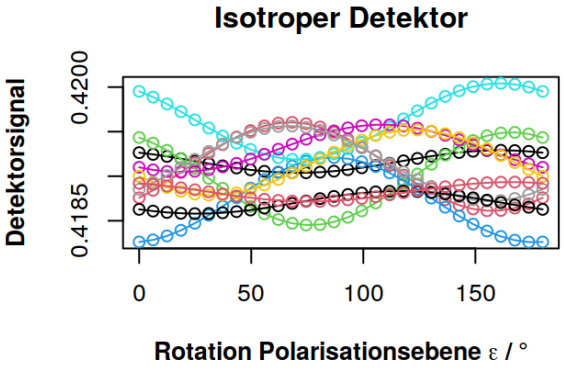
```
#
# HOW DOES THE ISOTROPIC DETECTOR RESPONSE CHANGE
# when repeating the Monte-Carlo-Simulation?
#
# Define CLI call to run 'polaram convert'
polaramTest.convert <- function(iterations=1e6, threshold=2) {
  # Delete previous output file
  # If polaram convert fails, it won't generate any output
  # If this file is not removed, an exception in convert may raise
  # no error and will stay undetected, because the results
  # of an old simulation can be read from the file
  file.remove(polaramTest.convert.outputFile)
  # Create cli call
  polaramTest.cli <- paste(c(
      # Call the programm 'polaram convert'
      Sys.getenv("POLARAM"), "convert",
      # Path to raman tensor
      polaramTest.tensorFile,
      # Path to output file
      paste("--output", polaramTest.convert.outputFile),
      # How many iterations should be done?
      paste("--iterations", format(iterations, scientific=F)),
      # How picky should the valdidation process be?
      paste("--threshold", format(threshold, scientific=F)),
      # Show warnings
      "--verbose"
    ), collapse = " ")
  # Call PolaRam
  system(polaramTest.cli)
}

#
# RUN MONTE-CARLO AND MUELLER-SIMULATION
#
polaramTest.repetition <- 10
# Run simulations and compute detector responses
polaramTest.monteCarloTest <- replicate(polaramTest.repetition, {
  #
  # RUN the Monte-Carlo
  #
  polaramTest.convert(iterations=1e6)
  #
  # RUN mueller-simulation with variing angle of polarisation
  #
  polaramTest.simulate(stokesvec = polaramTest.laserAngle,
                       muellermatrix = polaramTest.convert.outputFile)
  # Read results
  angleSim <- read.table(file = polaramTest.outputFile, comment.char = "#")[,c(1,6:9)]
  # Replace descriptive first column by the initial angle of the plane of polarisation
  angleSim[,1] <- polaramTest.angle
  # Give the columns descriptive names
  colnames(angleSim) <- c("sigma", "S0", "S1", "S2", "S3")
  # Compute isotropic detector response
  angleSim$angleFairResponse <- polaramTest.detectorResponse(angleSim, bias=1)
  angleSim$angleBiasResponse <- polaramTest.detectorResponse(angleSim, bias=2)
  #
  # RUN mueller-simulation with variing degree of polarisation
  #
  polaramTest.simulate(stokesvec = polaramTest.laserDegree,
```

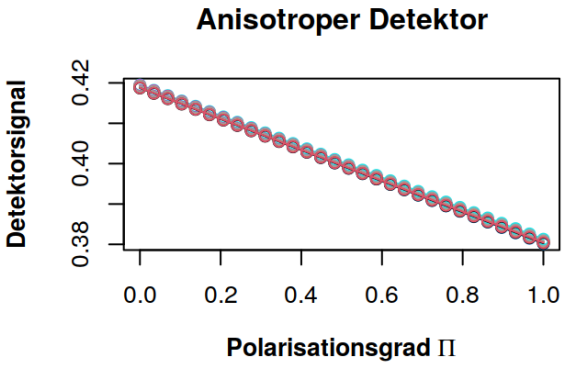
```
                      muellermatrix = polaramTest.convert.outputFile)
  # Read results
  degreeSim <- read.table(file = polaramTest.outputFile, comment.char = "#")[,c(1,6:9)]
  # Replace descriptive first column by the initial angle of the plane of polarisation
  degreeSim[,1] <- polaramTest.degree
  # Give the columns descriptive names
  colnames(degreeSim) <- c("Pi", "S0", "S1", "S2", "S3")
  # Compute isotropic detector response
  degreeSim$degreeFairResponse <- polaramTest.detectorResponse(degreeSim, bias=1)
  degreeSim$degreeBiasResponse <- polaramTest.detectorResponse(degreeSim, bias=2)

  #
  # RETURN detector responses
  #
  return( data.frame(angleSim[, c("sigma", "angleFairResponse", "angleBiasResponse")],
                     degreeSim[, c("Pi", "degreeFairResponse", "degreeBiasResponse")]) )
})
```

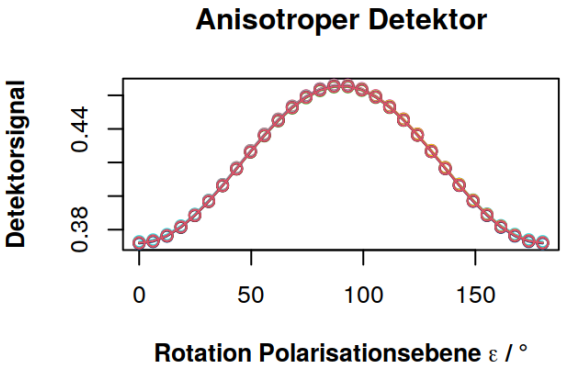
```
#
# REORGANISE data
#
# Extract simulation results for variing angle of polarisation
polaramTest.rotationFairDeviation <- matrix(ncol=polaramTest.repetition+1, byrow=F,
                                            data=c(
                                              polaramTest.monteCarloTest[,1]$sigma,
                                              apply(polaramTest.monteCarloTest, 2, function(da
ta) {
                                                data$angleFairResponse
                                              })
                                            )) %>% as.data.frame()
polaramTest.rotationBiasDeviation <- matrix(ncol=polaramTest.repetition+1, byrow=F,
                                            data=c(
                                              polaramTest.monteCarloTest[,1]$sigma,
                                              apply(polaramTest.monteCarloTest, 2, function(da
ta) {
                                                data$angleBiasResponse
                                              })
                                            )) %>% as.data.frame()
# Add descriptive column names
colnames(polaramTest.rotationFairDeviation) <- c("sigma",
                                                 sapply(1:polaramTest.repetition, function(i)
{
                                                   paste0("fairResponse", i)
                                                 }))
colnames(polaramTest.rotationBiasDeviation) <- c("sigma",
                                                 sapply(1:polaramTest.repetition, function(i)
{
                                                   paste0("biasResponse", i)
                                                 }))
# Extract simulation results for variing degree of polarisation
polaramTest.polarisationFairDeviation <- matrix(ncol=polaramTest.repetition+1, byrow=F,
                                            data=c(
                                              polaramTest.monteCarloTest[,1]$Pi,
                                              apply(polaramTest.monteCarloTest, 2, function(da
ta) {
                                                data$degreeFairResponse
                                              })
                                            )) %>% as.data.frame()
polaramTest.polarisationBiasDeviation <- matrix(ncol=polaramTest.repetition+1, byrow=F,
                                            data=c(
                                              polaramTest.monteCarloTest[,1]$Pi,
                                              apply(polaramTest.monteCarloTest, 2, function(da
ta) {
                                                data$degreeBiasResponse
                                              })
                                            )) %>% as.data.frame()
# Add descriptive column names
colnames(polaramTest.polarisationFairDeviation) <- c("Pi",
                                                 sapply(1:polaramTest.repetition, function(i)
{
                                                   paste0("fairResponse", i)
                                                 }))
colnames(polaramTest.polarisationBiasDeviation) <- c("Pi",
                                                 sapply(1:polaramTest.repetition, function(i)
{
                                                   paste0("biasResponse", i)
                                                 }))
```

```
#
# PLOT RESULTS
#
# Plot angle and degree of polarisation in two plots
# next to each other
par(mfrow=c(2,2))
# Degree of polarisation ISOTROPIC
plot( x=polaramTest.polarisationFairDeviation$Pi,
      y=polaramTest.polarisationFairDeviation$fairResponse1,
      main="Isotroper Detektor",
      xlab=expression(bold("Polarisationsgrad "*Pi)),
      ylab=expression(bold("Detektorsignal")),
      type="n", ylim=c( min(polaramTest.polarisationFairDeviation[,-1]),
                        max(polaramTest.polarisationFairDeviation[,-1]) ))
for(i in 1:polaramTest.repetition) lines(x=polaramTest.polarisationFairDeviation$Pi,
                                         y=polaramTest.polarisationFairDeviation[,i+1],
                                         col=i, type="o")
# Angle of polarisation ISOTROPIC
plot( x=polaramTest.rotationFairDeviation$sigma*180/pi/2,
      y=polaramTest.rotationFairDeviation$fairResponse1,
      main="Isotroper Detektor",
      xlab=expression(bold("Rotation Polarisationsebene "*epsilon*" / °")),
      ylab=expression(bold("Detektorsignal")),
      type="n", ylim=c( min(polaramTest.rotationFairDeviation[,-1]),
                        max(polaramTest.rotationFairDeviation[,-1]) ))
for(i in 1:polaramTest.repetition) lines(x=polaramTest.rotationFairDeviation$sigma*180/pi/
2,
                                         y=polaramTest.rotationFairDeviation[,i+1],
                                         col=i, type="o")
# Degree of polarisation ANISOTROPIC
plot( x=polaramTest.polarisationBiasDeviation$Pi,
      y=polaramTest.polarisationBiasDeviation$biasResponse1,
      main="Anisotroper Detektor",
      xlab=expression(bold("Polarisationsgrad "*Pi)),
      ylab=expression(bold("Detektorsignal")),
      type="n", ylim=c( min(polaramTest.polarisationBiasDeviation[,-1]),
                        max(polaramTest.polarisationBiasDeviation[,-1]) ))
for(i in 1:polaramTest.repetition) lines(x=polaramTest.polarisationBiasDeviation$Pi,
                                         y=polaramTest.polarisationBiasDeviation[,i+1],
                                         col=i, type="o")
# Angle of polarisation ANISOTROPIC
plot( x=polaramTest.rotationBiasDeviation$sigma*180/pi/2,
      y=polaramTest.rotationBiasDeviation$biasResponse1,
      main="Anisotroper Detektor",
      xlab=expression(bold("Rotation Polarisationsebene "*epsilon*" / °")),
      ylab=expression(bold("Detektorsignal")),
      type="n", ylim=c( min(polaramTest.rotationBiasDeviation[,-1]),
                        max(polaramTest.rotationBiasDeviation[,-1]) ))
for(i in 1:polaramTest.repetition) lines(x=polaramTest.rotationBiasDeviation$sigma*180/pi/
2,
                                         y=polaramTest.rotationBiasDeviation[,i+1],
                                         col=i, type="o")
```









- Wie erwartet ergeben alle Müllermatrizen für den anisotropen Detektor ähnliche Werte
- Wie erwartet resultieren verschiedene Müllermatrizen für den isotropen Detektor in verschiedene Verläufe
- Mittlere Standardabweichung des Detektorsignals für den anisotropen Detektor, wenn der Winkel der Polarisationsebene variiert: 2.8480299×10^{-4}
- Mittlere Standardabweichung des Detektorsingals für den anisotropen Detektor, wenn der Polarisationsgrad variiert: 3.6705849×10^{-4}