Locating Optimal Navigation Satellite Configurations

Markus Rippl

May 24, 2006

Contents

[1]GNSS: Global Navigation Satellite System

1 Abstract

This article describes the effort to find optimal stationary satellite constellations with respect to the position dependent function DOP *(Dilution of Precision)* for any given number of satellites. After investigating the possibilities to find an analytical solution, a simulation using the *Simulated Annealing* algorithm is implemented in MATLAB. While refining the algorithm, very simple arrangement rules become visible, which are further investigated using the implemented simulation method and various plots.

2 Introduction

2.1 Motivation

When the *Global Positioning System* (GPS) became fully operational in February 1995, navigation finally started to become an everyday application, available to anybody and usable at every spot on and around our planet. Nowadays, many areas fully depend on the availability of global navigation, be it personal travel, logistics, aviation, or even emergency services. Many more applications have recognized the rich set of possibilities available through GPS: Hikers use its pathfinding capabilities in unknown terrain, geologists and geodesists gain precise position information from the satellites above them; and even electronic toll systems on highways trust blindly on GPS' availability, integrity and accuracy. Soon, Europe will run its own and independent system for global navigation, the upcoming Galileo; and although the total number of *Global Satellite Navigation Systems* (GNSS) is unlikely to increase significantly after Galileo's start, research and development on the subject itself is still far from being completed.

With all those applications, the task of maximizing the robustness and quality of the delivered service becomes crucial for a systems' success. The service quality again is a function of several different factors; one of those factors is the satellites' constellation from the user's point of view. Said in short, the result will always be better if the satellites visible by the user are reasonably distributed in the sky. If all satellite signals arrive from roughly the same direction, positioning accuracy will be poor. Although this interrelation is probably self-explanatory to the reader, an explicit and simple term for describing the optimal constellation for a given number of visible satellites is not known yet.

The assignment addresses the issue of finding an optimal constellation of satellites regarding the best possible positioning accuracy with a given number of satellites.

2.2 Related work

Finding an optimal constellation can also be an important task when so-called pseudolite setups are used for precise positioning in geodesy or other application.

In [PM97], a setup inverse to the usual GPS-like arrangements is used to gain accurate position information about a plane carrying navigation facilities to be tested. In this research, a moving plane carries one transmitter *(pseudolite)*, while several receiver stations situated on the ground measure their pseudoranges to the plane simultaneously. The collected data is then processed to estimate the position of the plane. Again, the challenge was to optimize the positions of the receiving stations such that the achieved accuracy is maximized; but with only a restricted set of receiver positions available, the resulting set of equations is likely to perform much worse than in a regular GPS setup.

The paper proposes the condition of the geometry matrix as an alternative cost function; and the correlation between the matrix condition and DOP is investigated. This cost function will also be examined in this work.

Weighting the individual satellites' contribution when computing the position estimate is a way to prevent satellites with a known degradation of their signal quality from influencing the estimate too much. This approach is described in [WE95], where it is used for integrity monitoring. In the present work, the weighted position solution constitutes a basis for an alternative cost function, the weighted estimate accuracy.

2.3 Overview

In this thesis, optimal GNSS satellite constellation with respect to DOP are found. The optimization problem in general is explained in Section 3; furthermore, basic methods of position estimation using ranging signals as well as the derivation of DOP from the satellite positions is shown.

Section 4 covers the first part of the assignment, which is the effort to minimize the cost function by solving it analytically.

The subsequent simulation was implemented in a numerical software package. This approach is functionally explained in detail in Section 5, which covers the explanation of the simulation algorithms, and details about the implementation as well. Alternative cost functions are also discussed in this section.

In Section 6, the results of the simulation are presented and discussed.

As a closing thought, a conclusion on the present work can be found in Section 7.

3 Problem description

3.1 Measurement accuracy in GNSS[2]

The accuracy of a position estimate using ranging sources such as GPS or Galileo is determined by two major factors. One of those two factors are measurement errors in the involved systems, which can make the measured position deviate from the true position. These measurement errors typically consist of delays due to atmospheric effects (ionospheric delay, tropospheric delay), as well as multipath propagation, clock offsets and receiver noise. In the following, their combined contribution will be referred to as σ_{UERE} [3].

However, the impact of these measurement errors on the overall positioning accuracy is not constant. The second factor is determined by the satellite constellation, which plays an important role on how exact the user's position can be estimated. Depending on where the satellites are, the resulting positioning inaccuracy can theoretically be more than hundred times higher than the underlying measuring uncertainty. In reality, though, such a constellation occurs rarely because the satellite orbits are chosen carefully to prevent this.

As it will be demonstrated later on in Section 3.3, this factor impacting the overall accuracy is a numeric value called the Dilution of Precision, or DOP. The DOP can be obtained from the geometry matrix $\mathbf{H}$, which in turn contains the arguments of elevation and azimuth of all satellites used for the estimate.

Hence, a knowledge of both measuring error and DOP is essential for knowing how accurate the positioning is. The measuring error can be modeled through a statistical process in the receiver — its components are well known and examined — and the DOP can easily be gained in the process of position estimation. This is shown in the following sections.

3.2 Position estimation using ranging

This section gives a short overview about how positioning can be done with GPS. The comprehension of those basic formulae is needed to understand how the numeric value DOP is derived from the satellite positions. The following overview is adopted from [ME01], pages 176 et sqq.

The calculation of the user's position in GPS is done by measuring the distance to the available satellites; this procedure is called ranging. As stated before, different measurement errors are introduced in this process, but several of them can be corrected by means of error estimation or external data. The measured distance is called *pseudorange* to distinguish it from the real range to the satellite.

[2]GNSS: Global Navigation Satellite System

[3]UERE: User Equivalent Range Error

For the satellite k, the pseudorange can be written as

$$\rho^{(k)}(t) = r^{(k)}(t, t-\tau) + c[\delta t_u(t) - \delta t^{(k)}(t-\tau)] + I^{(k)}(t) + T^{(k)}(t) + \varepsilon_\rho^{(k)}(t) \quad (1)$$

where $r^{(k)}(t, t-\tau)$ is the real range to the satellite k; δt_u is the user clock offset, $\delta t^{(k)}$ is the k^{th} satellite's clock offset; and I, T and ε are errors superposed to the measurement which are introduced through ionospheric delay, tropospheric delay and residual unspecified effects, respectively.

For ionospheric propagation delay, a correction can be done either by dual frequency receivers — the delay is frequency dependent and thus can be calculated if two frequencies are used simultaneously. Tropospheric correction data is either estimated, or gained from DGPS[4] messages together with other correction data. The satellite's clock offset can be read from the satellite's navigation message. Thus, the corrected pseudorange can be denoted as

$$\rho_c^{(k)} = r^{(k)} + c \cdot \delta t_u + \tilde{\varepsilon}_\rho^{(k)}, \quad (2)$$

where residual errors are summarized in $\tilde{\varepsilon}_\rho^{(k)}$.

The range $r^{(k)}$ will now be represented by the vector difference between the satellite position and the user position; and the temporal deviation introduced through the user clock offset $c \cdot \delta t_u$ is transformed to a spatial offset b:

$$\rho_c^{(k)} = \left\| \mathbf{x}^{(k)} - \mathbf{x} \right\| + b + \tilde{\varepsilon}_\rho^{(k)}, \quad (3)$$

where $\mathbf{x}^{(k)}$ is the position vector of satellite k, and $\mathbf{x}$ is the user's position. An East-North-Up coordinate system is used in the local context of the user, where the x axis is assigned to East, the y axis to the North, and the z axis upwards, pointing into the zenith.

3.2.1 Linear model

In (3), the unknowns to be determined are $\mathbf{x}$ and b. This makes a total of four unknowns in a nonlinear equation. To unambiguously estimate the user's position, at least four of those equations are needed, corresponding to a total of at least four satellites visible to the user $(K \geq 4)$.

In order to solve the equation system, a good approach is to linearize them about an estimated position and time using *Newton's Method*, and let the system converge to the solution in iterations.

As a starting point, a first pseudorange ρ_0 is *calculated* out of the initial guesses $\mathbf{x_0}$ and b_0:

$$\rho_0^{(k)} = \left\| \mathbf{x}^{(k)} - \mathbf{x}_0 \right\| + b_0, \quad (4)$$

[4]DGPS: Differential GPS; ground based or satellite supported additional data to increase accuracy

and the difference between the (measured) corrected pseudorange ρ_c and ρ_0 is plugged into (3):

$$\begin{aligned}
\delta\rho^{(k)} &= \rho_c^{(k)} - \rho_0^{(k)} \\
&= \left\| \mathbf{x}^{(k)} - \mathbf{x} \right\| - \left\| \mathbf{x}^{(k)} - \mathbf{x}_0 \right\| + (b - b_0) + \tilde{\varepsilon}_\rho^{(k)} \\
&= \left\| \mathbf{x}^{(k)} - \mathbf{x}_0 - \delta\mathbf{x} \right\| - \left\| \mathbf{x}^{(k)} - \mathbf{x}_0 \right\| + (b - b_0) + \tilde{\varepsilon}_\rho^{(k)} \\
&\approx -\frac{(\mathbf{x}^{(k)} - \mathbf{x}_0)}{\left\| \mathbf{x}^{(k)} - \mathbf{x}_0 \right\|} \cdot \delta\mathbf{x} + \delta b + \tilde{\varepsilon}_\rho^{(k)} \\
&= -\mathbf{l}^{(k)} \cdot \delta\mathbf{x} + \delta b + \tilde{\varepsilon}_\rho^{(k)}
\end{aligned} \tag{5}$$

To make this transformation understandable, it has to be noted that it applies a Taylor series approximation of a vector norm. It then leads to the estimated line-of-sight vector $\mathbf{l}^{(k)}$ pointing from the estimated position towards satellite k.

3.2.2 The geometry matrix H

If (5) is used to set up a system of K linear equations, the whole set can be written as

$$\delta\boldsymbol{\rho} = \begin{bmatrix} \delta\rho^{(1)} \\ \delta\rho^{(2)} \\ \vdots \\ \delta\rho^{(k)} \end{bmatrix} = \underbrace{\begin{bmatrix} (-\mathbf{l}^{(1)})^T & 1 \\ (-\mathbf{l}^{(2)})^T & 1 \\ \vdots & \\ (-\mathbf{l}^{(K)})^T & 1 \end{bmatrix}}_{\mathbf{H}} \cdot \begin{bmatrix} \delta\mathbf{x} \\ \delta b \end{bmatrix} + \tilde{\varepsilon}_\rho \ , \tag{6}$$

where we have assigned the symbol $\mathbf{H}$ to the matrix containing the line-of-sight unit vectors $-\mathbf{l}$. The fourth column is multiplied with the spatial equivalent of the user clock offset — since the direction of the incoming satellite ranging signals does not affect the user clock offset's impact on the measured pseudorange, this column contains only ones.

The resulting matrix $\mathbf{H}$ is called *geometry matrix*, because it contains the geometric information about the satellite constellation. The coordinates $\mathbf{x}^{(k)}$ of each satellite can also be described in terms of their azimuth and elevation angles. The term azimuth is normally used for the angle of orientation of the satellite; its reference direction is defined to be North, and its orientation is clock wise. For the computations in the geometry matrix, and for coordinate transformations which are used later on, however, an angle referring to the primary axis x, with a counter clockwise orientation, is more appropriate. Therefore, the identifier Φ is now assigned to the angle known as azimuth, whereas Φ' is defined to be the angle referring to East, orientated in the opposite direction. It can be stated as

$$\Phi' = \frac{\pi}{2} - \Phi \ . \tag{7}$$

Then, the new line-of-sight vectors, now called $\mathbf{e}_k$, can be written as

$$\mathbf{e}_k{}^T = \left[\begin{array}{ccc} \cos E_k \cos \Phi'_k & \cos E_k \sin \Phi'_k & \sin E_k \end{array} \right] , \tag{8}$$

and the geometry matrix $\mathbf{H}$ is

$$\mathbf{H} = \left[\begin{array}{c} \begin{array}{cccc} -e_{1x} & -e_{1y} & -e_{1z} & 1 \\ -e_{2x} & -e_{2y} & -e_{2z} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \underbrace{-e_{Kx} \quad -e_{Ky} \quad -e_{Kz}}_{-\mathbf{e}_K{}^T} & & & 1 \end{array} \end{array} \right] ; \tag{9}$$

Here, E_k denotes the k^{th} satellite elevation from the horizon and Φ'_k the angle computed from the azimuth in (7). Again, the 1 is connected to the temporal impact of the satellite position, which is neutral. The line-of-sight vectors still have the same orientation: from the user position towards the satellite. This notation will become more important later on, when DOP is parameterized with the satellite positions.

If the set of equations (6) is now resolved for the unknowns $\delta\hat{\mathbf{x}}$ and $\delta\hat{b}$, the resulting equation

$$\left[\begin{array}{c} \delta\hat{\mathbf{x}} \\ \delta\hat{b} \end{array} \right] = \left(\mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T \delta\boldsymbol{\rho} \tag{10}$$

can be used to calculate the new estimate for the next iteration step:

$$\begin{array}{rcl} \hat{\mathbf{x}} & = & \mathbf{x}_0 + \delta\hat{\mathbf{x}} \\ \hat{b} & = & b_0 + \delta\hat{b}. \end{array} \tag{11}$$

3.2.3 Correlation of measurements

Up to now, all ranging measurements are assumed to be completely uncorrelated, and it is supposed that they can be used at an equal basis to determine the estimated user position. In reality, however, correlation between the satellites exists strongly: Ionospheric effects as well as residual tropospheric delay have always a quite common impact on the visible satellites' signals; especially if their associated line-of-sight vectors are particularly similar.

Also, a certain dependency between the quality of the signal and the satellite's position in the sky is evident. This addresses the satellite's elevation in special — if a satellite is very close to the horizon, the corresponding errors introduced through atmospheric effects are much larger — and that fact can be taken into account by weighting the individual satellites' contribution to the position estimation accordingly. This is described in particular in [WE95]. In contrast to that, minimizing the sum of the squared residuals as done above implies that all residing errors are worth the same, and therefore, the same effort is undertaken to minimize a satellite's error with respect to the global solution, no matter how qualified its signal is.

This impact on the signal quality can also be included in the cost function for the optimization problem that is subject to this thesis. Such an alternative cost function to obtain the perfect satellite constellation will be introduced in Section 5.4.1.

Correlation between the measurements, however, is usually completely neglected. The reason for this does not lie in the complicated procedure of taking it into account; it is because characterization of such correlation is too complex to do it within a reasonable amount of work. Furthermore, the original GPS SA[5] signal was in fact downgraded artificially by means of overlaying an uncorrelated error with zero-mean and a variance of 25m. This errors did absolutely dominate all the system-inherent errors; hence the assumption to neglect correlations was even more appropriate when SA was still active.

Under the stated assumption, the measurement error expextation and covariance can be modeled as

$$\begin{aligned} E(\tilde{\varepsilon}_\rho) &= \mathbf{0} \\ Cov(\tilde{\varepsilon}_\rho) &= E(\tilde{\varepsilon}_\rho \tilde{\varepsilon}_\rho^T) = \sigma_{UERE}^2 \mathbf{I}, \end{aligned} \tag{12}$$

where $\mathbf{I}$ denotes the identity matrix and σ_{UERE} is the standard deviation of the user range error.

3.3 Derivation of DOP

3.3.1 Example: position estimation in 2-D

The following simplified example shows the difference between the quality of a measured signal and the resulting quality of the gained position estimate. It is taken from [ME01], pg. 182.

In this example, a user determines his position in two dimensions by measuring the range to two available satellites, S_1 and S_2. His position has to be located on one of the intersection points of the circles drawn around S_1

[5]Selective Availability: artificial errors were superposed to the signal to degrade civil positioning until May 2000

and S_2, with the circles' radii set to the measured distances. It is assumed that the user has a-priori knowledge about which of the two intersections is the right one.

However, if the intersections are points, the measurements are to be perfect and not degraded by any uncertainty. In contrast, for a measurement of radii r_1 and r_2 with an uncertainty of $\pm\varepsilon$ each, the user's position is no more defined by a simple intersection of two circle lines; instead, it is now only known to be within an *area* defined by two pairs of circles: One pair around satellite S_1, with their radii of $r_1 - \varepsilon$ and $r_1 + \varepsilon$, and the other pair of circles around S_2, with radius of $r_2 - \varepsilon$ and $r_2 + \varepsilon$, respectively.

Here, the geometric arrangement of the two points $S1$, $S2$ around U becomes important. Figure 1 shows three distinct positionings, which greatly affect the *area of uncertainty* of the resulting position estimate. It is identifiable that, for large radii, the area of uncertainty can be approximated by the norm of the cross product:

$$A \approx \|\vec{e}_1 \times \vec{e}_2\|, \tag{13}$$

with the vectors $\vec{e}_1$ and $\vec{e}_2$ defined by

$$\vec{e}_i = \varepsilon \cdot \frac{\vec{U} - \vec{S}_i}{\|\vec{U} - \vec{S}_i\|}; \qquad i \in \{1; 2\}; \varepsilon = \|\vec{e}_i\| \ll r_i \tag{14}$$

The vectors' norms are ε, while their directions are defined by the line-of-sight vectors between the user and the corresponding satellite S_i.

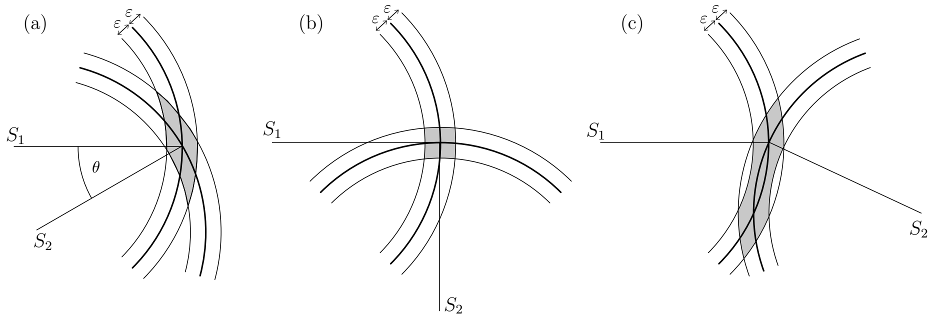

Figure 1: Positioning in the plane with three different satellite settings

In case (a), the satellites appear under an acute angle θ. Thus, the resulting area has a shape close to a rhombus. (*The rhombus, in fact, is obtainable for infinite radii. In reality, under the assumption $r_i \gg \varepsilon$, the shape will be much closer to a rhombus than in this example.*)

In case (b), a right angle lies between S_1 and S_2. The area's resulting shape is, again under the assumption $r_i \gg \varepsilon$, quadratic and has a significantly smaller area than in case (a).

In case (c), the two satellites are almost vis-a-vis — the angle is obtuse. The resulting shape has a very large area, resulting from the combination of two effects: The two intersection points are so close that together that the measurement uncertainty makes the two resulting areas of possible location join to one interconnected area; therefore a distinction made out of *a priori* knowledge such as the last position can not be taken into account anymore.

As it appears, for the special constellation of two satellites in two dimensions, the ideal angle is $90\,^\circ$, leading to a quadratic shape which has the lowest possible area of all rhombic shapes with edges of the length ε. In the next section, this basic principle will be extended to a more general setup of satellites, using linear algebraic expressions already provided by the positioning approach explained before.

3.3.2 Covariance and DOP

The principle derived from the example above can be applied to position estimation in 3-D as well. We revert to the assumption made in (12) which stated that the measurement errors are zero-mean; this leads to the fact that also the user's position estimation error is zero mean:

$$\begin{aligned} E(\Delta\mathbf{x}) &= E(\hat{\mathbf{x}} - \mathbf{x}) = \mathbf{0} \\ E(\Delta b) &= E(\hat{b} - b) = 0 \end{aligned} \tag{15}$$

Furthermore, under the assumption of common variance in the uncorrelated measurements, the estimates' covariance can be stated as:

$$Cov\begin{bmatrix} \Delta\mathbf{x} \\ \Delta b \end{bmatrix} = Cov\begin{bmatrix} \hat{\mathbf{x}} \\ \hat{b} \end{bmatrix} = \sigma^2_{UERE}(\mathbf{H}^T\mathbf{H})^{-1} \tag{16}$$

If every component is analyzed independently, the variances for the position estimates in the three axes and in the time domain can be written as

$$\begin{aligned} \sigma_E^2 &= \sigma^2_{UERE} \cdot \left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{11}; \\ \sigma_N^2 &= \sigma^2_{UERE} \cdot \left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{22}; \\ \sigma_V^2 &= \sigma^2_{UERE} \cdot \left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{33}; \\ \sigma_T^2 &= \sigma^2_{UERE} \cdot \left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{44}; \end{aligned} \tag{17}$$

The new subscripts E,N and V refer to the three axes in the local ENU coordinate system. while T refers to the local "time axis". With resulting the positioning accuracies defined as

$$E(\Delta\underline{u}\Delta\underline{u}^T) =: \begin{bmatrix} \sigma_E^2 & \bullet & \bullet & \bullet \\ \bullet & \sigma_N^2 & \bullet & \bullet \\ \bullet & \bullet & \sigma_U & \bullet \\ \bullet & \bullet & \bullet & \sigma_T^2 \end{bmatrix}, \tag{18}$$

and the four basic DOP values as

$$(\mathbf{H}^T\mathbf{H})^{-1} =: \begin{bmatrix} \mathrm{EDOP}^2 & \bullet & \bullet & \bullet \\ \bullet & \mathrm{NDOP}^2 & \bullet & \bullet \\ \bullet & \bullet & \mathrm{VDOP}^2 & \bullet \\ \bullet & \bullet & \bullet & \mathrm{TDOP}^2 \end{bmatrix}, \tag{19}$$

the positioning accuracy can now be written as

$$\begin{aligned} \sigma_E^2 &= \sigma_{UERE}^2 \cdot \mathrm{EDOP}; \\ \sigma_N^2 &= \sigma_{UERE}^2 \cdot \mathrm{NDOP}; \\ \sigma_V^2 &= \sigma_{UERE}^2 \cdot \mathrm{VDOP}; \\ \sigma_T^2 &= \sigma_{UERE}^2 \cdot \mathrm{TDOP};\ . \end{aligned} \tag{20}$$

The components noted as ($\bullet$) are left unspecified.

3.4 DOP types

Depending on what kind of positioning is done, different types of DOP become applicable. From the user's point of view, four values define his position: The eastern and northern component *(longitude and latitude)* of his position on the ground, his height and the exact time. The importance of the latter for an exact position estimate might not be clear at first; however, it is needed in the process of determining the signal's traveling time, which is the basis for the measurement of the signal range. Thus, a very exact knowledge of the current time is crucial, and positioning is always done in four dimensions, regardless of whether the measuring time is relevant to the application or not. The time base obtainable from GPS even creates its own applications and is frequently used where an exact time is needed, providing an alternative to the ground based time signal radio stations.

For each of the four axes, one DOP type is declared; in addition, three more parameters combine different sets of the first four values - HDOP combines EDOP and NDOP; the "position" PDOP combines all three spatial DOPs EDOP, NDOP and VDOP; and the "global" or "geometry" GDOP combines all four basic DOPs. An overview is given in Table 1.

In ground navigation applications such as car navigation, the most deciding parameter is usually HDOP, which combines NDOP and EDOP. Height information is of no importance in most en route navigation scenarios because map data is only provided in two dimensions. As opposed to this, the height information can have as much as or even more importancy as latitude and longitude in aviation applications; for example when performing an instrumental landing approach, where GPS acts as a navigation aid.

A coarse overview about typical DOP values can be obtained from Figure 2. In this plot, all DOP values introduced are shown for setups from 4 to 20 satellites, where one satellite stays at the zenith, while the remaining ones are deployed equidistantly on the horizon. It is very likely that, at least for some of the DOPs and small k, some constellations come close to or reach the minimally possible DOP.

3.5 Optimization of DOP

It is clearly visible that, in order to have an accurate position estimate, the user depends heavily on the geometry of the satellites that supply the ranging signals. The quality of the available geometry can be stated in terms of DOP, which can be derived from the geometry matrix H. A lower DOP results in better accuracy, so the ambition of the following research is minimizing a function that maps the satellite positions to DOP:

$$f_t(\mathbf{P}) : \mathbf{P} \rightarrow DOP_t \tag{21}$$

Here, the matrix $\mathbf{P}$ contains all the satellites' positions in terms of elevation E_k and the orientation angle Φ'_k. Hence, in an environment of k visible satellites, $\mathbf{P}$ is a $(k \times 2)$ matrix. The subscripts t in f_t and DOP_t accentuate that this mapping can be done for all different *types* of DOP.
$\mathbf{P}$ contains the elevation and orientation arguments for every satellite $1 \ldots k$:

$$\mathbf{P} = \begin{pmatrix} E_1 & \Phi'_1 \\ E_2 & \Phi'_2 \\ \vdots & \vdots \\ E_k & \Phi'_k \end{pmatrix}, \tag{22}$$

Table 1: DOP types

Name *(Desc.)*	**Affected accuracy**	σ
EDOP *(East)*	longitude	σ_E
NDOP *(North)*	latitude	σ_N
VDOP *(Vertical)*	altitude	σ_V
TDOP *(Time)*	local time	σ_T
HDOP *(Horizontal)*	planar accuracy	σ_H
PDOP *(Positional)*	3D accuracy	σ_P
GDOP *(Global)*	overall accuracy, including time domain	σ_G

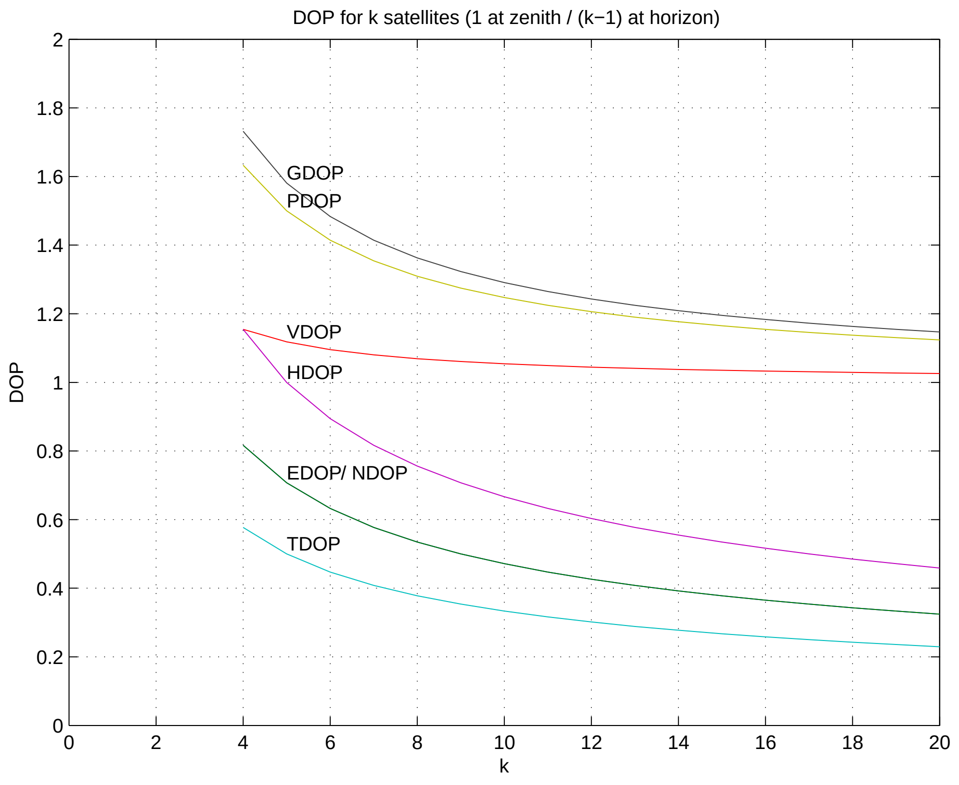

Figure 2: DOP values for different numbers of equidistantly arranged satellites

where E_i and Φ_i are limited to the following bounds:

$$\begin{aligned} E_i &\in \left[0 \ldots \frac{\pi}{2}\right[\\ \Phi_i' &\in \left[0 \ldots 2\pi\right[\qquad 0 \leq i \leq k \end{aligned} \tag{23}$$

While only stationary constellations are considered in the scope of this paper, it should not remain unstated that in "real life", the satellite locations will typically be distributed all over the visible area of the sky above the user. The knowledge of the stationary optimum could indeed be used to optimize GNSS satellite orbits with respect to optimal DOP — but other factors like the probability of masking in near-horizon positions or orbital constraints will probably still impact those considerations on a higher level.

All research was done in consideration of all DOP types shown in Table 1; for a better readability, similar results are summarized and described under the label "DOP" in the following sections.

4 Analytic approach

4.1 Manual solution of $(\mathbf{H}^T\mathbf{H})^{-1}$

As it can be seen in Section 3.3, an elementary DOP value can be read off directly as the square root of an element of the matrix $(\mathbf{H}^T\mathbf{H})^{-1}$, so the primary goal is to minimize those elements. It would be nice if we could just set its first derivative with respect to the position matrix $\mathbf{P}$ to zero and find the minimum:

$$\frac{\partial}{\partial \mathbf{P}} \sqrt{\left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{xx}} \overset{!}{=} 0 \; ; \tag{24}$$

with the subscript xx denoting the diagonal element of the matrix which contains its particular elementary DOP value EDOP, NDOP, VDOP or TDOP. For the combined values HDOP, PDOP and GDOP, a sum of the corresponding matrix elements would be plugged in instead of the single element. The cost function DOP_x of type x, which is to be minimized, is then given by

$$\arg\min_{\mathbf{P}} \left(\text{DOP}_x\right) = \arg\min_{\mathbf{P}} \sqrt{\sum_{i=1}^{4} a_i \cdot \left\{(\mathbf{H}^T\mathbf{H})^{-1}\right\}_{ii}} \tag{25}$$

Table 2: Coefficients a_i for Equation (25)

x	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$
EDOP	1	0	0	0
NDOP	0	1	0	0
VDOP	0	0	1	0
TDOP	0	0	0	1
HDOP	1	1	0	0
PDOP	1	1	1	0
GDOP	1	1	1	1

It is very likely that the input values for any found minima will touch their boundaries, especially the horizon. If this happens, setting the derivative to zero will, most probably, not result in any valid set of positions. This correlation can be easily observed for a constellation of four satellites: In general, the best available GDOP will probably be achieved for satellites that are distributed equally around the user in a fully accessible three-dimensional space. For a number of 4 satellites, the resulting shape will be a tetrahedron. However, the lower semi-sphere is not valid for any satellites; it is therefore quite likely that at least some of them will have their best valid position on the horizon, where the derivative of the resulting DOP cost function is *not* zero yet.

Apart from that, the effort's expected result was hoped to be a simplistic term, which could lead to a deeper understanding on how the positioning impacts the resulting DOP. If the term $(\mathbf{H}^T\mathbf{H})^{-1}$ is calculated for a set of four satellites and leads to a clear "best positioning", which could be read off easily, maybe this would indicate the right direction to have the same principles applied to any set of K satellites.

4.1.1 Coordinate System

In Section 3.2.2, the content of the geometry matrix $\mathbf{H}$ was denoted by line-of-sight vectors pointing from the satellites to the users and 1 for the temporal contribution. For this application, it is sufficient to look at the whole problem set in an ENU[6] context, so there is no need to consider the users real position in an earth centered (ECEF[7]) coordinate system. We assume that the users' position is known to be in the center of our coordinate system, with the x axis pointing towards East, the y axis pointing towards the North, and the z axis pointing up in the zenith, right above the user. Figure 3 shows one satellite in the ENU coordinate system. U denotes the user's position in the coordinate system's origin; S_k is satellite k; E_k is its elevation angle. Φ denotes the satellite azimuth, whereas Φ'_k is the angle of orientation referred to the x axis (East), in counter-clockwise direction.

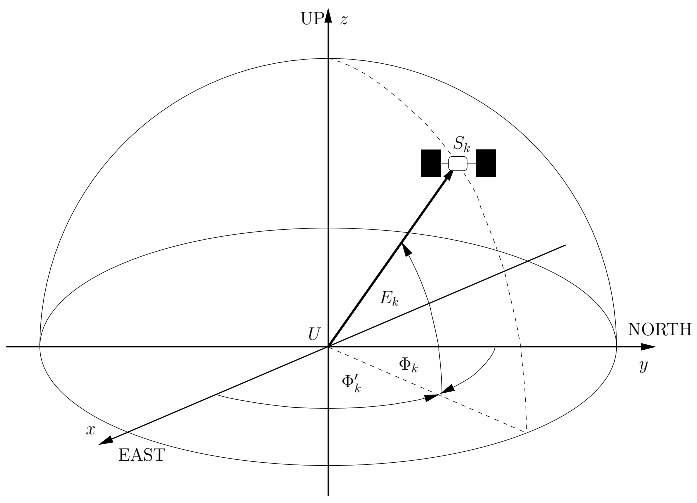

Figure 3: Local ENU coordinate system used in the problem

[6]East-North-Up
[7]Earth Centered/Earth-Fixed

4.1.2 The components of $\mathbf{H}^T\mathbf{H}$

With the line-of-sight vectors $\vec{e}_k$ given with respect to elevation and azimuth arguments, as shown in (8), the geometry matrix can also be written as

$$\mathbf{H} = \begin{bmatrix} -\cos E_1 \cos \Phi'_1 & -\cos E_1 \sin \Phi'_1 & -\sin E_1 & 1 \\ -\cos E_2 \cos \Phi'_2 & -\cos E_2 \sin \Phi'_2 & -\sin E_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -\cos E_k \cos \Phi'_k & -\cos E_k \sin \Phi'_k & -\sin E_k & 1 \end{bmatrix}, \tag{26}$$

where Φ' is derived from the azimuth according to (7).

The multiplication of the transpose of $\mathbf{H}$ with $\mathbf{H}$ leads to a symmetric (4×4) matrix with the positional arguments of the satellites well mixed up throughout the whole matrix. For simplification and in favor of readability, $\mathbf{H}^T\mathbf{H}$ is presented as a sum in the following equation (27).

$$\mathbf{H}^T\mathbf{H} = \sum_{k=1}^{K} \left[\begin{matrix} \cos^2 E_k \cos^2 \Phi'_k & \cos^2 E_k \cos \Phi'_k \sin \Phi'_k \\ \cos^2 E_k \cos \Phi'_k \sin \Phi'_k & \cos^2 E_k \sin^2 \Phi'_k \\ \cos E_k \cos \Phi'_k \sin E_k & \cos E_k \sin \Phi'_k \sin E_k \\ -\cos E_k \cos \Phi'_k & -\cos E_k \sin \Phi'_k \end{matrix} \right. \cdots$$
$$\cdots \left. \begin{matrix} \cos E_k \cos \Phi'_k \sin E_k & -\cos E_k \cos \Phi'_k \\ \cos E_k \sin \Phi'_k \sin E_k & -\cos E_k \sin \Phi'_k \\ \sin^2 E_k & -\sin E_x \\ -\sin E_x & 1 \end{matrix} \right] \tag{27}$$

4.1.3 The Inverse $(\mathbf{H}^T\mathbf{H})^{-1}$

As it can be seen from (27), the desired inverse of $\mathbf{H}^T\mathbf{H}$ is probably quite complex. Basically, there are two possibilities to compute the inverse of a non-singular symmetric matrix: One is the procedure known as the Co-Factor Method; the other method is called the Gauss-Jordan Method or Jacobi Method.

The Co-Factor Method defines the Inverse $\mathbf{A}^{-1}$ by calculating every single element of it following the rule

$$\{\mathbf{A}^{-1}\}_{ij} = \frac{(-1)^{i+j}\mathbf{D}_{ji}}{\det \mathbf{A}}, \tag{28}$$

where $\mathbf{D}_{ji}$ denotes the sub-determinant that results from deleting the row i and the column j from det $\mathbf{A}$.

For a (4×4) matrix, this involves computing 16 sub-determinants and the determinant itself, which also involves the usage of four of the sub-determinants. Each (3×3) sub-determinant can be computed directly using the formula

$$\begin{aligned}\det \mathbf{A} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - \\ a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}\end{aligned} \tag{29}$$

Every element of the initial matrix $\mathbf{H}^T\mathbf{H}$ contains a k-fold sum of several sine and cosine functions — although the computation is supposed to be straight-forward, complexity and available paper sheet sizes put a limit to this effort.

The Jacobi Method uses the Gauss algorithm to convert the matrix $\mathbf{A}$ into an identity matrix. If the same conversion steps are applied to an identity matrix, the resulting matrix is the inverse to $\mathbf{A}$.

$$\begin{aligned}[\mathbf{A}|\mathbf{I}] = \left[\begin{array}{ccc:ccc} a_{11} & \cdots & a_{1n} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} & 0 & \cdots & 1 \end{array}\right] \sim \\ \sim \left[\begin{array}{ccc:ccc} 1 & \cdots & 0 & b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & b_{n1} & \cdots & b_{nn} \end{array}\right] = [\mathbf{I}|\mathbf{B}] \\ \Rightarrow \mathbf{B} = \mathbf{A}^{-1}\end{aligned} \tag{30}$$

For this method to succeed, one must find the right conversion steps to result in an identity matrix, starting from the original matrix $\mathbf{A}$, or in our case, $\mathbf{H}^T\mathbf{H}$. This effort has been made, but also considered as too complex to be finished within a reasonable amount of time.

4.1.4 Conclusion to the manual approach

As calculation of $(\mathbf{H}^T\mathbf{H})^{-1}$ is quite complex, a manual minimization of only one of its components in respect of any of its arguments Φ_k or A_k would exceed the allocated editing time for this work. Thus, a computer aided approach seemed more feasible in order to find an optimal constellation.

4.2 Computer based analytic solution

The abilities to process symbolic equations, provided by add on packages to mathematical software as MATLAB, give the possibility to let the computer calculate the inverse of our self-multiplied geometry matrix $\mathbf{H}$. Therefore, all the input values have been introduced as symbolic values into a MATLAB matrix $\mathbf{H}$. $(\mathbf{H}^T\mathbf{H})^{-1}$ has been calculated and examined. The resulting term fills tens of screen pages, and could not be simplified with any of the available mechanisms.

4.3 Conclusions to the analytic approach

Obviously, the approach of resolving the requested cost functions manually wrt the satellite elevations and azimuths does not easily lead to the desired results. The term of the cost function, containing all position arguments, is too complex to simply read off the best satellite positions. A simulation may, however, find optimal constellations without knowing the cost function's inner details. This approach is described in the next section.

5 Optimization with Simulated Annealing

5.1 Simulation requirement

This section describes the reason why the optimization problem can be solved by means of simulation, and what attributes are required to be fulfilled for the simulation algorithm.

As shown in Section 4, the analytic approach did not render results satisfying the initial problem description — the term is too complex to derive a "best solution" by just looking at it; and computational methods do not give the wanted results either. Thus, the problem is now examined numerically. A method has to be found, which can resolve a number of $2K$ unknown variables such that the cost function becomes minimal.

Optimizing this problem could be done by searching the complete input domain. This would mean calculating the examined DOP value for every single combination of possible satellite positions, and saving the best value along with its belonging satellite positions. The position space for K satellites is, as we know, $2K$-dimensional, if the satellites' positions are described by their arguments of azimuth and elevation. A reasonably low granularity, in other words, large steps between the possible positions of one satellite, could keep the amount of computational work small. But the characteristics of the examined cost function are not known very well — there could be a global minimum hidden in between jump discontinuities, which would remain undiscovered if the step width of the search did not allow going there.

A more intelligent approach would start at an arbitrary set of satellite positions, and let the elevation and azimuth values move such that the cost function always returns a better value than before. This method is known as downhill simulation and is guaranteed to lead to a minimum, if there is one accessible directly from the starting position. However, the goal is to find the global minimum, which is unaccessible to such an algorithm if it is "hidden" behind a region that would require the simulation to move upwards. Since we cannot tell how the cost function exactly looks like, this attribute makes the downhill algorithm unacceptable for the application.

Hence, the simulation algorithm that is needed for this problem must be able to

- take care of local minima, while
- performing better than a "full search".

The following section describes such an algorithm.

5.2 The Simulated Annealing algorithm

This section describes why the Simulated Annealing algorithm is chosen for the simulation. It points out its fundamental idea and how the algorithm works. In this analysis, two different ways of “Simulated Annealing” were implemented — they are both introduced, and the main difference between them is depicted.

5.2.1 Principle of the algorithm

The Simulated Annealing algorithm was first published by S. Kirkpatrick et al. in 1983 [KJV83]. It is based on an algorithm of Metropolis et al. [MRR+53] and belongs to the class of heuristic optimization methods. Those methods generate, in contrast to exact methods such as full search, a computational requirement which is proportional to only small powers of the problem's complexity N.

The ability to find a global minimum to a problem such as the optimization of navigation satellite positions, while keeping the computational cost low, makes it a good choice for the simulation. In fact, the algorithm does not *guarantee* finding a global minimum; this is what the class of so-called exact algorithms can do. The quality of the result depends on many factors, but if applied reasonably, the probability to find a global minimum is quite high.

The main idea behind this algorithm is to apply the principles of statistical mechanics to a problem of combinatoric optimization. The term “Annealing” origins in metallurgy and describes the process of cooling down a (metallic) material very slowly in order to increase the size of the produced crystal structures. Crystal structures represent an energetic minimum for the involved atoms. If the whole material cools down to a monocrystal, the system is settled in a global minimum.

Atoms in heated material do not inevitably change their states towards lower energy all the time; instead, there is a certain probability that they will accept a higher energetic state as well. This probability depends strongly on the temperature of the material. If the temperature is reasonably high, the movement of the atoms is approximately random, in total disregard of any energetic gradient. In contrast, when the temperature reaches the vicinity of absolute zero, the probability to accept a worse state also converges to zero. Then, the material behaves in analogy to what was explained as the downhill algorithm in Section 5.1 — any state change will only happen if the energetic difference is negative; in other words, if it leads to a lower energetic level.

Any other temperature in between results in an equilibrium of the specified material, where a certain part of the atoms are settled at a state of minimal energy, contributing to a crystal structure; while other atoms are not linked and still retain to their significantly higher energy state. This balance can be described mathematically through a transition probability equation introduced in Metropolis' Algorithm [MRR+53]. It assigns the probability of 1 to an atom state transit towards an energetically lower level as 1, and a temperature dependent probability for a transit to a higher level:

$$P(\Delta E) = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ \mathrm{e}^{-\frac{\Delta E}{k_B T}} & \text{if } \Delta E > 0 \end{cases} \tag{31}$$

where ΔE is the energetic difference of the transition, k_B is the Boltzmann constant, and T is the absolute temperature in Kelvin; P denotes the probability of accepting this transition.

If the temperature of the material is lowered very slowly and carefully, it is very likely that each of the atoms in the system will remain in the lowest energetical state possible, making the whole system rest in its *global minimum*. If the temperature decay is too fast, more atoms do not have the probabilistic chance to find an energetically profitable molecular binding, and stay in a state of higher energy. This makes the system's (or material's) overall energy stay higher as well. Although this is a minimum as well — some of the atoms are in states out of which they can only get by absorbing energy — it is not a global, but only a *local* one. The global minimum system energy can only be reached by first heating the whole material, and then cooling it down again.

This physical principle can be easily applied to any mathematical problem by finding the correct analogies between the two worlds. The energetic states of the individual atoms can mostly be easily mapped to states of the elements in a problem. The energetic difference assigned to an atom transit between two states has to be substituted by an appropriate cost function very specific to the problem — the cost is normally predefined by the problem itself, as it is the parameter to be minimized. The probability to transit into a new state is defined by the temperature in "real life"; hence, a simulation temperature has to be introduced and scheduled properly throughout the simulated time period. A difficult part can be finding the neighbors of a state, which means deciding upon possible edges in the state network of the problem. This is especially difficult, if the simulation model has an infinite number of states.

In the context of optimizing satellite constellations with respect to DOP, definitions have been made to assign the required parameters in the following manner:

The output function to be minimized is the demanded DOP value. The state belonging to a specific DOP is defined as the set of satellite positions. Among other things, this means that every position set, where at least one satellite's position differs, represents another state. Neighbor states are defined as all states where at least one, but usually all satellite positions differs from the originating state within a certain distance. The distance between the old and the new positions is picked by a random process with normal distribution. The position change takes place on the surface of a unit sphere; hence, a two-dimensional random number is the basis of the random step. This step is then transformed into the arguments of elevation and azimuth. The complete procedure is described in detail in Section 5.3.4 An additional cut-off mechanism filter deviations that would exceed a position alteration of more than 45 degrees.

Letting the positions change in such a way makes the number of possible neighbors infinite. This means that the neighbors can not just be tested for transiting in a sequential or random order, but we have to assume that only one random neighbor is picked at a time. The decision on transiting leads either to a new state, or the whole loop starts over from the beginning, with the original state. Each satellite is allowed to travel independently, and the decision upon transition is made after each satellites move. Satellites are moved cyclically.

The simulating temperature is defined abstractly, and is used to influence the state transition probability as well as the mean distance between a originating state and a neighbor state. As mentioned, the step width is given as a standard deviation to a two-dimensional Gaussian random process. Coupling the standard deviation to temperature lets the simulation run faster, in general, when the chaotic contribution is still high, while allowing a finer granularity when temperature drops and the system starts heading towards the states of minimal cost.

With the simulation parameters defined like this, the satellites would move around randomly in the hemisphere of possible positions, as long as the temperature is high enough. When the simulation temperature is cooled down slowly, the satellite behavior changes into seeking better and better positions, until at last, they will remain in states that hopefully constitute the global optimum for the examined cost function.

5.2.2 Alternative implementation

Annealing can also be described in a slightly different way. One drawback of the above algorithm is that its movements have no orientation by themselves throughout the whole simulation process. Only the decision upon accepting a new state gives a guide towards a lower energetic state, but for a large number of satellites, the chance of randomly selecting a better state at all becomes more and more improbable. For a low temperature, the simulation might get stuck, because all state candidates that are picked do not meet the requirement of resulting in a lower cost function output.

The classical problems associated with the Metropolis algorithm and the method of Simulated Annealing, like the traveling salesman problem, feature a finite number of neighbor states, which avoid above complications more effectively.

With the current definition of neighbor states, a finite number could only be accomplished by letting the positions of the satellites change only in a predefined step width, each argument independently. For K satellites, there would be $4K$ neighbors to each state, because each of every satellite's two arguments (elevation and azimuth) could only be altered up- or downwards by a fixed number. But then, the search space would be severely narrowed, probably eviting minima that are desirable as solutions. On the other hand, if the step width was lowered to avoid this, simulation speed would degrade. It is mandatory to check whether the algorithm's functionality is not affected by a modification that allows it to handle an infinite number of neighbors.

The alternative implementation simply divides each subsystem's movement into two parts, of which each is weighted differently throughout the simulation. One part of the movement is the downhill part, which tries to drag the state to a place where the resulting cost will be lower. The second part applies random noise to the first position deviation, thus disturbing the downhill movement. The downhill contribution depends on the cost function's slope at the position, whereas the random noise is coupled to the simulation temperature. The physical analogon can be seen in atoms trying to reach lower states, but disturbed by the Brownian motion.

The strength of the random noise with respect to the downhill force has to be adjusted such that for a reasonably high enough temperature, the downhill force's contribution to the overall movement can be neglected. For zero temperature, the random noise must also be zero.

For this to work, the gradient of the cost function with respect to every input parameter must be known. In this simulation, the overall system state can be separated into subsystems of individual satellites, and the gradient can be calculated with respect to each satellite's elevation and azimuth. The satellite is then moved on the unit sphere towards the optimum, and a random noise is superposed afterwards.

Differences between the two implementations imply, first of all, the lack of a decision in the second one. The satellites move in every simulation cycle, except if there is no better position to go. Depending on the temperature, this movement is then disturbed by "thermal noise". In contrast to that, the original Simulated Annealing method tends to get stuck in environments where many worse neighbor states and only a few better ones are available, while the temperature is too low to let the worse be accepted.

It can be assumed that the "Noisy Downhill" algorithm will converge faster, but on the other hand, it might be more vulnerable to a steep local minimum. This assumption is motivated by the following consideration: If the state (or a sub-state, say, one satellite of the system) runs into a local minimum which is very steep, the downhill force has a substantial contribution to the satellites further movements, whereas the random noise is fixed in its maximum strength by the simulation temperature. If the minimum is too steep, and the temperature is already too low, downhill movement will *always* overvote the random part, making the satellite unable to escape from the local minimum until simulation time ends. It is possible, though, that the downhill part results in a movement far enough to kick the state out of the minimum on the opposite side, but the point remains that under normal circumstances, the chance to leave the pit are zero, if the minimum is steep and the temperature is low enough.

The original Simulated Annealing, however, has still a small probability to climb up, since it is possible that several consecutive steps lead uphill, as long as the temperature is not completely zero. The downhill movement does not depend on the slope of the underlying cost function, because the accepting probability in such cases is always one.

In order to analyze the specific pros and cons of both implementations, they have to be implemented and watched closely during simulation runs. The implementation details are described in the following section.

5.3 Description of the implementation using mathematical software

This section describes the steps necessary to implement the simulation and points out some details that needed special attention, such as the correct mapping of random distributions in the context of a satellite position on the virtual sky view hemisphere.

5.3.1 Functional overview

Figure 4 gives a coarse overview on how the Simulated Annealing algorithm was implemented. The main simulation loop is located in `SimAnn`, from where also the setup is initiated. `RandomSatelliteSetup` returns a position matrix with satellites equally distributed over the virtual sky-view semisphere (see also Section 5.3.3). For the purpose of creating a sky view plot, `SetupSkyView` is called. A row vector containing the temperature for every particular simulation time is returned by the function `TempSchedule`.

`SimAnn` then enters the main simulation loop and calls the simulation cycle function periodically. The simulation cycle function is set as a function handler, and can either point to `SimCycle_SimAnn` for Simulated Annealing, or to `SimCycle_NoisyDownhill` for the Noisy Downhill algorithm. If a position change is reported back, the `SkyView` function is called to update the screen.

If the Kirkpatrick simulation algorithm is defined as the simulation method, the invoked function `SimCycle_SimAnn` loops through satellites 1 to K for every invocation. It calls `RandomSatStep` in order to have the particular satellite's position changed. The satellite position is checked for not

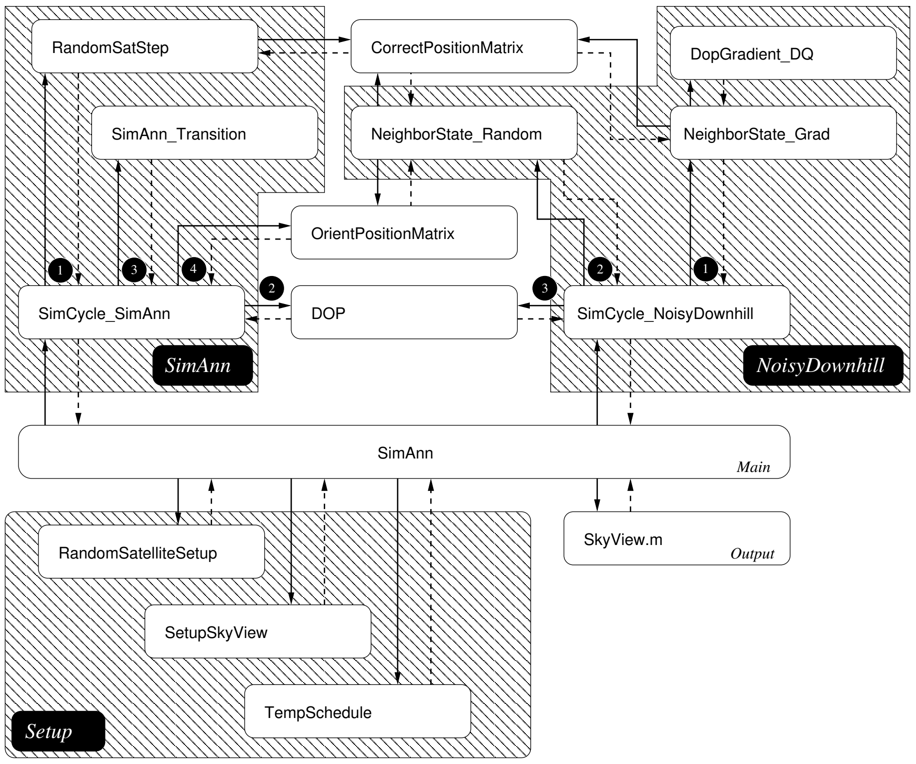

Figure 4: Functional chart of the Simulated Annealing implementation

exceeding the defined boundaries in `CorrectPositionMatrix`. The decision making function `SimAnn_Transition` is responsible for determining whether the new state will be accepted or not. Therefore, the new DOP has to be calculated in `DOP`, and the cost difference is provided to the decision function. If the new state is accepted, an orientation of the matrix is performed in `OrientPositionMatrix`, meaning that one satellite points to the East. This makes the graphical output nicer, helps avoiding duplicity in the solutions and makes the evaluation of the collected data easier.

If the "Noisy Downhill" method is used for simulation, the function `SimCycle_NoisyDownhill` is invoked. The new state is defined in two steps: First, the function `NeighborState_Grad` moves each satellite's position towards a better resulting DOP individually. The required gradient is approximated by a differential quotient implemented in `DopGradient_DQ`. Then, the new positions are again changed by superposing a random noise in `NeighborState_Random`. After the new state has been completed, the new DOP value is calculated by calling `DOP`. Corrections are made every time after shifting the positions, whereas the orientation of the solution is only done once, at the end of the state creating process.

The main loop in `SimAnn` updates simulation time and takes care of some parallel output on the console. When the simulation time is over, the loop terminates and the results are printed to the console. If the simulation states do not change during a predefined amount of simulation time, the result is considered stable and the simulation terminates as well.

5.3.2 Implementation platform

As implementation platform for the simulation algorithm, the plots and the evaluation scripts, MATLAB was chosen. It provides easy accessibility to matrix-like data structures, and lots of methods to visualize the results. With the Symbolic Math package installed, it can also resolve symbolic problems, as it was done in Section 4.

5.3.3 Setup

To initialize the simulation, a random setup of k satellites is preferred in the most cases. For further investigation of certain situations, however, a predefined setup can be supplied as well.

In this implementation, the MATLAB function returns a position matrix

`POS=RandomSatelliteSetup(`*`NumberOfSatellites`*`)`

as described in Equation (22) on page 16.

Uniformly distributed random setup

In the software model, each satellite's position is represented by its elevation and its azimuth. If both azimuth and elevation are chosen directly by a random process with a uniformly distributed probability density function, then the resulting distribution of satellites over the available surface of the sky-view hemisphere will not be uniform. The satellites will rather concentrate around the zenith. A random setup of a large number of satellites is shown in Figure 5.

It should be noted that, by projecting a spherical surface into a circle, the observed distortion effect is even amplified. Still, if the presentation form was a sphere instead of a sheet of paper, the accumulation of satellites around the zenith would be still noticeable.

The reason for the concentration of the satellites in Figure 5 is obviously the uniform distribution of the randomly chosen elevation and azimuth angles for each satellite — the concentration appears to be dependent only from elevation, but not from the azimuth. Since a uniform distribution of the satellites *with respect to area* is desired, the correlation between elevation and its corresponding surface element on the sphere has to be known. This connection is to be examined in this section.

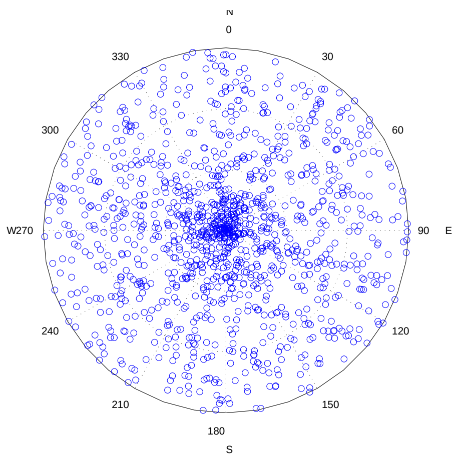

Figure 5: Random satellite setup with uniform elevation distribution

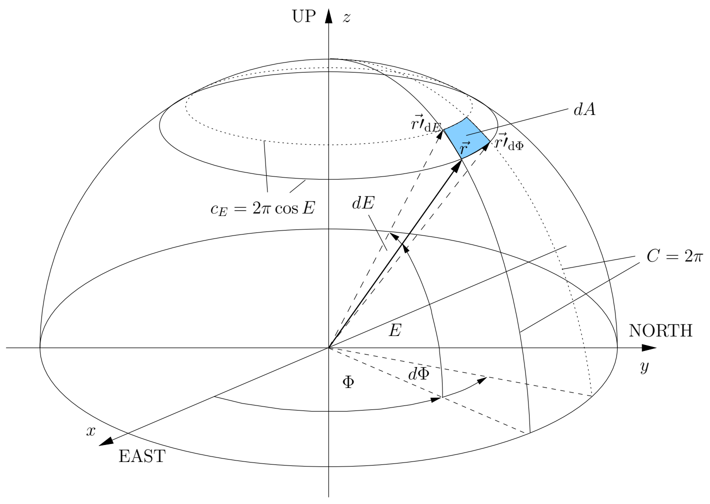

Figure 6: Infinitesimal surface element on the sphere

Let $\vec{r}$ in Figure 6 be a satellite position on the virtual sky-view hemisphere with radius 1, described in spherical coordinates:

$$\vec{r} = \begin{pmatrix} 1 \\ E \\ \Phi' \end{pmatrix} \tag{32}$$

For better readability, the prime denoting the reference and orientation of Φ' is neglected subsequently for all small differential angles in this section.

The vectors $\vec{r'}_{\mathrm{d}E}$ and $\vec{r'}_{\mathrm{d}\Phi}$ result from rotating the original vector $\vec{r}$ by infinitesimally small angles $\mathrm{d}\Phi$ and $\mathrm{d}E$:

$$\begin{aligned}\vec{r'}_{\mathrm{d}E} &= \begin{pmatrix} 1 \\ E + \mathrm{d}E \\ \Phi' \end{pmatrix} \\ \vec{r'}_{\mathrm{d}\Phi} &= \begin{pmatrix} 1 \\ E \\ \Phi' + \mathrm{d}\Phi \end{pmatrix} \end{aligned} \tag{33}$$

For *infinitesimally small* $\mathrm{d}E$ and $\mathrm{d}\Phi$, the resulting area element $\mathrm{d}A$ can be approximated by the cross product of the two vector differences

$$\mathrm{d}A = |(\vec{r'}_{\mathrm{d}E} - \vec{r}) \times (\vec{r'}_{\mathrm{d}\Phi} - \vec{r})| = \sin\alpha \cdot |(\vec{r'}_{\mathrm{d}E} - \vec{r})| \cdot |(\vec{r'}_{\mathrm{d}\Phi} - \vec{r})|, \tag{34}$$

where the angle α is $90°$, so the sine term can be neglected. With the vector differences expressed by the parameters $\mathrm{d}\Phi$ and $\mathrm{d}E$, the area $\mathrm{d}A$ can be approximated as

$$\mathrm{d}A \approx \underbrace{\mathrm{d}E}_{|(\vec{r'}_{\mathrm{d}E} - \vec{r})|} \cdot \underbrace{\cos E \, \mathrm{d}\Phi}_{|(\vec{r'}_{\mathrm{d}\Phi} - \vec{r})|}, \tag{35}$$

where both vector lengths are approximated by their small angle approximation

$$\begin{aligned} \sin \mathrm{d}E &\approx \mathrm{d}E \\ \sin \mathrm{d}\Phi &\approx \mathrm{d}\Phi\ . \end{aligned} \tag{36}$$

The cosine term in (35) takes account of the fact that $\vec{r'}_{\mathrm{d}\Phi}$ lies on a small circle parallel to the xy plane, which has a circumference that is scaled down with respect to the great circle's circumference of 2π that is used for the vector $\vec{r'}_{\mathrm{d}E}$:

$$\begin{aligned} C &= 2\pi \\ c &= 2\pi \cos E \end{aligned} \tag{37}$$

Given the above connection, the ratio of an area $\mathrm{d}A_E$, located at an arbitrary elevation E, and $\mathrm{d}A_0$ located at the horizon is

$$\mathrm{d}A_E = \cos E \cdot \mathrm{d}A_0\ . \tag{38}$$

The requirement for the satellite distribution is, that the probability to have a satellite present at a particular spot on the semi-sphere is equal through the whole defined area. In other words, the probability P(S) for a surface element to contain the randomly chosen satellite position shall be proportional to the surface element's area. With respect to E, it can be written as

$$P(S \in A_E) = \frac{\mathrm{d}A_E}{2\pi} = \cos E \cdot \underbrace{\frac{\mathrm{d}A_0}{2\pi}}_{\text{const.}} \ , \tag{39}$$

where the areas are normalized to the overall area 2π of the semi-sphere with radius 1.

With $P(S)$ desired to be uniformly distributed, the distribution for E must therefore follow a cosine shaped random distribution. Since this custom distribution is not available in MATLAB, an available distribution has to be transformed by a mapping function such that the resulting distribution has the desired cosine shape. The easiest way to achive this is to start from a uniform distribution, which is also readily available in most programming languages.

In order to get a continuous probability density function of arbitrary shape, random values from the input function $P(X)$ have to be mapped to values in the domain of the output function $P(Y)$. The distribution of the input function is known; in this case we have a uniform distribution, which is given in its cumulative probability distribution function by

$$P(X < x_i) = x_i \ . \tag{40}$$

In order to map each value gained from the stochastic process X to a value in Y such that the probability distribution $P(Y)$ is met, each value x_i in the input function must be assigned to a value y_i in the output function, where the probabilities

$$P\Big(X \in [x_i; x_{i+1}[\Big) \overset{!}{=} P\Big(Y \in [y_i; y_{i+1}[\Big) \qquad \forall x_i; \tag{41}$$

match for all intervals. Then, the value pairs (x_i, y_i) define the mapping function $f(x) : x \mapsto y$, so the objective is describing y_i by x_i.

For the input function, the probability density given in (40) is uniform; for the output, it is required to be cosine shaped, where the overall sum must be 1:

$$P(Y < y_i) := \int_0^{y_i} \cos Y' \ dY \ , \tag{42}$$

where the total probability sums up to 1:

$$P(Y \in [0; \frac{\pi}{2}]) = \int_0^{\frac{\pi}{2}} \cos Y' \ dY = \big[\sin Y\big]_0^{\frac{pi}{2}} \overset{!}{=} 1 \tag{43}$$

If we plug the probabilities into (41), we can rearrange the resulting equation

$$\begin{aligned} P(X < x_i) &= P(Y < y_i) \\ x_i &= \int_0^{y_i} \cos Y' \, dY \\ x_i &= \sin y_i \\ y_i &= \arcsin x_i \qquad (-1 \leq x_i \leq 1) \end{aligned} \tag{44}$$

Consequently, the arcsine of a uniformly distributed random variable X can be used to pick a sample elevation angle e

$$e = E = \arcsin X \qquad \text{with } P(X) \equiv 1 \ , \tag{45}$$

where the cumulative probability density function of the generated random variable E is given by

$$P(E < E_i) = \int_0^{E_i} \cos E_i' \, dE_i. \tag{46}$$

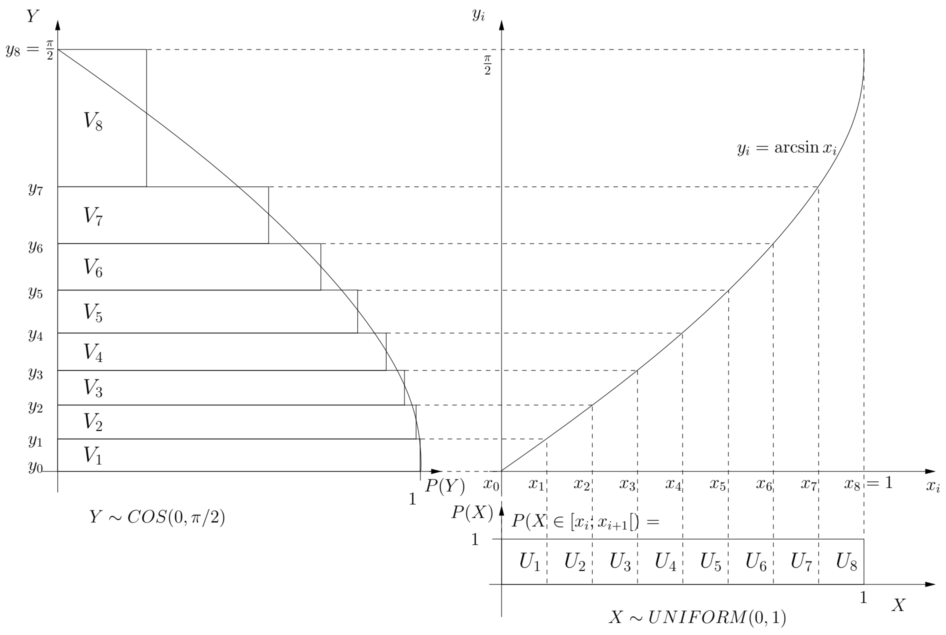

Figure 7: Transformation of random variables

Figure 7 demonstrates the transformation of the random variable X to E. The uniformly distruted X is drawn in the bottom, divided into intervals by an equally distributed set of x_i. The resulting probability masses U_i are therefore equal ($U_1 = U_2 = \ldots = U_8$). The transformation function *arcsin* maps each x_i to its corresponding interval boundary in the Y domain,

y_i. The resulting areas V_i between the interval boundaries are also equal ($V_1 = V_2 = \ldots = V_8$), and sum up to 1 in total. In the figure, they are approximated by rectangles of the dimensions $U_i = (y_i - y_{i-1}) \cdot cos(\frac{y_i - y_{i-1}}{2})$.

Using the described transformation for elevation angles, a histographic accumulation of several hundred randomly created setups in a sky-view results in a evenly distributed field of satellites like Figure 8.

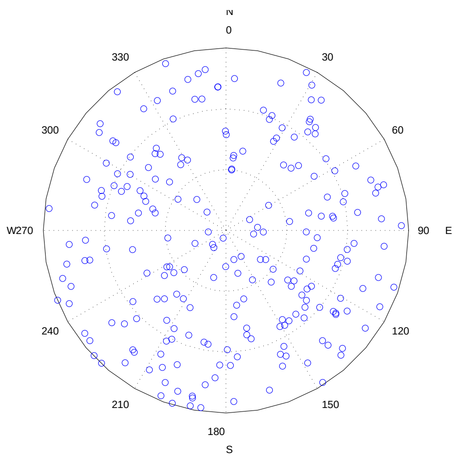

Figure 8: Random satellite setup with the fitted elevation distribution

5.3.4 Random stepping of satellite positions

When a satellite should perform a random step on the sphere, it is desirable that the average step width, which is referenced to the covered distance on the sphere surface, is fixed and does not depend on parameters such as elevation. As in Section 5.3.3, the description in terms of azimuth and elevation does not simply allow to assign uniformly distributed random numbers to these two parameters, but instead, a coordinate transformation has to take place.

In order to accomplish this, a two-dimensional movement in a fixed (and Cartesian) coordinate system is created by two random numbers, and is then transformed by means of vector rotation into the place of the desired originating satellite position vector.

The random movements of satellites were decided to have a two-dimensional Gaussian distribution $X \sim \mathcal{N}^2(0, \sigma^2)$, with the default standard deviation set to 22.5°.

The random stepping vector

$$\delta\vec{r}_{yz} = \begin{pmatrix} 0 \\ X_x \\ X_y \end{pmatrix} \tag{47}$$

is first created in the y-z-plane, as it can be seen in Figure 9. It corresponds to a random step for a position vector at $(1\ 0\ 0)^T$, and is thus normal to the x axis. This vector now has to be pitched into the elevation E by rotating it about the y-axis; the resulting vector is denoted as $\delta\vec{r}_{yE}$. Then, the rotation of Φ' is done about the z-axis. After this two-step rotation, the resulting stepping vector $\delta\vec{r}_{E\Phi}$ is normal to the position vector $\vec{r}$.

The new position vector $\vec{r}_{new}$ can then be obtained by adding the stepping vector to the old position, and dividing the result through its vector norm in order to have a unit vector — the result of the vector addition must have a norm greater than one, since the two vectors are normal. Because the new position has to be known in terms of elevation and azimuth, the normalization is in fact done by converting the vector to spherical coordinates and neglecting the radius.

The rotation can be described with the rotation matrix

$$\mathbf{R}_{yz} = \begin{pmatrix} \cos\Phi'\cos E & -\sin\Phi' & \cos\Phi'\sin E \\ \sin\Phi'\cos E & \cos\Phi' & \sin\Phi'\sin E \\ -\sin E & 0 & \cos E \end{pmatrix}, \tag{48}$$

so the resulting vector $\vec{r}'$ (not normed) can be calculated by

$$\vec{r}' = \vec{r} + (\mathbf{R}\delta\vec{r}_{yz}) = \vec{r} + \delta\vec{r}_{E\Phi} \tag{49}$$

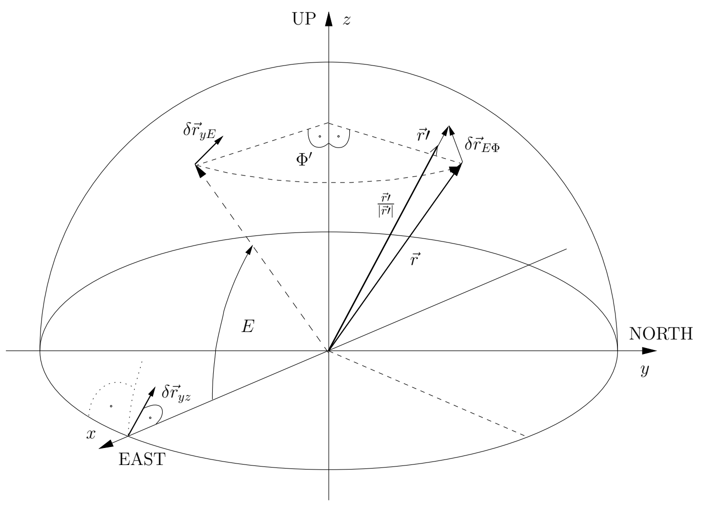

Figure 9: Rotation of the random satellite step vector

5.3.5 Correction and orientation of satellite positions

When random steps are applied to the satellite positions, the position matrix is always checked for boundary violations after the random movements have been added to the old position vectors.

For the elevation, the argument must not become negative on the one hand, and must not exceed $\frac{\pi}{2}$ on the other hand. In the case of using the weighted positioning solution, an elevation of 0 is even already forbidden, because it makes the satellite's weighting coefficient zero and thus, the geometry matrix is likely to become singular, especially for small numbers of satellites. When the elevation argument falls below the lower bound, it is simply set to it. When it exceeds the zenith, the elevation excess is subtracted from $\frac{\pi}{2}$, and the azimuth is flipped around by π. In the case of satellite azimuths, the boundary values are actually only to keep the values within a nice region. Azimuth arguments are kept within their bounds with the modulus operation. Both of the arguments are taken care of in the function `CorrectPositionMatrix`, which is invoked once after every alteration to the position.

To make different simulation results comparable to each other, fight infinite solution duplicity, and in order to bring some stability to the position sets, the whole constellation can be oriented towards a fixed point. Except for the EDOP and NDOP cost function, the azimuthal orientation of the system in whole does not play a role at all — all positions resulting from turning every satellite by the same azimuth result in the same DOP. Thus, it was decided to orient the position matrix by fixing one satellite's azimuth to zero degrees, corresponding with east in the used ENU coordinate system. This is done by subtracting the "Orientation Satellite"'s azimuth from all azimuths every time a new position set is completed. A special mechanism transfers the role of the orientation satellite to a new satellite whenever the old one rises to high. This is done because satellites were found out to tend to center in the zenith if they get close enough, and a satellite in the zenith changes its azimuth very chaotically. When this happens, all other azimuth angles follow the movement and the orientation mechanism does not work any more.

The number of the orientation satellite is kept in a global variable called `EASTSAT`. The function name to orient the position matrix is called `OrientPositionMatrix`. If the cost function is EDOP, NDOP, σ_E or σ_N, the orientation is skipped.

5.3.6 Computation of DOP

One of the central parts of the simulation is the cost function. Since most of the desired cost functions are obtained quite similarly, all required functionality was put into the function `DOP`. A string parameter is passed along with the position matrix to decide which cost function to apply. Valid functions are all seven DOPs, the corresponding estimate accuracies σ, when weighted positioning is used (see Section 5.4.1); and the matrix condition (see Section 5.4.2). It turned out that the symbolic computation does not lead to a gain in computation speed, thus the required inverse on the geometry matrix product $(\mathbf{H}^T\mathbf{H})^{-1}$ is computed numerically every time the function is invoked.

5.3.7 Simulation cycles and neighbor states

Each simulation cycle needs to pick a neighbor state to the present satellite constellation. The big difference between the two introduced simulation algorithms is that Simulated Annealing may reject the new state in favor of the old one, while the Noisy Downhill algorithm will always change to the new state. Naturally, the mechanisms for obtaining such a neighbor state are quite different.

In Simulated Annealing, the new state is obtained by moving a satellite around randomly. The function `RandomSatStep` takes care about that. Afterwards, a decision for the new state is done in `SimAnn_Transition`, according to the cost difference and a random contribution depending on simulation temperature. During first simulations, it was observed that, for low temperatures, the states were unlikely to ever change, if all satellites were moved at once and the decision was done only after that. Hence, the simulation cycle was modified to move each satellite individually and decide for each "sub-state" with only one satellite position altered. This lead to a more vivid simulation characteristic for small temperatures.

Noisy Downhill neighbor states are obtained in two steps, as described before. First, a down-hill movement of each satellite is executed in `NeighborState_Grad`. Therefore, the function `DopGradient_DQ` linearizes the cost function at the given position, and computes the differential quotient of the free parameter. The result is distorted by applying an *arcustangens* to it in order to circumvent too high gradients, which would have to be trapped later otherwise. After the downhill movement, a random noise generated by `NeighborState_Random` is superposed to new position. The variance of the noise is temperature dependent and converges to zero during the simulation process, meaning that at the end, only the downhill movement will still have an impact on the positions, which is crucial for finding the minima precisely.

5.3.8 Temperature schedule

In order to find good minima, the simulation temperature has to be lowered carefully throughout the simulation process. It has also been discussed whether an intermediate re-heating of the system would benefit the resulting found minima, but the results of first simulations made the author believe that a monotonously falling temperature would cope with the used cost functions.

Experience shows that a schedule following a negative exponential function suits best. The simulation was also tried with linearly falling temperatures, but it is easy to see that most of the important position arranging is done when the temperature is really low. Compared with the physical counterpart, the numerical value used here is already the product of the Boltzmann constant k_B and the arbitrary simulation temperature, showing up as the denominator in the probability equation for the uphill transition probability:

$$P(\Delta E) = \mathrm{e}^{-\frac{\Delta E}{k_B T}} \tag{50}$$

To provide some numbers for k_BT, a value of 10 provides enough “heat” to let the satellite positions move around completely chaotically in both simulation algorithms. Values below 1 lead to a very weak convergence towards a minimum, and below 0.1, the states show a strong convergence without any noticable uphill movements.

5.3.9 Plots

In order to examine the implementation as well as the simulation results, different plot functions are provided.

DOP plot for equidistantly placed satellites

In order to easily validate the implementation of the DOP function, the different resulting DOP values for equidistantly set up satellites in numbers between 4 and 20 were plotted, and the result was compared to an existing plot from [Gün05]. Figure 2 in section 3.4 shows the result.

SkyView

The sky view is a figure showing the satellite's line-of-sight vectors relative to the user, projected into a plane. A sky-view shows the zenith above the user as a point in the middle of the figure, and the horizon around the user as a circle concentric to the zenith. For a given satellite position, its argument of elevation is transformed linearly to the distance from the horizon, heading towards the zenith. The sky view is an azimuthal projection type and is therefore distance-preserving, but not shape- and size preserving. This leads to an additional distortion when a random distribution is shown, as it is discussed in Section 5.3.3. Figure 10 shows a sky-view for a randomly chosen setup of five satellites.

Although the term "Sky View" might usually associate a figure showing the view of the sky from the user's point of view, the sky views used in literature ([ME01]) as well as the one used in the author's GPS handheld receiver show the sky from above, hence assigning the left side West and the right side East. This orientation has therefore been adopted for the work. It can be seen as a "map" of satellites, centered on the user.

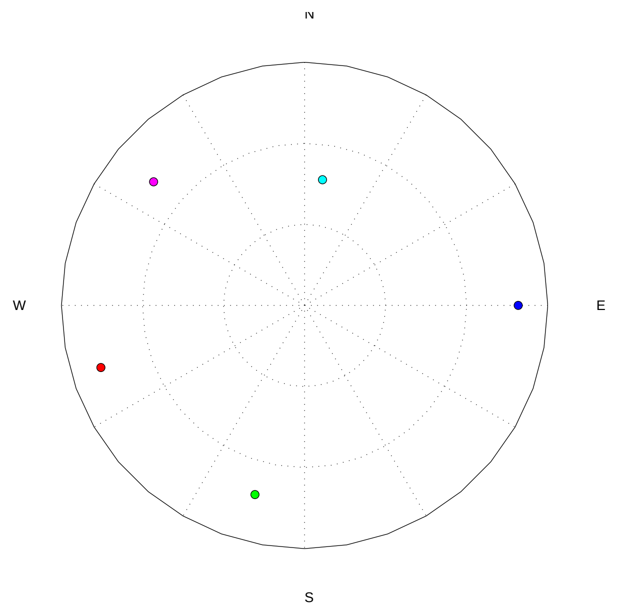

Figure 10: Sky view for a random setup of 5 satellites

Simulation Panel

When the simulation is running, it is sometimes desirable to watch the evolution of the satellites azimuths and elevations together with their resulting cost. This was implemented in the "Simulation Panel", which is integrated into the main loop in `SimAnn`. The updating behavior of the panel can be set to either update every 50^{th} simulation step, every single step, only once after the simulation, or not to be drawn at all (for batched simulations). A typical run of the curves can be seen in Figure 11 for a GDOP simulation with four satellites. The characteristical slow-down of the simulation parameter movements, including the cost function GDOP, can be seen.

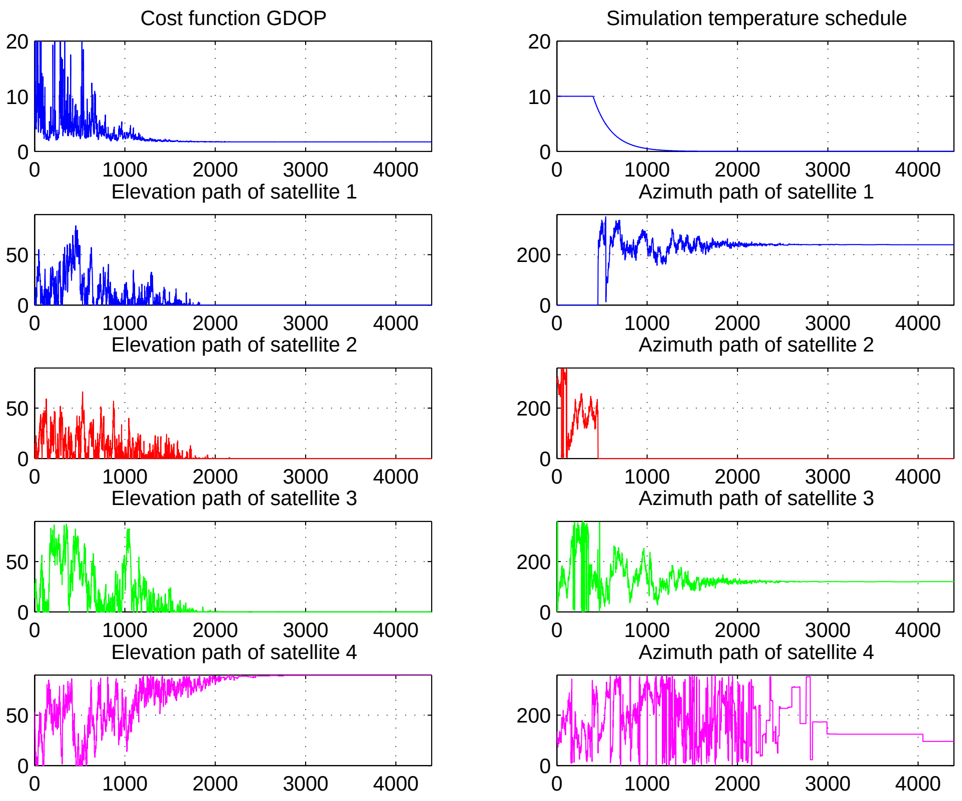

Figure 11: Simulation panel for a 4 satellites GDOP simulation

Surface plot for one free satellite

To get a deeper insight to the different cost functions' anatomies, special scenarios are investigated in Section 6.1.4, where only one satellite is allowed to travel, whereas the other satellites are fixed to their predefined positions. In this case, a real simulation can be omitted in favor of a full search over the allowed elevation and azimuth range for the free satellite. The result is displayed in a surface plot similar to a sky view, but with the z axis assigned to the resulting cost. Some of the surface plots can be seen in Appendix C.

Other plots include the plot for the resulting weighting function and the comparison of GDOP to σ_G using weighted positioning, both introduced in Section 5.4.1, as well as the comparison of GDOP to the matrix condition $\text{cond}(\mathbf{H}^T\mathbf{H})^{-1}$ (see Section 5.4.2).

5.3.10 Simulation wrappers

In order to easily process lots of simulation runs automatically, the functions `MultiSim` and `SimulationCampaign` can be used. MultiSim takes the number of simulation runs as an argument and runs that number of simulations with the same parameters, handing back the simulation results in a cell array. The SimulationCampaign script processes several of those multiple simulations for different parameter sets, and saves each parameter set's resulting cell array in a file, along with the set parameters.

The resulting files can be re-read by a function called `DisplayCampaignResults`, which outputs the resulting sky views for each run, and characterizes the overall result by means of categorizing the satellite positions into groups and computing statistical values like means and variances of elevations, azimuthal spacings etc. It can then decide upon different interesting resulting factors, like the stability of the solution for the parameter set or for subsets of the solution.

5.4 Alternative cost functions

5.4.1 Introducing weighted positioning as a basis for an alternative cost function

First simulations with the described implementation showed that in most of the cases, a certain number of satellites is likely to move onto the horizon line, resulting in zero elevation satellites. This behavior is easily explained when we remember that the satellites are located best, when they are located *all around* the user, that is, also beneath the horizon. The line of zero degrees is an artificial boundary to the mathematical system, created by the natural environment the GPS is supposed to work in. It is, furthermore, neither very likely that a satellite on the horizon will be receivable by the user, nor that the satellite will provide a very good signal suitable for positioning under these conditions.

The concept of the weighted position solution can take account of this fact. In the classical approach of minimizing the residual square error of the position solution with more than four satellites, and/or imperfect measurements, every satellite's signal contributes equally strongly to the computed position estimate. As already touched in Section 3.2.3, the signal quality rapidly degrades the nearer the satellite gets to the horizon. This is constituted by fact that the signal has to travel much longer distances through the atmospheric layers, which disperse the signal. Also, distortions to the signal caused by multipath get stronger.

The weighted position solution described in [WE95] combines the geometrical contribution with an elevation dependent measurement error variance into a new characteristical matrix. Because of the unequal weighting, the DOP can not be computed separately anymore. Instead, the weighted product of the measurement accuracy and the observed DOP is minimized. Therefore, the simulation has to be provided with typical measurement accuracy values. The values in this implementation are mean values obtained from the North American WAAS[8]; they are taken from the paper [WE95], where the simulation was also run with these parameters.

The weighted least squares position solution is found by minimizing

$$\mathbf{x} = (\mathbf{H}^T \cdot \mathbf{W} \cdot \mathbf{H})^{-1}\mathbf{H}^T\mathbf{W} \cdot \mathbf{y} \ , \tag{51}$$

where the product $\mathbf{H}^T\mathbf{H}$ has been provided with the $(K \times K)$ weighting matrix $\mathbf{W}$. This matrix contains the inverse satellite covariances in the diagonal elements, and zeros in all other elements — cross correlation between the satellites' errors is continuously neglected

$$\mathbf{W} = \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\sigma_k^2} \end{pmatrix} \tag{52}$$

The interesting values are not DOPs anymore, but the expected positioning accuracy or confidence for the three coordinate axes, and their combinations respectively. For example, the vertical accuracy can now be written as

$$\sigma_v = \sqrt{\left\{(\mathbf{H}^T\mathbf{W}\mathbf{H})^{-1}\right\}_{33}} \ ; \tag{53}$$

and can be used as a cost function to be minimized in the simulation.

The weighting function for computing the elements of $\mathbf{W}$ considers the elevation dependent error variance contributions of the troposphere and ionosphere, the multipath degradation and also the SNR contribution in the following form for the user equivalent range error:

[8]Wide Area Augmentation System: DGPS Network providing correction data for the United States

$$\sigma^2_{UERE,k} = \sigma^2_{UDRE,k} + \mathrm{F}^2(E)\sigma^2_{UIVE,k} + \sigma^2_{SNR,k} + \frac{\sigma^2_{m45}}{\tan^2 E} + \frac{\sigma^2_{trv}}{\sin^2 E} \; ; \tag{54}$$

where the subscript k denotes the satellite number k. The individual factors are given in Table 3.

Table 3: Variances for the weighting function

Identifier	**Value**	**Variance for**
$\sigma^2_{UDRE,k}$	0.5m	ionosphere-free and tropo-free pseudorange correction
$\sigma^2_{UIVE,k}$	0.5m	vertical ionosphere correction
$\sigma^2_{SNR,k}$	0.22m	receiver noise
σ^2_{m45}	0.22m	multipath contribution at 45 degrees
σ^2_{trv}	0.15m	vertical tropospheric delay estimate

The obliquity function $\mathrm{F}(E)$ converts the provided vertical-referenced ionospheric contribution into slant:

$$\mathrm{F}(E) = \frac{1}{\sqrt{1 - \cos^2 E \left(\frac{R_E}{R_E + h}\right)^2}} \tag{55}$$

The weighting curve resulting from (54) shows that below an elevation of around 5°, the error variance increases significantly and thus, the satellite's contribution to the solution can be more or less neglected.

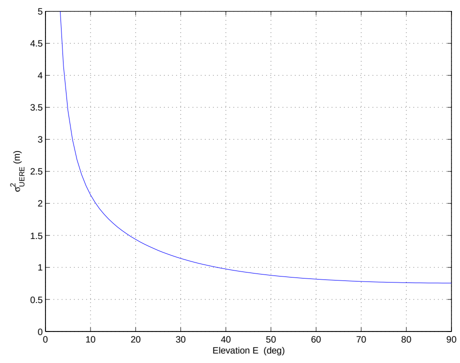

Figure 12: Pseudorange uncertainty as function of elevation angle

Using the weighted positioning uncertainty will then degrade the "quality" of locations near the horizon such that the satellites' observed drive towards lower elevations is confronted with an opposite force, driving them back higher in the sky. The result was observed to be a circle of satellites at a certain elevation, where the satellites were stuck to the horizon using the conventional DOP cost functions. Depending on the type of positioning uncertainty, the mean elevation of these circles changes. The sky views in Figure 13 show the simulation result of a 4 satellite setup using GDOP, and the same setup using the global positioning uncertainty σ_G respectively.

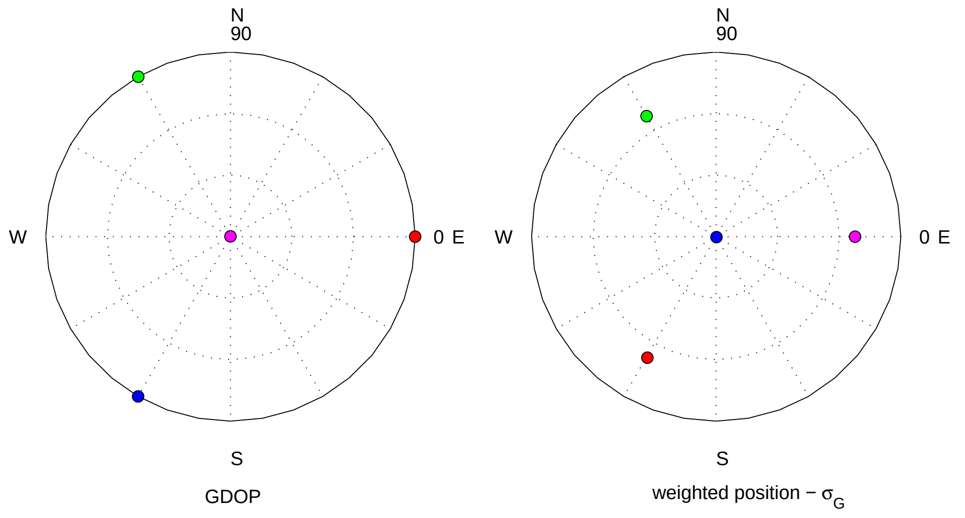

Figure 13: Sky views showing a GDOP and a weighted σ_G simulation result

5.4.2 Matrix condition as a measure for the constellation quality

As a measure of the position solution's quality, another possible can be the condition of the geometry matrix. This approach is taken in [PM97], where the geometry of a navigation reference system is to be optimized. The condition of a matrix $\mathbf{A}$ is defined as the ratio of the largest and smallest singular values of $\mathbf{A}$:

$$\text{cond}(\mathbf{A}) = \sqrt{\frac{\max\ \lambda(\mathbf{A}^T\mathbf{A})}{\min\ \lambda(\mathbf{A}^T\mathbf{A})}}\ , \tag{56}$$

with $\lambda(\bullet)$ denoting the eigenvalues of the matrix. It gives an indication about how close the matrix is to singularity. A singular geometry matrix means that the satellite positions lie in a plane, and thus do not suffice for a three-dimensional position solution. For such a position set, the computation of a cost function like DOP is not possible and does not make sense either, because the under-determined position solution can not be solved, anyway.

In fact, the DOP value GDOP, which can also be written as the square root of the trace of $(\mathbf{H}^T\mathbf{H})^{-1}$

$$GDOP = \sqrt{\mathrm{tr}\{(\mathbf{H}^T\mathbf{H})^{-1}\}} \tag{57}$$

is strongly correlated to the condition of the matrix **H**. The plot in Figure 14 shows cond(**H**) against GDOP for a number of 500 randomly selected position sets with 4, 6 or 10 satellites each. It can be seen that, although the minimally achievable GDOP is smaller for a higher number of satellites, the corresponding matrix condition does not fall below a constant value of about 3. Apart from that, the correlation between the two functions is almost linear. Therefore, cond(**H**) can be a substitute cost function for GDOP. Simulations have shown that the optimal results for both of them are the same for all examined simulation setups.

Detailed results using all discussed cost functions will be presented in the following Section.

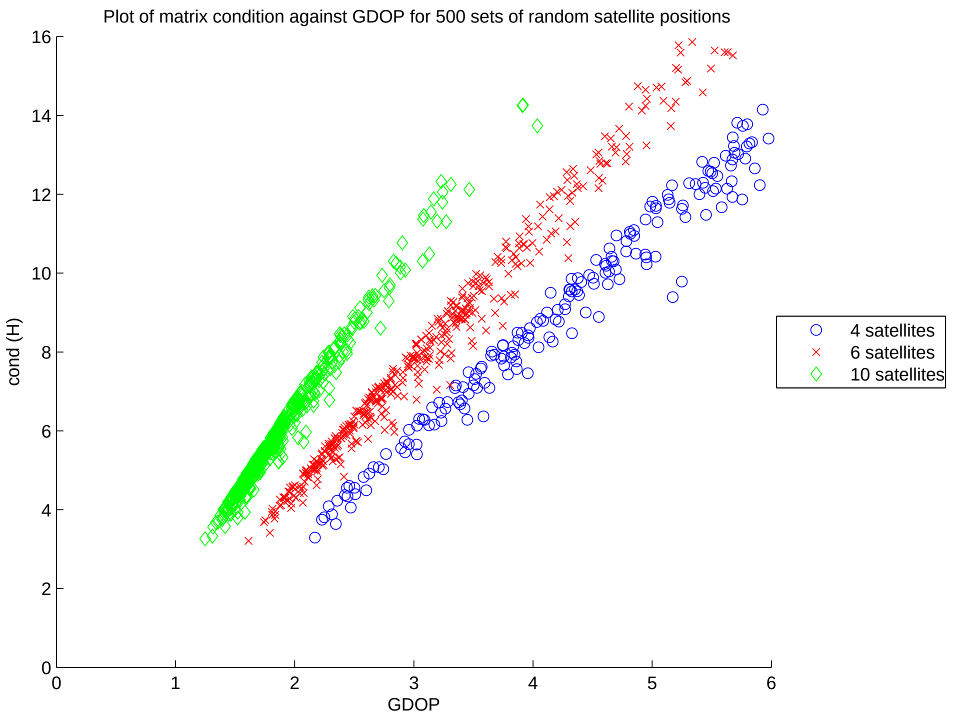

Figure 14: Comparison of GDOP and matrix condition for random satellite position sets

6 Simulation results and result characterization

6.1 Result inspection for different parameter sets

6.1.1 General observations

For almost every possible parameter setup, the satellite positions can be divided into two classes — those that are close to or at the horizon, and those in the zenith. For larger numbers of satellites K, the zenith position is often crowded by multiple satellites; if there are two satellites in the zenith, it is likely that they arrange vis-a-vis at an elevation of around 85°.

6.1.2 The cost function's impact on the result

Of course, there are special cost functions that lead to a different picture. The cost functions without vertical contribution, namely EDOP, NDOP and their combination HDOP, lack the satellite in the zenith. It is obvious that, in order to have the best possible geometry for horizontal positioning, the satellites' positions must be on the ground as well, hence have zero elevation. This, however, contradicts with the fact that the geometry matrix $\mathbf{H}$ is needed to be full-rank in order to have an equation system that can be solved. The resulting product $\mathbf{H}^T\mathbf{H}$ would be singular; the inverse could not be computed.

Hence, the horizontal cost functions make the simulation produce results where some of the satellites are set off the horizontal position just a little, so the geometry matrix becomes full-rank again. The same behavior can be observed for the horizontal weighted positioning accuracies σ_H, σ_E and σ_N, where all this happens at a particular elevation between the horizon and the zenith.

Figure 15 shows a result for a HDOP simulation. The two satellites on the horizon can be easily distinguished from the two satellites at a elevation of about 5°.

6.1.3 Satellite partitioning

One interesting number is the ratio of satellites in the zenith to those on the horizon. There seems to be a certain ratio that is always tried to be approximated by the solution, no matter how many satellites exist in the simulation. In fact, this assumption is true for almost every setup, although the ratio's value depends on the particular cost function.

6.1.4 Scenarios with $K - 1$ fixed satellites

In some scenarios, one satellite seems to be pulled back and forth between multiple good positions. In order to have a graphical means of visualizing the cost function for that particular satellite in a context of pre-defined

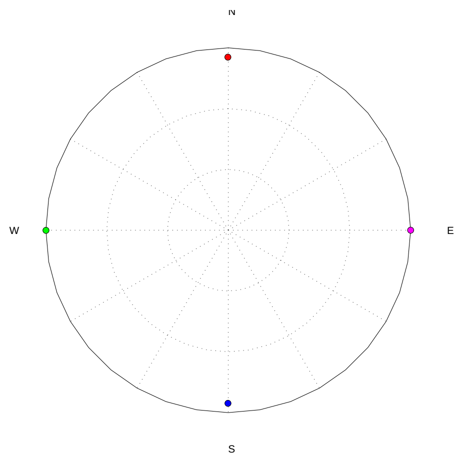

Figure 15: HDOP simulation result with 4 satellites

remaining satellite positions, a surface plot over the available positions was found to be helpful. This plot shows the cost function as a z-axis surface over the area as it is seen in a sky view.

The plot can be used to get to a deeper understanding of the correlations in the geometry matrix, but also to examine the corner case of satellites that show a very chaotic behavior in the simulation.

One scenario where this plot can show valuable information is the TDOP cost function. As it can be seen in the results in Section 6.2.2, there is always one “free satellite” moving around randomly, even for lowest temperature. The corresponding surface plot in Figure 17 in Appendix C shows that the TDOP (plotted as an inverse $\frac{1}{TDOP}$ here) is completely flat. In comparison with that, Figure 18 shows that the GDOP cost function has a well-defined minimum in the middle (highest point in the invertedly plotted surface). The corresponding sky view shows the positions of the 3 fixed satellites.

It can also be seen that positions that lie on or near a plane with the other satellites result in a much worse cost for all cost functions. Figure 19 shows an equidistant setup with an elevation of 25°. The free satellite creates a very high DOP for positions that are on the circle defined by the fixed satellites. Planes not parallel to the ground create an elliptic track on the sky view, which is made visible in Figure 20.

With the weighted positioning accuracy as a cost function, the weighting function’s impact on the satellite elevations also can be seen off the plot. Notably, the DOP increases really fast if the satellite’s elevation gets close to zero. Figure 21 shows the 25° setup again, but for the σ_G cost function.

6.2 Characterization of the simulation results

In order to have significant results available to the simulations with different cost functions and numbers of satellites, each simulation type was run multiple times, and the resulting data was saved to disk. Multiple simulation runs with the same starting configuration could then be compared among themselves. The gained data is processed by different statistical methods, in order to characterize solutions for different scenarios. The main aim of this simulation campaign is to find out which resulting characteristics stay the same for a field of configuration values, and which change. It is also tried to interpret the reasons for the observed behavior using this data.

6.2.1 Characteristics

In the following tables, some interesting characteristic values can be read off easily for each constellation size out of $K \in \{4; 5; 6; 7; 8; 10; 12; 14\}$ satellites. Therefore, the large number of available statistical values which are computed from the simulation data have been compressed into significant “markers”. Those markers will be described below. Some of the markers refer to the whole solution, while other exist for each class of satellites. For the latter, the subscript H, M, or Z assigns them to their particular class.

These classes are used as a means to divide the provided position solution into distinguishable parts. It was observed before, that the returned satellite positions tend to be located either near the horizon, or near the zenith; alternatively, the horizon positions move a little bit away from the horizon when the weighted position solution is used. Therefore, each satellite is assigned to one of three classes H, M, or Z, depending on its elevation. Good thresholds were found to be 1° for the upper limit of “Horizon” satellites, and 80° for the lower limit of “Zenith” satellites. The latter value might appear quite distant from the real zenith; however, satellites sharing their position in the zenith frequently arrange on a small circle around the zenith, making the elevation fall below 85° if enough satellites share the spot.

To understand the markers and their statement on the characteristics of the individual simulation setup, the overview below gives some explanations to the abbreviated markers used in the tables.

K: The total number of satellites available in this setup.

Rows H, M and Z: Each satellite position is put into one of three classes — the Horizon Class (H), if the satellite has a very low elevation ($E \in [0\ 1°]$); the Zenith Class (Z), if its elevation is close to the zenith ($E \in [80°\ 90°]$); and the so-called Medium Class (M) for all remaining satellites ($E \in]1°\ 80°[$). Different examinations are then done per-class.

$\mathbf{K}_H$, $\mathbf{K}_Z$ and $\mathbf{K}_M$: Number of satellites per class. The value is given as the mean calculated over all simulation runs.

CC: Stability of the class counts. If the satellite count within a particular class changes between individual simulation runs, the overall result can be considered less stable. The marker is derived from the sum of the variances of K_H, K_M and K_Z. It can be set to "S" if the class count is considered stable (low variance); "U" if it is considered unstable (high variance); and "M" for a marginally stable result, when only one simulation result differs from all the others. The thresholds have been picked such that a marginal stability is usually indicated, if only a small portion of the simulations result in a different position (statistical outlier), to distinguish that from cases where the partitioning is really unstable.

$\mathbf{E}_{mean,\{H|M|Z\}}$: Mean elevation in each class: Describess the average angle elevation of the classes' satellites. This value is especially interesting when weighted positioning variances is used as the cost function.

$\mathbf{ELST}_{\{H|M|Z\}}$: Stability of the Elevation. This marker shows if the mean elevation of a classes' satellites stays the same between simulation runs. It can be considered another indicator of the solution's stability; its values are again "U","S" or "M". A dash "-" denotes that a marker is not applicable, because no satellites are found in this class. The marker's setting is based on the evaluation of the variance over the simulation-run-wise elevation mean; its most interesting use lies in examining the M class satellites, naturally. However, it can also give information about the other classes' satellites' behavior.

$\mathbf{ELUN}_{\{H|M|Z\}}$: Elevation uniformity per simulation run. This marker is derived from each simulation run's elevation variance in the particular class. It says "S" if all satellites are at the same elevation, or "U" for solutions where not all satellites in the class have the same elevation. "M" denotes the edge case, again.[9]

$\mathbf{AZSP}_{\{H|M|Z\}}$: Stability of the azimuth spacing: The variance of the azimuth difference between satellites in a class describes if they are deployed equidistantly on their circular line. Generally, this is more likely to be the case when the number of satellites is low.

COST: Stability of the least cost value. The marker is based on the resulting cost's variance.

[9] "Unstable" elevation uniformity will happen if there are no satellites in other classes, for instance with HDOP — otherwise the matrix would become singular.

STEP: Stability of elapsed simulation steps. The SimAnn algorithm terminates automatically, when it considers the system in a stable state. Thus, the variance on the elapsed steps can give information about whether the simulations run similarly every time, or whether random influences disturb it a lot.

The analysis was done individually for each subset of the two simulation parameters "Number of satellites" and "Cost function". It was observed that a number of ten passes for each setup was enough to find out about the general characteristics. For the unstable cases like TDOP, further investigation could use a larger number of simulation passes to generate more exact statistical values. The feature of instability, however, is obvious already after ten simulations.

It was observed that the choice of the simulation algorithm did not change the results in any case. Since the Simulated Annealing algorithm terminates itself when stability is reached, its use was preferred in the simulation campaign. Random samples with the Noisy Downhill algorithm showed that this assumption was correct. Since the resulting positions were the same. they are not demonstrated here.

6.2.2 Simulation results with DOP cost functions

In the following, a tabular overview of all simulation runs is given for every cost function that was implemented. Since NDOP shows the behavior of EDOP turned by $\frac{\pi}{2}$, NDOP will be neglected in this overview. The same does apply for the discussion of the weighted positioning accuracy σ_N.

The first cost function being examined is **GDOP**. It combines all diagonal elements of the inverted matrix, so it gives a good rating about the general quality of a satellite constellation in terms of geometry. Table 4 shows a quite stable partitioning property for all values of K. If only 4 satellites are visible, the best geometry is obtained with one satellite at the zenith, and three distributed equidistantly at the horizon. For 5 satellites, the most likely situation is two satellites in the zenith. The marker AZSP$_H$ shows that the two satellites are equally spaced at an elevation of around 83°.

The azimuth spacing behavior of the horizon satellites is stable for $K < 8$. For 8 or more satellites, they line up more chaotically, which can be attributed to the better geometry in general, making the cost function flatter. This is probably also the reason for one satellite assigned to the M class in one single simulation for $K = 7$.

The reader is encouraged to view the detailed simulation results with the provided MATLAB function `DisplayCampaignResults(FILENAME)`. The function plots all resulting sky views, and gives the numerical statistics as well as the derived markers. The simulation results used for this overview are located in the subfolder `SimCampaign/`.

Table 4: Simulation Results for GDOP

K	**CC**	$\mathbf{K_Z}$	$\mathbf{E_{mean,Z}}$	$\mathbf{ELST_Z}$	$\mathbf{ELUN_Z}$	$\mathbf{AZSP_Z}$
	COST	$\mathbf{K_M}$	$\mathbf{E_{mean,M}}$	$\mathbf{ELST_M}$	$\mathbf{ELUN_M}$	$\mathbf{AZSP_M}$
	STEP	$\mathbf{K_H}$	$\mathbf{E_{mean,H}}$	$\mathbf{ELST_H}$	$\mathbf{ELUN_H}$	$\mathbf{AZSP_H}$
GDOP						
4	S	1	89.8°	S	-	-
	S	0	n/a	-	-	-
	S	3	0°	S	S	S
5	M	1.9	83.3°	M	S	S
	S	0	n/a	-	-	-
	S	3.1	0°	S	S	S
6	S	2	89.7°	S	S	S
	S	0	n/a	-	-	-
	M	4	0°	S	S	S
7	M	2	89.4°	M	M	S
	M	0.1	1.3°	S	-	-
	M	4.9	0.0163°	S	S	S
8	M	2.4	88.8°	M	M	U
	M	0	n/a	-	-	-
	M	5.6	0°	S	S	U
10	S	3	89.8°	S	S	U
	S	0	n/a	-	-	-
	M	7	0°	S	S	U
12	S	4	89.8°	S	S	U
	S	0	n/a	-	-	-
	M	8	0°	S	S	U
14	M	4.2	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	9.8	0°	S	S	U

The **PDOP** cost function (Table 5) can be compared to GDOP very well; in fact it only lacks the time component, which is always set to one by definition (see Section 3.4). The resulting characteristics should not differ too much therefore. Surprisingly, the overall stability of the setup appears a little worse than with GDOP; it is not fully understood why this is the case. One outstanding feature is that in two cases, a "lost satellite" in the M class is quite likely for $K = 7$ and $K = 10$, in 4 of 10 cases for the former, and 1 of 10 cases for the latter constellation size ($K_M = 0.4$ and $K_M = 0.1$, respectively).

Table 5: Simulation Results for PDOP

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
PDOP						
4	S	1	89.9°	S	-	-
	S	0	n/a	-	-	-
	M	3	0°	S	S	S
5	M	1.7	83.1°	M	S	S
	M	0.2	69.9°	S	M	S
	M	3.1	0°	S	S	S
6	S	2	89.7°	S	S	S
	S	0	n/a	-	-	-
	M	4	0°	S	S	S
7	U	1.8	88.3°	M	S	S
	M	0.4	71.6°	M	M	S
	M	4.8	0°	S	S	S
8	M	2.5	89.5°	M	M	U
	S	0	n/a	-	-	-
	M	5.5	0°	S	S	M
10	M	3	89.4°	M	M	U
	M	0.1	75.9°	S	-	-
	M	6.9	0°	S	S	U
12	S	4	89.8°	S	S	U
	S	0	n/a	-	-	-
	M	8	0°	S	S	U
14	S	5	89.7°	S	S	U
	S	0	n/a	-	-	-
	M	9	0°	S	S	M

HDOP is the combined DOP type which contains only the horizontal contributions EDOP and NDOP (Table 6). Geometry will be considered good, if the satellites are on low elevations. A satellite in the zenith does not contribute anything to horizontal accuracy. Therefore, it is no surprise that the mean elevation of all satellites for the HDOP optimization lies below 5°. Still, the satellites do not all have 0°elevation. This behavior roots in the contradiction between perfect geometry and the need for a determined equation system. If all satellites lie in the horizon plane, the resulting matrix $\mathbf{H}^T\mathbf{H}$ becomes singular, and no position solution can be found at all. In the results, the overall mean of the ratio $\frac{K_M}{K_H}$ is 1 : 1, although it varies a lot between the constellation sizes. This value, however, can be considered non-significant because of the small elevation difference between the two classes.

The azimuth spacing is only stable for very low numbers of satellites.

Table 6: Simulation Results for HDOP

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
HDOP						
4	S	0	n/a	-	-	-
	S	2	4.89°	M	S	S
	M	2	0.00443°	S	S	S
5	M	0	n/a	-	-	-
	S	3.2	4.33°	M	M	U
	M	1.8	0.11°	S	M	S
6	U	0	n/a	-	-	-
	S	3.4	4.42°	M	M	U
	M	2.6	0.0763°	M	S	M
7	U	0	n/a	-	-	-
	S	2.7	4.34°	M	M	U
	M	4.3	0.081°	S	M	U
8	U	0	n/a	-	-	-
	S	3.3	4.19°	M	M	U
	M	4.7	0.044°	S	S	U
10	U	0	n/a	-	-	-
	S	4.2	3.84°	M	M	U
	M	5.8	0.0761°	S	S	U
12	U	0	n/a	-	-	-
	S	3.9	3.84°	M	M	U
	M	8.1	0.0546°	S	S	U
14	U	0	n/a	-	-	-
	S	4.4	3.45°	M	M	U
	M	9.6	0.0627°	S	S	U

The cost function **EDOP** results in a very special constellation — since the resulting cost describes only the geometry's quality wrt the x axis (see 4.1.1), the resulting satellite positions line up nearly linearly in an East-West axis. The need for a determined equation system inhibits a fully linear setup, again. All the satellites are equally allocated to two mid-size areas located on the horizon, in the very east and in the very west, respectively. This assumption holds for all constellation sizes (see sky view in Figure 16)). Roughly, half of the satellites stick to the horizon, while the rest is elevated by around 5°.

The same assumptions can be made to the **NDOP** cost function, where the whole system is rotated by $\frac{\pi}{2}$ in azimuth. Hence, the table for NDOP does not show up here; still, the results are proven to be the same.

Table 7: Simulation Results for EDOP

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
EDOP						
4	M	0	n/a	-	-	-
	S	2.9	6.28°	M	M	U
	M	1.1	0.0547°	S	S	S
5	U	0	n/a	-	-	-
	S	3.2	5.53°	M	M	U
	M	1.8	0.203°	S	M	U
6	U	0	n/a	-	-	-
	M	3.5	4.71°	M	M	U
	M	2.5	0.0848°	S	M	U
7	U	0	n/a	-	-	-
	S	3.3	4.92°	M	M	U
	M	3.7	0.0874°	S	M	U
8	U	0	n/a	-	-	-
	M	4.4	4.38°	M	M	U
	M	3.6	0.0832°	S	M	U
10	U	0	n/a	-	-	-
	M	5.7	4.1°	M	M	U
	M	4.3	0.0366°	S	S	U
12	U	0	n/a	-	-	-
	S	5.4	4.62°	M	M	U
	M	6.6	0.0794°	S	M	U
14	U	0	n/a	-	-	-
	S	5.7	3.93°	M	M	U
	S	8.3	0.0747°	S	M	U

VDOP needs only a good vertical geometry. This would, if the whole surroundings of the user were available, result in a quasi-linear arrangement over and below the user position, similar to the EDOP/NDOP. However, negative elevations are forbidden; so the satellites dedicated to provide the lower contribution get stuck on the horizon (Table 8). For $K = 4$, this results in a linear arrangement similar to EDOP, but with two satellites near the zenith. Again, they keep a certain distance to each other to prevent the matrix from becoming singular. For $K > 4$, the lower satellites arrange first equidistantly spaced, and then randomly to the horizon for higher K.

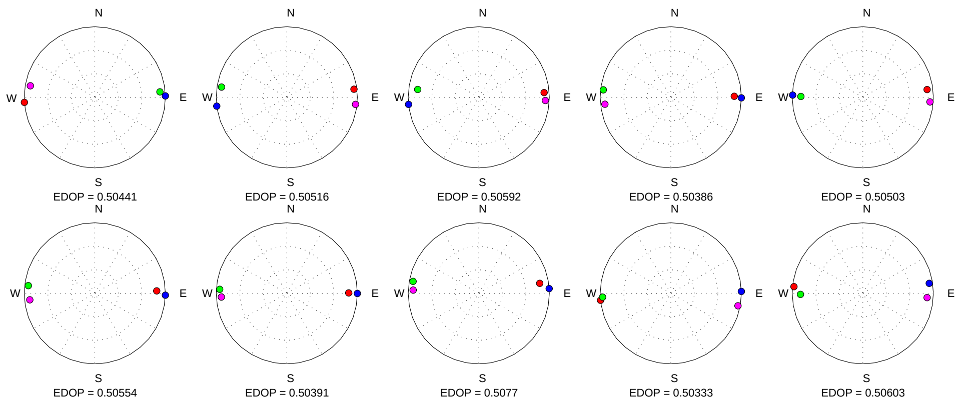

Figure 16: Sky View plots for Simulated Annealing with EDOP and 4 satellites

The last DOP cost function to be discussed is **TDOP**. The simulation results are summarized in Table 9. Here, no spatial orientation has preference to the others. For $K = 4$, three satellites arrange equidistantly at the horizon (AZSP$_H$=S), while the fourth satellite is totally randomly positioned somewhere in the sky. It seems that the cost function for this "free“ satellite is completely flat over the whole sky area, except the horizon areas. Thus, the class count is completely stable and the class ratio is $1 : (K - 1)$ for all examined K. From the STEP marker, we can see that the simulation always terminates after the same number of simulation steps — in fact, it is canceled after the hard maximum of simulation steps has been reached, because the free satellite keeps on moving.

Table 8: Simulation Results for VDOP

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
VDOP						
4	U	1.9	86.2°	M	M	S
	M	0.3	26.5°	S	U	S
	S	1.8	0°	S	S	S
5	M	2.2	89.1°	M	M	S
	S	0	n/a	-	-	-
	M	2.8	0°	S	S	S
6	M	2.9	88.9°	M	M	U
	M	0	n/a	-	-	-
	M	3.1	0°	S	S	U
7	M	3.3	89.8°	S	S	U
	S	0	n/a	-	-	-
	M	3.7	0°	S	S	U
8	M	3.8	89.7°	M	M	U
	M	0	n/a	-	-	-
	M	4.2	0°	S	S	U
10	M	4.8	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	5.2	0°	S	S	U
12	M	5.4	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	6.6	0°	S	S	U
14	M	6.7	89.9°	S	S	U
	S	0	n/a	-	-	-
	M	7.3	0°	S	S	U

Table 9: Simulation Results for TDOP

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
TDOP						
4	S	0	n/a	-	-	-
	S	1	32.1°	M	-	-
	S	3	0°	S	S	S
5	S	0	n/a	-	-	-
	S	1	24.3°	M	-	-
	S	4	0°	S	S	U
6	S	0	n/a	-	-	-
	S	1	30.7°	M	-	-
	S	5	0°	S	S	U
7	S	0	n/a	-	-	-
	S	1	31°	M	-	-
	S	6	0°	S	S	U
8	S	0	n/a	-	-	-
	S	1	35.5°	M	-	-
	S	7	0°	S	S	U
10	S	0	n/a	-	-	-
	S	1	28.8°	M	-	-
	S	9	0°	S	S	U
12	S	0	n/a	-	-	-
	S	1	39.4°	M	-	-
	S	11	0°	S	S	M
14	M	0	n/a	-	-	-
	S	0.9	29.8°	M	-	-
	S	13.1	0.00033°	S	S	U

6.2.3 Simulation results with other cost functions

This section describes the simulation results using the positioning accuracy of a weighted position solution and the matrix condition.

The weighted positioning accuracy renders results that are obviously closely related to those gained with DOP as a cost function. The main difference is, that where satellites are positioned on the horizon for a DOP, they stay away from too low elevation angles because of the σ cost functions. The reason is the influence of the weighting function, which acts contrary to the force dragging the satellite down to the horizon.

For $\sigma_{\mathbf{G}}$, the horizon satellites from GDOP moved to an average elevation of 22°(Table 10). It should be noted that the average partition ratio between zenith and horizon has changed — a larger part of the satellites stays down. This can be connected to the weighting function's influence, which greatly reduces the contribution of lower satellites to the overall position solution, and thus, also to the accuracy. Thus, more satellites are needed at lower elevation angles than in the zenith.

The $\sigma_{\mathbf{P}}$ cost function (Table 11) features a slightly higher mean elevation of the M class satellites, compared to σ_G. However, the individual satellites' elevation angles are very simular (ELUN$_M$), thus the equilibrium of geometric influence and weighting is pretty stable. The azimuthal spacing gets unstable for $K \geq 8$.

The horizontal accuracy $\sigma_{\mathbf{H}}$ creates a constellation consisting of circularly arranged satellites at a mean elevation of 38°. Notably, the elevation uniformity ELUN$_M$ shows "M“ for all K — the satellites can not all have the same elevation, as the zenith satellite is missing here and they would otherwise all lie in one plane. Table 12 shows the resulting characteristics.

$\sigma_{\mathbf{E}}$, the weighted variant of EDOP, produces results that also look like a resized version of its relative EDOP. The arrangement is quasi-linear, but the satellites now rest at an elevation of 38°. Again, the neighbor satellites keep a certain distance between them, to prevent the matrix from becoming singular.

Where the mean elevation for the best horizontal accuracy was quite high, it is really low for $\sigma_{\mathbf{V}}$. The importance of having an antipole for the zenith satellite is obviously stronger. Also, the ratio $\frac{K_Z}{K_M}$ is again lower than $\frac{K_Z}{K_H}$ in the VDOP result, meaning that more satellites are needed to provide the contribution needed from low-elevation positions.

Table 10: Simulation Results for σ_G

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_G						
4	S	1	89.7°	S	-	-
	S	3	22.2°	S	S	S
	M	0	n/a	-	-	-
5	S	1	89.4°	M	-	-
	M	4	21°	M	M	M
	S	0	n/a	-	-	-
6	M	1.4	89.7°	M	S	S
	M	4.6	21.6°	M	S	S
	S	0	n/a	-	-	-
7	M	1.7	89.7°	S	S	S
	M	5.3	21.8°	M	S	M
	M	0	n/a	-	-	-
8	S	2	89.7°	S	S	S
	S	6	22.2°	S	S	U
	M	0	n/a	-	-	-
10	M	2.1	89.7°	S	S	U
	M	7.9	21.3°	M	S	M
	M	0	n/a	-	-	-
12	M	2.8	89.8°	S	S	U
	M	9.2	21.8°	M	S	U
	M	0	n/a	-	-	-
14	S	3	89.8°	S	S	U
	S	11	21.5°	S	S	M
	M	0	n/a	-	-	-

The $\sigma_\mathbf{T}$ results exhibit an interesting change compared to TDOP: The "free satellite“ that was always somewhere in the sky is now dragged to the zenith for all values of K. This behavior can be explained through the influence of the weighting function, which now makes the satellites position most valuable for a maximum angle of elevation. For high values of K, the probability to have two satellites in the zenith also increases, because the lower positions are weighted less.

The condition of the matrix shows almost the same behavior as GDOP. The typical partitioning has twice the satellites at the horizon than at the zenith, and for small K, the horizon satellites are equally spaced. For higher satellite quantities, the horizontal distributions get more and more chaotic.

Table 11: Simulation Results for σ_P

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_P						
4	S	1	89.8°	S	-	-
	S	3	23.5°	S	M	S
	S	0	n/a	-	-	-
5	S	1	89.3°	M	-	-
	M	4	22.4°	M	M	S
	M	0	n/a	-	-	-
6	M	1.8	89.1°	M	S	S
	M	4.2	24.2°	M	M	S
	S	0	n/a	-	-	-
7	S	2	89.6°	S	S	S
	S	5	24.2°	S	S	S
	S	0	n/a	-	-	-
8	S	2	89.7°	S	S	S
	S	6	23.5°	S	S	U
	M	0	n/a	-	-	-
10	M	2.5	89.7°	S	S	U
	M	7.5	23.4°	M	S	M
	S	0	n/a	-	-	-
12	S	3	89.7°	S	S	U
	S	9	23.5°	S	S	U
	M	0	n/a	-	-	-
14	M	3.7	89.8°	S	S	U
	M	10.3	23.8°	M	S	M
	M	0	n/a	-	-	-

Table 12: Simulation Results for σ_H

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_H						
4	S	0	n/a	-	-	-
	M	4	38.8°	M	M	S
	M	0	n/a	-	-	-
5	S	0	n/a	-	-	-
	S	5	38.4°	M	M	S
	M	0	n/a	-	-	-
6	S	0	n/a	-	-	-
	S	6	38.2°	M	M	U
	M	0	n/a	-	-	-
7	S	0	n/a	-	-	-
	S	7	38.5°	M	M	U
	S	0	n/a	-	-	-
8	S	0	n/a	-	-	-
	S	8	38°	M	M	U
	M	0	n/a	-	-	-
10	S	0	n/a	-	-	-
	S	10	38.1°	M	M	M
	M	0	n/a	-	-	-
12	S	0	n/a	-	-	-
	S	12	38.1°	M	M	M
	M	0	n/a	-	-	-
14	S	0	n/a	-	-	-
	S	14	38.4°	M	M	M
	M	0	n/a	-	-	-

Table 13: Simulation Results for σ_E

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_E						
4	S	0	n/a	-	-	-
	M	4	37.3°	M	M	U
	M	0	n/a	-	-	-
5	S	0	n/a	-	-	-
	S	5	37.6°	M	M	U
	S	0	n/a	-	-	-
6	S	0	n/a	-	-	-
	M	6	37.4°	M	M	U
	S	0	n/a	-	-	-
7	S	0	n/a	-	-	-
	M	7	38.1°	M	M	U
	M	0	n/a	-	-	-
8	S	0	n/a	-	-	-
	M	8	38.2°	M	M	U
	M	0	n/a	-	-	-
10	S	0	n/a	-	-	-
	M	10	37.6°	M	M	U
	M	0	n/a	-	-	-
12	S	0	n/a	-	-	-
	M	12	38.2°	M	M	U
	M	0	n/a	-	-	-
14	S	0	n/a	-	-	-
	S	14	38.5°	M	M	U
	M	0	n/a	-	-	-

Table 14: Simulation Results for σ_V

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_V						
4	S	1	88.4°	M	-	-
	M	3	15.1°	M	M	S
	M	0	n/a	-	-	-
5	M	1.3	89.5°	M	S	S
	M	3.7	15.2°	M	S	U
	S	0	n/a	-	-	-
6	M	1.7	89.2°	M	S	S
	M	4.3	17.2°	M	M	U
	M	0	n/a	-	-	-
7	M	2.1	89.7°	S	S	U
	M	4.9	16°	M	S	U
	M	0	n/a	-	-	-
8	M	2.2	89.8°	S	S	U
	M	5.8	15.6°	M	S	U
	M	0	n/a	-	-	-
10	M	3.2	89.8°	S	S	U
	M	6.8	16.3°	M	S	U
	M	0	n/a	-	-	-
12	M	3.6	89.8°	S	S	U
	M	8.4	16°	M	S	U
	M	0	n/a	-	-	-
14	M	4.2	89.8°	S	S	U
	S	9.8	16°	M	S	U
	M	0	n/a	-	-	-

Table 15: Simulation Results for σ_T

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
σ_T						
4	S	1	89.6°	M	-	-
	S	3	18.7°	S	S	S
	S	0	n/a	-	-	-
5	S	1	89.5°	M	-	-
	S	4	18°	S	M	U
	S	0	n/a	-	-	-
6	S	1	89.8°	S	-	-
	S	5	17.5°	S	S	U
	M	0	n/a	-	-	-
7	S	1	89.8°	S	-	-
	S	6	17°	S	S	U
	S	0	n/a	-	-	-
8	S	1	89.8°	S	-	-
	S	7	16.5°	S	S	U
	M	0	n/a	-	-	-
10	M	1.4	89.8°	S	S	S
	M	8.6	16.7°	M	S	U
	M	0	n/a	-	-	-
12	M	1.7	89.8°	S	S	S
	S	10.3	16.8°	M	S	U
	M	0	n/a	-	-	-
14	M	1.9	89.9°	S	S	U
	M	12.1	16.7°	M	S	U
	M	0	n/a	-	-	-

Table 16: Simulation Results for COND

K	**CC** **COST** **STEP**	$\mathbf{K_Z}$ $\mathbf{K_M}$ $\mathbf{K_H}$	$\mathbf{E_{mean,Z}}$ $\mathbf{E_{mean,M}}$ $\mathbf{E_{mean,H}}$	$\mathbf{ELST_Z}$ $\mathbf{ELST_M}$ $\mathbf{ELST_H}$	$\mathbf{ELUN_Z}$ $\mathbf{ELUN_M}$ $\mathbf{ELUN_H}$	$\mathbf{AZSP_Z}$ $\mathbf{AZSP_M}$ $\mathbf{AZSP_H}$
COND						
4	S	1	89.8°	S	-	-
	S	0	n/a	-	-	-
	S	3	0°	S	S	S
5	S	2	89.7°	S	S	S
	S	0	n/a	-	-	-
	M	3	0°	S	S	S
6	S	2	89.7°	M	S	S
	S	0	n/a	-	-	-
	S	4	0°	S	S	M
7	M	2.2	89.7°	S	S	U
	M	0	n/a	-	-	-
	M	4.8	0°	S	S	U
8	M	2.9	89.4°	M	M	U
	M	0	n/a	-	-	-
	M	5.1	0°	S	S	U
10	M	3.4	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	6.6	0°	S	S	U
12	M	4.1	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	7.9	0°	S	S	U
14	M	4.8	89.8°	S	S	U
	M	0	n/a	-	-	-
	M	9.2	0°	S	S	U

7 Conclusion

The scope of this assignment was to find optimal satellite positions in a GNSS, with respect to the various DOP functions. The first approach was to analyze the geometry matrix manually and to find out which constellations are good for DOP "by inspection". This attempt, as it turned out, was too complicated to be finished; it would have taken thousands of operations and probably months to calculate the inverse of a geometry matrix manually, for only one constellation size. Symbolic computation done in MATLAB showed that even with the computed inverse, there is no easy way to find good position sets. Computers can do these investigations much faster, and this was done in a simulation using the Simulated Annealing algorithm.

With the resulting optimal position constellations of all DOP cost functions, and for constellation sizes between 4 and 14 satellites, the cost functions could be investigated more closely. It can be stated that the assumed complexity of DOP does not exist that strongly; instead, the preliminary estimations of optimal constellations such as the equidistant setup were indeed confirmed with the simulation results.

However, some characteristics of the solutions could not be anticipated; it was, for instance, not supposed that the aggregation of multiple satellites on one single spot, namely the zenith, could result in a cost minimum. By contrast, the situation of many satellites stuck to the horizon was foreseen and is now approved. This optimum is only of a theoretical value; in reality, very low elevation will lead to satellite masking in nearly every case; and other side effects like strong multipath and atmospheric errors make the situation even worse.

In order to examine also cost functions with practical value, the estimate accuracy derived with a weighted position solution was examined. The results could be compared to those gained from DOP simulations, with the big difference that the lower satellites have their best elevation somewhere between 15°and 35°, depending on the cost function. Other small characteristical changes in behavior due to the weighting function could be especially seen for the temporal estimate accuracy σ_T.

As a third cost function, the matrix condition of the geometry matrix was examined briefly. It exhibits a simulation behavior that is almost identical with the one known from GDOP. The correlation between both of them for a large set of random positions fortifies this observation. In fact, these two cost functions have a strong mathematical similarity, which is responsible for the congeneric results.

The results showed that the assumptions about good and bad satellite constellations, based on gut feeling could be approved in the Simulated Annealing algorithm. Furthermore, the best satellite positions seemed to be a very determined state in all analyzed cases; this could be seen from the simulation series which returned very stable minima. The algorithms capable of avoiding local minima, however, were probably the best choice to get those results.

In addition to the printed version in the appendix, the simulation's source code is provided on CD, along with the simulation results that were discussed. Every function can provide a help text which should be, together with the description in the present document, sufficient information to start own investigations on optimal satellite locations.

A References

[Gün05] Christoph Günther. Lecture "Satellite Navigation", TU München, 2005.

[KJV83] S. Kirkpatrick, C.D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), May 1983.

[ME01] Pratap Misra and Per Enge. *Global Positioning System: Signals, Measurements and Performance.* Ganga-Jamuna Press, 2001.

[MRR$^+$53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1953.

[PM97] M. Pachter and J. B. McKay. Geometry Optimization of a GPS-Based Navigation Reference System. In *NAVIGATION - Journal of the Institute Of Navigation*, volume 44, pages 457–470. ION, 1997.

[WE95] Todd Walter and Per Enge. Weighted RAIM for Precision Approach. ION GPS-95. ION, ION, 1995.

B Source code

B.1 CorrectPositionMatrix

```
function pos=CorrectPositionMatrix(pos)
% CORRECTPOSITIONMATRIX catch bound violations of satellite
    positions and correct
% POS=CORRECTPOSITIONMATRIX(POS)
%       checks for all elevation and azimuth arguments in POS and
%       corrects them if they are out of bounds.
%       The matrix is NOT oriented.
% See also ORIENTPOSITIONMATRIX

global ELEV_LOWLIMIT


% correct bound violations (too high/low elevation or azimuth)
for i=1:size(pos,1)
    % check for too high elevation
    if pos(i,1)>pi/2
        % flip over the azimuth (+pi)
        pos(i,2)=pos(i,2)+pi;
        % correct the elevation
        pos(i,1)=pi-pos(i,1);
    end
    % check for too low elevation (beyond horizon)
    if pos(i,1) < ELEV_LOWLIMIT
        pos(i,1)= ELEV_LOWLIMIT;
    end

    % azimuth
    if (pos(i,2)<0) || (pos(i,2)≥2*pi)
        pos(i,2)=mod(pos(i,2),2*pi);
    end
end
```

B.2 deg2rad

```
function rad=deg2rad(deg)

rad=deg*pi/180;
```

B.3 DOP

```
function DOPValue = DOP(positions, doptype)
% DOP   compute the DOP value to a given set of satellite
   positions
%
% DOPVALUE=DOP(POSITION_VECTOR, DOPTYPE)
%   returns the DOP value
%       of the specified type.
%       These include
%           DOP:    EDOP, NDOP, VDOP, TDOP, or one of
%                   the combined values HDOP, PDOP or GDOP
%           SGM:    SGME, SGMN, SGMV, SGMT, SGMH, SGMP, SGMG
%                   cost function using weighted positioning
%           COND:   Matrix condition cond(H) as a measure for the
%                   constellation quality.
%           ALL:    returns a row vector of all DOP types
%
%       POSITION_VECTOR = [ E1 A1 ; E2 A2 ; ... En An ]
%       contains the positions of the satellites in elevation and
%       azimuth, one in a row. The values must be radian within
    the
%       bounds 0≤A<pi for all azimuths, and 0≤E≤pi/2 for
    elevation.
%
%       DOPTYPES
%       DOP can return the specified scalar DOP value if a DOPTYPE
     is
%       given. If the DOPTYPE value is left out, DOP returns the
    vector
%       DOPValue = [ EDOP NDOP VDOP TDOP HDOP PDOP GDOP ]
%
%   With the concept of the weighted position solution implemented
    ,
%   further DOPTYPEs include 'SGMX', where X can be E,N,V,T,H,P,
    or G.
%       In this case, DOP does _not_ return a real DOP value, but
    a standard
%   deviation σ, which contains the particular DOP as well as a
%       standardized term of error contributions. The point is,
    that in this
%       case, the elevation of the particular satellite is
    considered as a
%       measure for the error, and thus, satellites closer to the
    horizon do
%       not contribute as much to the position solution as the
    higher ones do.
%       Minimizing the returned value wrt position returns
    positions optimized
%       for weighted positioning.
%
% ——— BEGIN PARAM CHECK ———
```

```
error(nargchk(1,2,nargin,'struct'));
num_satellites=size(positions,1);
%if num_satellites < 4
%    % to few satellite parameters given - we need at least 4
%    disp('please supply a larger number of satellites (at least
    4)!')
%    return
%end
if size(positions,2)≠2
    disp('wrong number of arguments in the positon matrix!')
    disp('the desired form is:')
    disp('positions =')
    disp('   E1 A1')
    disp('   E2 A2')
    disp('   .. ..')
    disp('   En An')
    disp('               n≥4')
    return
end
% check bounds of positions
%if min(positions(:,1))<0
%    disp('One of the satellites has an elevation less than 0
    degrees!')
%    return
%end
%if max(positions(:,1))>(pi/2)
%    disp('One of the satellites has an elevation greater than 90
    degrees!')
%    return
%end
%if min(positions(:,2))<0
%    disp('One of the satellites has an azimuth less than 0
    degrees!')
%    return
%end
%if max(positions(:,2))≥2*pi
%    disp('One of the satellites has an azimuth equal or greater
    than 360 degrees!')
%    return
%end

% check the requested doptype
if nargin==2
    switch lower(doptype)
    case {'ndop','edop','vdop','tdop','hdop','pdop','gdop','all','
        cond'}
    % conventional DOP or condition
    weighted=false;
    case {'sgmn','sgme','sgmv','sgmt','sgmh','sgmp','sgmg'}
    weighted=true;
    otherwise
        disp('No valid DOP type selected!')
        disp('See help for SIMANN')
        return
    end
else
    doptype='all';
    weighted=false;
end
% ——— END PARAM CHECK ———

% ——— BEGIN COMPUTATION ———
%tic

% set up the geometry matrix
H=zeros(num_satellites,4);
for l=1:num_satellites
    % pos(l,1) is elevation of sat. #l
    % pos(l,2) is azimuth
    H(l,:)=[ cos(positions(l,1))*cos(positions(l,2)) ...
             cos(positions(l,1))*sin(positions(l,2)) ...
                                 sin(positions(l,1)) ...
             1 ];
end

%if weighted positioning is requested, compute the satellites' variances
%  into W. Otherwise, W is the identity matrix so we dont have to change
%  anything else below.

if weighted==true
    Ref_SSQ=Weighted_SigmaSQ(pi/2);
    for l=1:num_satellites
        E=positions(l,1);
        % trap for singularity with zero—elevation sats
        if E≠0
            W(l,l)=1/(Weighted_SigmaSQ(E)/Ref_SSQ);
        else
            % variance would be inf, so we set its inverse to zero.
            % this means that in the calculation of (H^TWH)^-1
            % below, the particular satellite is canceled out.
            W(l,l)=0;
        end
    end
else
    W=eye(num_satellites,num_satellites);
end

% The inverse is always computed with the covariance matrix included
if ¬strcmp(lower(doptype),'cond')
    HtH_inv=inv(transpose(H)*W*H);
end

switch lower(doptype)
case {'edop','sgme'}
```

```
    DOPValue=sqrt(HtH_inv(1,1));
case {'ndop','sgmn'}
    DOPValue=sqrt(HtH_inv(2,2));
case {'vdop','sgmv'}
    DOPValue=sqrt(HtH_inv(3,3));
case {'tdop','sgmt'}
    DOPValue=sqrt(HtH_inv(4,4));
case {'hdop','sgmh'}
    DOPValue=sqrt(HtH_inv(1,1)+HtH_inv(2,2));
case {'pdop','sgmp'}
    DOPValue=sqrt(HtH_inv(1,1)+HtH_inv(2,2)+HtH_inv(3,3));
case {'gdop','sgmg'}
    DOPValue=sqrt(HtH_inv(1,1)+HtH_inv(2,2)+ ...
                                    HtH_inv(3,3)+HtH_inv(4,4));
case 'all'
    DOPValue= [ sqrt(HtH_inv(1,1)) ...
                sqrt(HtH_inv(2,2)) ...
                sqrt(HtH_inv(3,3)) ...
                sqrt(HtH_inv(4,4)) ...
                sqrt(HtH_inv(1,1)+HtH_inv(2,2)) ...
                sqrt(HtH_inv(1,1)+HtH_inv(2,2)+HtH_inv(3,3)) ...
                sqrt(HtH_inv(1,1)+HtH_inv(2,2)+ ...
                                    HtH_inv(3,3)+HtH_inv(4,4)) ];
case 'cond'
    DOPValue=cond(H);
end
%toc
```

B.4 DopGradient_DQ

```
% DOPGRADIENT_DQ    return the numerical gradient in respect to a
    specified
%           parameter
%           Uses a numerically determined difference quotient
% GRADIENT = DOPGRADIENT_DQ(POS,PARAMETER,DOPTYPE)
%
%   where POS denotes the position matrix (k x 2)
%   PARAMETER contains the "position" of the parameter for which
%       to find the gradient as a vector denoting row and column,
%       e.g. [1 2] for azimuth (2nd col) of the first satellite (1
    st row)
%   DOPTYPE is a string giving the type of cost function to use.
    It can be
%       'EDOP','NDOP','VDOP','TDOP','HDOP','PDOP','GDOP'
%

function gradient = DopGradient_DQ(pos,parameter,doptype)

% find the difference quotient to requested parameter
%   we compute it in both directions and return the average, so in
     case
%   of a local extremum, the returned value is likely to be zero

DELTA=pi/18000; % 1/100 degree

% position 'right' of the original one - add DELTA to the free
    parameter
posplus=pos;
posplus(parameter(1),parameter(2))=pos(parameter(1),parameter(2))+
    DELTA;

% same for the 'left' side
posminus=pos;
posminus(parameter(1),parameter(2))=pos(parameter(1),parameter(2))
    -DELTA;

% compute the function values for every position
yplus=DOP(posplus,doptype);
yminus=DOP(posminus,doptype);

% return the gradient
gradient=(yplus - yminus) / ( 2 * DELTA );
```

B.5 EquidistantSetup

```
% EQUIDISTANTSETUP      build a uniform satellite constellation
%
% POS = EQUIDISTANTSETUP (NUMSATS [,ELEVATION]) returns a matrix
    containing
%    the satellite setup of one satellite at the zenith plus
%    NUMSATS-1 satellites equally spaced on the horizon.
%
%    An optional second argument 0 ≤ ELEVATION < (pi/2) denotes
%    the elevation of the satellites (excluding the one at the
%    zenith), which is otherwise set to zero.
%
%   See also RANDOMSATELLITESETUP

function POS=EquidistantSetup(SatNumber,Elevation)
if nargin==1
    Elevation=0;
else
    if (Elevation<0) | (Elevation≥(pi/2))
        disp('ERROR: Elevation is out of bounds!')
        return
    end
end
if SatNumber<2
    disp('ERROR: at least 2 satellites are required')
    return
end

for i=1:SatNumber-1
    Azimuth=(i-1)*2*pi/(SatNumber-1);
    POS(i,:)=[ Elevation Azimuth ];
end
% add the satellite in the zenith
POS=[ POS; pi/2  0 ];
```

B.6 Geometry_Matrix

```
function H=Geometry_Matrix(POS)
% GEOMETRY_MATRIX   compute the geometry matrix to a position
    matrix
%
% H = GEOMETRY_MATRIX(POS)
%

% set up the geometry matrix
H=zeros(size(POS,1),4);
for l=1:size(POS,1)
    % pos(l,1) is elevation of sat. #l
    % pos(l,2) is azimuth
    H(l,:)=[ cos(POS(l,1))*cos(POS(l,2)) ...
             cos(POS(l,1))*sin(POS(l,2)) ...
                                 sin(POS(l,1)) ...
             1 ];
end
```

B.7 MultiSim

```
function SimResults = MultiSim(NumPasses)

% MULTISIM    batched simulation wrapper
%
% SIMRESULTS = MULTISIM(NUMPASSES)
%
%       simulates NUMPASSES times without graphical outputs. The simulation
%       results (optimal position and corresponding cost) are returned in
%       the cell array SIMRESULTS, one sim result per row:
%           SIMRESULTS =
%               [ BESTPOS_1     BESTCOST_1      SIMTIME_1 ]
%               [ BESTPOS_2     BESTCOST_2      SIMTIME_2 ]
%               [ ..            ..              ..        ]
%               [ BESTPOS_N     BESTCOST_N      SIMTIME_N ]
%
%       where SIMTIME contains the simulation duration the pass took.
%
%       All configuration regarding the individual simulations, such as
%       cost function, number of satellites, can be provided directly to
%       SIMANN via the global config options mechanisms.
%
%   See also SIMANN

global SIMANN_LIVEUPDATES;
SIMANN_LIVEUPDATES='NONE';

SimResults=cell(NumPasses,3);

for pass=1:NumPasses
    disp(['------------------------ SIMULATION ' num2str(pass) ' OF ' num2str(NumPasses) ' ------------------------'])
    [foo bar SimResults{pass,1} SimResults{pass,2} SimResults{pass,3}] = SimAnn;
end
```

B.8 NeighborState_Grad

```
function returnPosition = NeighborState_Grad(pos,doptype)
% NEIGHBORSTATE_GRAD    return better neighbor state to a
    satellite position set
%
% RETURNPOSITION = NEIGHBORSTATE_GRAD(POS,DOPTYPE)
%
%   returns a position matrix of the same dimensions as the given
%   position POS.
%   For each argument, the cost gradient is computed individually.
%       The argument is then shifted towards a lower cost. If the
    derivative is zero,
%       argument the position is kept.
%       For a nonzero derivative, the position is shifted linearly
     with the derivative.
%
%   See also NEIGHBORSTATE_RANDOM

% init variable
returnPosition=zeros(size(pos,1), size(pos,2), 'double');

% coefficient for the movement speed
MOVINGSPEED=0.05;

% for every argument, move towards a better solution
% we can only do this sequentially by computing the gradient at
    the current
% position in respect to one of the arguments, and move that
    argument downhill.
for i=1:size(pos,1)
    EGradient=DopGradient_DQ(pos,[i 1],doptype);
    AGradient=DopGradient_DQ(pos,[i 2],doptype);
    % compute the new position. atan is used to prevent too high
    % gradients from messing up the new position. High gradients
        may occur
    % if the matrix is nearly singular.
    returnPosition(i,:)=[ pos(i,1)-atan(EGradient)*MOVINGSPEED ...
                  pos(i,2)-atan(AGradient)*MOVINGSPEED ];
end

returnPosition=CorrectPositionMatrix(returnPosition);
```

B.9 NeighborState_Random

```
function returnPosition = NeighborState_Random(pos,stddev)
% NEIGHBORSTATE_RANDOM  return random neighbor state to a satellite position set
%
% RETURNPOSITION = NEIGHBORSTATE_RANDOM(POS,SIGMA)
%
%   returns a position matrix of the same dimensions as the given
%   position POS, which is randomly chosen around POS using a
%   gaussian probability density function.
%   The parameter SIGMA controls the average distance between given
%   position and returned neighbor position. The standard deviation
%   is kept independent from elevation and azimuth of the given positions.
%
%   See also NEIGHBORSTATE_GRAD

returnPosition=zeros(size(pos,1),size(pos,2),'double');

% avoid too high stepwidths.
% 3 sigma < 45 degrees
SIGMABOUND=pi/12;
if stddev>SIGMABOUND
    stddev=SIGMABOUND;
end

% put up the new position
for i=1:size(pos,1)
    % get a random "step vector", rotated into position
    d=RotateVector( [0; tan(randn*stddev); tan(randn*stddev) ] , ...
                -pos(i,1),pos(i,2));

    % [px py pz] is the cartesian (old) position vector
    [px py pz]=sph2cart(pos(i,2), pos(i,1), 1);

    % add to position (in cartesian coords)
    newvector_cart=[px; py; pz] + d;

    % convert back to spherical coordinates
    [ A, E, R ] = cart2sph(newvector_cart(1), newvector_cart(2), newvector_cart(3));

    % we need only elevation and azimuth back from that
    returnPosition(i,:)=[E A];
end

% Correct again because orientation will shift azimuths
    % second: orient corrected position
    % first: correct
returnPosition=CorrectPositionMatrix(...
    OrientPositionMatrix(...
    CorrectPositionMatrix(returnPosition)));
```

B.10 OrientPositionMatrix

```
% ORIENTPOSITIONMATRIX  orient a position matrix
%
% POS=ORIENTPOSITIONMATRIX(POS)
%   modifies the position matrix such that the first satellite's
    azimuth
%   is always zero

% The global variable EASTSAT is the number of the satellite that
    is used for
% orientation. It may be changed by the function when the old
    satellite
% walks into the zenith.
function pos=OrientPositionMatrix(pos)

global EASTSAT;
% trap external calls, where EASTSAT is not defined as global
if isempty(EASTSAT)
    EASTSAT=1;
end

if EASTSAT>size(pos,1)
    EASTSAT=1;
end

if EASTSAT>0   % correct only if not zero (used for EDOP/NDOP/...)
% if the satellite's elevation is too high
    if pos(EASTSAT,1)>deg2rad(75)
        % calculate the "distance" from the north horizon
        % the tangens is added to cancel out the highly al
        eastdistances=abs(mod((pos(:,1)+pi),2*pi) -pi )+0.85*tan(
            pos(:,2));
        % set the nearest satellite to be used for orientation
        [FOO EASTSAT] = min(eastdistances);
        %disp(['Changed orientation satellite to No. ' EASTSAT]);
    end

    % set the first azimuth to the north
    pos(:,2)=pos(:,2)-pos(EASTSAT,2);
end
```

B.11 Plot_Comparison_GDOP_COND

```
function FH = Plot_Comparison_GDOP_COND (varargin)
% PLOT_COMPARISON_GDOP_COND     plot GDOP and COND for a set of
    randomized
%                               sat constellations
%
% FIGUREHANDLE = PLOT_COMPARISON_GDOP_COND ([ NUMSATS [, NUMSETS]
    ])
%
%   returns a figure handle to the created plot.
%
%   NUMSATS defines the number of satellites to consider,
    optionally.
%   Defaults to 4.
%
%   NUMSETS defines the number of satellite constellations to
    calculate.
%   It defaults to 500 and can only be defined as second
    parameter.


% parameter handling
if nargin≥1
    NumSats=varargin{1};
else
    NumSats=4;
end

if nargin≥2;
    NumSets=varargin{2};
else
    NumSets=500;
end

MarkerColor=['b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ];
MarkerShape=['o' 'x' 'd' 'p' 'h' ...
'o' 'x' 'd' 'p' 'h' ...
'o' 'x' 'd' 'p' 'h' ...
'o' 'x' 'd' 'p' 'h' ...
'o' 'x' 'd' 'p' 'h'];

% allocate the vars first
GDOPS=NaN(1,NumSets);
```

```
CONDS=NaN(1,NumSets);

FH=figure;
if size(NumSats)==1
    title(['Plot of matrix condition against GDOP for ' int2str(
        NumSets) ' sets of ' int2str(NumSats) ' random satellite
        positions']);
else
    title(['Plot of matrix condition against GDOP for ' int2str(
        NumSets) ' sets of random satellite positions']);
end
hold on;

for P=1:length(NumSats  )

    % compute NumSets value pairs (GDOP,COND)
    for SatSet=1:NumSets
       Pos=RandomSatelliteSetup(NumSats(P));
       GDOPS(SatSet)=DOP(Pos,'GDOP');
       CONDS(SatSet)=DOP(Pos,'COND');
    end

    MED_GDOPS=median(GDOPS);
    MED_CONDS=median(CONDS);

    %text(MED_GDOPS,MED_CONDS,[num2str(NumSats(P)) ' satellites'])
        ;

    lshandle(P)=plot(GDOPS,CONDS,[MarkerColor(P) MarkerShape(P)]);

    maxxscale(P)=2*fix(median(GDOPS));
    maxyscale(P)=2*fix(median(CONDS));

    xlabel('GDOP');
    ylabel('cond (H)');
    set(lshandle(P),'DisplayName',[num2str(NumSats(P)) '
        satellites']);
end

axis([0 median(maxxscale) 0 median(maxyscale)]);
legend('Location','EastOutside');
```

B.12 Plot_DOP_surf

```
% PLOT_DOP_SURF create a DOP plot for one free satellite
%
%   FIGHANDLE = PLOT_DOP( POS [, DOPTYPE])
%
%   plots a 3-D surface of the value of DOP if the last satellite
%   in POS is moved around any possible locations. All but the
    last
%   satellite stay fixed at their given spots.
%
%   DOPTYPE may specify any valid computation type and is simply
%   handed over to DOP.
%

function returnfigurehandle=Plot_DOP_surf(pos,doptype)

if nargin==1
    doptype='all';
end

% create a new figure
returnfigurehandle=figure;

title('inverse DOP plot for one free satellite')

% basic plot grid
X=-1:0.02:1;
Y=-1:0.02:1;
if strcmp(lower(doptype),'all')
    EDOP=zeros(length(X),length(Y));
    NDOP=zeros(length(X),length(Y));
    VDOP=zeros(length(X),length(Y));
    TDOP=zeros(length(X),length(Y));
    HDOP=zeros(length(X),length(Y));
    PDOP=zeros(length(X),length(Y));
    GDOP=zeros(length(X),length(Y));

    % create six plots and select upper left
    skyviewhandle=subplot(4,2,1);
else
    Z=zeros(length(X),length(Y));

    % create two subplots, select left one
    skyviewhandle=subplot(1,2,1);

end
% show the positions of the satellites
skyviewhandle=SetupSkyView(skyviewhandle);
title(skyviewhandle,'Positions of the fixed satellites')
SkyView(pos(1:end-1,:));

```

```
% the last satellite is now altered

for xcount=1:length(X)
    for ycount=1:length(Y)
        [TH R]=cart2pol(X(xcount),Y(ycount));
        % check whether inside the circle
        if R≤1
            % substitute satellite
            pos(end,:)=[(1−R)*pi/2 TH];
            % calculate cost
            if strcmp(lower(doptype),'all')
                dopval=dop(pos);
                EDOP(xcount,ycount)=dopval(1);
                NDOP(xcount,ycount)=dopval(2);
                VDOP(xcount,ycount)=dopval(3);
                TDOP(xcount,ycount)=dopval(4);
                HDOP(xcount,ycount)=dopval(5);
                PDOP(xcount,ycount)=dopval(6);
                GDOP(xcount,ycount)=dopval(7);
            else
                Z(xcount,ycount)=DOP(pos,doptype);
            end
        else
            if strcmp(lower(doptype),'all')
                EDOP(xcount,ycount)=NaN;
                NDOP(xcount,ycount)=NaN;
                VDOP(xcount,ycount)=NaN;
                TDOP(xcount,ycount)=NaN;
                HDOP(xcount,ycount)=NaN;
                PDOP(xcount,ycount)=NaN;
                GDOP(xcount,ycount)=NaN;
            else
                Z(xcount,ycount)=NaN;
            end
        end
    end
end

if strcmp(lower(doptype),'all')
    % GDOP upper right
    plothandle=subplot(4,2,2);
    surf(plothandle,X,Y,GDOP,'FaceColor','interp','EdgeColor','
       none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'GDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,−1.05,0,'SOUTH');
    text(−1.05,0,0,'WEST');


    % 2nd row left: NDOP
    plothandle=subplot(4,2,3);
    surf(plothandle,X,Y,NDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'NDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,−1.05,0,'SOUTH');
    text(−1.05,0,0,'WEST');

    % 2nd row, right: EDOP
    plothandle=subplot(4,2,4);
    surf(plothandle,X,Y,EDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'EDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,−1.05,0,'SOUTH');
    text(−1.05,0,0,'WEST');


    % 3rd row left: HDOP
    plothandle=subplot(4,2,5);
    surf(plothandle,X,Y,HDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'HDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,−1.05,0,'SOUTH');
    text(−1.05,0,0,'WEST');

    % 3nd row, right: PDOP
    plothandle=subplot(4,2,6);
    surf(plothandle,X,Y,PDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'PDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,−1.05,0,'SOUTH');
    text(−1.05,0,0,'WEST');


    % 4th row left: VDOP
    plothandle=subplot(4,2,7);
    surf(plothandle,X,Y,VDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'VDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,-1.05,0,'SOUTH');
    text(-1.05,0,0,'WEST');

    % 4th row right: TDOP
    plothandle=subplot(4,2,8);
    surf(plothandle,X,Y,TDOP,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    title(plothandle,'TDOP for the free satellite');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,-1.05,0,'SOUTH');
    text(-1.05,0,0,'WEST');



else % only one plot
    % select right subplot
    plothandle=subplot(1,2,2);
    %contour(plothandle,X,Y,Z,50);
    surf(plothandle,X,Y,1./Z,'FaceColor','interp','EdgeColor','
        none','FaceLighting','phong');
    %set(plothandle,'ZLim',[0 20]);
    %set(plothandle,'ZLimMode','manual');
    %set(plothandle,'ZScale','log');
    %set(plothandle,'CLim',[0 20]);
    %set(plothandle,'CLimMode','manual');
    plottitle=['\textsf{' 1/upper(doptype) ' for the free satellite}' ];
    titlehandle=title(plothandle,plottitle);
    set(titlehandle,'Interpreter','latex');
    text(0,1.05,0,'NORTH');
    text(1.05,0,0,'EAST');
    text(0,-1.05,0,'SOUTH');
    text(-1.05,0,0,'WEST');
end
```

B.13 Plot_EquiDistZeroElevation

```matlab
% PLOT_EQUIDISTZEROELEVATION plots the DOPs for equidistantly spaced
%   satellites on the horizon, for different numbers of satellites.
%
% FigureHandle=Plot_EquiDistZeroElevation(lowbound,hibound,doptype)
%
%   lowbound
%       is the number of satellites to start with
%   hibound
%       is the maximum number of satellites
%   doptype
%       is handed over to the DOP() function. If it is set to 'all',
%       all DOPs are plotted

function FigureHandle=Plot_EquiDistZeroElevation(lowbound,hibound,doptype)
if nargin≠3
    disp('ERROR: Wrong number of arguments! See help');
    return;
end
% compute the DOP values
for SatNumber=lowbound:hibound
    pos=EquidistantSetup(SatNumber);
    dops(SatNumber,:)=DOP(pos,doptype)';
end

for SatNumber=1:lowbound-1
    if lower(doptype)=='all'
        dops(SatNumber,:)=[ NaN NaN NaN NaN NaN NaN NaN ];
    else
        dops(SatNumber)=NaN;
    end
end

% plot the figure
FigureHandle=figure;
if lower(doptype)=='all'
    plot(dops);
    title('DOP for k satellites (1 at zenith / (k-1) at horizon)');
    xlabel('k');
    ylabel('DOP');
    %legend('EDOP','NDOP','VDOP','TDOP','HDOP','PDOP','GDOP', ...
    %        'Location','NORTHEAST');
    DOPTYPES={'EDOP' '          / NDOP' 'VDOP' 'TDOP' 'HDOP' 'PDOP' 'GDOP'};
    for i=1:7
```

```
        dops(5,i)
        DOPTYPES{i}
        text(5,dops(5,i)+0.03,DOPTYPES{i})
    end
    grid ON;
    axis( [ 0 hibound 0 ceil(max(max(dops)))] );
else
    plot(dops);
    title({doptype,' for k satellites (zenith/horizon)'});
    xlabel('k');
    ylabel(upper(doptype));
    axis([lowbound hibound 0 ceil(max(dops))]);
    grid ON;
end
```

B.14 Plot_EquiElev_HDOP_SGMH

```
function fighandle = PLOT_EQUIELEV_DOP_WSGM(num_sats,doptype,
    sigmatype,varargin)
% PLOT_EQUIELEV_DOP_WSGM    plot the DOP and the corresponding
    weighted
%                           sigma
% FIGHANDLE = PLOT_EQUIELEV_DOP_WSGM( NUM_SATS, DOPTYPE, SIGMATYPE
     [, SCOPE] )
%
% plot the DOP and the corresponding weighted \sigma for
    equidistant
% setups with k-1 satellites elevated increasingly, while sat k is
     at zenith
%   if SCOPE = [MIN MAX] is given, it defines the plot's X axis
    minimum
%       and maximum. Else, the plot goes from 1 degree to 90
    degrees.

if nargin≥4
    bounds=varargin{1}
    XSTART=bounds(1);
    XSTOP=bounds(2);
else
    XSTART=pi/180;
    XSTOP=pi/2-pi/180;
end

granlty=pi/180; % 1 degree
i=0;

% skip 0 and 90 degrees (singular matrix)
for elevation=XSTART:granlty:XSTOP
    i=i+1;
    pos=EquidistantSetup(num_sats,elevation);;
    dops(:,i)=[ elevation; ...
            DOP(pos,doptype); ...
            DOP(pos,sigmatype) ];
end

% plot the result
fighandle=figure;
plot(rad2deg(dops(1,:)),dops(2,:),'b',rad2deg(dops(1,:)),dops(3,:)
    ,'r');
grid on
legend(upper(doptype),upper(sigmatype),'Location','North');
```

B.15 Plot_Weighting_Function

```
function [FH WEIGHTINGFACTORS] = PLOT_WEIGHTING_FUNCTION
% PLOT_WEIGHING_FUNCTION    Plot the weighing function

WEIGHTINGFACTORS=zeros(2,90);

% reference to normalize the variances to DOP (make comparable)
%Ref_SSQ=Weighted_SigmaSq(pi/2);
Ref_SSQ=1;

i=0;
for E=pi/180:pi/180:pi/2
    i=i+1;
    WEIGHTINGFACTORS(:,i)=[E; sqrt(Weighted_SigmaSQ(E)/Ref_SSQ) ];
end

FH=figure;
plot(rad2deg(WEIGHTINGFACTORS(1,:)),WEIGHTINGFACTORS(2,:));
axis([0 90 0 5]);
grid on;
xlabel('Elevation E  (deg)')
ylabel('\sigma^2_{UERE} (m)')
```

B.16 rad2deg

```
function deg=rad2deg(rad)

deg=rad*180/pi;
```

B.17 RandomSatStep

```
function returnPosition = RandomSatStep(pos,stddev)
% RANDOMSATSTEP return random neighbor state to a single satellite
     position
%
% RETURNPOSITION = RANDOMSATSTEP(POS,SIGMA)
%
%   returns a position vector describing a satellite LOS
%   which is randomly chosen around POS using a gaussian
    probability
%   density function.
%   The parameter SIGMA controls the average distance between
    given
%   position and returned neighbor position. The standard
    deviation
%   is kept independent from elevation and azimuth of the given
    positions.
%   SIGMA is to be kept below pi/12 to avoid too high values
%
%   In contrast to NEIGHBORSTATE_RANDOM, this function only steps
    a single
%       satellite, and not a whole set. Thus, the input POS should
     be of size (1x2).
%
%   See also NEIGHBORSTATE_RANDOM

% 3 sigma < 45 degrees
SIGMABOUND=pi/12;
if stddev>SIGMABOUND
    stddev=SIGMABOUND;
end

% get a random "step vector", rotated into position
d=RotateVector( [0; tan(randn*stddev); tan(randn*stddev) ] ,...
         -pos(1,1),pos(1,2));

% [px py pz] is the cartesian (old) position vector
[px py pz]=sph2cart(pos(1,2), pos(1,1), 1);

% add to position (in cartesian coords)
newvector_cart=[px; py; pz] + d;

% convert back to spherical coordinates
[ A, E, R ] = cart2sph(newvector_cart(1), newvector_cart(2),
    newvector_cart(3));

% we need only elevation and azimuth
% position is corrected before returned (bounds, flipovers)
returnPosition=CorrectPositionMatrix( [E A] );
```

B.18 RandomSatelliteSetup

```
% RANDOMSATLELITESETUP returns a position vector with (valid)
    random values.
% MYPOS = RANDOMSATELLITESETUP(NUMBEROFSATELLITES)

function positions = RandomSatelliteSetup(number)
randmat=rand(number,2);
% catch 1.000 values in the random numbers
while max(max(randmat))==1
    randmat=rand(number,2);
end

% sort order: Elevation, Azimuth
positions=sortrows([ asin(randmat(:,1))  randmat(:,2)*2*pi ],[1
    2]);

% orient the returned matrix to the north
positions=OrientPositionMatrix(positions);

% perform correction (azimuths will be shifted after orientation)
positions=CorrectPositionMatrix(positions);
```

B.19 RotateVector

```
% ROTATEVECTOR  rotate a 3 dimensional vector by elevation and
    azimuth
%
% [ XN; YN; ZN ] = ROTATEVECTOR([X; Y; Z], E, A)
%
%   where X, Y, Z gives the cartesian coordinates of the vector to
     be
%   translated.
%   E is the elevation of the rotation
%   A is the azimuth of the rotation

function vector=RotateVector(vector,e,a)
R=[ cos(a)*cos(e)   -sin(a) cos(a)*sin(e) ;...
    sin(a)*cos(e),  cos(a)  sin(a)*sin(e) ;...
    -sin(e)     0       cos(e) ];
vector=R*vector;
```

B.20 SetupSkyView

```
function figureHandle = SetupSkyView(varargin)
% SETUPSKYVIEW  Setup Skyview figure.
%       SETUPSKYVIEW([FIGHANDLE]) plots a Skyview
%       diagram for the display of satellites' positions as seen
%       from the user.
%       The circular diagram shows the zenith in the middle, and
%       the horizon as seen from the user's position at the circle
%       edges.
%       It returns a handle to the figure created, which must be handed
%       over to SkyView() when plotting data.
%
%       An optional handle to an existing figure may be provided in
%       FIGHANDLE. Otherwise, a new figure is created.
%
%       See also SKYVIEW
error(nargchk(0,1,nargin,'struct'))
% figure handle provided?
if nargin==0
    % create a new figure and set some defaults
    figureHandle=figure;
    ScreenSize=get(0,'ScreenSize');
    posx=ScreenSize(3)/2-200;
    posy=ScreenSize(4)-420;
    %set(figureHandle,'Name','Skyview');
    set(figureHandle,'Position',[ posx posy 400 400 ]);
    set(figureHandle,'ToolBar','none');
else
    figureHandle=cell2mat(varargin(1));
end
% draw the circles
horizon=0:pi/180:2*pi;
horX=cos(horizon);
horY=sin(horizon);
% horizon line
plot(horX,horY,'k');
axis image off;
hold on;
% 60 degree line
plot(horX/3,horY/3,':k');
% 30 degree line
plot(2*horX/3,2*horY/3,':k');
for diagonals=0:pi/6:pi-pi/6
    plot( [cos(diagonals) -cos(diagonals)], ...
          [sin(diagonals) -sin(diagonals)], ':k');
%    text( cos(diagonals)*1.05, sin(diagonals)*1.1, ...
%          num2str(rad2deg(diagonals)));
%    text( -cos(diagonals)*1.05-0.1, -sin(diagonals)*1.1, ...
%          num2str(rad2deg(diagonals+pi)));
```

```
end
texthandle=text( 1.1, 0, 'E');
set(texthandle,'HorizontalAlignment','center');
texthandle=text( 0, 1.1, 'N');
set(texthandle,'HorizontalAlignment','center');
texthandle=text(-1.1, 0, 'W');
set(texthandle,'HorizontalAlignment','center');
texthandle=text( 0, -1.1,'S');
set(texthandle,'HorizontalAlignment','center');

% corners to avoid wrong scaling for prints

%plot(1.1,1.1);
%plot(-1.1,-1.1);
```

B.21 SimAnn

```
function [PosTrack CostTrack returnpos returndop SimSteps SimDuration] = SimAnn
% SIMANN    Simulation to determine optimal GNSS satellite configurations
%
% [ POSTRACK COSTTRACK BESTPOS BESTCOST SIMSTEPS SIMDURATION ] = SIMANN()
%
%   If NUM_SATS is omitted, the simulation is run with 4 satellites.
%
%   The simulation returns the cost track, the best obtained position and the
%   corresponding cost function's value.
%
%   POSTRACK is a 3–D matrix of size (NUM_SATS,2,SIMTIME+1) containing
%   all the positions.
%
%   COSTTRACK is a (1,SIMTIME+1) sized row vector, containing all the
%   simulation costs consecutively.
%
%   BESTPOS     contains the position matrix of the best value found.
%   BESTCOST    contains the corresponding cost.
%   SIMSTEPS    contains the number of simulation steps executed.
%   SIMDURATION contains the time it took.
%
%
% GLOBAL CONFIG OPTIONS
%
%   The behavior of SimAnn can be fine tuned by setting global variables
%   prior to invoking SIMANN().
%
%   List of global configuration options
%
%  SIMANN_ALGO       CHAR ARRAY
%       Algorithm to be used. Can be 'SIMANN' or 'NOISYDOWNHILL'.
%       Defaults to 'SIMANN'.
%
%  SIMANN_COSTF
%       Cost function. Possible values are:
%           EDOP    East DOP
%           NDOP    North DOP
%           VDOP    Vertical DOP
%           TDOP    Time DOP
%           HDOP    Horizonal DOP
%           PDOP    Position DOP
%           GDOP    Geometry (Global) DOP
%           SGME    East Weighted Position Solution Error
%           SGMN    North Weighted Position Solution Error
%           SGMV    Vertical Weighted Position Solution Error
%           SGMT    Time Weighted Position Solution Error
%           SGMH    Horizonal Weighted Position Solution Error
%           SGMP    Position Weighted Position Solution Error
%           SGMG    Geometry Weighted Position Solution Error
%       Defaults to GDOP.
%
%   SIMANN_DURATION
%       Maximum simulation time, if not finished before. Defaults
%       to 10000.
%
%   SIMANN_LIVEUPDATES
%       Controls updates of the plots. 'FULL' updates every simulation
%       step, 'SEMI' updates every 50th step. 'ONCE' plots only after
%       finishing. 'NONE' does not output anything graphical at all and is
%       intended for use in batched environments. Defaults to 'SEMI'.
%
%   SIMANN_NUMSAT
%       Number of satellites to simulate. Defaults to 4 and is ignored
%       if SIMANN_POS is set.
%
%   SIMANN_POS
%       Position matrix of satellites. If not set, a random setup is
%       created. If set, SIMANN_NUMSAT is ignored. Format of the matrix:
%           SIMANN_POS =
%               EL1     AZ1
%               EL2     AZ2
%               ..      ..
%               ELk     AZk
%
%   SIMANN_STEPWIDTH
%       Average stepwidth for random satellite steps. The random step is
%       implemented as a 2-D gaussian normal distribution, SIMANN_STEPWIDTH
%       defines the standard deviation. Defaults to pi/60.
%
%   SIMANN_STUCKLIMIT
%       Maximum number of consectutive steps without a state change to
%       allow, before simulation is ended. Note that for algo
%       'NOISYDOWNHILL', this option has no effect, since it does not
%       decide upon accepting a new state or not.
%       Defaults to 500.
%
%  SIMANN_TEMPSCHED
%       Temperature schedule parameters. Data structure with
    fields
%           Type:       'LIN' or 'EXP', default: 'EXP'. Shape of
    the
%                       temperature curve. EXP is recommended.
%           InitTemp:   Initial temperature, defaults to 10.
%           HoldingTime: How long to keep the initial temperature,
     before
%                       starting to lower it. Default: 400
%           DecaySpeed: Speed factor for the temperature decay.
    Defaults to
%                       0.005.
%
%   If a global option is set, but empty, or if the value does not
     make
%   sense or is of the wrong type, SIMANN will silently ignore it
    and use
%   the default. Any successfully read config option will be
    prompted to
%   the console.
%
%   Default values can be loaded from the MAT file
%   CONFIG_OPTS_DEFAULT, while a set of empty globals can be
    loaded from
%   CONFIG_OPTS_EMPTY.
%

% ============== GLOBAL CONFIG OPTIONS ==============
%   -   these options can be set by defining the particular
    globals in the
%   workspace before invoking SimAnn

% COST FUNCTION
global SIMANN_COSTF;
if ¬isempty(SIMANN_COSTF)
    CostF=SIMANN_COSTF;
else
    CostF='GDOP';
end
clear SIMANN_COSTF;

% number of the satellite used for orientation
%   must be defined before parsing global opts
global EASTSAT;
if strcmp(upper(CostF),'EDOP') | ...
        strcmp(upper(CostF),'NDOP') | ...
        strcmp(upper(CostF),'SGME') | ...
        strcmp(upper(CostF),'SGMN') %| ...
        %strcmp(upper(CostF),'VDOP') | ...
        %strcmp(upper(CostF),'SGMV')
    EASTSAT=0; % do not orient the matrix since it disturbs
```

```
                % the simulation here!
else
    EASTSAT=1;
end

% POSITION AND NUMBER OF SATS
global SIMANN_POS;
if ¬isempty(SIMANN_POS) % SIMANN_POS set?
    Pos=SIMANN_POS
    NumSat=size(SIMANN_POS,1);
else
    global SIMANN_NUMSAT;
    if ¬isempty(SIMANN_NUMSAT) % SIMANN_NUMSAT set?
        NumSat=SIMANN_NUMSAT;
    else
        NumSat=4;
    end
    clear SIMANN_NUMSAT;
    % set up the satellites randomly
    Pos=RandomSatelliteSetup(NumSat);
end
clear SIMANN_POS;

% STEPWIDTH FOR RANDOM STEPS
global SIMANN_STEPWIDTH;
if ¬isempty(SIMANN_STEPWIDTH)
    stddev=SIMANN_STEPWIDTH
else
    stddev=pi/60;
end
clear SIMANN_STEPWIDTH;

% SIMULATION ALGORITHM
global SIMANN_ALGO;
if ¬isempty(SIMANN_ALGO)
    if strcmp(upper(SIMANN_ALGO),'NOISYDOWNHILL')
        SimCycleHandle=@SimCycle_NoisyDownhill;
        SimType='noisy';
    else
        SimCycleHandle=@SimCycle_SimAnn;
        SimType='simann';
    end
else
    SimCycleHandle=@SimCycle_SimAnn;
    SimType='simann';
end
clear SIMANN_ALGO;

% SIMULATION DURATION
global SIMANN_DURATION;
if ¬isempty(SIMANN_DURATION)
    StopTime=SIMANN_DURATION
else
    StopTime=10000;
```

```
end
clear SIMANN_DURATION;

% TEMPERATURE SCHEDULE
global SIMANN_TEMPSCHED;
% check for correct data structure
if ¬isempty(SIMANN_TEMPSCHED) & ...
        isstruct(SIMANN_TEMPSCHED)
    if isfield(SIMANN_TEMPSCHED,'Type') & ¬isempty(
        SIMANN_TEMPSCHED.Type)
        if strcmp(upper(SIMANN_TEMPSCHED.Type),'LIN')
            ScheduleType='lin'
        else
            ScheduleType='exp'
        end
    else
        ScheduleType='exp';  % if field is missing, set silently
    end
    if isfield(SIMANN_TEMPSCHED,'InitTemp') & ¬isempty(
        SIMANN_TEMPSCHED.InitTemp)
        ScheduleInitTemp=SIMANN_TEMPSCHED.InitTemp
    else
        ScheduleInitTemp=10;
    end
    if isfield(SIMANN_TEMPSCHED,'HoldingTime') & ¬isempty(
        SIMANN_TEMPSCHED.HoldingTime)
        ScheduleHoldingTime=SIMANN_TEMPSCHED.HoldingTime
    else
        ScheduleHoldingTime=400;
    end
    if isfield(SIMANN_TEMPSCHED,'DecaySpeed') & ¬isempty(
        SIMANN_TEMPSCHED.DecaySpeed)
        ScheduleDecaySpeed=SIMANN_TEMPSCHED.DecaySpeed
    else
        ScheduleDecaySpeed=0.005;
    end
else
    % default values
    ScheduleType='exp';
    ScheduleInitTemp=10;
    ScheduleHoldingTime=400;
    ScheduleDecaySpeed=0.005;
end
clear SIMANN_TEMPSCHED;

% LIMIT OF CONSECUTIVE NON CHANGING STATES TO ABORT SIMULATION
global SIMANN_STUCKLIMIT
if ¬isempty(SIMANN_STUCKLIMIT)
    StuckLimit=SIMANN_STUCKLIMIT
else
    StuckLimit=500;
end

% LIVE UPDATING OF PLOTS
```

```
global SIMANN_LIVEUPDATES
if ¬isempty(SIMANN_LIVEUPDATES)
    if strcmp(upper(SIMANN_LIVEUPDATES),'FULL')
        LiveUpdate=true;
        SemiLiveUpdate=false;
        NoGUI=false;
    elseif strcmp(upper(SIMANN_LIVEUPDATES),'SEMI')
        LiveUpdate=false;
        SemiLiveUpdate=true;
        NoGUI=false;
    elseif strcmp(upper(SIMANN_LIVEUPDATES),'ONCE')
        LiveUpdate=false;
        SemiLiveUpdate=false;
        NoGUI=false;
    elseif strcmp(upper(SIMANN_LIVEUPDATES),'NONE')
        LiveUpdate=false;
        SemiLiveUpdate=false;
        NoGUI=true;
    else
        LiveUpdate=false;
        SemiLiveUpdate=true;
        NoGUI=false;
    end
else
    SemiLiveUpdate=true;
    LiveUpdate=false;
    NoGUI=false;
end

% ____________END OF GLOBAL CONFIG OPTS____________

MarkerColor=['b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ];

% configure the lower limit for elevation here (in radians)
global ELEV_LOWLIMIT;
if isempty(strfind(upper(CostF),'SGM')) % => DOP
    ELEV_LOWLIMIT=0; % 0 degrees
else % => SIGMA
    ELEV_LOWLIMIT=pi/180; % 1 degree to avoid zero—weighted sats
end

Tvec=TempSchedule(ScheduleType,ScheduleInitTemp,...
        ScheduleHoldingTime,ScheduleDecaySpeed,StopTime);

LastDOP=DOP(Pos,CostF);

```

```
SimTime=0;

BestPos=Pos;

BestLevel=LastDOP;

BestLevelTime=SimTime;

% contains the costs for the whole simulation (upper row time,
    lower row
% cost)
CostTrack=[LastDOP NaN(1,StopTime)];

% saves all the position matrices
% this is a 3—dimensional array where the 3rd dim is simulation
% time, while the first two are the same than in Pos
PosTrack=NaN(size(Pos,1),size(Pos,2),StopTime+1);
PosTrack(:,:,1)=rad2deg(Pos);

% true when simulating, false when not.
SimState=true;
userexit=false;

% stuck counter to be incremented for every no—change
StuckCounter=0;

% print some information to the console
disp(['Starting ' upper(SimType) ' Simulation with cost function '
     CostF ...
        ' and ' int2str(NumSat) ' satellites'])

% INITIALIZATION OF PLOTS
if ¬NoGUI
    % initialize panel
    fh=figure;
    if strcmp(SimType,'simann')
        set(fh,'Name',['Simulated Annealing with ' num2str(size(
            Pos,1)) ' satellites']);
    else
        set(fh,'Name',['Noisy Downhill Simulation with ' num2str(
            size(Pos,1)) ' satellites']);
    end
    % set the figure size to something reasonable
    winsz=get(0,'ScreenSize');
    winsz=fix([0.1.*winsz(:,3:4) 0.8.*winsz(:,3:4)]);
    set(fh,'Position',winsz);

    % set up the cost plot
    CostTrackHandle=subplot(NumSat+1,2,1);
    CostTrackLSOHandle=plot(CostTrack);
    title(['Cost function ' upper(CostF)]);
    axis([0 100 0 20]);
    grid on;
    % prepare plot updates
    set(CostTrackLSOHandle,'YDataSource','CostTrack');

    % set up temperature plot
    TempTrackHandle=subplot(NumSat+1,2,2);
    TempTrackLSOHandle=plot(Tvec);
    title('Simulation temperature schedule');
    grid on;

    % plot the satellites' paths
    for sat=1:size(Pos,1)
        ElevationPath=squeeze(PosTrack(sat,1,:));
        AzimuthPath=squeeze(PosTrack(sat,2,:));

        % plot the satellites elevation plot and set some
            preferences
        PosTrackHandle(sat,1)=subplot(NumSat+1,2,2*sat+1);
        PosTrackLSOHandle(sat,1)=plot(squeeze(PosTrack(sat,1,:)),
            MarkerColor(sat));
        set(PosTrackLSOHandle(sat,1),'YDataSource','ElevationPath'
            );
        title(['Elevation path of satellite ',num2str(sat)]);
        axis([0 100 0 90]);
        grid on;

        % plot the satellites azimuth plot and set some
            preferences
        PosTrackHandle(sat,2)=subplot(NumSat+1,2,2*sat+2);
        PosTrackLSOHandle(sat,2)=plot(squeeze(PosTrack(sat,2,:)),
            MarkerColor(sat));
        set(PosTrackLSOHandle(sat,2),'YDataSource','AzimuthPath');
        title(['Azimuth path of satellite ',num2str(sat)]);
        axis([0 100 0 360]);
        grid on;
    end

    % initialize skyview
    SkyViewHandle=figure;
    SetupSkyView(SkyViewHandle);
    set(SkyViewHandle,'Name','SkyView');
    axh=SkyView(Pos);

end

tic;
% main loop:
while userexit==false

    if SimState
        % perform simulation cycle
        [Pos LastDOP poschange]=SimCycleHandle(Pos,LastDOP,CostF,
            Tvec(SimTime+1));

        % save the values
        CostTrack(SimTime+1)=LastDOP;
        PosTrack(:,:,SimTime+1)=rad2deg(Pos);

        % only execute if the satellite positions have changed
        if poschange==true
            % update graphical output
            if ¬NoGUI
                SkyView(Pos,axh);
                if LiveUpdate
                    for sat=1:size(Pos,1)
                        ElevationPath=squeeze(PosTrack(sat,1,:));
                        AzimuthPath=squeeze(PosTrack(sat,2,:));

                        refreshdata(PosTrackLSOHandle(sat,1),'caller');
                        refreshdata(PosTrackLSOHandle(sat,2),'caller');
                         % grey out azimuth if satellite in zenith
                        if Pos(sat,1)>1.56
                            get(PosTrackHandle(sat,2),'Type');
                            set(PosTrackHandle(sat,2),'Color',[0.8 0.8 0.8]);
                        else
                            set(PosTrackHandle(sat,2),'Color','w');
                        end
                    end
                    refreshdata(CostTrackLSOHandle,'caller');
                end
                drawnow;
            end

            % reset the StuckCounter
            StuckCounter=0;
            if LastDOP<BestLevel
                BestLevel=LastDOP;
                BestLevelTime=SimTime;
                BestPos=Pos;
            end
        else
            % StuckCounter counts how many simulation cycles the position
            % did not change any more. If the value hits a limit, the
            % simulation is stopped.
            StuckCounter=StuckCounter+1;
        end

        % update simulation time
        SimTime=SimTime+1;

        % some output every 50 cycles
        if mod(SimTime,50)==0
            %disp([ 'Simulation Time:        ' num2str(SimTime) ]);
            %disp([ 'Simulation Temperature: ' num2str(Tvec(SimTime+1))]);
            %disp([ 'Best DOP value:         ' num2str(BestLevel)]);
            %disp(' '); % dirty hack for a CR/LF..

            % semi-live data plotting
            if ¬NoGUI
                if SemiLiveUpdate
                    for sat=1:size(Pos,1)
                        ElevationPath=squeeze(PosTrack(sat,1,:));
                        AzimuthPath=squeeze(PosTrack(sat,2,:));

                        refreshdata(PosTrackLSOHandle(sat,1),'caller');
                        refreshdata(PosTrackLSOHandle(sat,2),'caller');
                    end

                    refreshdata(CostTrackLSOHandle,'caller');
                end

                % update of axis scaling
                if (LiveUpdate || SemiLiveUpdate)
                    %TimeScale=2^ceil(log2(SimTime*1.1))
                    TimeScale=SimTime+50;

                    axis(CostTrackHandle,[TimeScale-400 TimeScale 0 20]);
                    axis(TempTrackHandle,[TimeScale-400 TimeScale 0 20]);
                    for sat=1:NumSat
                        axis(PosTrackHandle(sat,1),[TimeScale-400 TimeScale 0 90]);
                        axis(PosTrackHandle(sat,2),[TimeScale-400 TimeScale 0 360]);
                        % grey out azimuth if satellite in zenith
                        if Pos(sat,1)>1.56
                            get(PosTrackHandle(sat,2),'Type');
                            set(PosTrackHandle(sat,2),'Color',[0.8 0.8 0.8]);
                        else
                            set(PosTrackHandle(sat,2),'Color','w');
                        end
                    end
                    drawnow;
                end
            end

            if LastDOP<BestLevel
                BestLevel=LastDOP;
                BestLevelTime=SimTime;
                BestPos=Pos;
                % reset the StuckCounter
                StuckCounter=0;
            end
        end

        % interrupt conditions
        if (SimTime≥StopTime) || (StuckCounter≥StuckLimit)
            % cut off the last STUCKCOUNTER simulations if simulation was
            % stopped due to this limitation
            if StuckCounter≥StuckLimit
                disp([’Simulation did not change states for ’ ...
                        num2str(StuckLimit) ’ consecutive steps!’ ]);
                disp(’Ending simulation and removing the trailing non−changing steps.’);
                EffectiveTrackSize=SimTime−StuckLimit+1;
                CostTrack=CostTrack(1:EffectiveTrackSize);
                PosTrack=PosTrack(:,:,1:EffectiveTrackSize);
                SimTime=SimTime−StuckLimit;
            end;
            if ¬NoGUI
                % update the axis scaling once more and draw the plots
                TimeScale=SimTime;

                axis(CostTrackHandle,[0 TimeScale 0 20]);
                axis(TempTrackHandle,[0 TimeScale 0 20]);
                for sat=1:NumSat
                    axis(PosTrackHandle(sat,1),[0 TimeScale 0 90]);
                    axis(PosTrackHandle(sat,2),[0 TimeScale 0 360]);
                    ElevationPath=squeeze(PosTrack(sat,1,:));
                    AzimuthPath=squeeze(PosTrack(sat,2,:));
                    refreshdata(PosTrackLSOHandle(sat,1),’caller’);
                    refreshdata(PosTrackLSOHandle(sat,2),’caller’);

                    % grey out azimuth if satellite in zenith
                    if Pos(sat,1)>1.56
                        set(PosTrackHandle(sat,2),’Color’,[0.8 0.8 0.8]);
                    else
                        set(PosTrackHandle(sat,2),’Color’,’w’);
                    end
                end
                refreshdata(CostTrackLSOHandle,’caller’);
                drawnow;
            end

            SimState=false;
            SimDuration=toc;
            disp(['Simulation finished in ' num2str(SimDuration) '
                 seconds!'])
            disp('Simulation time limit reached')
            SimSteps=SimTime;

            returnpos=BestPos;
            returndop=BestLevel;
            userexit=true;
        end
    end
end
```

B.22 SimAnn_Transition

```
% SIMANN_TRANSITION decide upon transition to new state
%
%   RESULT=SIMANN_TRANSITION(E,EN,T)
%
%   returns FALSE or TRUE in RESULT, depending on the old energy E
    ,
%   the new energy EN and the current simulation temperature T.


function result=SimAnn_Transition(e,en,T)

% classical approach (Kirkpatrick et al): Probability is always 1
% if en<e, otherwise P=exp( (e-en)/T )

if en<e
    result=true;
else
    % compare to a random number
    %   e-en=0 => exp()=1 => condition is always true
    %   e-en -> inf => exp()->0 => condition always false
    result= ( rand ≤ (exp((e-en)/T)) );
end;
```

B.23 SimCycle_NoisyDownhill

```
% SIMCYCLE_NOISYDOWNHILL    perform a simulation cycle using a
%                noise superposed downhill algorithm
% [ NEWPOS NEWDOP POSCHANGE ] = SIMCYCLE_NOISYDOWNHILL(POS,DOPVAL,
    DOPTYPE,T)
%   executes a simulation cycle after an algorithm modified after
    the
%   original Simulated Annealing algorithm.
%   Here, the position is moved towards the minimum for each
    satellite
%   independently.
%   A temperature dependent gaussian noise is superposed to the
    movement
%   afterwards. The
%   POS contains the current positions
%   DOPVAL must contain the dop to POS
%   DOPTYPE is a string containing the cost function to use (e.g.
    'gdop')
%   T contains the current simulation temperature
%
%   The function always returns POSCHANGE=TRUE here (kept for
    compatibility
%   with the SimAnn algo).
%   NEWPOS and NEWDOP are set to POS and DOPVAL if no transition.



function [ newpos newdop poschange ] = SimCycle_NoisyDownhill(pos,
    dopval,doptype,T)


% move the position to an better level argument by argument. At
    the moment
%  the stepsize only depends on the gradient, not on the
    temperature.
newpos=NeighborState_Grad(pos,doptype); % compute better pos in
    elevation

% add gaussian noise. Here, the temperature schedule has an
    effect.
newpos=NeighborState_Random(newpos,T);

newdop=DOP(newpos,doptype);

poschange=true;
```

B.24 SimCycle_SimAnn

```matlab
% SIMCYCLE_SIMANN   perform a simulation cycle
%
% [ NEWPOS NEWDOP POSCHANGE ] = SimCycle_SimAnn(POS,DOPVAL,DOPTYPE
    ,T)
%   executes a simulation cycle after Kirkpatrick's Simulated
    Annealing
%   algotrithm.
%   POS contains the current positions
%   DOPVAL must contain the dop to POS
%   DOPTYPE is a string containing the cost function to use (e.g.
    'gdop')
%   T contains the current simulation temperature
%
%   The function returns POSCHANGE=TRUE if the new state was
    accepted.
%   NEWPOS and NEWDOP are set to POS and DOPVAL if no transition.

function [ newpos,newdop,poschange ] = SimCycle_SimAnn(pos,dopval,
    doptype,T)

newpos=pos;
newdop=dopval;
poschange=false;

% examine every single satellite on its own
for i=1:size(pos,1)
    % random change of the ith satellite's position
    newpos(i,:)=RandomSatStep(pos(i,:),pi/24);
    % get cost for this "sub-state"i
    olddop=newdop;
    newdop=DOP(newpos,doptype);

    % decide upon accepting the new position
    if SimAnn_Transition(dopval,newdop,T)
        % keep that position and set a reminder
        poschange=true;
    else
        newpos(i,:)=pos(i,:);
        newdop=olddop;
    end
end

% if the position changed, do the orientation of the position
    matrix here.
if poschange==true
    newpos = CorrectPositionMatrix(OrientPositionMatrix(newpos));
end
```

B.25 SimpleDownhillSimulation

```
function pos=SimpleDownhillSimulation(pos,stddev,doptype)

SetupSkyView;
newdop=dop(pos,doptype)
lsh=SkyView(pos);
drawnow;
while true
    % cycle thru single satellites
    for i=1:size(pos,1)
        olddop=newdop;
        oldpos=pos;
        pos(i,:)=RandomSatStep(pos(i,:),stddev);
        if i==1
            % get the matrix newly oriented
            pos=CorrectPositionMatrix(OrientPositionMatrix(pos));
        end
        newdop=dop(pos,doptype);
        if newdop<olddop
            bestdop=olddop
            lsh=SkyView(pos,lsh);
            drawnow;
        else
            newdop=olddop;
            pos=oldpos;
        end
    end
end
```

B.26 SimulationCampaign

```
function SimResults = SimulationCampaign(varargin)
% SIMRESULTS = SIMULATIONCAMPAIGN(PREFIX, ALGOS,
%                            COSTFUNCTIONS, CONSTELLATIONSIZES,
    PASSES)
%
%   performs multiple runs of simulations (PASSES times) for a
    collection
%   of parameter sets. The results are saved for each parameter
    set
%   individually
%
%   PREFIX          string  Prefix for the filenames
%   ALGOS           cell array
%                   contains strings with the Algos -
%                   { 'SIMANN' 'NOISYDOWNHILL' }
%   COSTFUNCTIONS   cell array
%                   contains strings with the cost functions
%                   { 'EDOP'  'HDOP'  'SGME'  'SGMH' }
%   CONSTELLATIONSIZES  vector
%                   containing satellite quantities
%                   [ 4 6 8 10 15 ]
%   PASSES          number of passes for every parameter set

if nargin==0
   Prefix=[date '-']
else
    if isempty(varargin{1})
        Prefix='';
    else
        Prefix=[varargin{1} '-']
    end
end

%if nargin==5 % use parameters
%    ALGOS=varargin{2}
%    COSTFUNCTIONS=varargin{3}
%    CONSTELLATIONSIZES=varargin{4}
%    PASSES=varargin{5}
%else
%    ALGOS={'SIMANN'};
%    COSTFUNCTIONS={'EDOP' 'VDOP' 'TDOP' 'HDOP' 'PDOP' 'GDOP' '
    SMGE' 'SGMV' 'SGMT' 'SGMH' 'SGMP' 'SGMG' 'COND'};
%    CONSTELLATIONSIZES=[4 5 6 7 8 10 12 14];
%    PASSES=10;
%end
% dont care about that - load from file!
load CampaignOptions

ALGOS
COSTFUNCTIONS
```

```
CONSTELLATIONSIZES
PASSES

SimResults=cell(length(ALGOS),length(COSTFUNCTIONS),length(CONSTELLATIONSIZES));

global SIMANN_ALGO;
global SIMANN_COSTF;
global SIMANN_NUMSAT;

for algorithm=1:length(ALGOS)
    SIMANN_ALGO=ALGOS{algorithm};
    disp(['======== SIMULATION ALGORITHM: ' SIMANN_ALGO])
    for costfunction=1:length(COSTFUNCTIONS)
        SIMANN_COSTF=COSTFUNCTIONS{costfunction};
        disp(['================ COST FUNCTION: ' SIMANN_COSTF])
        for constellation=1:length(CONSTELLATIONSIZES)
            SIMANN_NUMSAT=CONSTELLATIONSIZES(constellation);
            disp(['================================ CONSTELLATION SIZE: ' num2str(SIMANN_NUMSAT)])
            SimResult=MultiSim(PASSES);
            ResultsFilename=[Prefix 'SimulationResults_' SIMANN_ALGO '-' SIMANN_COSTF '-' int2str(SIMANN_NUMSAT) '-' int2str(PASSES)    ];
            disp(['Saving results to ' ResultsFilename])
            disp(' ')
            save(ResultsFilename,'SimResult','SIMANN_ALGO','SIMANN_COSTF',...
            'SIMANN_NUMSAT','PASSES');
            SimResults{algorithm,costfunction,constellation}=SimResult;
        end
    end
end
```

B.27 SkyView

```
function positionHandles = SkyView(pos,lineserobj)
% SKYVIEW   Plot position data in Skyview figure
%       SkyView(POSITIONS,[LINESEROBJ])
%       plots the position data of satellites given in POSITIONS
    to
%       the (previously set up) figure in FIGUREHANDLE.
%
%       POSITIONS
%       is a Nx2 matrix consisting of one row each satellite, with
%       the radian elevation in the first column and the radian
%       azimuth in the second one.
%       Example for an equidistant setup of 4 satellites:
%       POSITIONS =
%
%           1.5708         0
%                0   −2.0944
%                0   −0.0000
%                0    2.0944
%
%       LINESEROBJ
%       can be supplied to update previously plotted data.
%       If it is ommitted, the lineseries is plotted in the
%       current axes and will return a new object.
%
% H=SKYVIEW(POSITIONS,LINESEROBJ) returns a handle to the plotted
%       positions, similar to plot().
%
%       See also SETUPSKYVIEW

% definitions of satellite colors
MarkerColor=['b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ...
'b' 'r' 'g' 'm' 'c' ];

error(nargchk(1,2,nargin,'struct'));
% set up the coordinantes
PlotPositions=[ (1−(2*pos(:,1)./pi)).*cos(pos(:,2)) ...
                (1−(2*pos(:,1)./pi)).*sin(pos(:,2)) ];
% new plot?
if nargin==1
    % plot into a new set of data)
    for sat=1:size(pos,1)
        positionHandles(sat)=plot(PlotPositions(sat,1), ...
                            PlotPositions(sat,2),'o');
        set(positionHandles(sat),'MarkerEdgeColor','k');
        set(positionHandles(sat),'MarkerFaceColor',MarkerColor(sat
            ));
    end

% or update data?
else
    % get the data out of the lineseries object
    %myxdata=get(lineserobj,'XData');
    for sat=1:size(pos,1)
        set(lineserobj(sat),'XData',PlotPositions(sat,1));
        set(lineserobj(sat),'YData',PlotPositions(sat,2));
    end
    positionHandles=lineserobj;
end
```

B.28 TempSchedule

```
% TEMPSCHEDULE      Compute the temperature schedule for the simulation
%
% T=TEMPSCHEDULE(SCHEDULETYPE,SCHEDULEINITTEMP,SCHEDULEHOLDINGTIME,SCHEDULEDECAYSPEED,ENDTIME)
%   returns a vector T with temperatures for every step of the simulation from 0 to ENDTIME
%   size(T)=ENDTIME+1
%
%   SCHEDULETYPE        can be either 'lin' or 'exp'.
%   SCHEDULEINITTEMP    defines the initial temperature
%   SCHEDULEHOLDINGTIME initial temperature is held constant for that time
%   SCHEDULLEDECAYSPEED if linear, this parameter defines the linear gradient of the temperature
%                  after the constant phase.
%                 if exponential, it is used as the coefficient a in exp(-ax)
%   ENDTIME          to define the length of the returned vector
%
% NOTE: In linear mode, the function will not return any negative temperature values, instead all of them
%   will be set to zero.

function T=TempSchedule(ScheduleType,ScheduleInitTemp,ScheduleHoldingTime,ScheduleDecaySpeed,EndTime)

switch lower(ScheduleType)
case 'lin'
    % when does the temperature reach zero?
    ZeroCrossingTime=floor(ScheduleInitTemp/ScheduleDecaySpeed + ScheduleHoldingTime);
    % constant until SchedHoldTemp (included), then decay linearly till zero, then zeroes till end
    T=[ ones(1,ScheduleHoldingTime+1)*ScheduleInitTemp ...
        ScheduleInitTemp-ScheduleDecaySpeed * ( [ScheduleHoldingTime+1:ZeroCrossingTime] - ScheduleHoldingTime) ...
        zeros(1,EndTime-ZeroCrossingTime) ];
case 'exp'
    T=[ ones(1,ScheduleHoldingTime)*ScheduleInitTemp ...
        ScheduleInitTemp*exp(-ScheduleDecaySpeed*( [ScheduleHoldingTime:EndTime] -ScheduleHoldingTime)) ];
otherwise
    T=0;
end
% cut back the vector if it is too long
if length(T)>EndTime+1
    T=T(1:EndTime+1);
end
```

B.29 Weighted_SigmaSQ

```
function SIGMASQ = WEIGHTED_SIGMASQ(E)
% WEIGHTED_SIGMA    return the elevation-weighted variance for a
    satellite
%
%   SIGMA = WEIGHTED_SIGMA(E)
%
%   returns a variance consisting of contribution by atmospheric
    effects,
%   multipath, SNR and residual effects. The tropospheric,
    ionospheric
%   and multipath contributions are elevation-dependent.
%
%   The variance is needed in the weighting matrix W when using
    the
%   weighted navigation solution.

% contributing variances. The predefined values are average values
     obtained
% from the north american WAAS network
%    residual
sUDREsq=0.5^2;
%    ionospheric, vertical
sUIVEsq=0.5^2;
%    SNR
sSNRsq=0.22^2;
%    multipath at 45 degrees
sM45sq=0.22^2;
%    troposheric, vertical
sTRVsq=0.15^2;



% this is (R_e/(R_e+h))^2
EarthFactor=(6378/6728)^2;
% the squared obliquity factor Fsq(E)
Fsq=1/(1-EarthFactor*cos(E)^2);

% this is the resulting variance
SIGMASQ = sUDREsq + Fsq*sUIVEsq + sSNRsq + ...
    sM45sq/(tan(E)^2) + sTRVsq/(sin(E)^2);
```

C Plots

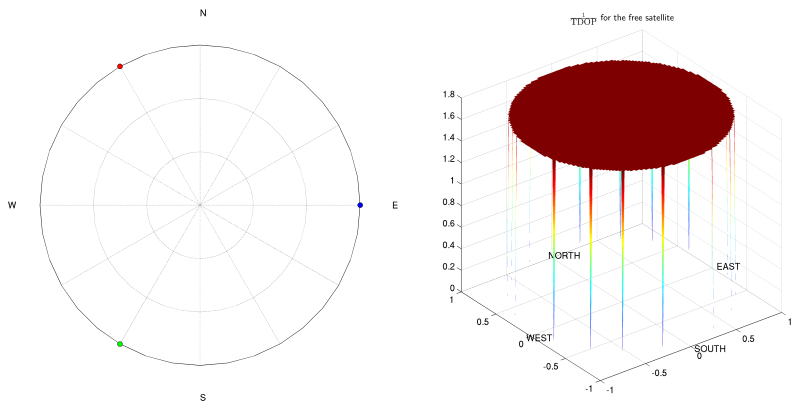

Figure 17: TDOP surface plot for an equidistant setup of 4 satellites

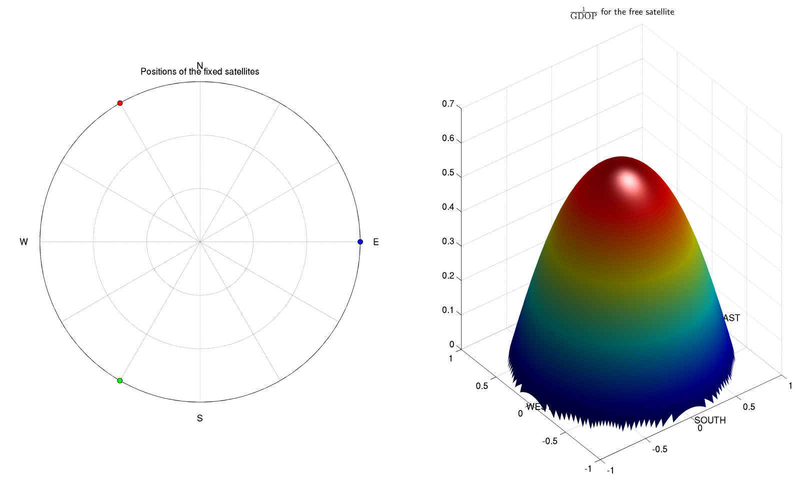

Figure 18: GDOP surface plot for an equidistant setup of 4 satellites

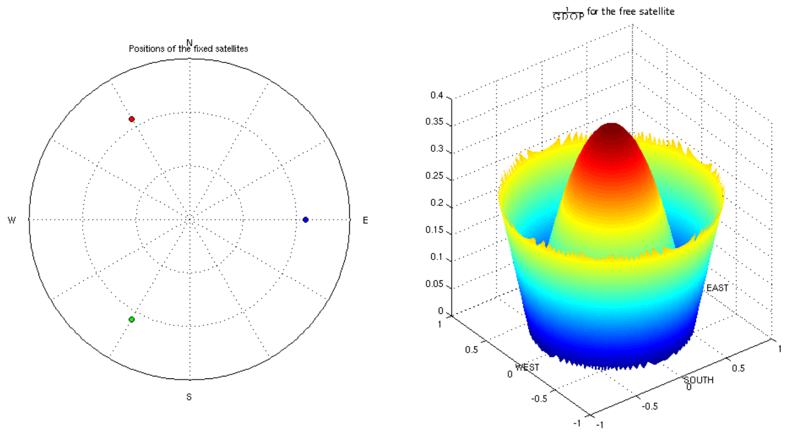

Figure 19: GDOP surface plot for a 25° elevated equidistant setup of 4 satellites

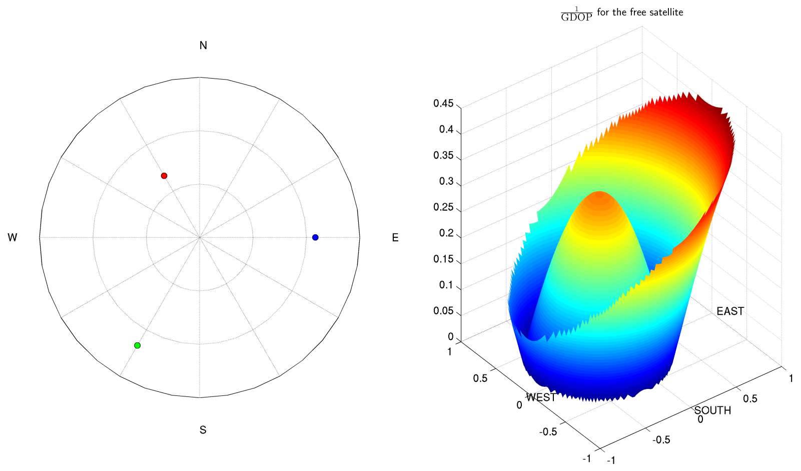

Figure 20: GDOP surface plot for 4 satellites, fixed satellites define a slanted plane

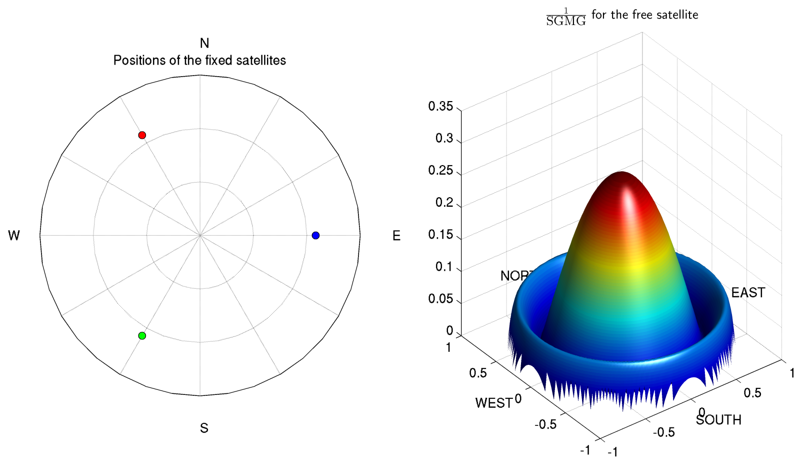

Figure 21: σ_G surface plot for a 25° elevated equidistant setup of 4 satellites