Arrow Grouping Indicator AGI

Jochen Hertle

July 26, 2025

1 Scatter ellipse

The concept of a *confidence interval* for the mean of a variable can be extended to two dimensions. This is then called a scatter (error) ellipse[1]

The main axis of the scatter ellipse points into direction of the largest dispersion of the combined data. The minor axis will show the direction of the smallest dispersion. The lengths of the axis are analogue to the widths of the confidence intervals and will be expressed in a multiple of the standard deviation of the data in the respective directions.

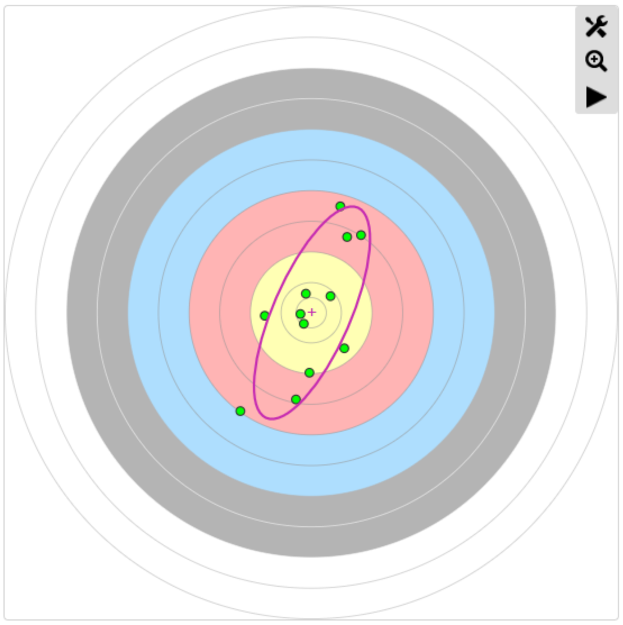

Figure 1: scatter ellipse

2 AGI

The goal of the **Arrow Grouping Indicator (AGI)** is to provide a single number representing the dispersion of arrow positions measured by tuples (x_i, y_i) of coordinates.

The main idea is to take the standard deviations in main and minor direction as calculated for the scatter ellipse and combine them with a procedure called the *pooled variance*. Then, this pooled variance is divided by the shooting distance which makes it a variance of the angle dispersion. The angle dispersion is therefore independent of the shooting distance and allows comparison of results between different shooting distances.

[1]see e.g.`https://en.wikipedia.org/wiki/Confidence_region`, or, `https://scholarship.depauw.edu/cgi/viewcontent.cgi?article=1010&context=math_facpubs`

The final step is to linearly transform the angle dispersion onto the interval $[0, 100]$ to make it easier to memorize. 100 will represent a set of arrows with zero dispersion, i.e., all having identical coordinates and hitting the same point on the target.

3 Formulae

3.1 Calculation of the scatter ellipse

Let $[(x_1, y_1), \dots (x_n, y_n)]$ be a sequence of coordinates of n arrows.

The standard statistical numbers of the coordinates are given by

$$\bar{x} = \frac{1}{n}\sum_i^n x_i \qquad \text{mean of x} \tag{1}$$

$$\bar{y} = \frac{1}{n}\sum_i^n y_i \qquad \text{mean of y} \tag{2}$$

$$s_x^2 = \frac{1}{n-1}\sum_i^n (x_i - \bar{x})^2 \qquad \text{variance of x} \tag{3}$$

$$s_y^2 = \frac{1}{n-1}\sum_i^n (y_i - \bar{y})^2 \qquad \text{variance of y} \tag{4}$$

$$c_{xy} = \frac{1}{n-1}\sum_i^n (x_i - \bar{x})(y_i - \bar{y}) \qquad \text{covariance of } xy \tag{5}$$

The *covariance matrix* is given by

$$C = \begin{pmatrix} s_x^2 & c_{xy} \\ c_{xy} & s_y^2 \end{pmatrix} \tag{6}$$

The main and minor directions of the 2-dimensional dispersion are given by the eigenvectors e_M, e_m of the covariance matrix. The standard deviation in each direction are the square roots of the corresponding eigenvalues.

For a 2x2 covariance matrix, these eigenvalues can be explicitly calculated:

$$det(C) = (s_x^2 + s_y^2)^2 - 4(s_x^2 * s_y^2 - c_{xy}^2) \qquad \text{determinant of } C \tag{7}$$

$$\lambda_M = \frac{1}{2}(s_x^2 + s_y^2 + \sqrt{det(C)}) \qquad \text{main eigenvalue} \tag{8}$$

$$\lambda_m = \frac{1}{2}(s_x^2 + s_y^2 - \sqrt{det(C)}) \qquad \text{minor eigenvalue} \tag{9}$$

$$s_M = \sqrt{\lambda_M} \qquad \text{sd main direction} \tag{10}$$

$$s_m = \sqrt{\lambda_m} \qquad \text{sd minor direction} \tag{11}$$

The measurement unit of the standard deviations s_M, s_m is the same as for the given coordinates x_i, y_i (e.g., in mm).

Finally, the rotation angle of the main axis of the ellipse is given by

$$\alpha_{rad} = \text{atan2}(\lambda_M - s_x^2,\ c_{xy}) \qquad \text{in rad} \tag{12}$$

$$\alpha = \frac{180}{\pi}\alpha_{rad} \qquad \text{in degrees} \tag{13}$$

In order to get a scatter ellipse that contains about 90% of all hits, the length of the axis are the standard deviations multiplied by a factor 1.645 (z-value for confidence level 0.9).

In summary, a 90% scatter ellipse is defined by these numbers

$$(cx, cy) := (\bar{x}, \bar{y}) \qquad \text{center of ellipse = center of arrow group} \tag{14}$$

$$rx := 1.645 \cdot s_M \qquad \text{length of main axis} \tag{15}$$

$$ry := 1.645 \cdot s_m \qquad \text{length of minor axis} \tag{16}$$

$$a := \alpha \qquad \text{rotation angle of ellipse} \tag{17}$$

Using these numbers, the svg element to draw the 90% scatter ellipse is given by

```
<ellipse cx="cx" cy="cy" rx="rx" ry="ry"
   transform="rotate(a, cx, cy)"></ellipse>
```

3.2 Calculation of the AGI

The *pooled standard deviation* of both standard deviations s_M, s_m is given by

$$psd = \sqrt{\frac{n-1}{2n}(s_M^2 + s_m^2)} \qquad \text{pooled standard deviation} \tag{18}$$

Dividing the *psd* by the shooting distance d, it becomes a pooled standard deviation for the angle dispersion

$$asd = psd/d \qquad \text{pooled standard deviation of angle dispersion} \tag{19}$$

Note that the shooting distance d must have the same measurement dimension as the arrow coordinates x_i, y_i to make it work (e.g., in mm). Then, the *asd* is a pure number without dimension and represents an angle in rad.

Let $\alpha_{max} = 1.22/70$ be the maximal grouping angle in rad. This is the maximum angle between two arrows on a 122 cm target at 70 m shooting distance, as seen from the shooting line[2].

The linear transformation to map the *asd* onto the interval $[0, 100]$ is called the *Arrow Grouping Indicator (AGI)* and is given by

$$AGI = 100 \cdot \frac{\alpha_{max} - asd}{\alpha_{max}} \qquad \text{Arrow Grouping Indicator} \tag{20}$$

The *AGI* has got the following properties:

- it is independent of the shooting distance and represents the angle dispersion of a group of arrows on the interval $[0, 100]$
- if the angle dispersion is zero ($asd = 0$), then $AGI = 100$
- if the angle dispersion is maximal ($asd = \alpha_{max}$), then $AGI = 0$
- recurve archers on the international competition level achieve $AGI > 95$ for all arrows of a competition
- recurve archers in indoor competitions can even achieve $AGI > 98$ during a (short) sequence of ends

[2]the angle is small enough to justify the approximation $\operatorname{atan2}(1.22, 70) \approx 1.22/70$ (≈ 1 deg)

3.3 JavaScript listings

```
function scatterEllipse(hits) {
    // calculates the parameters of the scatter ellipse
    // hits is a list of arrays: [[x1, y1], ..., [xn, yn]]

    let sumX = 0, sumY = 0, sumXX = 0, sumYY = 0, sumXY = 0;

    let n = hits.length;
    if (n <= 1){
        // no calculation possible
        return undefined;
    }

    // optimized calculation of hit statistics
    let x, y;
    for (let hit of hits) {
        x = hit[0];
        y = hit[1];
        sumX += x;
        sumXX += x * x;
        sumY += y;
        sumYY += y * y;
        sumXY += x * y
    }
    let meanX = 1 / n * sumX;
    let meanY = 1 / n * sumY;
    let varX = 1 / (n - 1) * sumXX - n / (n - 1) * meanX ** 2;
    let varY = 1 / (n - 1) * sumYY - n / (n - 1) * meanY ** 2;
    let covXY = 1 / (n - 1) * sumXY - n / (n - 1) * meanX * meanY;

    // calculation of eigenvalues of covariance matrix
    let lambda1, lambda2, r1, r2;
    let det = (varX + varY) ** 2 - 4 * (varX * varY - covXY ** 2);

    if (det <= 0){
        // no eigenvalues, return just hit statistics
        return {
            'cx': Math.round(meanX),
            'cy': Math.round(meanY),
            'rx': varX === 0 ? 0 : undefined,
            'ry': varY === 0 ? 0 : undefined,
            'angle': 0,
            'sdx': Math.round(Math.sqrt(varX)),
            'sdy': Math.round(Math.sqrt(varY))
        };
    }

    lambda1 = 0.5 * (varX + varY + Math.sqrt(det));
    lambda2 = 0.5 * (varX + varY - Math.sqrt(det));

    if (lambda1 >= 0) {
        r1 = Math.sqrt(lambda1);
    } else {
        return undefined;
    }

    if (lambda2 >= 0) {
        r2 = Math.sqrt(lambda2);
    } else {
        return undefined;
    }

    // rotation angle of main axis of ellipse in degrees
    let angle = Math.atan2((lambda1 - varX), covXY) * 180 / Math.PI;

    return {
        'cx': Math.round(meanX),
        'cy': Math.round(meanY),
        'rx': Math.round(1.645 * r1), // confidence level = 90%
        'ry': Math.round(1.645 * r2),
        'angle': Math.round(angle),
        'sdx': Math.round(Math.sqrt(varX)),
        'sdy': Math.round(Math.sqrt(varY))
    };
}
```

```
function agi(hits, dist) {
    // calculates the AGI for a given set of hits and shooting distance dist
    // hits is a list of coorinates [xi, yi], dist has the same units as the hits (e.g., mm)

    let n = hits.length;
    if (n < 3) return undefined; // AGI requires >=3 hits

    let ellipse = scatterEllipse(hits);
    if (ellipse == undefined || ellipse.rx == undefined || ellipse.ry == undefined) return
        undefined;

    let pooledStandardDeviation = Math.sqrt((n - 1) / (2 * n) * ((ellipse.rx / 1.645) ** 2
        + (ellipse.ry / 1.645) ** 2));
    // pooled standard deviation for angle dispersion
    let groupingAngle = pooledStandardDeviation / dist;

    // linear transformation to [0, 100]
    let maxGroupingAngle = 1.22 / 70;
    return Math.max((maxGroupingAngle - groupingAngle) / maxGroupingAngle * 100, 0.0);
}
```

3.4 Python listings

```python
import numpy as np
from typing import Optional


def calculate_agi(values: np.array, distance: int) ->Optional[float]:
    """ calculates the AGI from a given numpy array of hit coordinates
    :param values: np.array [[cx-values], [cy values]], in mm
    :param distance: int (in m)
    :returns agi: float
    """

    n = values.shape[1] # number of observations
    if n <3:
        # agi calculation requires at least 3 hits
        print(f"Not enough values to calculate AGI: n={n}")
        return None

    # covariance matrix
    cov_xy =np.cov(values)

    # get 2 positive eigenvalues = variances along principal axes
    eigen =np.linalg.eig(cov_xy)
    try:
        r1, r2 =np.sqrt(eigen[0]) # standard deviations along principal axes (in mm)
    except ValueError:
        print(f"no positive eigenvalues: {eigen[0]}")
        return None

    # pooled standard deviation
    pooled_sd =np.sqrt((n -1) /(2 * n) * (r1 **2 + r2 **2))  # in mm

    # as angle dispersion (using tan(alpha) = alpha for small alpha)
    pooled_sd_angle =pooled_sd /1000 /distance

    # normalization of AGI=0 <=> max scattering on a 122 target @ 70m
    max_grouping_angle =1.22 /70

    return np.max((max_grouping_angle -pooled_sd_angle) /max_grouping_angle *100, 0)
```

```python
# example usage
x = [10, 20, 0, 60, 20] # x-coordinates of 4 hits (in mm)
y = [20, -20, 0, 10, 2] # y-coordinates of 4 hits (in mm)
hits =np.array([x, y])
d = 18  # shooting distance (in m)

agi =calculate_agi(hits, d)
if agi is None:
    print("error in input data")
else:
    print(f"agi = {agi:.2f}")
```