Institut für Geodäsie
Steyrergasse 30, A-8010 Graz

LV 522.319
Informatik I für Geodäsie
Wintersemester 22/23
20. Dezember 2022

Übungsprogramm I

Rutschung Blaubachgraben

Daniel Ebert
Matr.Nr.: 12216327
daniel.ebert@student.tugraz.at

Inhaltsverzeichnis

1 Aufgabenstellung

Mittels verschiedener Dateien (Messdaten und Ortho-Bildern) sollen eine Reihe von Visualisierungen erstellt werden, welche die gemessenen Erdbewegung an einem Rutschhang (Blaubachgraben) in der Nähe der Gemeinde Krimml darstellen.

Dabei wurden in 3 verschiedenen Jahren (1953, 1999 und 2004) verschiedene Messungen durchgeführt, die miteinander verglichen werden können. Insbesondere sollen die Veränderungen des Geländes graphisch ausgewertet werden.

Die Details der Visualisierung sind dabei ausführlich vorgegeben. Ergebnis sollen insgesamt 6 Plots sein, deren Hintergrund ein Orthofoto bildet, auf das zusätzliche Details eingezeichnet werden.

Diese Plots (via `matplotlib`) sollen mit einer Auflösung von 400 DPI als Bilder abgelegt werden.

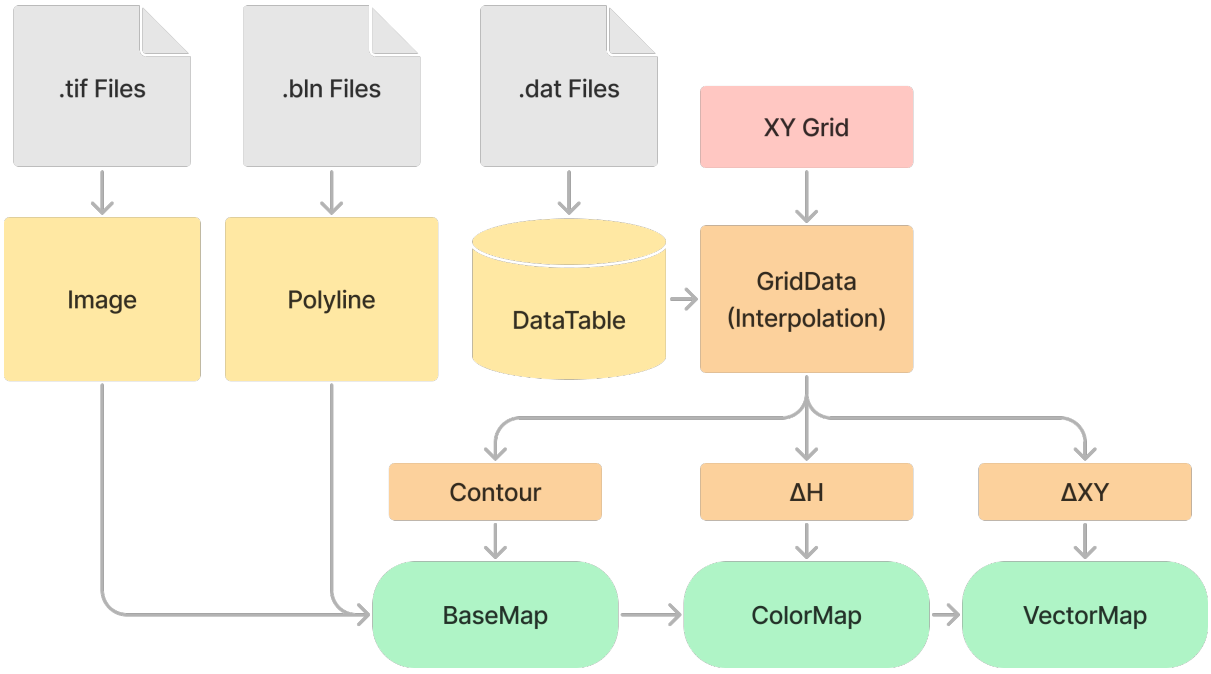
Anschließend folgt eine kurze Interpretation der aufbereiteten Daten.

2 Durchführung

2.1 Verwendete Programmbibliotheken

- `matplotlib` zur Erstellung der Plots.
- `numpy` für Datenimport und vielfältige Berechnungen.
- `pandas` für einfacheren Zugriff auf Messreihen.
- `scipy` für Interpolation.

2.2 Architektur



Gegeben sind Dateien (grau), aus denen Rohdaten (gelb) gelesen werden können.

Außerdem ist ein Raster (rot) vordefiniert.

Aus diesen werden dann angereicherte Daten (orange) berechnet.

Die Roh-Daten und angereicherten Daten fließen in die Erstellung von Karten ein (grün). Dabei bauen 3 Sets von Karten aufeinander auf, indem für jede Stufe weitere Elemente hinzugefügt werden.

Eine detaillierte Beschreibung dieser Abläufe folgt in den weiteren Kapiteln.

2.3 Datenimport

Zunächst müssen die vorhandenen Daten gesichtet und in das Programm importiert werden. Dabei können 3 Arten von Datenformaten unterschieden werden:

- Orthobilder im `.tif` Format. Importiert als `np.ndarray` .
- Linienzüge im `.bln` Format. Importiert als `pd.DataFrame` .
- Messreihen im `.dat` Format. Importiert als `pd.DataFrame` .

Für jedes Format muss eine entsprechende Importmethode gefunden werden.

Die Orthobilder können via `matplotlib.pyplot.imread` in ein 2d-Array eingelesen werden, dass direkt an `imshow` übergeben werden kann. Da hier kein weiterer Zugriff auf die Daten notwendig ist, wurde der Wert nicht extra in ein `DataFrame` überführt.

Die anderen Datenformate liegen im Rohtextformat als DSV (delimiter separated values) vor, und können mit `pd.read_csv` eingelesen werden. Dabei ist Acht zu geben auf das verwendete Trennzeichen (Komma bei .bln Dateien, Leerzeichen bei .dat Dateien), und ob ein Header in der Datei verfügbar ist.

Bei fehlendem Header wurden sinnvolle Spalten-Bezeichner beim Import angelegt, damit mit sprechenden Namen auf die Spalten zugegriffen werden kann.

Die `.dat` Dateien lassen sich nicht direkt importieren, da der Header dort fälschlicherweise mit einem zusätzlichen Leerzeichen beginnt. Es muss also zunächst der Dateiinhalt in einen String geladen werden und das Leerzeichen entfernt werden. Erst dann kann der Inhalt an `pd.read_csv` übergeben werden.

Außerdem muss beim Importieren auf Text-Encoding geachtet werden - die Dateien verwenden noch das veraltete Windows-1252 Encoding statt dem UTF-8 Standard.

```
with open(filename, "r", encoding="cp1252") as f:
    content = f.read()[1:] # fehlerhaftes Leerzeichen entfernen
    return pd.read_csv(io.StringIO(content), sep=" ", header=0)
```

2.4 Interpolation

In den verschiedenen Messungen, die über die letzten 70 Jahre durchgeführt wurden, wurden die dreidimensionalen Position einer Vielzahl von Punkten gemessen. Diese Punkte bilden jedoch kein gleichverteiltes Raster über das gesamte Messgebiet. Um die Messwerte in Einklang mit der Darstellung im Orthophoto zu bringen, und eine bessere Vergleichbarkeit über mehrere Jahre hinweg zu gewährleisten, sollten Zahlenwerte über alle Messreihen an ganz konkreten Punkten interpoliert werden.

Da alle Orthofotos die selbe Bildgröße und Positionierung aufweisen, macht es Sinn ein Gitter zu bilden, das zu jedem Pixel im Bild eine 2d-Koordinate definiert. So kann der Wert jeder Messgröße einheitlich für jeden Pixel bestimmt (linear aus umliegenden Punkten interpoliert)

werden, sofern es umliegende Punkte gibt - was für die Randbereiche nicht zutrifft, aber innerhalb des Studienbereichs immer gegeben ist.

⚠ In der Angabe zum Übungsprogramm werden geodätisch orientierte Koordinaten (x-Achse nach Norden) verwendet. In den Messreihen werden jedoch mathematisch orientierte Koordinaten (x-Achse nach Osten) angegeben. In diesem Dokument und im Programm werden fortan nur noch mathematisch orientierte Koordinaten erwähnt, da die verwendeten Programmbibliotheken ebenfalls mit dieser Orientierung arbeiten.

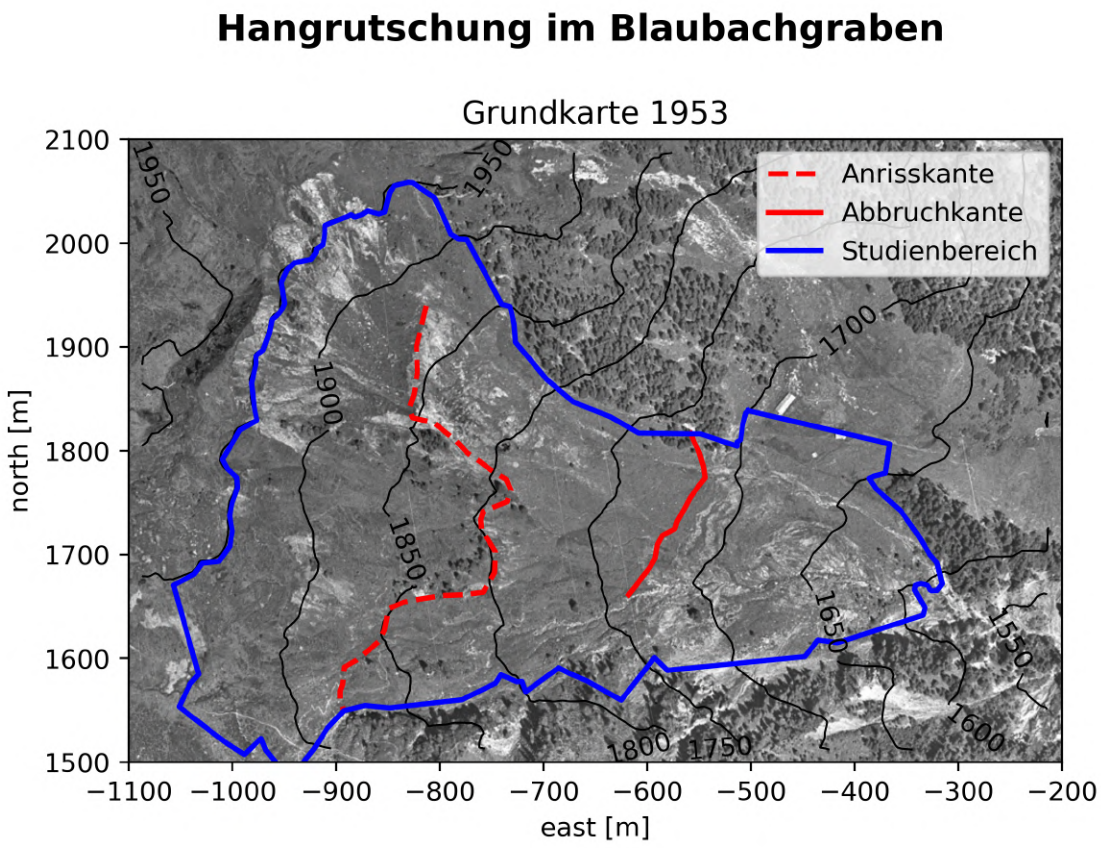
Für die Interpolation wurde zuerst ein Raster definiert, das den vorgegebenen Koordinatenbereich vollständig abdeckt und die selbe Dimension wie die Bilder hat (900x600). Für diese Rasterpunkte lassen sich nun interpolierte Werte für Höhe oder horizontale Verschiebung aus den Rohdaten berechnen.

```
width, height = 900, 600
x_min, x_max = -1100, -200
y_min, y_max = 1500, 2100

"""
Aus den oben gegebenen Begrenzungen lassen sich 2d-Arrays
berechnen, welche den selben `shape` haben wie die Orthobilder,
und jedem Element eine x bzw. y-Koordinate im verwendeten
Koordinatensystem zuweisen.
"""
grid_x, grid_y = np.meshgrid(
    np.linspace(x_min, x_max, width),
    np.linspace(y_min, y_max, height),
)

"""
Für diese Koordinaten lassen sich nun Werte aus einem `dataframe`
interpolieren. Dazu müssen die Koordinaten der Rohdaten, sowie
ein zu interpolierender Wert (hier `Z`)  angegeben werden.
Alle Eingangsdaten müssen 1d-Arrays der gleichen Länge sein.
Das Ergebnis ist dann ein 2d-Array des selben `shape` wie die
anderen Gitterkoordinaten.
"""
grid_z = interpolate.griddata(
    points=(dataframe["X[m]"], dataframe["X[m]"]),
    values=dataframe["Z[m]"],
    xi=(grid_x, grid_y),
    method="linear",
)
```

2.5 Grundkarten

Die Basis für alle weiteren Visualisierungen stellt eine Reihe von Grundkarten. Diese zeigen Orthofotos des Messgebiets mit einigen zusätzlichen Features: Anrisskante, Abbruchkante, Studienbereich und Höhenlinien.

Beispiel:



2.5.1 Messbereich

Außerdem legen die Grundkarten den Achsenbereich und -beschriftungen für alle Plots fest. Um zu verhindern, dass `matplotlib` einen über die Orthofotos hinausragenden Bereich für den Plot annimmt, sollten die Grenzen genau festgelegt werden:

```
ax.set_xlim(x_min, x_max)
ax.set_ylim(y_min, y_max)
```

2.5.2 Orthobilder

Das Orthobild (in Form des `np.ndarray` wie vorher importiert) kann dann direkt in den Bereich gezeichnet werden. Zu beachten ist die Angabe der Colormap - denn ansonsten würde für Graustufenbildern automatisch eine Falschfarbendarstellung gewählt werden.

```
ax.imshow(
    ortho_image,
    extent=[x_min, x_max, y_min, y_max],
    cmap="gray",
)
```

2.5.3 Höhenlinien

Über das Bild können nun Höhenlinien gezeichnet werden. Dazu nutzt man das Raster mit interpolierten Z-Werten, das direkt an einen Contour-Plot übergeben werden kann. Als Höhen-Level wurden 50m Schritte zwischen 1500m und 2100m gewählt, was den gesamten Messbereich abdeckt. Um Beschriftungen entlang der Contour-Linien abzuzeigen, muss zusätzlich noch eine Funktion aufgerufen werden:

```
height_contours = ax.contour(
    grid_x,
    grid_y,
    grid_z_1999,
    levels=np.arange(z_min, z_max, z_step),
    colors="black",
    linewidths=0.75,
)
ax.clabel(height_contours, inline=True, fontsize=10)
```

2.5.4 Linienzüge

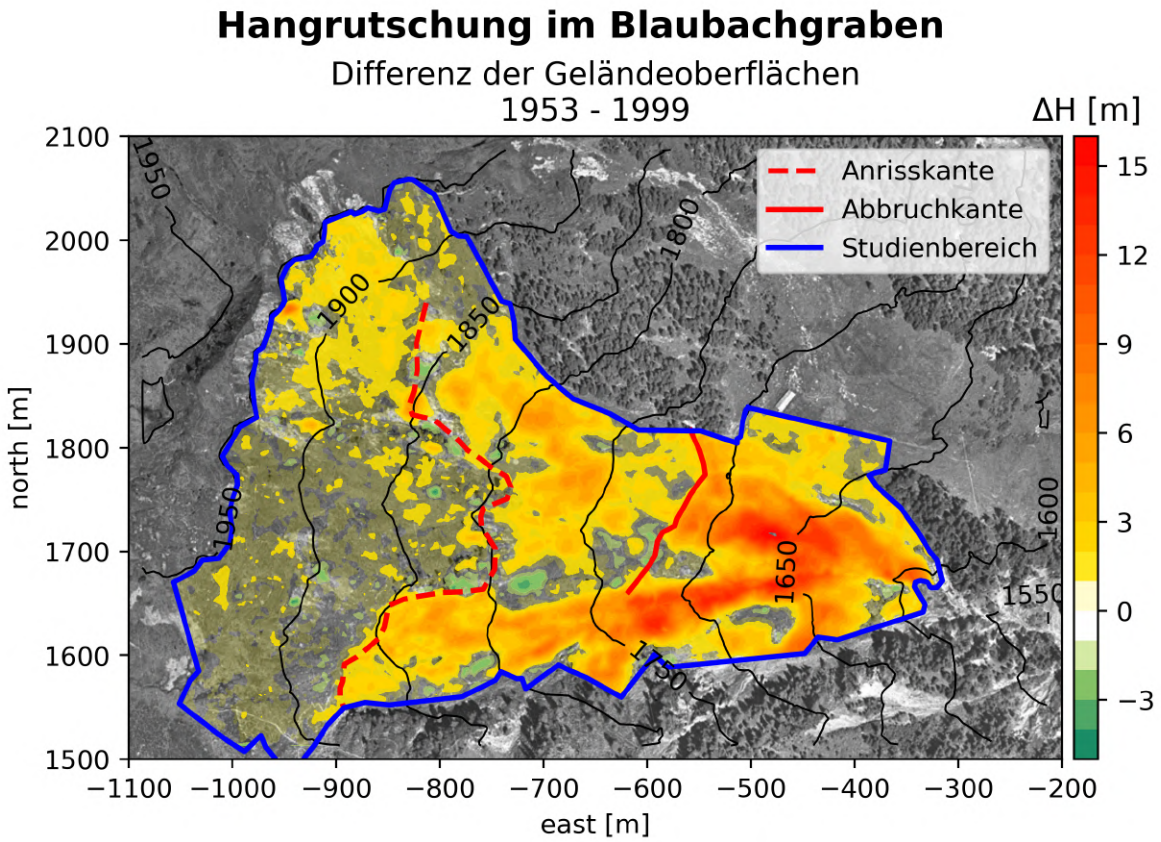
Anschließend können Linienzüge aus den `.bln` Dateien eingezeichnet werden. Diese wurden zuvor in ein `pd.DataFrame` mit selbstgewählten Spaltenbezeichnern geladen, und lassen sich somit ganz einfach als xy-Linienplot zeichnen, beispielsweise:

```
ax.plot(
    polyline_tear_edge["X[m]"],
    polyline_tear_edge["Y[m]"],
    color="red",
    linewidth=2,
    linestyle="--",
    label="Anrisskante",
)
```

2.6 Thematische Plots

Über die Grundkarten hinausgehend werden nun Veränderungen an der Oberflächenhöhe des Messgebietes eingezeichnet. Die Art, wie diese Höhendifferenzen berechnet werden, unterscheidet sich leicht zwischen den Jahren.

Beispiel:



2.6.1 Differenzmessung

Zwischen 1953 und 1999 wird die Differenz ΔH aus den interpolierten Höhenwerten beider Messjahre genommen:

```
grid_z_diff_1999 = grid_z_1953 - grid_z_1999
```

Dabei ist zu beachten, dass das Ergebnis die Anzahl der Höhenmeter ist, die in der Zeitspanne verschwunden sind.

Die Differenz zwischen 1999 und 2004 kann direkt aus der Spalte `"dZ[m]"` der 2004er-Messdaten entnommen werden.

2.6.2 Maskierung

Die Farbdarstellung der Höhenänderung soll auf das Analysegebiet beschränkt werden. Innerhalb des Rasters gibt es für alle Bildpunkte einen Differenzwert (abgesehen von Randeffekten), es müssen also vor dem Plotten die Werte gelöscht werden, die außerhalb des Wertebereiches liegen.

Das Analysegebiet ist definiert als Linienzug aus `diff_analyse_grd.bln` . Mittels 'matplotlib.Path' lässt sich daraus ein geschlossener Pfad (Polygon) generieren, über den getestet werden kann, ob ein Punkt innerhalb oder außerhalb des Polygons liegt. Führt man diese Überprüfung für jedes x/y-Wertepaar durch, die das Gitter bilden, erhält man ein 2d-Array von Boolean-Werten mit der Information, ob ein Punkt innerhalb oder außerhalb des Analysegebiets liegt.

```
region_path = Path(polyline_region[["X[m]", "Y[m]"]].values)
```

Aus den x/y-Koordinaten pro Gridpunkt wird ein 1d-Array mit der Liste aller x/y-Paare erstellt, für die dann getestet werden kann ob sie innerhalb des Gebiets liegen.

```
grid_xy = np.vstack((grid_x.flatten(), grid_y.flatten())).T
```

Die x/y-Paare werden nun auf ein Boolean gemapped, das `True` ist wenn der Punkt innerhalb des Analysebereichs liegt. Da das Ergebnis zunächst ein 1d-Array ist, wird es anschließend wieder in ein (900,600)-Array umgeformt. Am Ende wird das Array invertiert - `True` ist nun ein Bildpunkt außerhalb des Analysegebiets. Dies entspricht der Anwendung von Masken in `numpy` , bei der alle Werte gelöscht werden bei denen die Maske `True` ist.

```
region_mask = region_path.contains_points(grid_xy)
region_mask = region_mask.reshape(grid_x.shape)
region_mask = np.logical_not(region_mask)
```

Anschließend kann die Maske auf alle 2d-Arrays angewandt werden, die auf den Analysebereich begrenzt werden sollen:

```
grid_values = np.ma.masked_array(grid_values, mask=region_mask)
```

2.6.3 Farbverläufe

Um die Höhendifferenzen visual darstellen zu können, werden die Werte innerhalb eines vorgegebenen Bereichs ($-5[m] \leq \Delta H \leq 17[m]$) mit verschiedenen Farben an die entsprechende Stelle auf der Karte eingezeichnet.

Für besseres Verständnis wurde ein speziell zugeschnittener Farbverlauf definiert, der folgende Eigenschaften aufweist:

- Positive Höhendifferenz in gelb bis orange
- Negative Höhendifferenz in Grün-Schattierungen
- Geringe Höhendifferenz ($|\Delta H| < 0.5$) ohne Farbdarstellung, danach gradueller Übergang in den Farbbereich

Ein Farbverlauf in `matplotlib` lässt sich durch ein (256,4) Array beschreiben, also 256 RGBA-Werte. Um die Farbe für jeden Index in 0-255 zu bestimmen, wurde dieser Index zunächst unter Berücksichtigung des oben genannten Wertebereichs auf ΔH gemapped um zu entscheiden, ob man positive oder negative Höhendifferenz hat, bzw. welcher Transparenzwert angelegt werden sollte.

Der Absolutwert der Höhendifferenz wird anschließend wieder auf einen neuen Bereich von 0-255 gemapped und je nach Vorzeichen aus einem von 2 in der Programmbibliothek vorhandenen Basis-Farbverläufen entnommen. Somit ergibt sich ein zusammengesetzter Farbverlauf mit einer Transparenzlücke nahe $\Delta H = 0$.

```
min, max = -5, 17

top = cm.get_cmap("autumn_r")
bottom = cm.get_cmap("summer")

colors = np.ndarray((256, 4))
for i in range(256):
    dz = np.interp(i, [0, 255], [min, max])
    if dz > 0:
        colors[i, :] = top(int(np.interp(dz, [0, max], [0, 255])))
    else:
        colors[i, :] = bottom(int(np.interp(dz, [min, 0], [0, 255])))
    colors[i, -1] = np.interp(abs(dz), [0.5, 2], [0, 1])

cmap = ListedColormap(colors)
cmap.set_over(top(255))
cmap.set_under(bottom(0))
```

Dieses Vorgehen kann und wird für andere Wertebereiche wiederholt, um jeweils wieder einen der Aufgabe angepassten Farbverlauf zu generieren.

2.6.4 Farbflächen und Legende

Mittels Maskierung und Farbverläufen können die Daten nun in den Plot gezeichnet werden:

```
diff_colors = ax.contourf(
    grid_x,
    grid_y,
    grid_z_masked,
    levels=np.arange(-5, 17, 1),
    cmap=cmap,
)
```

Die Angabe von ‘levels‘ führt hier dazu, dass Werte auf 1m Genauigkeit zwischen -5m und 17m beschränkt werden. Die Farbverläufe sind dadurch weniger glatt, als würde man ohne Level-Angabe plotten.

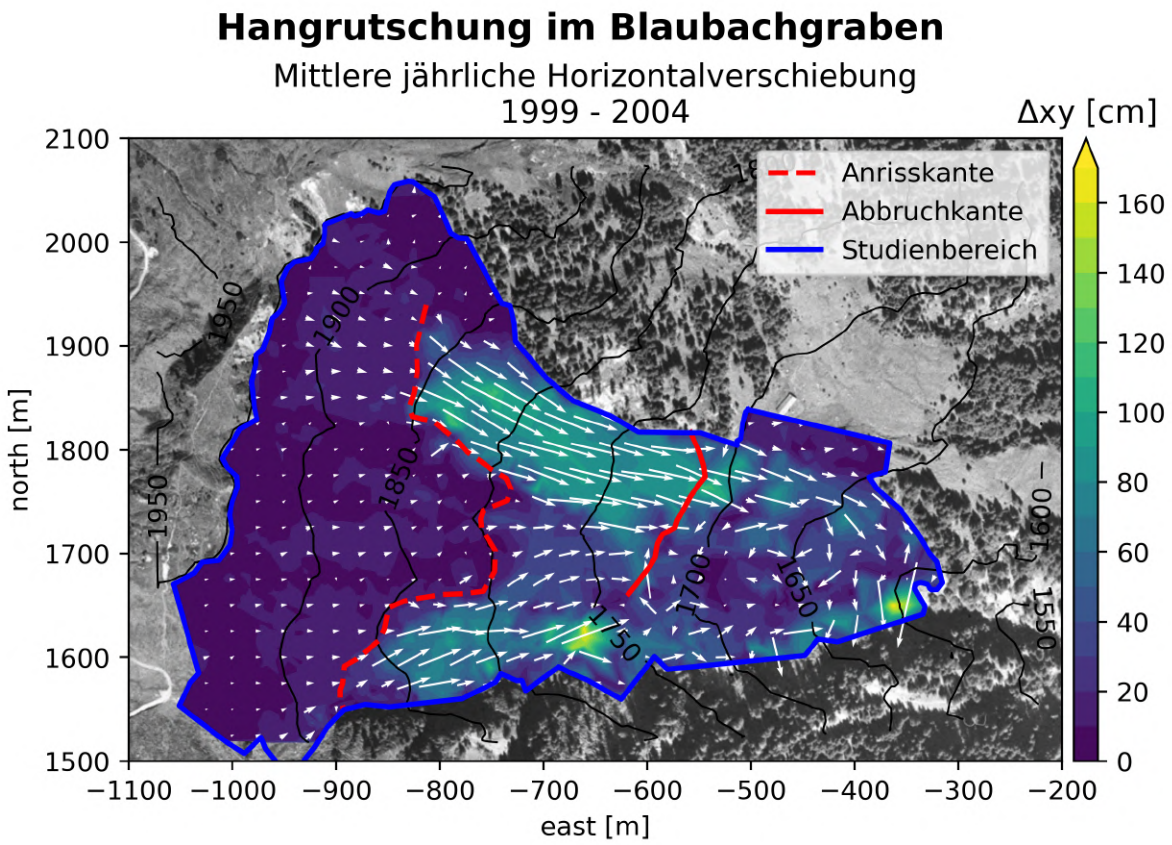
Um eine Legende zu erstellen, welche die Bedeutung der verschiedenen Farben erklärt, muss eine `colorbar` erstellt werden. Die von `matplotlib` gewählte Standard-Anordnung ist allerdings unschön, daher wird zunächst noch händisch ein Achsenbereich im Diagramm definiert, in den die Farbskala gezeichnet wird:

```
cax = fig.add_axes(
    [
        ax.get_position().x1 + 0.01,
        ax.get_position().y0,
        0.02,
        ax.get_position().height,
    ],
)
diff_colors_bar = fig.colorbar(diff_colors, cax=cax)
diff_colors_bar.ax.set_title(ax_label)
```

2.6.5 Vektorfelder

Der letzte Plot verwendet ebenfalls eine flächige Farbdarstellung. Statt der Differenz der Geländehöhen wird hier allerdings der Betrag der horizontalen Verschiebung von Geländepunkten dargestellt, entnommen aus dem Wert `"dS_2D/T[cm]"` der 2004er-Messreihe.

Zusätzlich wird über das Farb-Feld noch ein Vektor-Feld gezeichnet, das die Richtung und Stärke der horizontalen Verschiebung anzeigt:



Das Vektorfeld setzt sich aus zwei 2d-Arrays zusammen für die x- und y-Verschiebung. Diese werden in gewohnter Weise linear aus den Spalten `"dX/T[cm]"` und `"dY/T[cm]"` interpoliert. Diese beiden Werte können dann an die `quiver` Funktion von `matplotlib` übergeben werden, um ein Vektorfeld mit Pfeilen über die Karte zu zeichnen.

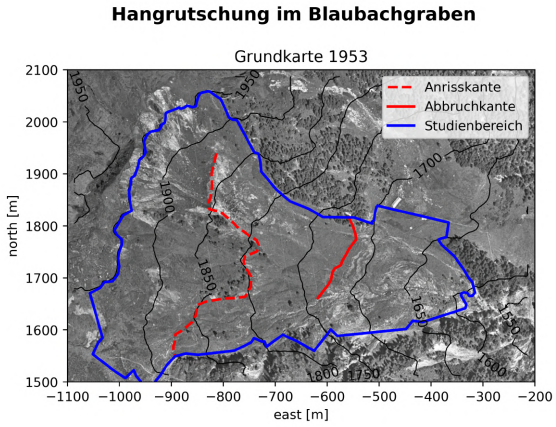
Die Interpolation liefert eine Richtung für jeden Pixel im Bild. Da dies die Darstellung überladen würde, könnte man eine geringere Gitter-Auflösung vor der Interpolation wählen. Oder, einfacher, nur jeden $n$ten Punkt aus dem Gitter für die Darstellung verwenden.

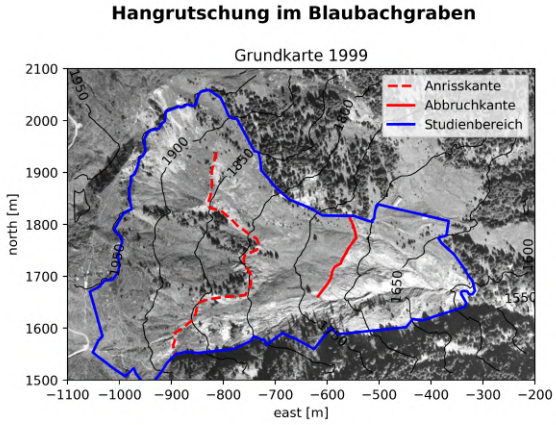
Laut Vorgabe soll jeder 5. Punkt verwendet werden, die Darstellung ist damit aber immer noch sehr überladen, also wird hier nur jeder 25. Punkt verwendet. Dazu wird die Slicing-Syntax mit Step size von 25 verwendet:

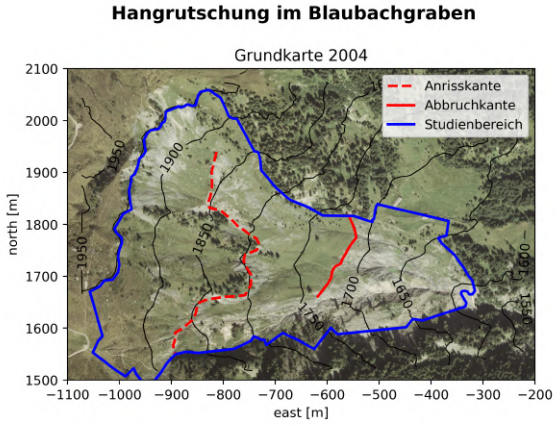
```
ax.quiver(
    grid_x[::25, ::25],
    grid_y[::25, ::25],
    grid_vec_u[::25, ::25],
    grid_vec_v[::25, ::25],
    units="xy",
    width=2,
    color="white",
    pivot="mid",
)
```

3 Ergebnisse und Interpretation

3.1 Grundkarten





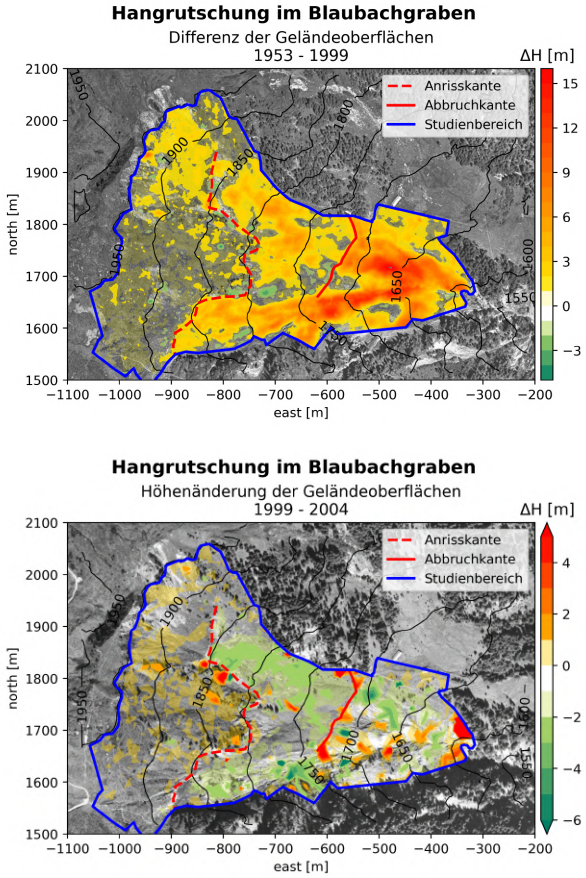


Die Grundkarten zeigen das Messgebiet aus der Vogelperspektive. Die Auflösung hat sich über die Jahrzehnte etwas verbessert, seit 2004 gibt es auch Farbinformationen über das Gebiet. Allerdings lässt sich aus den Orthofotos alleine wenig Aussagekraft über die Entwicklung des Rutschhanges ziehen.

Die eingezeichneten Höhenlinien sind mit Höhendaten des jeweiligen Jahres erstellt worden. Einige kleinere Abweichungen zwischen den Bildern lassen sich erkennen, ein direkter Vergleich ist mit dieser Darstellung aber sehr schwierig zu ziehen, und ähnelt eher den „Finde den Unterschied“-Bilderrätseln für Kinder.

Zusammenfassend lässt sich sagen, dass die Grundkarten gut geeignet sind um einen Überblick über das Messgebiet zu bekommen, aber kaum geeignet sind, die Veränderungen in dem Gebiet zu visualisieren.

3.2 Höhendifferenzen



Mittels Farbkodierung der Höhenunterschiede zeichnet sich ein deutlich genaueres Bild. Aufgrund des zusammengesetzten Farbverlaufs lässt sich leicht erkennen, wo Gelände abgetragen wurde, und wo es sich angesammelt hat.

Wenn man beachtet, dass rote Färbung den Bereich markiert, in der Höhe verloren wurde (siehe 2.6.1 Differenzmessung), ergibt sich das Bild, dass zwischen 1953 und 1999 großflächige Geländeverluste zu verzeichnen sind. Dies ist insbesondere östlich (d.h. unterhalb) der Anrisskante geschehen, reicht aber im Norden auch über die Anrisskante hinaus.

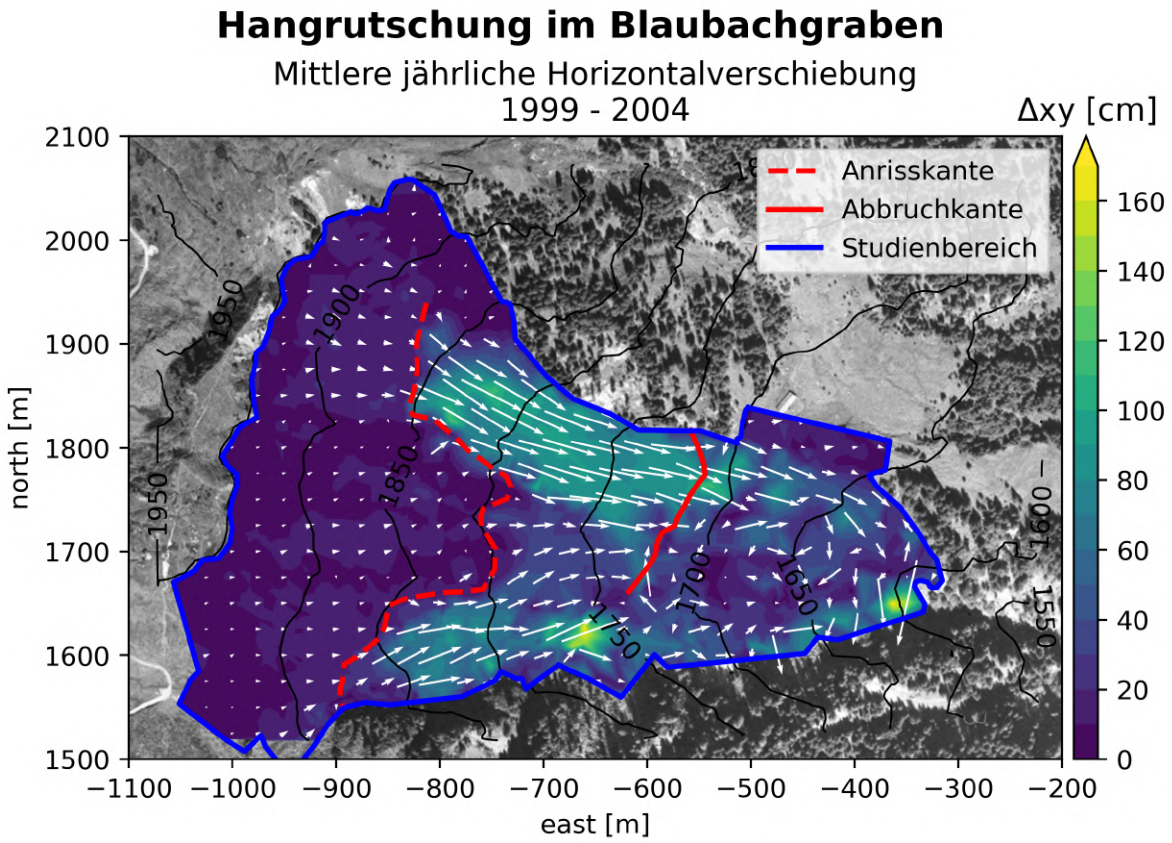
Unterhalb der Abbruchkante ist der Effekt nochmal deutlich stärker, mit Geländehöhendifferenzen über 15 Metern. Unklar ist, wohin sich das Material bewegt hat, innerhalb des Analysegebietes gibt es nur wenige Stellen mit einer leichten Anhäufigung. Es ist daher davon auszugehen, dass viel Material weiter bergab, außerhalb des Analysegebietes gerutscht ist.

Zwischen 1999 und 2004 zeigt sich ein differenzierteres Bild, mit deutlich geringeren Höhendifferenzen, die sowohl positiv wie auch negativ ausfallen. Da hier ein kleinerer Zeitbereich (5 statt 46 Jahren) zugrunde liegt, ist eine geringere Differenz auch zu erwarten.

Mit einem angenommenem Maximum von 15m für den ersten Zeitbereich und 5m für den zweiten, ist das Tempo der Bewegungen sogar deutich angestiegen (ca. 0.33m pro Jahr von 53-99, ca. 1m pro Jahr von 99-04). Mit nur drei Datenpunkten lässt sich hier aber nicht viel aussagen.

Auffällig ist, dass zwischen 1999 und 2004 unterhalb der Anrisskante Geländezugewinne zu vermerken sind, die von oberhalb der Kante zu kommen scheinen. Ein genauerer Ablauf der Erdbewegung lässt sich hier nur vermuten, dennoch kann man sagen, dass die Differenzbilder deutlich mehr Informationen vermitteln als die Grundkarten alleine.

3.3 Horizontale Verschiebungen



Die Karte mit den eingezeichneten Vektorfeldern liefert Aufschluss über die Bewegungsrichtungen am Rutschhang. Deutlich erkennt man eine nördliche und eine südliche Flanke zwischen Anriss- und Abbruchkante, die den Großteil der Bewegung talwärts (nach Osten) ausmachen. Unterhalb der Abbruchkante ist die Bewegung weniger geradlinig, die Erde bewegt sich hier in verschiedene Richtigungen, mit einem leichten Vorzug nach Süden aus dem Analysegebiet heraus.