BTO | Januari 2018

BTO 2018.008

Verkennend Onderzoek naar Visualisatie met Animaties

KWR Watercycle Research Institute

BTO

Verkennend Onderzoek naar Visualisatie met Animaties

BTO 2018.008 | Januari 2018

Opdrachtnummer
400695/048/004

Projectmanager
Jan Willem Kooiman

Opdrachtgever
BTO - Verkennend onderzoek

Kwaliteitsborger(s)
Tim van der Mast

Auteur(s)
Steven Ros

Verzonden aan

Jaar van publicatie
2018

Meer informatie
MSc, Steven Ros
T 0306069679
E steven.ros@kwrwater.nl

PO Box 1072
3430 BB Nieuwegein
The Netherlands

T +31 (0)30 60 69 511
F +31 (0)30 60 61 165
E info@kwrwater.nl
I www.kwrwater.nl

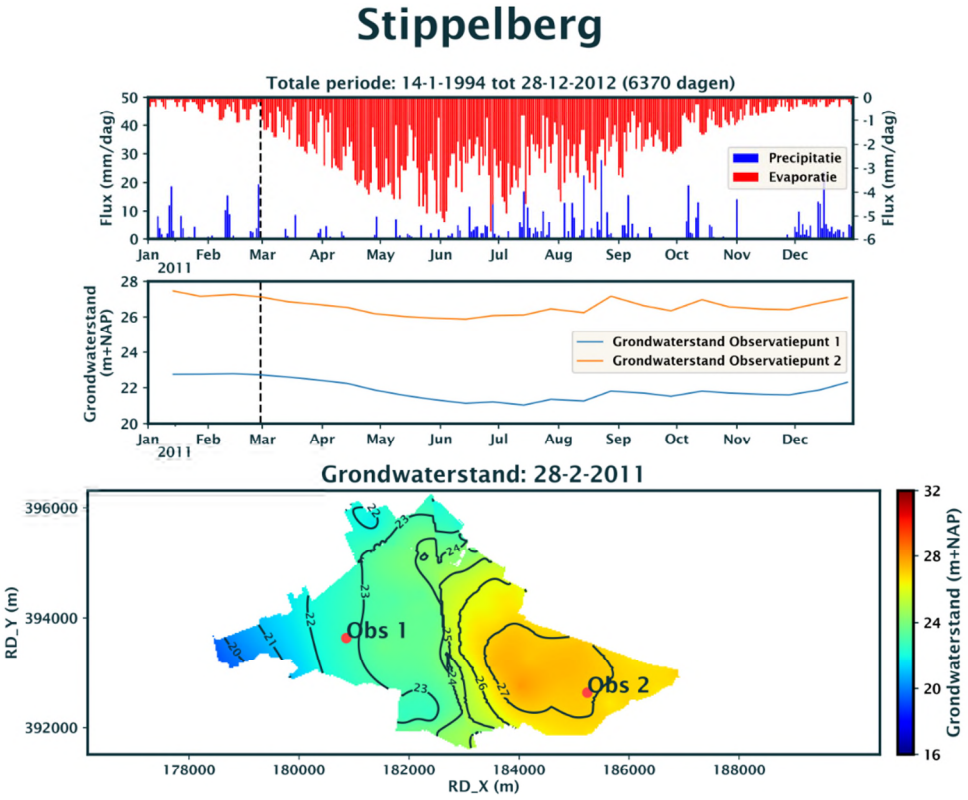
KWR Watercycle Research Institute

BTO *Managementsamenvatting*

Van ruwe data en figuren naar animaties: scripts en handleiding voor het visualiseren van onderzoeksresultaten

Auteur Steven Ros

Binnen het BTO-onderzoek is behoefte aan middelen om onderzoeksresultaten beter te visualiseren. In het Verkennend Onderzoek zijn daarom scripts in Python en een handleiding daarvoor ontwikkeld waarmee tijdsafhankelijke processen inzichtelijk kunnen worden gemaakt door tijdreeksen en aanzichten samen te voegen in overzichtelijke figuren en deze vervolgens als animatie aan elkaar te koppelen. Deze manier van presenteren kan voordelen bieden door meer inzicht te geven in de onderzochte situatie. Zo is het mogelijk om figuren met boven- en/of zijaanzicht te tonen, samen met tijdreeksen waarin veranderingen van gemodelleerde waarden (bv. concentraties of stijghoogten) met de tijd te zien zijn voor punten binnen het getoonde vlak. Dit geeft een beeld van hoe de verandering op een punt in relatie staat tot veranderingen in de ruimte (2D/3D). De handleiding maakt het voor iedereen met minimale voorkennis van de programmeertaal Python mogelijk om herkenbare animaties te ontwikkelen, die zijn voorzien van een KWR of BTO lay-out.



KWR

Onderste beeld: visualisatie met daarin stijghoogtereeksen voor twee observatiepunten op de Stippelberg, Obs 1 en Obs 2 op een specifieke datum (28-2-2011). Daarboven de verklarende tijdreeksen voor de grondwaterstanden en voor neerslag en verdamping (precipitatie en evaporatie) voor beide punten.

Belang: tijdsafhankelijke processen inzichtelijk maken

Bij onderzoeksrapportages en presentaties worden figuren, tabellen en grafieken gebruikt om uitkomsten inzichtelijk te maken en onderzoeksvragen te beantwoorden. Vaak is het praktisch niet mogelijk om alle relevante beelden te laten zien en worden bijvoorbeeld alleen eindsituaties of tijdreeksen voor enkele relevante punten getoond. In bepaalde gevallen is het echter niet alleen belangrijk om een tijdreeks te tonen, maar juist om te bestuderen hoe representatief dat ene punt is in het gebied, bijvoorbeeld om erachter te komen of een bepaalde problematiek elders ook kan spelen. Zelden kan er een goed beeld verkregen worden van de verandering in de omgeving met de tijd, omdat bij tijdreeksen en beelden óf de plaats gelijk blijft, óf de tijd. Het is daarom zeer relevant om te verkennen hoe tijdreeksen en boven- en/of zijaanzichten elkaar kunnen ondersteunen.

Aanpak: van ruwe data naar informatie aan de hand van de casus Stippelberg

In eerste instantie is gezocht naar een geschikte casus waarbij tijdreeksen kunnen bijdragen aan een verhoogd inzicht ten opzichte van een ruimtelijke (2D)-weergave. Tijdens een presentatie is gebruikers gevraagd naar hun mening, mogelijke onderwerpen en verwachtingen van de tool. Hieruit kwam de (BTO) Stippelberg casus naar voren als geschikt voor verdere uitwerking. Voor Stippelberg zijn scripts ontworpen om de beschikbare ruwe (ASCII)-data om te zetten naar informatie in de vorm van figuren. Deze figuren kunnen hierna achtereenvolgens worden getoond als animatie (met bijvoorbeeld een programma als ‘VirtualDub’).

Resultaten: handleiding voor het visualiseren van ruwe data met animaties

Er is een handleiding gemaakt waarmee iedereen met (enige) voorkennis van de programmeertaal Python tijdsafhankelijke processen inzichtelijk kan maken door tijdreeksen en aanzichten samen te voegen in overzichtelijke figuren en deze figuren vervolgens aan elkaar te koppelen in een animatie. De animaties worden door de software (tool) voorzien van een KWR of BTO lay-out en worden daardoor herkenbare digitale media voor externe communicatie.
Deze manier van presenteren kan voordelen bieden door meer inzicht te geven in de onderzochte situatie. Zo is het mogelijk om figuren met boven- en/of zijaanzicht te tonen, samen met tijdreeksen waarin veranderingen van gemodelleerde waarden (bv. concentraties of stijghoogten) met de tijd te zien zijn voor punten binnen het getoonde vlak. Dit geeft een beeld van hoe de verandering op een punt in relatie staat tot veranderingen in de ruimte (2D/3D).
De handleiding beschrijft het gebruik van de ontwikkelde vooropgezette scripts in Python-code en omvat een stappenplan voor de aanmaak van figuren en (eventueel) beelden.

Implementatie: Presentaties en ondersteunende handleiding

De ontwikkelde vooropgezette scripts en de handleiding kunnen worden ingezet om te komen van ruwe data naar informatie in de vorm van animaties. Hun resultaten kunnen dan dienen als naslagwerk voor de (casus-specifieke) opgeleverde scripts.
Na het schrijven van de conceptrapportage is de tool intern gepresenteerd aan KWR-onderzoekers als één van de nieuwe communicatiemiddelen, waarop een goede discussie volgde over mogelijk gebruik van de tool en aanvullende wensen. Dit kan worden gebruikt voor verdere ontwikkeling.

Rapport

Dit onderzoek is beschreven in rapport *Verkennend Onderzoek naar Visualisatie met Animaties* (BTO 2018.008).

Meer informatie
Steven Ros MSc
T 030 60 69 679
E steven.ros@kwrwater.nl

KWR
PO Box 1072
3430 BB Nieuwegein
The Netherlands

KWR Watercycle Research Institute

Samenvatting

Deze handleiding maakt het mogelijk om met minimale voorkennis van de programmeertaal Python animaties te ontwikkelen in KWR of BTO lay-out. Figuren worden aantrekkelijker en herkenbaarder in rapporten of als digitale media (qua type figuur, opmaak, legenda) wanneer ze een eenduidige stijl hebben en van hoge kwaliteit zijn. In dit verkennend onderzoek wordt specifiek aandacht besteed aan het inzichtelijk maken van tijdsafhankelijke processen door tijdreeksen en aanzichten samen te voegen in overzichtelijke figuren en deze als animatie te tonen. Dit brengt verschillende voordelen met zich mee.. Zo is het mogelijk om figuren met boven- en/of zijaanzicht te tonen, samen met tijdreeksen waarin veranderingen van gemodelleerde waarden (bv. concentraties of stijghoogten) met de tijd te zien zijn voor punten binnen het getoonde vlak. Dit geeft een beeld van hoe de verandering op een punt in relatie staat tot veranderingen in de ruimte (2D/3D) en leidt zo in sommige gevallen tot verhoogd inzicht in de onderzochte situatie.

Inhoud

1 Aanleiding en introductie

1.1 Aanleiding

Het BTO verkennend onderzoek Communicatiepilots heeft tot doel om de resultaten en inzichten die het BTO genereert met een vooropgestelde lay-out naar buiten te brengen. Dit kan door middel van figuren, maar ook met bewegend beeldmateriaal. Hiermee kan met minimale voorkennis de communicatie naar buiten toe met een zelfde type lay-out verzorgd worden. Figuren worden aantrekkelijker en herkenbaarder in rapporten of als digitale media (qua type figuur, opmaak, legenda) wanneer ze een eenduidige stijl hebben en van hoge kwaliteit zijn.

Er wordt veel gebruik gemaakt van figuren zoals grafieken, tabellen, zijaanzichten en bovenaanzichten, om de uitkomsten van een onderzoek inzichtelijk te maken en om onderzoeksvragen te kunnen beantwoorden. Doordat er een limiet is op het aantal figuren, worden binnen de hoofdtekst alleen figuren getoond die de belangrijkste conclusies onderbouwen. Denk hierbij aan het tonen van een eindsituatie (zij- en/of bovenaanzicht) of aan het plotten van tijdreeksen op slechts enkele punten in een groot gebied.

Boven- en zijaanzichten geven een tweedimensionale verdeling op één tijdstip weer en tijdreeksen tonen de verandering op één of enkele punten in een ruimte, of van één of meerdere gemeten parameters gedurende een meetperiode. In bepaalde gevallen is het echter niet alleen belangrijk om een tijdreeks te tonen, maar juist om te bestuderen hoe representatief dat ene punt is in het gebied. Bijvoorbeeld om erachter te komen of een bepaalde problematiek elders ook speelt. Zelden kan er een goed beeld verkregen worden van de verandering in de omgeving met de tijd, omdat bij tijdreeksen en aanzichten of de plaats gelijk blijft of de tijd. Het is daarom zeer relevant om te verkennen hoe tijdreeksen en boven- en/of zijaanzichten elkaar kunnen ondersteunen.

1.2 Introductie

In dit verkennend onderzoek wordt specifiek aandacht besteed aan het inzichtelijk maken van tijdsafhankelijke processen door tijdreeksen en aanzichten samen te voegen in overzichtelijke figuren en deze als animatie te tonen. Dit brengt verschillende voordelen met zich mee en leidt in sommige gevallen tot verhoogd inzicht in de onderzochte situatie. Zo is het mogelijk om figuren met boven- en/of zijaanzicht te tonen, samen met tijdreeksen waarin veranderingen van gemodelleerde waarden (bv. concentraties of stijghoogten) met de tijd te zien zijn voor punten binnen het getoonde vlak. Dit geeft een beeld van hoe de verandering op een punt in relatie staat tot veranderingen in de ruimte (2D/3D).

Voordelen van samenvoegen van figuren t.o.v. het gebruik van alleen een boven- en/of zijaanzicht:

- Inzicht krijgen in tweedimensionale processen doordat factoren/parameters die invloed kunnen hebben (zoals neerslag of evaporatie) bijgehouden kunnen worden in een meelopende tijdreeks.
- Op één of enkele observatiepunten kunnen bijhouden met de tijd of een limiet overschreden wordt om zo de oorzaak van de eventuele overschrijding te achterhalen. Dit kan bijvoorbeeld voor concentraties, stijghoogten, waterniveaus, temperaturen en drukken.

Voordelen samenvoegen van figuren t.o.v. het gebruik van alleen tijdreeksen:

- Inzichtelijk maken of metingen op een observatiepunt een representatief beeld schetsen voor het gebied.
- Inzichtelijk maken welke processen op de korte of lange termijn van invloed kunnen zijn, en wat de invloedsfeer is van deze effecten: waar kan een bepaalde problematiek (wel/verder) een rol spelen.

1.3 Opbrengsten

Directe opbrengsten van het project zijn:

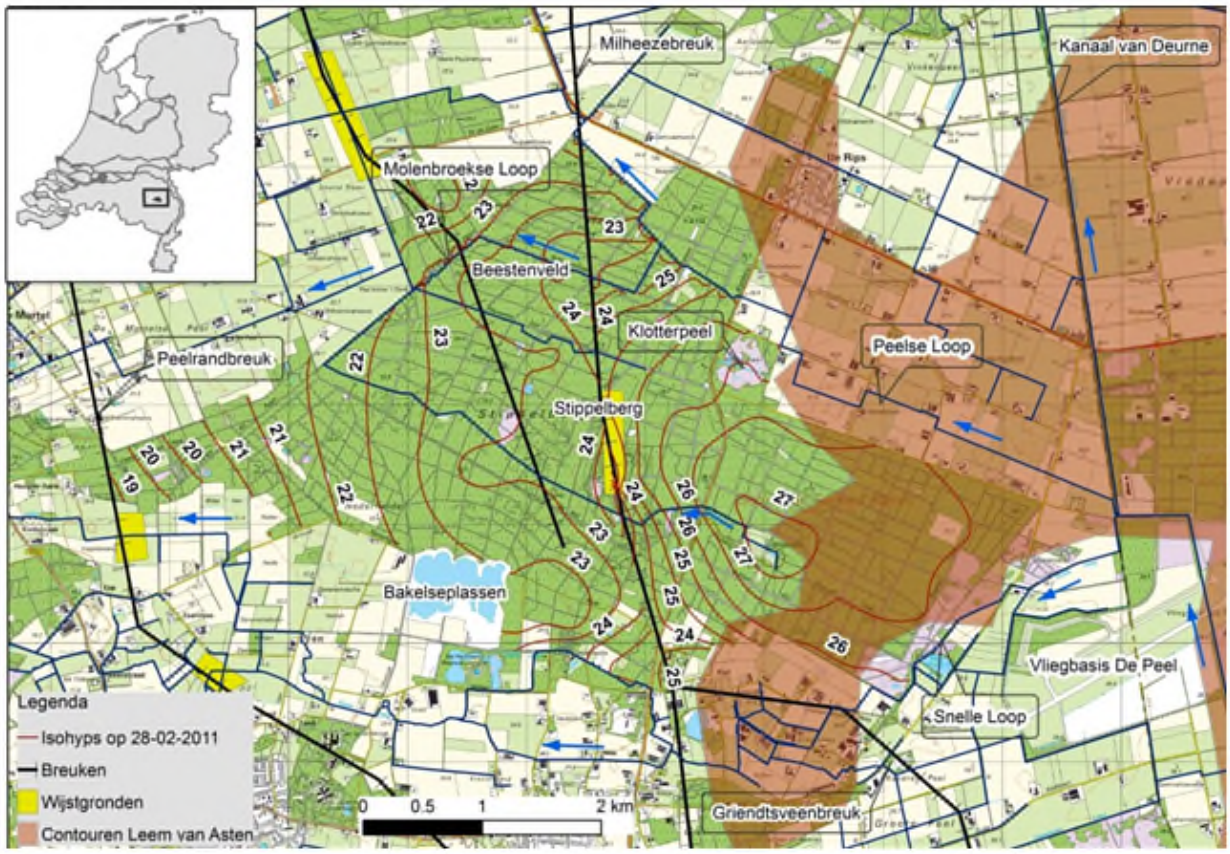
- Vooropgezette scripts in Python code (tool(s)) voor de aanmaak van figuren met een eenduidige stijl voor algemeen gebruik.
- Een handleiding met stappenplan voor het maken van figuren en (eventueel) beelden.

De opgeleverde tool(s) met bijbehorende handleiding zijn bedoeld voor belanghebbenden met (enige) voorkennis van de programmeertaal Python en dient ter ondersteuning voor het maken van visualisaties met animaties met een algemene lay-out.

Omdat uitkomsten van modelleringen niet op dezelfde manier worden weggeschreven, is het niet mogelijk om een algemeen script te maken die iedereen rechtstreeks kan gebruiken zonder voorbewerking van deze ruwe data. Binnen dit verkennend onderzoek (VO) is het BTO project Stippelberg gekozen als casus voor de handleiding. Om de in de handleiding bijgevoegde scripts overzichtelijk te houden wordt er onderscheid gemaakt tussen algemene secties en casus-specifieke secties. De aanpak van programmering van de Visualisatie-tool wordt beschreven in hoofdstuk 2.

1.4 Stippelberg

Binnen dit VO is gebruik gemaakt van data afkomstig van het BTO project Stippelberg. Voor de Stippelbergregio is onderzoek gedaan naar de verdroging van het natuurgebied en naar de mogelijkheden om de waterbeschikbaarheid voor de natuur rondom de Stippelberg weer te verhogen (van Loon et al., 2014; Figuur 1). Voor dit onderzoek is een infiltratieproef uitgevoerd op de Stippelberg. Daarnaast is er gemodelleerd door gebruik te maken van vlakdekkende tijdreeksanalyse. De uitkomsten van deze tijdreeksanalyse dienen als basis voor de visualisaties die gemaakt zijn binnen dit VO.



FIGUUR 1: TOPOGRAFISCHE KAART VAN DE REGIO STIPPELBERG MET GRONDWATERSTANDEN (ISOHYPSEN) VAN 28 FEBRUARI 2011, EEN DROOG VOORJAAR.

2 Aanpak modellering

2.1 Opzet Visualisatie-tool

Er zijn (binnen python) veel verschillende manieren om plots te maken van tijdreeksen en boven- en/of zijaanzichten. Het kan zijn dat er andere stappen gezet moeten worden of dat er stappen weggelaten kunnen worden om de visualisaties te maken, maar de hoofdlijnen zullen overeenkomen met de stappen die beschreven zijn binnen dit VO.

De output van de Stippelberg modellering bestaat uit ASCII-bestanden voor zowel het westelijk deel als het oostelijk deel – die gescheiden zijn door een breukvlak - binnen de gemodelleerde periode van 14 januari 1994 tot 28 december 2012. ASCII-bestanden ('.asc') zijn eenvoudige tekstbestanden zonder opmaak. Deze bestanden kunnen zoals veel simpele tekstbestanden (net als bijvoorbeeld '.txt') door (bijna) iedere tekstverwerker worden geopend. Het oostelijk en westelijk deel zijn in Python met elkaar verbonden voordat er visualisaties en een animatie van gemaakt worden. Voor de uiteindelijke visualisaties zijn drie scripts gebruikt:

- Stippelberg_convertASCII.py: leest de ASCII-output in en vertaalt deze naar numpy arrays voor het totale gebied van Stippelberg (West en Oost) voor iedere gemodelleerde datum.
- Visualisaties_VO.py: bevat het uitgewerkte script voor het maken van de visualisatiesom de specifieke input (numpy arrays voor iedere gemodelleerde datum) om te zetten, samen met dagelijkse data voor precipitatie en evaporatie. De figuren kunnen worden voorzien van een KWR of BTO lay-out.
- cross_sections.py: Het script 'cross_sections.py' ondersteunt de plotfuncties binnen 'Visualisaties_VO.py'. Dit script is niet (in detail) beschreven in deze handleiding.

2.2 Stappenplan: maken van visualisaties met animaties

Om de visualisaties te maken is het van belang dat de invoerdata uiteindelijk overeenkomt met het visualisatiescript 'Visualisaties_VO.py', d.w.z. numpy arrays met waarden voor iedere te plotten datum. Overige parameters die via tijdreeksen getoond kunnen worden, zoals neerslagdata, verdampingsdata, stijghoogtemetingen, gemeten concentraties of anderzijds, kunnen eveneens worden toegevoegd.

Binnen dit VO is de verwerking van data van ASCII-bestanden naar numpy arrays beschreven. Als het bestandstype van de modeloutput niet overeenkomt met ASCIIs (of een vergelijkbaar eenvoudig tekstbestandtype), dan is het noodzakelijk om de data op een andere manier om te zetten naar numpy arrays. De beschrijving van het script 'Stippelberg_convertASCII.py' kan in dat geval overgeslagen worden.

2.2.1 Stippelberg_convertASCII.py

Het script 'Stippelberg_convertASCII.py' leest ASCII-bestanden in voor iedere gemodelleerde tijdstap. Deze ASCII-bestanden bevatten stijghoogtewaarden per rij en voor iedere kolom van dit bovenaanzicht. Deze waarden worden opgeslagen in numpy arrays, voor zowel het westelijk als het oostelijk deel van de Stippelberg. Vervolgens worden de arrays samengevoegd in één numpy array, waarbij overlap tussen beide delen verwijderd wordt. De uiteindelijke stijghoogtearrays worden opgeslagen als numpy arrays, samen met numpy arrays waarin de X-coördinaten en Y-coördinaten per cel zijn opgeslagen.

2.2.2 Visualisaties_VO.py

De volgende invoerdata dienen beschikbaar te zijn om zonder (veel) extra aanpassingen visualisaties te kunnen maken (stap 0):

- Numpy arrays voor iedere gemodelleerde tijdstap
- Gegevens van celdikten, celbreedten en cellengten om de afbeeldingen op een juiste manier op schaal te kunnen tonen; X-coördinaten en Y-coördinaten voldoen voor bovenaanzichten.
- Eventueel: tijdreeksen van neerslag, verdamping of anderzijds om mogelijke verbanden met de boven- en/of zijaanzichten duidelijk te kunnen maken (als csv-bestand, excelbestand of anderzijds).

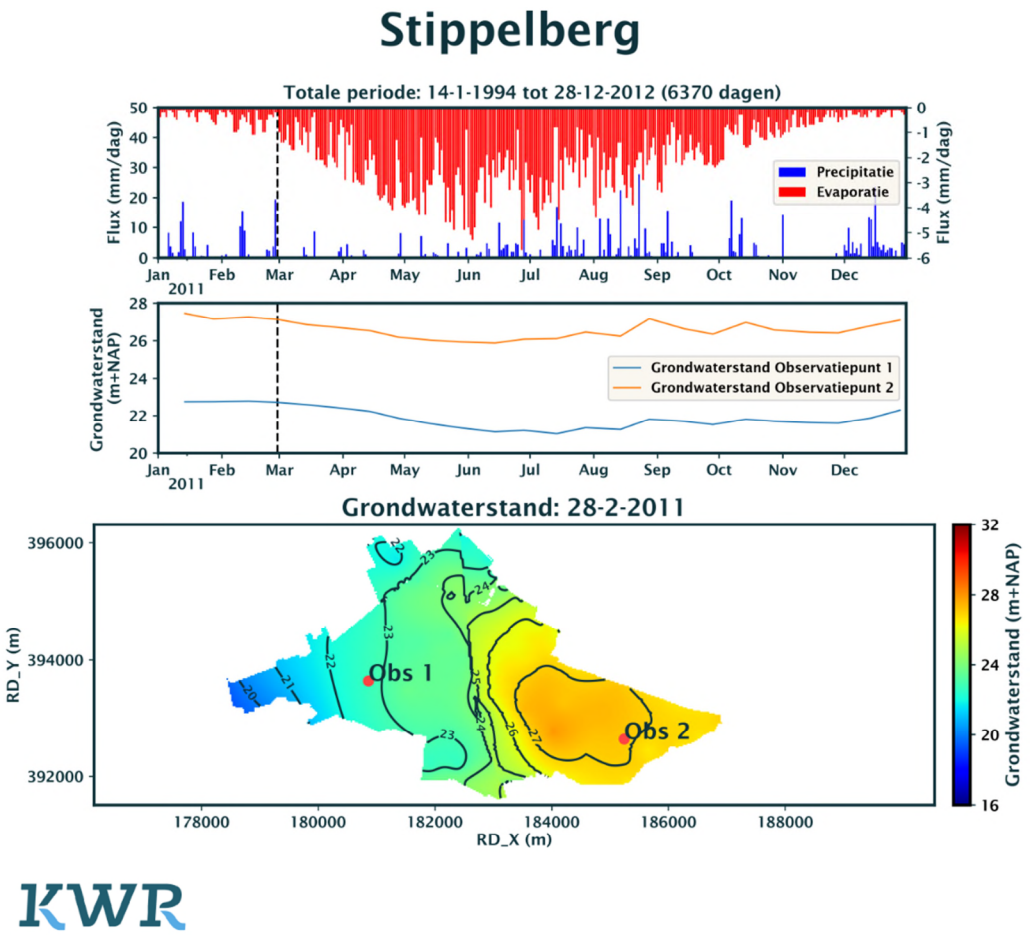
De volgende stappen dienen als leidraad voor het ontwerpen van de plots (en zijn inbegrepen in de tool), gebruik makend van het script 'Visualisaties_VO.py':

1. Bepaal welke modules of packages geïmporteerd moeten worden: aanbevolen is om de huidige packages te laten staan om een goede werking van het script te waarborgen.
2. Stel tijdreeksen in 'Pandas' (een vrij verkrijgbare python package) of soortgelijke type tijdreeksopslag op om voor iedere gemodelleerde tijdstap een visualisatie te kunnen maken, indien gewenst met ondersteunende parameters zoals neerslag en/of verdamping.
3. Laad numpy arrays in voor iedere tijdstap om boven- en/of zijaanzichten te kunnen maken.
4. Zorg ervoor dat arrays omgezet worden naar een homogeen grid met een vooraf gedefinieerd domein, zodat de plots op de juiste wijze worden getoond.
5. Definieer de positie van de subplots binnen de figuren en optimaliseer de lay-out afhankelijk van de te plotten minimum en maximumwaarden, de periode (maanden, jaren, decennia) en het domein; d.w.z. de afstand over de X-as en Y-as die men wil tonen, inclusief de X- en Y-coördinaat van de hoek linksonder in het aanzicht.
6. Pas bij de creatie van figuren waar nodig teksten en waardes van titels en labels aan en geef aan wat de minimum- en maximumwaarden in het boven- of zijaanzicht zijn, zodat de figuren overzichtelijk worden en de juiste informatie overbrengen.
7. Sla de figuren op en zorg voor juiste nummering ervan (001, 002, 999, 1001, etc.). Dit maakt het mogelijk om eenvoudig video's te maken met VirtualDub (binnen dit project gebruikt) of een soortgelijk programma.

3 Handleiding: Visualisatie-tool

In dit hoofdstuk worden de opgeleverde Python scripts 'Stippelberg_convertASCII.py' en 'Visualisaties_VO.py' besproken en waar nodig voorzien van extra commentaar, zie Tabel 2 en Tabel 3. Het commentaar wordt ondersteund met figuren. Om de in de handleiding bijgevoegde scripts overzichtelijk te houden is er onderscheid gemaakt tussen algemene secties (geel gearceerd) en casus-specifieke secties (groen gearceerd).

De in de scripts gebruikte termen zijn omschreven in Tabel 1. Een voorbeeldfiguur van de uiteindelijke visualisaties is getoond in Figuur 2. Aan het einde van dit hoofdstuk is beschreven hoe de figuren omgezet kunnen worden in een animatie. Deze animatie is hier te vinden: Animatie van grondwaterstanden Stippelberg (BTO-net).



FIGUUR 2: VISUALISATIE MET DAARIN STIJGHOOGTEREEKSEN VOOR TWEE OBSERVATIEPUNTEN OP DE STIPPELBERG - AANGEGEVEN IN HET BOVENAANZICHT MET GRONDWATERSTANDEN – AANGEVULD MET VERKLARENDE TIJDREEKSEN VOOR NEERSLAG EN VERDAMPING (DATUM: 28 FEBRUARI 2011).

TABEL 1: SCRIPT-SPECIFIEKE TERMEN MET VERKLARING.

Term	Verklaring
ASCII	Eenvoudig tekstbestandtype met extensie '.asc', te lezen als '.txt'-bestandstype
workspace/w_s	Werkmap: hierin bevinden zich relevante data of mappen of scripts
ncol	Totaal aantal kolommen
nrow	Totaal aantal rijen
delc	Onderlinge afstand tussen de rijen (van boven naar beneden)
delr	Onderlinge afstand tussen de kolommen (van links naar rechts)
well_DimX	Ligging observatiepunt binnen de doorsnedeplot (X-richting)
well_DimY	Ligging observatiepunt binnen de doorsnedeplot (Y-richting)
ex_vert	Verticale overdrijving binnen de figuur (doorsnedeplot)
nr_of_xlab	aantal labelpunten X-as
nr_of_ylab	aantal labelpunten Y-as
x_null	Locatie eerste X-label
y_null	Locatie eerste Y-label
stopX	Locatie laatste X-label
stopY	Locatie laatste Y-label
xtic_left	Getoonde waarde eerste X-label
xtic_right	Getoonde waarde laatste X-label

3.1 Handleiding Stippelberg_convertASCII.py

TABEL 2: SCRIPT OM MODEL OUTPUT DATA (ASCII-BESTANDEN) VOOR DE STIPPELBERG CASUS OM TE ZETTEN NAAR (NUMPY)-ARRAYS: 'STIPPELBERG_CONVERTASCII.PY'

Script	Commentaar
	Dit script zet ASCII-bestanden om naar numpy arrays (arrays aangemaakt met (de functionaliteit van) de numpy module) en voegt twee gebieden samen die voorafgaand aan de modellering gescheiden waren. De outputbestanden (numpy arrays) zijn vervolgens gebruikt om visualisaties te maken, zie hiervoor 'Visualisaties_VO.py'.
################## Importeer benodigde packages en modules ################## import glob import numpy as np import os import copy import re from datetime import datetime	De modules/packages zijn specifiek geïmporteerd voor dit script. Mocht de invoer (hier: ASCII-bestanden) afwijken, dan kan het nodig zijn om andere modules te importeren voor de verwerking van de data.

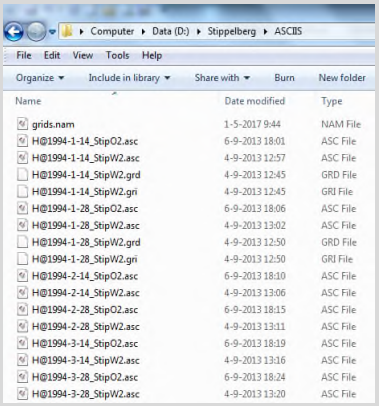
# Voeg de werkmap toe: 	
w_s = r'D:\Stippelberg'
	Voeg hier een verwijzing naar de werkmap toe: Gebruik r'Dir:\Map\Submap' (raw string)

```
# Controle of de map bestaat
if not os.path.exists(w_s):
    os.makedirs(w_s)

os.chdir(w_s)
```

Als de map nog niet bestaat dan wordt deze hier aangemaakt.

os.chdir() verandert de werkmap binnen Python

```
# Locatie ASCII-bestanden
ascii_file_loc = r'D:\Stippelberg\ASCIIS'
```




```
# Verzamel alle ASCII's
allFiles = glob.glob(ascii_file_loc + "/*.asc")
# Verzamel westelijk deel van gebied
westFiles = glob.glob(ascii_file_loc + "/*W2.asc")
# Verzamel oostelijk deel van gebied
eastFiles = glob.glob(ascii_file_loc + "/*O2.asc")
```

Slaat alle verwijzingen naar ASCII-bestanden op in een Python-list van strings eindigend op '.asc'. of 'W2.asc' of 'O2.asc'.
De 're' module (regular expressions) kan gebruikt worden om bestandsverwijzingen specifieker te selecteren.

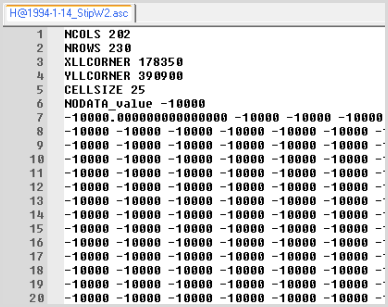
```
# Lees ASCII's in voor het westelijk deel
head_W = {}
head_E = {}
```

Maak 'dictionaries' aan om voor iedere tijdstap stijghoogten-arrays op te kunnen slaan voor zowel het westelijk als oostelijk deel.

```
head_input = {}
for file_path in westFiles:
    with open(file_path, 'r') as head_input[file_path]:
```

Voor ieder ASCII-bestand voor het westelijk deel, open bestand als:

De inhoud van het bestand ziet er als volgt uit, met spaties om woorden te scheiden




```
# Simulatiedatum:
```

iDate = datetime.strptime(re.findall('([1-9]+.*)_Stip',file_path)[0], "%Y-%m-%d").strftime("%Y-%m-%d")	Datum waarvoor stijghoogten zijn gemodelleerd (bv: 2008-04-28) wordt gehaald uit de bestandsnaam.

# Aantal kolommen 	
ncol_W= int(head_input[file_path].readline().strip().split()[1])
	De methode strip verwijdert eventuele (extra) spaties aan de linker- en rechterkant van woorden. De methode split maakt een lijst van strings aan. Item [1] is het tweede item in deze lijst.

# Aantal rijen 	
nrow_W=int(head_input[file_path].readline().strip().split()[1])	

# X-coordinaat linksonder XLL_WestCorner = float(head_input[file_path].readline().strip().split()[1])	
# Y-coordinaat linksonder YLL_WestCorner = float(head_input[file_path].readline().strip().split()[1])	

# Celgrootte (homogeen) 	
cellsize_W=float(head_input[file_path].readline().strip().split()[1])
# NaN-waarde van het grid 	
NAN_Value=float(head_input[file_path].readline().strip().split()[1])
	NaN (Not a Number) is de waarde die toegekend is aan niet-actieve cellen.

# Voor het aantal rijen binnen het grid: lees ncol aantal #stijghoogten in en schrijf deze weg in een initieel leeg 2D-array 	
head_W[iDate] = np.empty((nrow_W, ncol_W), dtype = np.float32)
for i in xrange(nrow_W):
  head_items = []
  while(True):
  lastline = head_input[file_path].readline()
  head_items.extend(re.findall('(\S+)',lastline))
	re.findall('(\S+)): zoek alle aparte aaneengesloten karakters, gescheiden door spaties, en zet deze om in een lijst van strings.

# Stop de loop als iedere kolom in de rij van een waarde voorzien is 	
if len(head_items) == ncol_W:
  break
for j in xrange(ncol_W):
  if float(head_items[j]) == NAN_Value or float(head_items[j]) < -999.:
  head_W[iDate][i,j] = -999.99
	Vul (zo nodig) cellen met NaN-waarden op.

```
            else:
                head_W[iDate][i,j] = float(head_items[j])
    head_items = []

# Lees ASCII's in voor het oostelijk deel
head_input = {}
for file_path in eastFiles:
    with open(file_path, 'r') as head_input[file_path]:
```

Anders: vul de stijghoogtewaarde in voor deze rij en kolom van het array.

Voor ieder ASCII-bestand voor het oostelijk deel, open bestand als:

Het bestand voor het oostelijk deel komt overeen met het westelijk deel

H@1994-1-14_StipO2.asc

```
1   NCOLS 200
2   NROWS 230
3   XLLCORNER 182000
4   YLLCORNER 390900
5   CELLSIZE 25
6   NODATA_value -10000
7   -10000.000000000000000 -10000
8   -10000 -10000 -10000 -10000 -1
9   -10000 -10000 -10000 -10000 -1
10  -10000 -10000 -10000 -10000 -1
11  -10000 -10000 -10000 -10000 -1
12  -10000 -10000 -10000 -10000 -1
13  -10000 -10000 -10000 -10000 -1
14  -10000 -10000 -10000 -10000 -1
15  -10000 -10000 -10000 -10000 -1
16  -10000 -10000 -10000 -10000 -1
```

```
        # Simulatiedatum:
        iDate = datetime.strptime(re.findall('([1-
9]+.*)_Stip',file_path)[0],
        "%Y-%m-%d").strftime("%Y-%m-%d")

        # Aantal kolommen
        ncol_E =
int(head_input[file_path].readline().strip().split()[1])

        # Aantal rijen
        nrow_E =
int(head_input[file_path].readline().strip().split()[1])

        # X-coordinaat linksonder
        XLL_EastCorner =
float(head_input[file_path].readline().strip().split()[1])

        # Y-coordinaat linksonder
        YLL_EastCorner =
float(head_input[file_path].readline().strip().split()[1])

        # Celgrootte (homogeen)
        cellsize_E =
float(head_input[file_path].readline().strip().split()[1])

        # NaN-waarde van het grid
        NAN_Value =
float(head_input[file_path].readline().strip().split()[1])

        # Voor het aantal rijen binnen het grid: lees ncol aantal
        # stijghoogten in en schrijf deze weg in een initieel leeg
```

Datum waarvoor stijghoogten zijn gemodelleerd (bv: 2008-04-28) wordt gehaald uit de bestandsnaam.

De methode strip verwijdert eventuele (extra) spaties aan de linker- en rechterkant van woorden. De methode split maakt een lijst van strings aan. Item [1] is het tweede item in deze lijst.

NaN (Not a Number) is de waarde die toegekend is aan niet-actieve cellen.

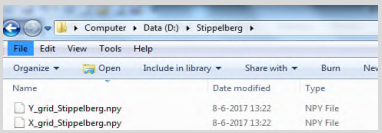
```
2D-array
        head_E[iDate] = np.empty((nrow_E, ncol_E), dtype =
np.float32)
        for i in xrange(nrow_E):
            head_items = []
            while(True):
                lastline = head_input[file_path].readline()
                head_items.extend(re.findall('(\S+)',lastline))



# Stop de loop als iedere kolom in de rij van een waarde
voorzien is
                if len(head_items) == ncol_E:
                    break
            for j in xrange(ncol_E):
                if float(head_items[j]) == NAN_Value or
float(head_items[j]) < -999.:
                    head_E[iDate][i,j] = -999.99

                else:
                    head_E[iDate][i,j] = float(head_items[j])
        head_items = []

# Gat in data West + Oost; Totale lengte array 1450 m korter
overlap = 56  # 58 * 25 m = 1450 m  (cellsize = 25 m)

# Afstanden samengevoegd array (in X-richting) t.o.v. X-
coördinaat linksonder
nCols = ncol_W + ncol_E - overlap
xmid = np.empty((nCols), dtype = np.float32)
xmid[0] = XLL_WestCorner + 0.5 * cellsize_W
for j in xrange(1, nCols):
    xmid[j] = xmid[j-1] + cellsize_W

# Afstanden samengevoegd array (in Y-richting) t.o.v. Y-
coördinaat linksonder
ymid = np.empty((nrow_W), dtype = np.float32)
ymid[0] = YLL_WestCorner + (nrow_W * cellsize_W - 0.5 *
cellsize_W)
for i in xrange(1,nrow_W):
    ymid[i] = ymid[i-1] - cellsize_W

# XY-coordinaten van iedere cel binnen samengevoegd grid
X_grid = np.empty((nrow_W, nCols), dtype = np.float32)
Y_grid = np.empty((nrow_W, nCols), dtype = np.float32)
for i in xrange(nrow_W):
    for j in xrange(nCols):
        X_grid[i,j] = xmid[j]
        Y_grid[i,j] = ymid[i]

# Sla arrays op als numpy arrays
```

re.findall('(\S+)): zoek alle aparte aaneengesloten karakters, gescheiden door spaties.

Vul (zo nodig) cellen met NaN-waarden op.

Anders: vul de stijghoogtewaarde in voor deze rij en kolom van het array.

```
np.save(os.path.join(w_s, "X_grid_Stippelberg.npy"), X_grid)
np.save(os.path.join(w_s, "Y_grid_Stippelberg.npy"), Y_grid)
```




```
# Gesorteerde datums in datetime-format
keys_sorted_dt = []
for key in sorted(map(lambda x: datetime.strptime(x, "%Y-%m-
%d"), head_E.iterkeys())):
    keys_sorted_dt.append(key)
```

Gesorteerd op datum door gebruik te maken van ‘datetime’-format

```
# Gesorteerde datums in string-format
keys_sorted_str = []
for key in map(lambda x: datetime.strftime(x, "%Y-%m-%d"),
keys_sorted_dt):
    keys_sorted_str.append(key)
```

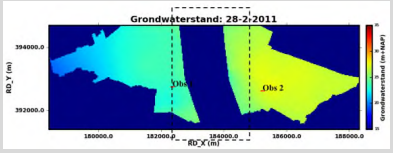
Zet vervolgens het type om naar string (tekst)

```
# Locatie outputfolder
outputdir = os.path.join(w_s, "Head_arrays")
if not os.path.exists(outputdir):
    os.makedirs(outputdir)
```

Indien nodig wordt er een map aangemaakt

```
# Startdatum
iTime = keys_sorted_str[0]

# Tel het aantal NaN-waarden vanaf de rechterkant van het
westelijk deel (= array)
count_W = np.zeros((nrow_W), dtype = np.int32)
for i in xrange(nrow_W):
    for j in xrange(ncol_W):
        if head_W[iTime][i,-(j+1)] < -999.:
            count_W[i] += 1
        else:
            break

# Tel het aantal NaN-waarden vanaf de linkerkant van het
oostelijk deel  (= array)
count_E = np.zeros((nrow_E), dtype = np.int32)
for i in xrange(nrow_E):
    for j in xrange(ncol_E):
        if head_E[iTime][i,j] < -999.:
            count_E[i] += 1
        else:
            break
```

De rechterkant van het westelijk deel en de linkerkant van het oostelijk deel bevatten NaN-waarden op het gebied waar beide delen gescheiden zijn. In dit gedeelte wordt per rij bepaald hoeveel cellen aan overlap er zijn in totaal (in zowel het westelijk deel als het oostelijk deel).




```
# Totaal aantal NaN-waarden tussen beide delen (overlap zou
moeten zijn: 56 cellen)
count_tot = count_W + count_E
```

# Combineer stijghoogtearrays door de NaN-waarden in het tussenliggende deel te 	
# verwijderen. Houdt Een groot gecombineerd array met stijghoogtewaarden over. 	
head_comb_list = []
head_comb = {}
 	
for iTime in keys_sorted_str:
  # list
  head_comb_list.append(np.empty((nrow_W, nCols), dtype = np.float32)) # Only if rows
  head_comb_list[-1][:, :ncol_W] = copy.deepcopy(head_W[iTime][:,:])
  for i in xrange(nrow_E):
  for j in xrange(min(ncol_W- count_W[i], ncol_E - count_E[i])):
  if 55 <= count_tot[i] <= 57:
  head_comb_list[-1][i, (ncol_W - 1 - count_W[i]) + j] = head_E[iTime][i,j + count_E[i]]
 	
  np.save(os.path.join(outputdir, "Head_" + iTime + ".npy"), head_comb_list[-1])
 	
# Einde script 	
	Voor iedere datum: maak een numpy array aan om de arrays samen te kunnen voegen. Hierbij wordt data van het westelijk deel gekopieerd. Per rij worden cellen zonder waarden (NaN) rechts van actieve cellen (indien aanwezig) aangevuld met stijghoogtewaarden die gemodelleerd zijn binnen het oostelijk deel.
	
	De numpy arrays worden voor iedere gemodelleerde tijdstap opgeslagen.

3.2 Handleiding Visualisaties_VO.py'

TABEL 3: SCRIPT VOOR HET OPSTELLEN VAN TIJDREEKSEN EN HET MAKEN VAN VISUALISATIES: 'VISUALISATIES_VO.PY'. ALGEMENE SECTIES ZIJN GEEL GEARCEERD EN CASUS-SPECIFIEKE SECTIES ZIJN GROEN GEARCEERD.

Script	**Commentaar**
	Dit script is gebruikt voor het maken van een algemeen ontwerp voor de visualisaties. Hiervoor zijn tijdreeksen van KNMI voor neerslag en verdamping gebruikt (csv-bestanden) naast 2D numpy arrays met stijghoogtewaarden per cel. Om problemen met de uitvoer van het script te beperken, is het aanbevolen om afwijkende input voorafgaand aan de relevante secties om te zetten.
################## Importeer benodigde packages en modules ################## import glob import os import numpy as np import matplotlib.pyplot as plt import pandas as pd	De modules/packages zijn specifiek geïmporteerd voor dit script. Mocht de invoer (hier: csv-bestanden en numpy arrays) afwijken, dan kan het nodig zijn om andere modules te importeren voor de verwerking van de data.

```
import matplotlib
import copy
import re  # Regular expressions
import matplotlib.gridspec as gridspec
from datetime import datetime
from matplotlib.dates import DateFormatter, YearLocator,
MonthLocator
from mpl_toolkits.axes_grid1 import make_axes_locatable,
axes_size

''' Lay-out KWR/BTO figuren:
KWR:
-    Hoofdkleuren:
o   Blauw: #00B9EA
o   Zwart/Donkergroen voor lopende tekst: #0E323B
-    Accentkleuren:
o   Beige voor tekstvakken: #F9F5ED
o   Rood voor knoppen: #FF4443
o   Groen voor overige accenten: #62F0A2
-    Lettertype: Lucida Sans
-    KWR-logo wordt gebruikt zonder Pay-off 'watercycle
research institute'.
-    KWR-logo wordt linksboven of linksonder geplaatst.

BTO:
-    Hoofdkleuren:
o   Tekstkleur: #000000
o   Accentkleur 1: #237183
o   Accentkleur 2: #429136
-    Lettertype voor Kantoor (Office-)toepassingen: Lucida
Sans
-    Logo: er bestaat géén BTO-logo. De letters BTO mogen
niet los gebruikt worden. KWR is de afzender, dus dat logo
mag eventueel gebruikt worden.
'''
# KWR lay-out of BTO layout (KWR layout = False)?
KWR_layout = True

if KWR_layout:
    color_main = "#00B9EA"
    color_text = "#0E323B"
    color_face = "#F9F5ED"
    color_button = "#FF4443"
    color_accents = "#62F0A2"
    font_type = "Lucida Sans"
else:
    color_main = "#00B9EA"  # Geen BTO hoofdkleur; --> KWR
    color_text = "#000000"
    color_face = "#FFFFFF"  # Standaard: Witte tekstvlakken
    color_button = "#FF0000"  # Standaard: Kleur rood ("red")
    color_accents = "#000000"  # Geen BTO kleur: Zwart voor
accenten
```

Figuren kunnen worden gemaakt met een KWR lay-out of een BTO lay-out.

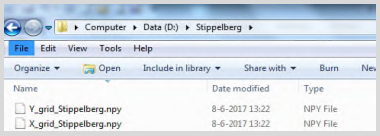
Geef hier de lay-out voor figuren weer. Zijn de figuren bedoeld voor een KWR rapport of voor een BTO rapport?

```
    font_type = "Lucida Sans"

# Voeg de werkmap toe:
workspace = r'D:\Stippelberg'
w_s = os.path.join(workspace)
# Controle of de map bestaat
if not os.path.exists(w_s):
    os.makedirs(w_s)

modelname = 'Stippelberg'  # Modelnaam

Lx = 10050.  # Lengte data (2D-doorsnede) in X-richting (X-as)
Ly = 5750.  # Lengte data (2D-doorsnede) in Y-richting (Y-as)

nrow = 230  # (Aantal rijen (2D-doorsnede) in dataset)
ncol = 346  # (Aantal kolommen (2D-doorsnede) in dataset)

# Onderlinge afstand tussen de rijen (van boven naar
beneden)
delc = np.empty((nrow), dtype=np.float64)
delc[:] = 25.
# Onderlinge afstand tussen de kolommen (van links naar
rechts)
delr = np.empty((ncol), dtype=np.float64)
delr[:] = 25.

# Map waarin (ruwe) gegevens opgeslagen zijn
data_loc = r'D:\Stippelberg\Head_arrays'

# Leest coordinatengrids in als (numpy-)arrays met grootte
(nrow, ncol):
xmid = np.load(os.path.join(w_s, "X_grid_Stippelberg.npy"))
ymid = np.load(os.path.join(w_s, "Y_grid_Stippelberg.npy"))



''' locatie van observatiepunten (optioneel) voor het uitlezen
van modelwaarden
## met de tijd op specifiek punten binnen het 2D-grid. '''
# Naam van punten
obs_punt = {1: "Observatiepunt 1", 2: "Observatiepunt 2"}
# Rij van de cel waarin een observatiepunt zich bevindt
obs_row = {1: [120], 2: [160]}
# Kolom van de cel waarin een observatiepunt zich bevindt
obs_col = {1: [100], 2: [275]}

## Lijst met bestandsnamen die ingeladen moeten worden
## Ieder bestand bevat een 2D-doorsnede op een bepaalde
datum/tijd.
headFiles = sorted(glob.glob(data_loc + "/*.npy"))
# N.B. In dit geval gesorteerd op datum; op basis van
bestandsnamen in data_loc
```

Voeg hier een verwijzing naar de werkmap toe: Gebruik r'Dir:\Map\Submap' (raw string)

Locatie van opgeslagen numpy arrays

Laad coördinaten van numpy arrays in

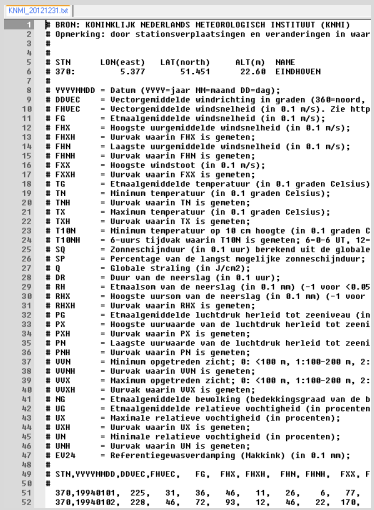


Rij en kolom van observatiepunt worden gebruikt om gemodelleerde waarden (van in dit geval grondwaterstanden) op te slaan in tijdreeksen.

Zorg ervoor dat er voor iedere datum/tijd numpy arrays zijn – op volgorde van datum/tijd - om deze te visualiseren in combinatie met tijdreeksen, en v.v. Het is waarschijnlijk dat een (for-)loop nodig zal zijn om de numpy arrays/tijdreeksen te plotten.

```
#Opzetten DataFrame met stijghoogtewaarnemingen op
verschillende locaties #
# Startdatum: Zoals "2012-01-13" (YYYY-MM-DD)
startdate =  re.findall('([1-9]+.*).npy',headFiles[0])[0]
# Einddatum: Zoals "2017-02-12" (YYYY-MM-DD)
enddate =    re.findall('([1-9]+.*).npy',headFiles[-1])[0]

## Weerdata - Hier: te Eindhoven, van 1 januari 1994 tot 31
december 2012
fpath_eindh = os.path.join(w_s, 'KNMI_20121231.txt')

# Kolomnamen
header = ['Date', 'RH', 'Ev24']
# Lees .csv-bestand in als DataFrame
ClimateSeries = pd.read_csv(fpath_eindh, skiprows = 50,
names = header,                    index_col = None, usecols
= [1,22,40],
                    na_values = [''], skipinitialspace=True)
```

De eerste 50 rijen worden overgeslagen, omdat deze verklarende tekst bevatten:



```
# BRON: KONINKLIJK NEDERLANDS METEOROLOGISCH INSTITUUT (KNMI)
# Opmerking: door stationsverplaatsingen en veranderingen in waar
#
#
# STN       LON(east)   LAT(north)     ALT(m)  NAME
# 370:          5.377       51.451      22.60  EINDHOVEN
#
# YYYYMMDD = datum (YYYY=jaar MM=maand DD=dag);
# DDVEC    = Vectorgemiddelde windrichting in graden (360=noord,
# FHVEC    = Vectorgemiddelde windsnelheid (in 0.1 m/s). Zie http
# FG       = Etmaalgemiddelde windsnelheid (in 0.1 m/s);
# FHX      = Hoogste uurgemiddelde windsnelheid (in 0.1 m/s);
# FHXH     = Uurvak waarin FHX is gemeten;
# FHN      = Laagste uurgemiddelde windsnelheid (in 0.1 m/s);
# FHNH     = Uurvak waarin FHN is gemeten;
# FXX      = Hoogste windstoot (in 0.1 m/s);
# FXXH     = Uurvak waarin FXX is gemeten;
# TG       = Etmaalgemiddelde temperatuur (in 0.1 graden Celsius)
# TN       = Minimum temperatuur (in 0.1 graden Celsius);
# TNH      = Uurvak waarin TN is gemeten;
# TX       = Maximum temperatuur (in 0.1 graden Celsius);
# TXH      = Uurvak waarin TX is gemeten;
# T10N     = Minimum temperatuur op 10 cm hoogte (in 0.1 graden C
# T10NH    = 6-uurs tijdvak waarin T10N is gemeten; 6-0-6 UT, 12-
# SQ       = Zonneschijnduur (in 0.1 uur) berekend uit de globale
# SP       = Percentage van de langst mogelijke zonneschijnduur;
# Q        = Globale straling (in J/cm2);
# DR       = Duur van de neerslag (in 0.1 uur);
# RH       = Etmaalsom van de neerslag (in 0.1 mm) (-1 voor <0.05
# RHX      = Hoogste uursom van de neerslag (in 0.1 mm) (-1 voor
# RHXH     = Uurvak waarin RHX is gemeten;
# PG       = Etmaalgemiddelde luchtdruk herleid tot zeeniveau (in
# PX       = Hoogste uuruaarde van de luchtdruk herleid tot zeeni
# PXH      = Uurvak waarin PX is gemeten;
# PN       = Laagste uuruaarde van de luchtdruk herleid tot zeeni
# PNH      = Uurvak waarin PN is gemeten;
# VVN      = Minimum opgetreden zicht; 0: <100 m, 1:100-200 m, 2:
# VVNH     = Uurvak waarin VVN is gemeten;
# VVX      = Maximum opgetreden zicht; 0: <100 m, 1:100-200 m, 2:
# VVXH     = Uurvak waarin VVX is gemeten;
# NG       = Etmaalgemiddelde bewolking (bedekkingsgraad van de b
# UG       = Etmaalgemiddelde relatieve vochtigheid (in procenten
# UX       = Maximale relatieve vochtigheid (in procenten);
# UXH      = Uurvak waarin UX is gemeten;
# UN       = Minimale relatieve vochtigheid (in procenten);
# UNH      = Uurvak waarin UN is gemeten;
# EV24     = Referentiegewasverdamping (Makkink) (in 0.1 mm);
#
# STN,YYYYMMDD,DDVEC,FHVEC,   FG,  FHX, FHXH,  FHN, FHNH,  FXX, F
#
  370,19940101,  225,   31,   36,   46,   11,   26,    6,   77,
  370,19940102,  228,   46,   72,   93,   12,   46,   22,  170,
```

De datums, neerslagdata en verdampingsdata staan in kolommen 2, 23 en 41, respectievelijk.

```
# Gebruik de kolom 'Date' als indexkolom voor de data
ClimateSeries.index = pd.to_datetime(ClimateSeries['Date'],
dayfirst = False,
                                format='%Y%m%d')
# Verwijder eerdere 'Date' kolom
ClimateSeries = ClimateSeries.drop('Date', 1)
# Behoud alleen de data tussen de startdatum en einddatum:
ClimateSeries = ClimateSeries.loc[(ClimateSeries.index >=
startdate) &
                                  (ClimateSeries.index <= enddate)]
```

De pandas-module is krachtig om berekeningen uit te voeren met dataframes en is hierdoor zeer geschikt voor de (tijdelijke) opslag van tijdreeksen.
In plaats van csv-bestanden kunnen o.a. excel-bestanden ingelezen worden via in dat geval pd.read_excel(). Zie de documentatie voor pandas voor de mogelijkheden betreffende dataframe invoer, bewerkingen en uitvoer.

Neerslag < 0 houdt hier in: minder dan

```
# Verwijder negatieve waarden
ClimateSeries['RH'][ClimateSeries['RH'] < 0] = 0
# Reken neerslag- en verdampingsdata om van 0.1 mm/dag
naar mm/dag
ClimateSeries['RH'] = ClimateSeries['RH'] / 10.
ClimateSeries['Ev24'] = -ClimateSeries['Ev24'] / 10.


# Opzetten DataFrame voor opslag van berekende waarden #
# Genereer een lijst met strings (tekst) voor datum-indexering
datums = map(lambda x: re.findall('([1-9]+.*).npy', x)[0],
headFiles)
# Opstellen lijst met "Datetime indices"; String naar datetime
formaat
date_index = map(lambda x: datetime.strptime(x, "%Y-%m-
%d"), datums)

# Kolomnamen voor Neerslag ('RH') en verdamping ('EV24')
columns = ['RH','Ev24']
# Kolomnamen voor opslag waarden gemodelleerde
stijghoogten
for iObs in obs_punt:
    columns.append('Stijghoogte observatiepunt ' + str(iObs))

# Creëer een lege Dataframe
df_output = pd.DataFrame(index = date_index, columns =
columns)



## Laad stijghoogten-arrays in: ##
heads = {}
# Voeg waardes toe van gemodelleerde stijghoogten voor
ieder observatiepunt
index_nr = 0  # Bijhouden indexgetal
for iHeadArray in headFiles:  # Lijst met 2D-arrays
(doorsneden)
    iDate = date_index[index_nr]  # Datum behorende bij
doorsnede
    index_nr += 1  # Bijhouden indexgetal

    # Neem Neerslag- en verdampingsdata over
    df_output.loc[iDate, 'RH'] = ClimateSeries.loc[iDate, 'RH']
    df_output.loc[iDate, 'Ev24'] = ClimateSeries.loc[iDate,
'Ev24']

    heads[iDate] = np.load(iHeadArray)  # Laad array in
    for iObs in obs_punt:  # Sla gemodelleerde waarden op in
Output Dataframe
        df_output.loc[iDate, 'Stijghoogte observatiepunt ' +
```

0.5 mm/tijdstap. Deze neerslag is niet meegenomen.

De pandas-module is gebruikt voor het wegschrijven van gemodelleerde stijghoogten per opgelegd observatiepunt. Ook de neerslag- en verdampingsdata zijn opgeslagen in de uitvoer dataframe, zodat datums en tijden overeen komen.

Dit kunnen (ook) waarnemingen zijn uit het veld of anderzijds.

```
str(iObs)] = (

copy.deepcopy(heads[iDate][obs_row[iObs],obs_col[iObs]]))

### Schrijf uitvoerdata weg
df_output.to_csv(os.path.join(w_s, 'df_output.csv'), sep='\t')
df_output.to_excel(os.path.join(w_s, 'df_output.xlsx'))

## Lees uitvoerdata in (mits er al een csv-bestand of excel-
bestand aanwezig is
## om in te lezen):
#df_output = pd.read_csv(os.path.join(w_s, 'df_output.csv'),
sep='\t')
## Opzetten DataFrame index##
#df_output.index = date_index

# Packages/modules die specifiek dienen voor het plotten van
de visualisaties
#from matplotlib import use

from matplotlib._png import read_png
from matplotlib.cbook import get_sample_data

# Logo KWR omzetten in grid:
kwr_logo = get_sample_data(os.path.join(w_s,
"KWR_BASIS_los.png"), asfileobj=False)
arr_kwr = read_png(kwr_logo)

os.chdir(w_s)  # Verander (zonodig) huidige werkmap
# Module ondersteunt het opstellen van de lay-out van de
visualisaties
import cross_sections as cs

# Locatie voor opslag van figuren
submap_fig = 'plots_vo_Stippelberg'
dstdirplots = os.path.join(w_s, submap_fig)
# Controleer of de map al bestaat
if not os.path.exists(dstdirplots):
    os.makedirs(dstdirplots)

## Isohypsen in de figuren
levels_hd = np.array(range(15,41))



index_nr = -1
# Indicatie begin en einddatum
startDay = df_output.index[0].day
startMonth = df_output.index[0].month
startYear = df_output.index[0].year
endDay = df_output.index[-1].day
```

Deepcopy zorgt voor een directe kopie van waarden zonder dat link met originele array behouden blijft.

Lees afbeeldingen in (hier: png's) en zet deze om in griddata om de afbeelding(en)/logo's te kunnen plotten in de figuren.

Contouren (1D-array) van hoogtelijnen die al dan niet toegevoegd kunnen worden aan de 2D-doorsnede.

Sla dag, maand en jaar op van het start- en eindpunt van de modellering

```
endMonth = df_output.index[-1].month
endYear = df_output.index[-1].year
# Aantal gemodelleerde perioden
nper = len(df_output.index) - 1
# Aantal gemodelleerde dagen
ndays = nper * 14  # tijdstap output is 14 dagen
plot_nr = 0
# Itereer over datums in outputtabel; genereer figuren per
datum
for iDate in df_output.index:

    # Volgende indexnummer (itereren over datums in
Dataframe)
    index_nr += 1
    plot_nr +=1

    # Bijhouden huidige datum
    iYear = iDate.year
    iMonth = iDate.month
    iDay = iDate.day

    # Initieel (mogelijk heterogeen) grid:
    head_arr = copy.deepcopy(heads[iDate]) # Eerder
aangemaakt stijghoogtegrid voor deze datum

    # Resolutie uiteindelijk homogeen grid (gebruikt voor
maken figuren)
    min_horX = 25.
    min_horY = 25.
    # XY-coordinaten
    xy = np.empty((nrow * ncol, 2), dtype = np.float)
    # Stijghoogtewaarden per cel in 1D-array
    head_values = np.empty((nrow * ncol, 1), dtype = np.float)

    # Stijghoogte dwarsdoorsneden
    cs_plot = cs.CrossSection()

    # extent: (xmin, xmax, ymin, ymax)
''' Het object cs_plot wordt gebruikt om de verticale
```

Houd plotnummer bij

(iDate: “28-4-1996”, “12-5-1996”, …) Vanaf deze regel (for-loop) worden per tijdstap binnen de dataframe de laatste berekeningen uitgevoerd en de plots gemaakt. N.B. Zorg ervoor dat er voor iedere datum/tijd numpy arrays zijn om te visualiseren in combinatie met tijdreeksen, en v.v.

Numpy-array met dimensie (nrow * ncol, 2) om voor iedere cel een X-en Y-waarde op te geven. Er hoeft slechts 1 stijghoogtewaarneming per cel(laag) te worden opgegeven.

cs.CrossSection() verwijst naar de ‘Class’ CrossSection binnen het script ‘cross_sections.py’. Deze klasse dient ter ondersteuning van het plotten van de figuren, zoals het verzorgen van de plotverhouding van de 2D-doorsnede.

Bepaling extent (ofwel omvang) van figuur: Er kan een verticale overdrijving toegepast worden als de lengte in de X-

overdrijving zodanig aan te passen dat de plotverhouding gelijk blijft (drie keer zo breed als hoog). ''' extent, ex_vert = cs_plot.set_extent(extent = (176162.5, 190562.5, 391512.5, 396312.5), ex_vert = None) # Toon extent van plot en uiteindelijke verticale overdrijving print extent, "Verticale overdrijving:", ex_vert	richting meer dan drie keer zo groot is als de lengte in de Y-richting. N.B. De Horizontale afstand moet minstens drie keer zo groot zijn als de verticale dimensie om een correcte plotfunctionaliteit te garanderen. Vergroot desnoods de afstand tussen xmin en xmax om incorrecte weergave te voorkomen (zoals de schaalverdeling in bovenaanzicht Figuur 2).

count = 0
# Bepaal gemodelleerde stijghoogten die behoren bij bepaalde XY-coordinaten 	
for i in xrange(nrow):
for j in xrange(ncol):
xy[count,0] = xmid[i,j]
xy[count,1] = ymid[i,j] * ex_vert
head_values[count,0] = head_arr[i,j]
count += 1
	Voor iedere cel: sla X- en Y-waarden op can het middelpunt in een numpy array - ((X1,Y1),(X2,Y2),…, (Xn,Yn)); sla ook de gemodelleerde parameterwaarde (hier: stijghoogtewaarden) op.
	
	In dit geval is het gemodelleerde grid homogeen (cellen hebben dezelfde dimensie in zowel de X- als Y-richting. Er mag echter een heterogeen grid aangeleverd worden. De X- en Y-coördinaten van iedere cel dienen opgegeven te worden in het array 'xy'.

# Tick-locatie X-as en Y-as samen met bijbehorende labelnamen x_tic, y_tic, x_tic_lab, y_tic_lab = cs_plot.xy_labels(nr_of_xlab = 6, nr_of_ylab = 3, x_null = 178000., y_null = 392000. * ex_vert, stopX = 188000., stopY = 396000. * ex_vert, xtic_left = 178000., xtic_right = 188000.) ''' Met nr_of_xlab: aantal labelpunten X-as, nr_of_ylab: aantal labelpunten Y-as, x_null: Locatie eerste X-label, y_null: Locatie eerste Y-label, stopX: Locatie laatste X-label, stopY: Locatie laatste Y-label, xtic_left: Getoonde waarde eerste X-label, xtic_right: Getoonde waarde laatste X-label '''	De labelnamen worden in geval van verticale overdrijving in de Y-richting automatisch behouden, mits de locatie van de eerste en laatste Y-ticks correct is opgegeven. De Y-waarden van de labels veranderen mee met de verticale overdrijving die is toegepast. Vermenigvuldig met verticale overdrijving (ex_vert) om de posities van de labels correct te bepalen.
# Homogeen XY-grid waarnaartoe het initiële (heterogene) grid geïnterpoleerd wordt xi, yi = cs_plot.set_xygrid(xmin = extent[0], xmax = extent[1], ymin = extent[2], ymax = extent[3], resolutionX =	Vul alle cellen binnen het extent met (bekende) bijbehorende X- en Y-coördinaten, respectievelijk.

```
min_horX, resolutionY = min_horY)

    # Homogeen Stijghoogtegrid waarnaartoe het initiële
(heterogene) grid geïnterpoleerd wordt
    corr_headgrid = cs_plot.set_corr_grid(name = 'Head', points
= xy, values = head_values,  xi = xi, yi = yi, extent = extent,
method='nearest')

    # Maak een grid aan welke als standaard dient om waarden
voor te maskeren
    if iDate == df_output.index[0]:
        layout_headgrid = cs_plot.set_corr_grid(name = 'Head',
points = xy, values = head_values,
                                        xi = xi, yi = yi, extent = extent,
method='nearest')

    # Bepaal plotlocatie van observatiepunten
    well_DimX, well_DimX_min, well_DimX_max = {}, {}, {}
    well_DimY, well_DimY_min, well_DimY_max = {}, {}, {}

    for iObs in obs_punt:
        well_DimX_min[iObs] = (xmid[obs_row[iObs][0],
obs_col[iObs][0]] -
                               0.5 * delr[obs_col[iObs][0]])
        well_DimX_max[iObs] = (xmid[obs_row[iObs][-1],
obs_col[iObs][-1]] +
                               0.5 * delr[obs_col[iObs][-1]])
        well_DimX[iObs]     = xmid[obs_row[iObs][0],
obs_col[iObs][0]]
        well_DimY_min[iObs] = (ymid[obs_row[iObs][-1],
obs_col[iObs][-1]] -
                               0.5 * delc[obs_row[iObs][-1]]) * ex_vert
        well_DimY_max[iObs] = (ymid[obs_row[iObs][0],
obs_col[iObs][0]] +
                               0.5 * delc[obs_row[iObs][0]]) * ex_vert
        well_DimY[iObs]     = ymid[obs_row[iObs][0],
obs_col[iObs][0]] * ex_vert

    # Vergrotingsfactor plot (standaard: 1.0)
    size_fig = 1.0

    # Locaties subplots binnen de figuren
    # Voor achtergrondinformatie, zie: help(gridspec.GridSpec)
    # Tijdreeksen
    gs1 = gridspec.GridSpec(nrows = 2, ncols =  1, top = 0.88,
bottom = 0.52,
                            left = 0.17, right = 0.87, hspace=0.30,
height_ratios=[1,1])
    # Logos
    gs2 = gridspec.GridSpec(nrows = 1, ncols =  4, top = 0.12,
bottom = -0.02,
                            left = 0.02, right = 0.92, height_ratios=[1],
```

Vul alle cellen binnen het extent met (bekende) bijbehorende stijghoogtewaarden

Deze regels beschrijven de middelpunten van de observatiepunten en zowel de boven- en ondergrens als de linker- en rechtergrens ervan. Er wordt eventueel gecorrigeerd voor de toegepaste verticale overdrijving (ex_vert).

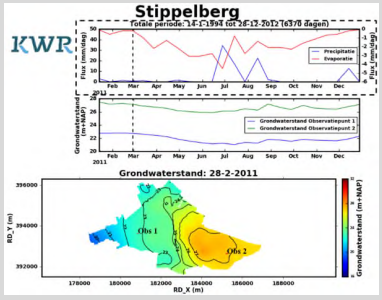
Via de gridspec module kan de precieze locatie van de subplots bepaald worden binnen de figuur, waarbij zowel de horizontale als verticale as tussen de nul en één zijn gedimensioneerd. Het wijzigen van deze waarden kan invloed hebben op de correcte weergave van de subplots en onder andere (label/titel)-teksten.

```
width_ratios=[1,1,1,1])
    # Cross-sectie
    gs3 = gridspec.GridSpec(nrows = 1, ncols = 1, top = 0.47,
bottom = 0.13,
                            left = 0.11, right = 0.93)

    # use('pdf') - Om opslaan als '.pdf'-bestand mee te
faciliteren

    ## Maak figuur aan ##
    fig_ch = plt.figure(figsize=(10 * size_fig, 8 * size_fig), dpi =
300)
    plt.rc('ps', usedistiller='xpdf')


    plt.rc('font', weight='bold')
    plt.rc('font', family = font_type)
    plt.rc('axes', linewidth= 2 * size_fig)
    plt.rc('xtick', labelsize = 10 * size_fig)
    plt.rc('ytick', labelsize = 10 * size_fig)

# Algemene titel boven grafiek
    fig_ch.suptitle("Stippelberg", fontsize = 30 * size_fig, family
= font_type, fontweight='bold', color = color_text)

    # tijdreeks 1 (subplot)
    ax = plt.subplot(gs1[0, :])
```

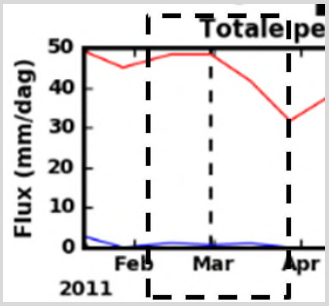
De huidige format resulteert in figuren zoals Figuur 2 (bij een keuze voor 'barplot'. Ook de mogelijkheid voor een 'line-plot' is uitgewerkt.

Maak teksten dikgedrukt

Bepaal asdikten
Grootte van x-ticks
Grootte van y-ticks




```
# Verticale lijn die datum/tijd van geplotte cross-sectie
bijhoudt
ax.axvline(pd.to_datetime(df_output.index[index_nr].strftime('
%Y-%m-%d')), color='k',
               linestyle='--', lw= 1.5 * size_fig)
    plt.tick_params(which='both', width=1 * size_fig)
```



Toon op de Y-as neerslagwaarden van hetzelfde jaar (bv 2016) als waarvan de geplotte bovenaanzicht is (index.year == iYear).

```
    # Plot tijdreeks voor precipitatie op linker Y-as
    line1 = ax.plot_date(x =
ClimateSeries.index[ClimateSeries.index.year == iYear],
                         y = ClimateSeries
['RH'][ ClimateSeries.index.year == iYear], fmt='-',
                         color = 'blue', label= 'Precipitatie',
                         tz=None, xdate=True, ydate=False, linewidth=
```

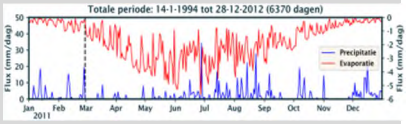
```
1. * size_fig)
    # Titel (linker) Y-as
    ax.set_ylabel("Flux (mm/dag)", fontsize = 11 * size_fig,
fontweight='bold')
```

Voeg label linker Y-as toe.

```
    # Plot een tweede Y-as (= rechter Y-as)
    ax1b = ax.twinx()
```

De methode twinx() maakt het mogelijk om een extra y-as te gebruiken. Evaporatiedata zijn negatief genomen.

‘line-plot’




```
## Voor een ‘line-plot’ gebruik onderstaande code: ##

    # Plot tijdreeks voor verdamping (= negatief) op rechter Y-
as

    line2 = ax1b.plot_date(x =
ClimateSeries.index[ClimateSeries.index.year == iYear],
                           y = ClimateSeries
['Ev24'][ ClimateSeries.index.year == iYear],
                           fmt='-', color = 'red', label= 'Evaporatie',
tz=None, xdate=True,
                           ydate=False, linewidth= 1. * size_fig)
#     xmin, xmax = ax.get_xlim()
    ax.set_xlim(ClimateSeries.index[ClimateSeries.index.year
== iYear][0],
                ClimateSeries.index[ClimateSeries.index.year ==
iYear][-1])
```

Definieer xmin en xmax van de subplot

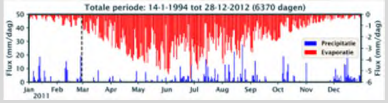
```
    # Legenda op basis van 1 Y-as
    # plt.legend(loc = 'best', prop={'size':7 * size_fig, 'weight':
'bold'},
    # ncol = 1)
```

Voeg deze regel toe als een simpele legenda binnen de subplot gewenst is.

```
    # Voeg lijnen samen om een gezamenlijke legenda te
maken
    lines = line1 + line2
    labels = [l.get_label() for l in lines]

## Einde code ‘line-plot’ ##
```

Hiermee is het mogelijk om legenda’s voor verschillende assen samen te voegen. De methode is iets anders voor een ‘barplot’.

‘Barplot’



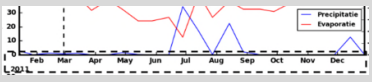

```
## Voor een ‘barplot’ vervang bovenstaande code door het
volgende: ##

    # Plot tijdreeks voor precipitatie op linker Y-as
    bar1 = ax.bar(ClimateSeries.index[ClimateSeries.index.year
== iYear],
                  ClimateSeries['RH'][ClimateSeries.index.year ==
iYear],
                  width = 0.8 * size_fig, color = 'b')
    ax.xaxis_date()
    # Titel (linker) Y-as
    ax.set_ylabel("Flux (mm/dag)", fontsize = 11 * size_fig,
```

```
fontweight='bold')
    # Plot een tweede Y-as (= rechter Y-as)


    ax1b = ax.twinx()
    # Plot tijdreeks voor verdamping (= negatief) op rechter Y-
as
    bar2 =
ax1b.bar(ClimateSeries.index[ClimateSeries.index.year ==
iYear],
                ClimateSeries['Ev24'][ClimateSeries.index.year ==
iYear],
                width = 0.8 * size_fig, color = 'r')
    ax1b.xaxis_date()
    ax.set_xlim(ClimateSeries.index[ClimateSeries.index.year
== iYear][0],
                ClimateSeries.index[ClimateSeries.index.year ==
iYear][-1])
    # Legenda op basis van 1 Y-as
    # plt.legend(loc = 'best', prop={'size':7 * size_fig, 'weight':
'bold'},
    # ncol = 1)

    # Voeg lijnen samen om een gezamenlijke legenda te
maken
    lines = [bar1, bar2]
    labels =  ['Precipitatie', 'Evaporatie'] #[l.get_label() for l in
lines]

## Einde code  'bar-plot' ##

    leg = plt.legend(lines, labels, loc = 'center right',
                prop = {'size':9 * size_fig, 'weight': 'bold', 'family':
font_type}, ncol = 1)
    # Verander kleur in tekstvak legenda
    leg.get_frame().set_facecolor(color_face)

    # Titel subplot
    ax.set_title("Totale periode: {0}-{1}-{2} tot {3}-{4}-{5} ({6}
dagen)".format(startDay,
                startMonth, startYear, endDay, endMonth, endYear,
ndays), fontweight='bold',
                fontsize = 12 * size_fig)

    ax1b.set_ylabel("Flux (mm/dag)", fontsize = 11 * size_fig,
fontweight='bold')

    # Plaatsing (en format) van X-as labels op basis van datums
    ax.xaxis.set_major_formatter(DateFormatter('%b'))  #
Maandnamen
    ax.xaxis.set_minor_formatter(DateFormatter('\n%Y'))  #
Jaarnummers
```

De uiteindelijke weergave is zoals:

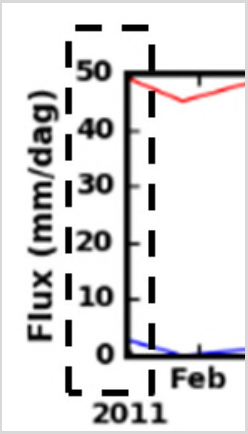
Voeg label rechter Y-as toe.

Een voorbeeldweergave van de X-as:

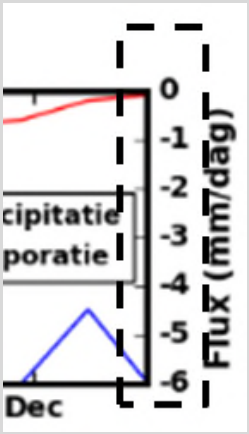


```
    #  Plaatsing maandnamen met een frequentie van 1 maand
beginnend bij maand 1
    ax.xaxis.set_major_locator(MonthLocator(bymonthday=1,
interval=1))
    # Plaatsing jaartal met een frequentie van 1 jaar op dag 15
in maand 1
    ax.xaxis.set_minor_locator(YearLocator(1, month = 1, day =
15))
```

```
    # Y-axis labels en ticks
    ax.set_ylim([0.,50.])  # Verander de (bestaande) Y-as limiet

    ax.set_yticks([0.,10., 20., 30., 40., 50.])
    # Labels met integerwaarden
    ax.set_yticklabels(['0','10','20','30','40','50'])
```

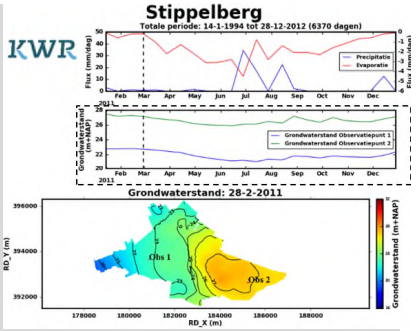



```
    # Y-axis labels en ticks
    ax1b.set_ylim([-6,0])  # Verander de (bestaande) Y-as limiet

    ax1b.set_yticks([-6.,-5.,-4.,-3.,-2.,-1.,0.])
    # Labels met integerwaarden
    ax1b.set_yticklabels(['-6','-5','-4','-3','-2','-1','0'])
```




```
    # Kleuren subplot 1 (precipitatie en evaporatie)
    ax.xaxis.label.set_color(color_text)  # X-as
    ax.yaxis.label.set_color(color_text)  # Y-as
    ax1b.yaxis.label.set_color(color_text)  # Y-as
    ax.title.set_color(color_text)
    ax1b.spines['bottom'].set_color(color_text)
    ax1b.spines['left'].set_color(color_text)
    ax1b.spines['right'].set_color(color_text)
    ax1b.spines['top'].set_color(color_text)
    ax.tick_params(axis='x', which = 'both', colors= color_text)
    ax.tick_params(axis='y', colors= color_text)
```

```
    ax1b.tick_params(axis='y', colors= color_text)

    # tijdreeks 2 (subplot)
    ax2 = plt.subplot(gs1[1, :])
    plt.tick_params(which='both', width= 1 * size_fig)
```



De lay-outs zijn conform de eerste tijdreeks; zie de eerdere voorbeelden

```
# Verticale lijn die datum/tijd van geplotte cross-sectie
bijhoudt
ax2.axvline(pd.to_datetime(df_output.index[index_nr].strftime
('%Y-%m-%d')),
            color='k', linestyle='--', lw= 1.5 * size_fig)
    # Plot tijdreeks met uit dwarsdoorsnede verkregen
stijghoogten voor ieder observatiepunt
    for iObs in obs_punt:
        ax2.plot_date(
            x=df_output.index[df_output.index.year == iYear],
            y=df_output['Stijghoogte observatiepunt ' +
str(iObs)][df_output.index.year == iYear],
            fmt='-', label= 'Grondwaterstand ' + obs_punt[iObs],
            tz=None, xdate=True, ydate=False, linewidth= 1. *
size_fig)
 #    xmin, xmax = ax2.get_xlim()
    ax.set_xlim(ClimateSeries.index[ClimateSeries.index.year
== iYear][0],
            ClimateSeries.index[ClimateSeries.index.year ==
iYear][-1])
```

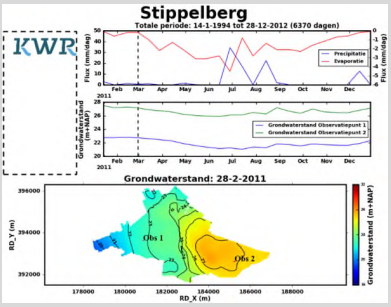
Definieer xmin en xmax van de subplot

```
    # Plot legenda
    leg2 = plt.legend(loc = 'center right',
            prop = {'size':9 * size_fig, 'weight': 'bold', 'family':
font_type},
            ncol = 1)
    # Verander kleur in tekstvak legenda
    leg2.get_frame().set_facecolor(color_face)

    # Plaatsing (en format) van X-as labels op basis van datums
    ax2.xaxis.set_major_formatter(DateFormatter('%b'))  #
Maandnamen
    ax2.xaxis.set_minor_formatter(DateFormatter('\n%Y'))  #
Jaartallen
    #  Plaatsing maandnamen met een frequentie van 1 maand
beginnend bij maand 1
    ax2.xaxis.set_major_locator(MonthLocator(bymonthday=1,
interval=1))
    # Plaatsing jaartal met een frequentie van 1 jaar op dag 15
```

De combinatie van karakters ‘\n’ beëindigt de huidige regel.

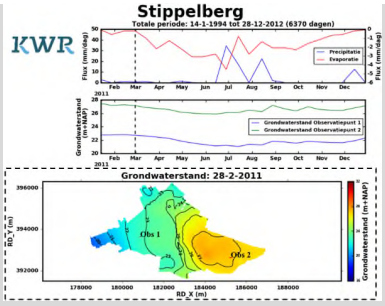
```
in maand 1
    ax2.xaxis.set_minor_locator(YearLocator(1, month = 1, day
= 15))

    # Y-axis labels en ticks
    ax2.set_ylim([20.,28.])  # Verander de (bestaande) Y-as
limiet

    ax2.set_yticks([20.,22., 24., 26., 28.])
    # Labels met integerwaarden
    ax2.set_yticklabels(['20','22','24','26','28'])
    # Geef een grenswaarde aan als horizontale lijn in de
tijdreeks
#     ax2.hlines(y= 0.018, xmin=xmin,xmax=xmax-1,color="k",
linestyle='-.', linewidth=3.0 * size_fig)

    # Label Y-as
    ax2.set_ylabel("Grondwaterstand\n    (m+NAP)", fontsize =
11 * size_fig, fontweight='bold')


    # Kleuren subplot 2 (grondwaterstanden)
    ax2.xaxis.label.set_color(color_text)  # X-as
    ax2.yaxis.label.set_color(color_text)  # Y-as
    ax2.spines['bottom'].set_color(color_text)
    ax2.spines['left'].set_color(color_text)
    ax2.spines['right'].set_color(color_text)
    ax2.spines['top'].set_color(color_text)
    ax2.tick_params(axis='x', which = 'both', colors=color_text)
    ax2.tick_params(axis='y', colors=color_text)

# Voeg logo toe: KWR of anderzijds
    ax3 = plt.subplot(gs2[0, 0])
    ax3.imshow(arr_kwr, extent=[0,610,0,248])
    ax3.axis('off')
    # Plaats een tweede logo zoals in vb.
    # ax4 = plt.subplot(gs2[1, 0])


    # Subplot dwarsdoorsnede
    ax5 = plt.subplot(gs3[0, 0])
```

Bij meer dan 2 logo's kan het noodzakelijk zijn om de indeling van 'gs2' aan te passen (meerdere kleinere logo's).



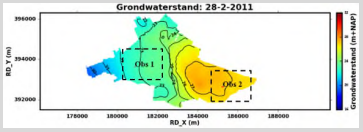
Subplot boven- of zijaanzicht

```
    # Locaties observatiepunten in dwarsdoorsnede met
tekstlabel
```

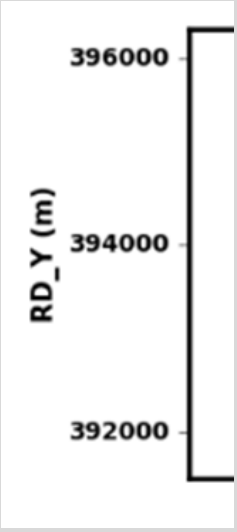



```
    for iObs in obs_punt:
        # Markeer het observatiepunt in de figuur
        ax5.scatter(well_DimX[iObs],well_DimY[iObs],
marker='o', color = color_button,
                    linewidth = 1.5 * size_fig)
        plt.text(well_DimX[iObs], well_DimY[iObs], 'Obs ' +
obs_punt[iObs][-1],
                 family= font_type, fontsize = 15 * size_fig,
                 color = color_text)

    # Voeg ticks en labelnamen toe aan X-as en Y-as
    ax5.set_xticks( x_tic )
#    ax5.set_xticklabels(x_tic_lab) # Kommagetallen voor
labels

    # Integer getallen voor labels
    ax5.set_xticklabels(map(lambda x:
str(int(float(x))),x_tic_lab))
    ax5.set_yticks( y_tic )
#    ax5.set_yticklabels(y_tic_lab) # Kommagetallen voor labels
    # Integer getallen voor labels
    ax5.set_yticklabels(map(lambda x:
str(int(float(x))),y_tic_lab))
```

De methode ‘scatter’ geeft het observatiepunt weer en plt.text is gebruikt voor annotatie van de punten.



‘Map’ functie: Omzetten van [‘0.0’, ’1.0’, …] naar [‘0’, ’1’, …] voor weergave van coördinaten als gehele getallen.




```
    # Maskeer NaN-waarden op basis van eerste headgrid
(gedefinieerd als layout_headgrid)
    corr_headgrid = np.ma.masked_where(layout_headgrid <=
0., corr_headgrid)
```

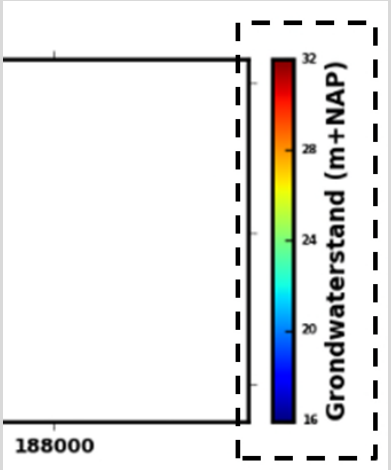
```
    im5 = ax5.imshow(corr_headgrid,  # Dwarsdoorsnede /
bovenaanzicht (array)

                     vmin = 16., vmax = 32., cmap = 'jet',
                     interpolation='nearest', extent= extent)
    ''' vmin: Minimumwaarde die getoond wordt,
        vmax: Maximumwaarde die getoond wordt,
        extent: Domein van geplotte array (dient overeen te
komen met inputarray
        (corr_headgrid), opgegeven als eerste parameter binnen
imshow). '''

    ## Optioneel: Contourlijnen (levels = np.array(0.,1.,2.,...))
    ## Negatieve waarden getoond als onderbroken lijn '- - -'
    # matplotlib.rcParams['contour.negative_linestyle'] =
'dashed'
    # Input array inverteren met np.flipud(inputarray)
    CS5 = ax5.contour(np.flipud(corr_headgrid), levels =
levels_hd, extent = extent,
                      colors = color_text, zorder=9)

    # Toon contour labelnamen
    plt.clabel(CS5, inline=1, fontsize=8 * size_fig, fmt='%i',
zorder=10, color = color_text)

    # Locatie en grootte kleurenbalk
    divider = make_axes_locatable(ax5)
    pad_fraction = 2.0
    aspect = 30.
    width = axes_size.AxesY(ax5, aspect=1/aspect)
    pad = axes_size.Fraction(pad_fraction, width)
    cax5 = divider.append_axes("right", size="2%", pad= pad)

    # Plaats kleurenbalk als legenda
    cbar5 = plt.colorbar(im5, cax = cax5, ticks= [16., 20., 24.,
28., 32.])
    cbar5.set_label('Grondwaterstand (m+NAP)', fontsize= 12 *
size_fig, fontweight='bold', labelpad = 4 * size_fig,
color = color_text)

    # Plot ticks
    cbar5.ax.tick_params(labelsize = 12 * size_fig)



    # Tick-opties aanpassen
    ax5.get_yaxis().set_tick_params(direction='out')  # Ticks
```

Eerste term: verwijs naar het te plotten 2D-numpy array (hier: corr_headgrid')

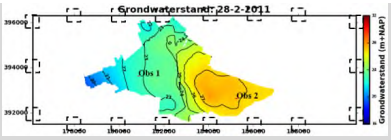
Cmap = 'jet' → Standaard 'Blauw geel rood' kleurenpatroon

De waarden gedefinieerd met 'levels_hd' worden weergegeven bij de contourlijnen.

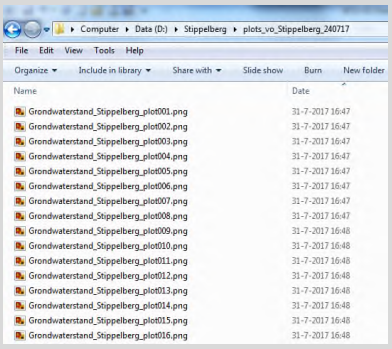
fmt (format) is integer ('%i')



```
naar buiten richten
    ax5.get_xaxis().set_tick_params(direction='out')  # Ticks
naar buiten richten
    plt.tick_params(which='both', width=1 * size_fig, labelsize
= 10 * size_fig, color = color_text)

    # Voeg titel toe aan subplot
    ax5.set_title('Grondwaterstand: {0}-{1}-
{2}'.format(iDay,iMonth,iYear), fontweight='bold',
                fontsize = 16 * size_fig)

    # Label Y-as
    ax5.set_ylabel('RD_Y (m)', fontweight='bold', fontsize = 11 *
size_fig, labelpad = 3 * size_fig)  # Show y-axis label
    # Label X-as
    ax5.set_xlabel('RD_X (m)', fontweight='bold', fontsize = 11
* size_fig, labelpad = 2 * size_fig)  # Show x-axis label

    # Kleuren in dwarsdoorsnede
    ax5.title.set_color(color_text)
    ax5.xaxis.label.set_color(color_text)  # X-as
    ax5.yaxis.label.set_color(color_text)  # Y-as
    ax5.spines['bottom'].set_color(color_text)
    ax5.spines['left'].set_color(color_text)
    ax5.spines['right'].set_color(color_text)
    ax5.spines['top'].set_color(color_text)
    ax5.tick_params(axis='x', which = 'both', colors=color_text)
    ax5.tick_params(axis='y', colors=color_text)


# Sla plot op in de map dstdirplots, bijvoorbeeld als .png-
bestand
    plt.savefig(os.path.join(dstdirplots,
'Grondwaterstand_Stippelberg_plot%(plotnr)03d' % {'plotnr':
plot_nr} + '.png') # Verwijder plot uit geheugen
    plt.close('all')
```



Houdt het plotnummer (plotnr) bij per figuur: bij meer dan 1000 figuren kan het nodig zijn om als formaat ‘04d’ aan te houden (4 ‘digits’ ofwel 4 getallen).



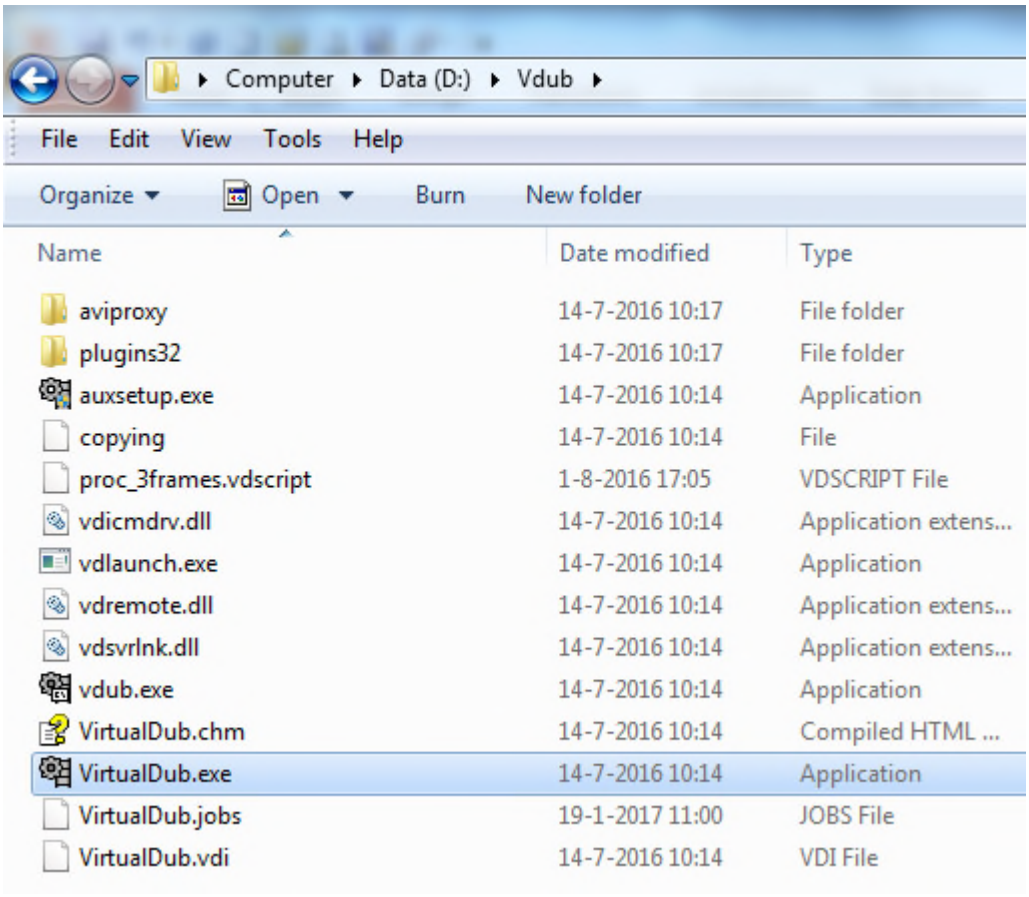
3.3 Figuren omzetten in een visualisatie

Om video's te maken is het gratis verkrijgbare programma VirtualDub gebruikt. Het programma voldoet (hier) om figuren in een compacte video om te zetten. Hieronder is een korte handleiding toegevoegd om met dit programma van de reeds opgeslagen figuren een video te maken. Er zijn er meerdere programma's die vergelijkbare functionaliteiten hebben.

3.3.1 Maken van visualisaties met VirtualDub

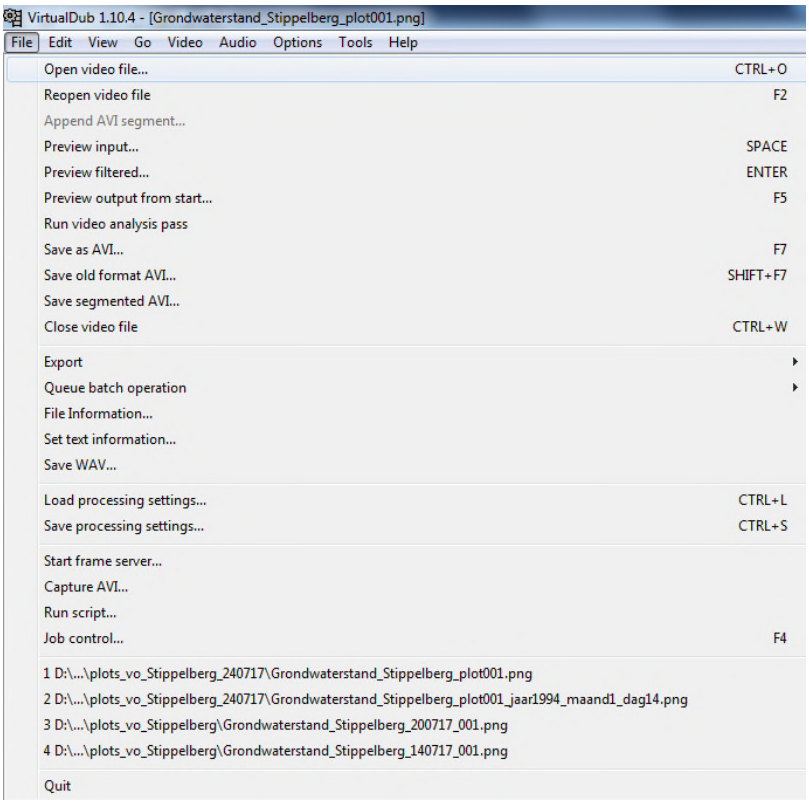
Met behulp van de volgende stappen worden de visualisaties via VirtualDub gemaakt:

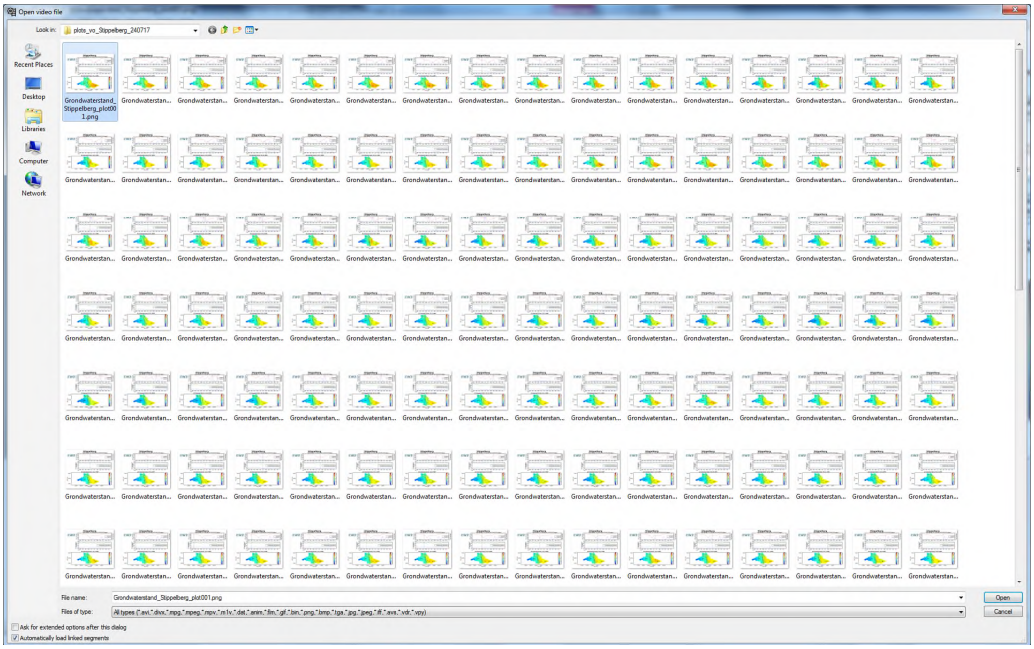
1. Open VirtualDub.exe



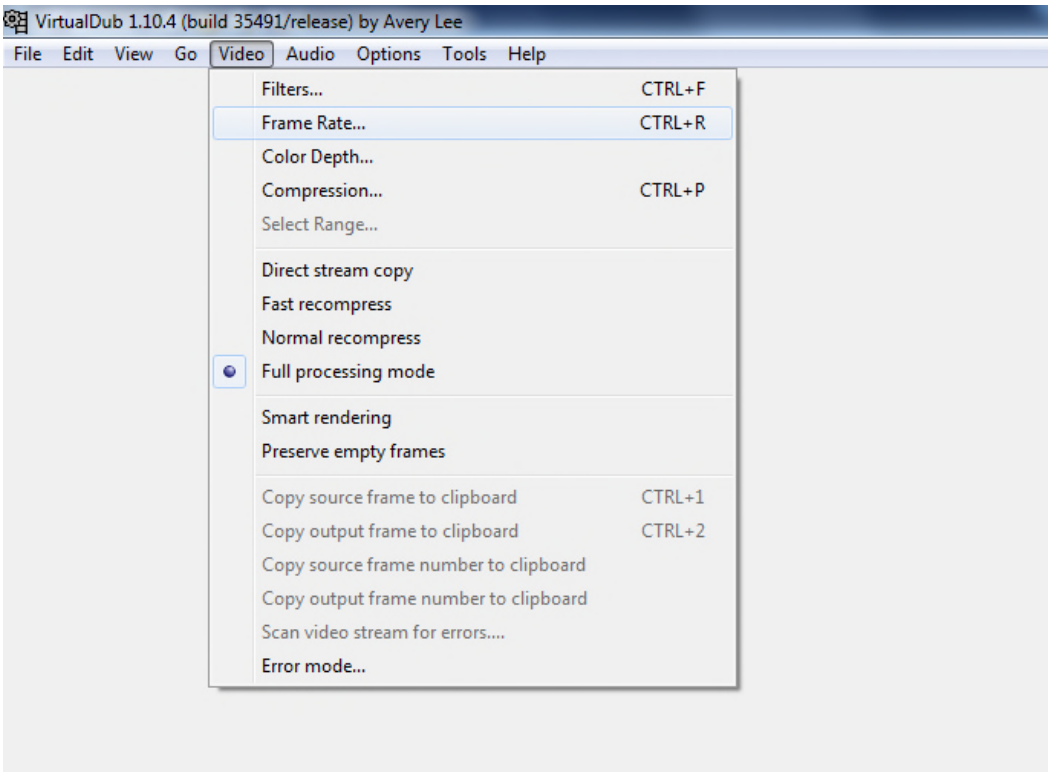
2. Laad de afbeeldingen in: ga naar de locatie waar de figuren zijn opgeslagen en selecteer de figuur eindigend op '001'. Ga naar 'File → Open video file'. In dit geval: open het bestand 'Grondwaterstand_Stippelberg_plot001.png'.

   N.B. Mochten de figuren niet vooraf genummerd zijn dan kan dit alsnog gedaan worden. Dit kan ook buiten Python met een programma als 'Total Commander' via de 'Multi-rename tool'. Echter, binnen dit VO is als voorbeeld de nummering met Python doorgevoerd en beschreven.

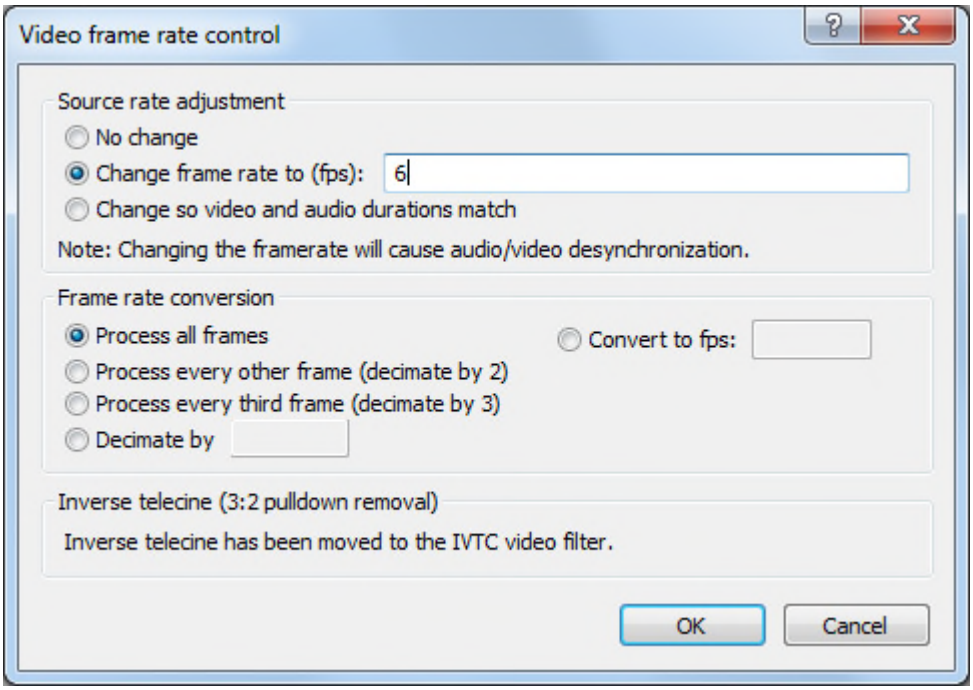




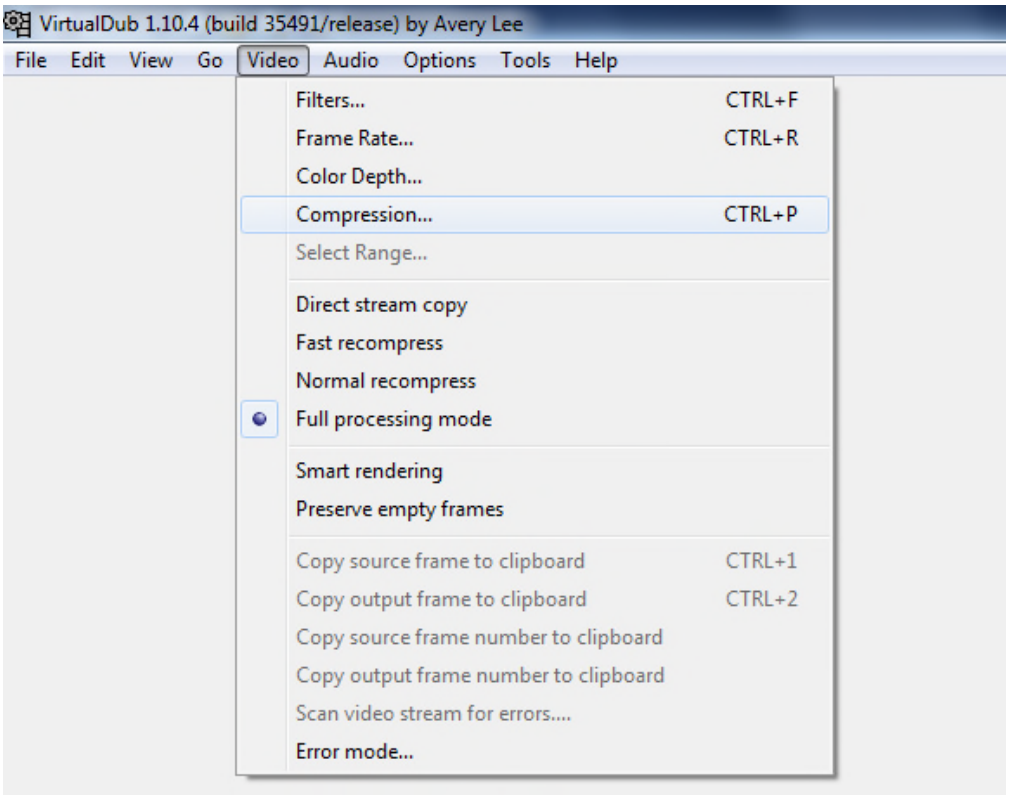
3. Ga naar ‘Video → Frame Rate’ om onder andere het aantal beelden per seconde in te stellen



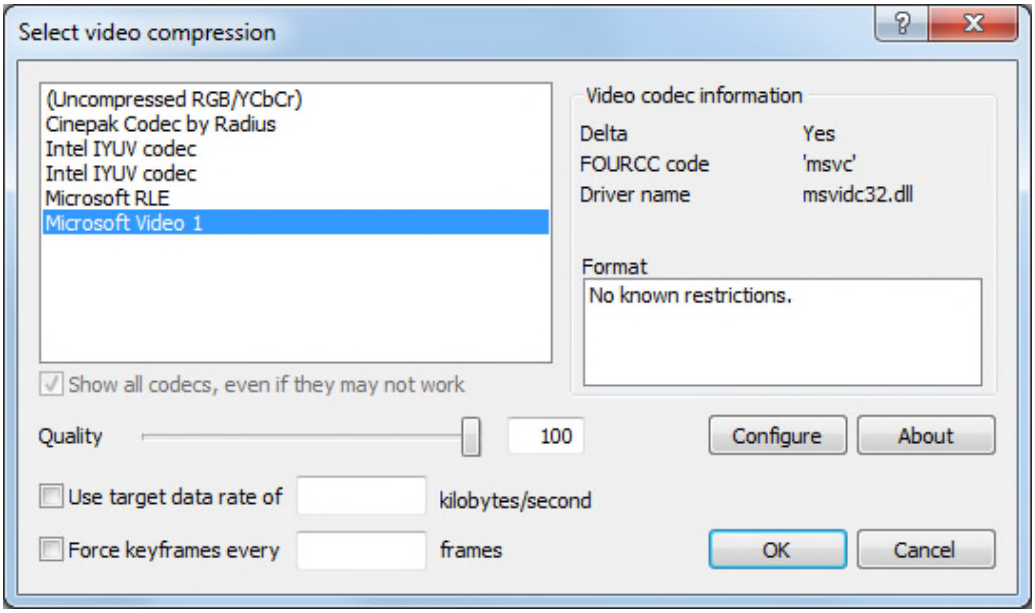
4. Bepaal het aantal beelden per seconde (fps): selecteer 'Change frame rate to (fps)' en vul een waarde in. Voor Stippelberg is gekozen voor zes beelden per seconden. Er kan eventueel gekozen worden om slechts een deel van de figuren te tonen. Klik op 'OK' om het venster te sluiten.



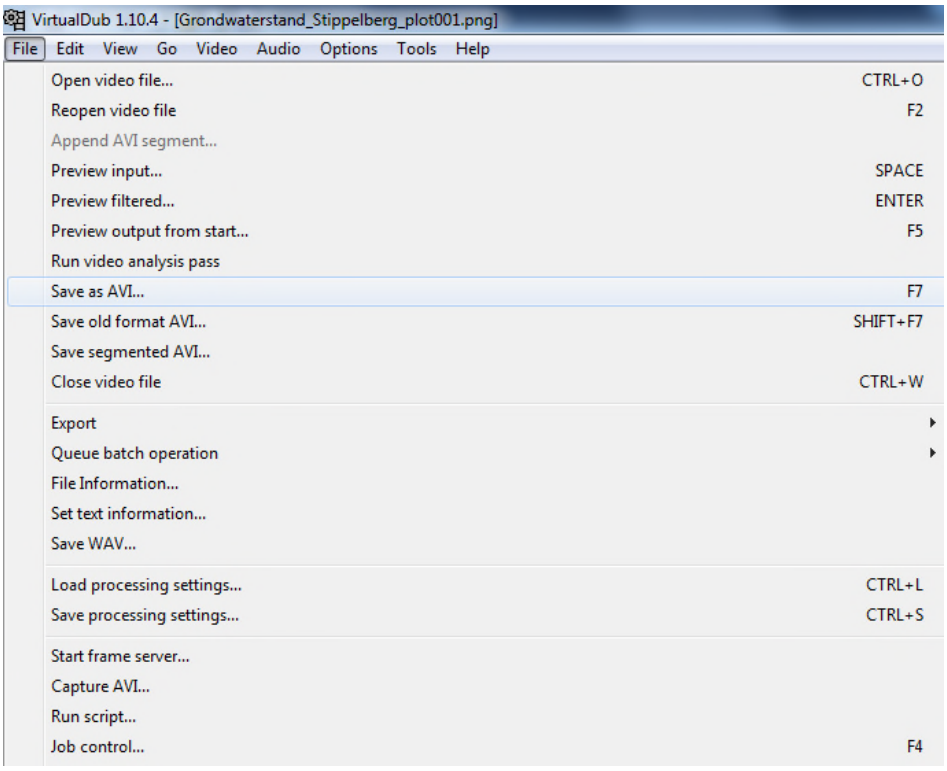
5. Om de video's op een compacte manier op te slaan kan gebruik worden gemaakt van compressie. Ga naar 'Video → Compression'.



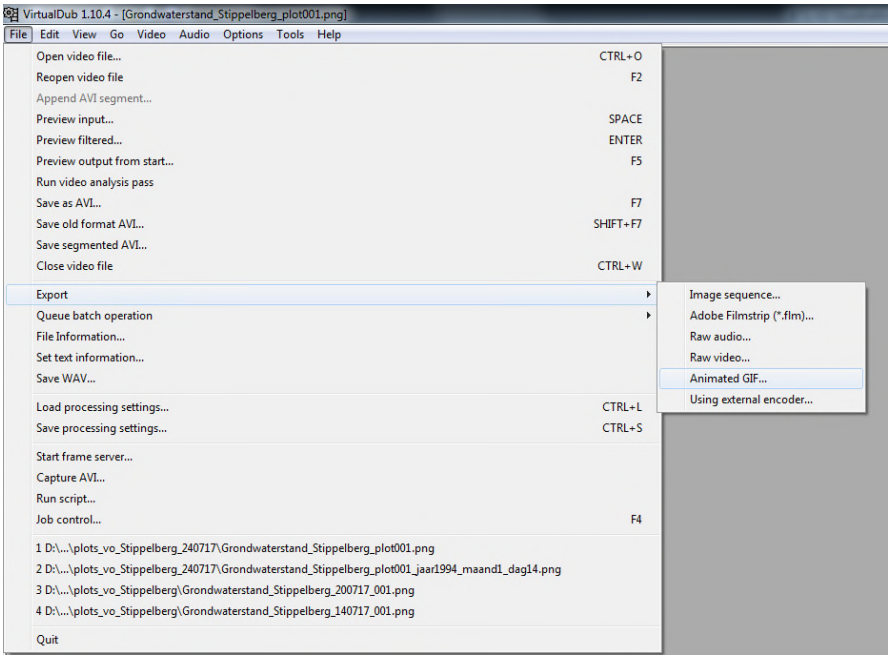
6. Selecteer ‘Microsoft Video 1’ als compressietype en pas zo nodig de kwaliteit van de video aan. Een ‘Quality’ van 100 behoudt de beeldkwaliteit, maar zorgt voor een minimale beperking van de bestandsgrootte. Het omgekeerde geldt voor een Quality van nul. Klik op ‘OK’ om het venster te sluiten.



7. Het is op verschillende manieren mogelijk om video’s te maken. Een AVI-formaat video is aan te maken via ‘File → Save as AVI’.



Om het bestandformaat nog verder te beperken kan de animatie als GIF-bestand worden opgeslagen (maximaal 256 kleuren: 8 bits). Selecteer 'File → Export → Animated GIF' en klik vervolgens op OK om de export te starten (verander eventueel de bestandslocatie.



De link naar het video-bestand is toegevoegd aan dit rapport: Animatie van grondwaterstanden Stippelberg (BTO-net).

4 Referenties

- van Loon, A.H., Lapperre, R., Mensink, J., Paalman, M.A.A. – Voorraadvorming van water in de Stippelberg (Noord-Brabant) – H2O-Online (2014) 11 juni)