Analytische oefeningen Voortoets

@Theo Olsthoorn (12-03-2024 .. 11-11-2024)

Intro, verantwoording

Na de vorige vergadering van 12 maart 2024 heb ik een aantal aspecten geanalyseerd die van belang zouden kunnen zijn voor een relatief eenvoudige, analytische analyse van de impact van een ingreep op een grondwatersysteem. De analyse heb ik uitgevoerd en gedocumenteerd in voorliggend Jupyter (Python) notebook, dat uitleg en code bevat om het uitgelegde te kwantificeren en te laten zien.

Voor een aantal complexere concepten zoals vertraagde nalevering, debietverloop van een bemaling met constante verlaging, hoe lang het duurt voor de verlaging stationair wordt e.d. zijn vereenvoudigingen voorgesteld die in de praktijk goed zullen werken.

Voor een zinvolle analyse van een fysische ingreep in het grondwatersysteem, zoals een nieuwe puntonttrekking, bemaling of verandering van de loop of het niveau van oppervlaktewater, is een beeld nodig van de opbouw van de ondergrond en van de drainage in het gebied (de randvoorwaarden). Bovendien is men geïnteresseerd in de overlap van de impact met bijzondere beschermingsgebieden. Deze drie vormen van informatie zijn gebonden aan beschikbare kaarten zoals die van de habitatgebieden, het oppervlaktewater, en bodemlagen. De laatste twee zijn aanwezig in de kaarten waarop het grondwatermodel van Vlaanderen is gebaseerd. Deze kaarten bevatten ook de benodigde informatie over de waarden van de bodemconstanten, zoals doorlaatvermogen, weerstand tussen lagen en bergingscoëfficiënten. Mogelijk kunnen op basis van de beschikbare laagverbreidingskaarten ook complexere situaties worden gesignaleerd, zoals breuken en andere scherpere overgangen tussen gebieden, die niet met eenvoudige formules kunnen worden geanalyseerd of waarvoor een aangepaste berekening kan worden voorgesteld of voorgeschreven. Het uit kaarten halen van de randvoorwaarden voor een berekening is vermoedelijk het meest complex of valt in de praktijk het meest op af te dingen. Uiteraard kunnen randvoorwaarden worden voorgeschreven zoals een invloedsradius of duur van de onttrekking waarmee moet worden gerekend, zoals dat nu reeds in de Voortoets het geval is.

Voor de benodigde onderliggende gegevens moet er i de Voortoets toegreep zijn tot het Vlaamse grondwatermodel, of althans de kaarten waar dit op gebaseerd is, zodat deze kaarten als een ruimtelijke database kunnen worden beschouwd en bevrraagd op bodemconstanten die voor een gegeven locatie moeten worden gebruikt.

Door op een dergelijke manier de voor elke vraag benodigde informatie op te vragen, kan

de onderliggende machinerie van de voortoets steeds gemakkelijk worden aangepast en verbeterd naar nieuwe inzichten.

Het resultaat van de Voortoets zol op deze wijze ook steeds in lijn zijn met de eventueel naderhand uit te voeren bredere analyse, waar dan zonodig een ruimtelijk grondwatermodel aan te pas komt, dat immers op dezelfde gegevens is gebaseerd. Een kernpunt zou dus moeten zijn dat de gegevens die gebruikt worden in de voortoets dezelfde zijn als die in het ruimtelijke model van Vlaanderen, waarbij de voortoetsberekeningen zich zullen baseren op de ondergrond gegevens ter plaatse van de ingreep en het ruimtelijk model met de ruimtelijke variatie rekening houdt.

De info-vraag van de Voortoets zal altijd zeer beperkt zijn, zodat de ICT-belasting navenant laag blijft en de informatie dus real-time over het internet (via een URL) moet kunnen worden opgevraagd en worden opgezocht op de ruimtelijke kaarten.

Imports voor de benodigde functionaliteit

In [1]:
```
import matplotlib.pyplot as plt
from matplotlib import patches
import numpy as np
from scipy.special import k0 as K0, j0 as J0, y0 as Y0, j1 as J1, y1 as Y1,
from scipy.integrate import quad
from scipy.signal import lfilter, filtfilt
import pandas as pd
from itertools import cycle

import ttim
```

Basisfuncties die verderop worden gebruikt:

In [132…
```
def newfig(title, xlabel, ylabel, xlim=None, ylim=None, xscale=None, yscale
    """Set up a new figure with a single axes and return the axes."""
    fig, ax = plt.subplots(1, figsize=figsize)
    ax.set_title(title)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    if xlim: ax.set_xlim(xlim)
    if ylim: ax.set_ylim(ylim)
    if xscale is not None: ax.set_xscale(xscale)
    if yscale is not None: ax.set_yscale(yscale)
    ax.grid(True)
    return ax


def Wh(u, rho=0):
    """Return Hantush's well function values.
```

```
    Parameters
    ----------
    u = r^s / (4 kD t)
    rho = r / lambda,    lambda = sqrt(kD c)

    >>>Wh(0.004, 0.03)
    4.894104204671381
    """
    def kernel(y, rho):
        """Return the function to be integrated."""
        return np.exp(-y - (rho / 2) ** 2 / y) / y
    def w(u, rho): # Integrate the argument
        return quad(kernel, u, np.inf, args=(rho,))[0]
    wh = np.frompyfunc(w, 2, 1) # Vectorize function w(u, rho) so we can us
    return np.asarray(wh(u, rho), dtype=float)

Wh(0.004, 0.03)


def Wb(tau, rho=0):
    """Return Hantush well function values using the Bruggeman (1999) form

    Parameters
    ----------
    tau = t/ (cS)
    rho = r / lambda, lambda= sqrt(kD c)

    >>>Wb(0.05625, 0.03)
    4.894104204671381
    """
    def kernel(x, rho):
        """Return the function to be integrated."""
        return np.exp(-x - (rho / 2) ** 2 / x) / x
    def w(tau, rho): # Integrate the argument
        return quad(kernel, 0, tau, args=(rho,))[0]
    wh = np.frompyfunc(w, 2, 1) # Vectorize function w(u, rho) so we can us
    return np.asarray(wh(tau, rho), dtype=float)


def Wb1(tau, rho=0):
    """Return Hantush well function values using the Bruggeman (1999) form

    Parameters
    ----------
    tau = t/ (cS)
    rho = r / lambda, lambda= sqrt(kD c)

    >>>Wb(0.05625, 0.03)
    4.894104204671381
    """
    u = rho ** 2  / (4 * tau)
    return Wh(u, rho)

def SRtheis(T=None, S=None, r=None, t=None):
    """Return Step Responss for the Theis well function."""
    dt = np.diff(t)
    assert(np.all(np.isclose(dt[0], dt))), "all dt must be the same."
```

```
    u = r ** 2 * S / (4  * T * t)
    return np.hstack((0, 1 / (4 * np.pi * T) * exp1(u[1:])))
def BRtheis(T=None, S=None, r=None, t=None):
    """Return Block Response for the Theis well function"""
    SR = SRtheis(T, S, r, t)
    return np.hstack((0, SR[1:] - SR[:-1]))
def IRtheis(T=None, S=None, r=None, t=None):
    dt = np.diff(t)
    assert np.all(np.isclose(dt[0], dt))
    u = np.hstack((np.nan, r ** 2 * S / (4 * T * t[1:])))
    return np.hstack((0, 1 / (4 * np.pi * T) *  np.exp(-u[1:]) / t[1:])) *

#u = np.logspace(-4, 1, 51)
u, rho = 0.004, 0.03
print("Wh(u={:.4g}, rho={:.4g}) = {}".format(u, rho, Wh(u, rho)))
tau = rho ** 2 / (4 * u)
print("Wb(tau={:.4g}, rho={:.4g}) = {}".format(tau, rho, Wb(tau, rho)))
```

```
Wh(u=0.004, rho=0.03) = 4.894104204671381
Wb(tau=0.05625, rho=0.03) = 4.894104204671358
```

De nalijlende verlaging

Hoeveel blijft er na verloop van tijd nog over van een tijdeijke onttrekking in het verleden? Deze vraag is eigenlijk alleen relevant voor de Theis situatie met freatische berging, omdat de semi-gespannen situatie volgens Hantush al snel stationair wordt en dan ook snel geen restantvergelaging meer heeft nadat de onttrekking is gestopt.

De verlaging door een onttrekking in een situatie zonder randvoorwaarden wordt goed beschreven door Theis

$$\frac{Q}{2\pi kD}W(u), \quad u = \frac{r^2 S}{4kDt}$$

Er is in een dergelijke Theis situatie na beëndiging van de onttrekking nog zeer lange tijd sprake van een resterende verlaging, die samen met andere niet permanente onttrekkingen toch tot een cumulatief effect zal leiden.

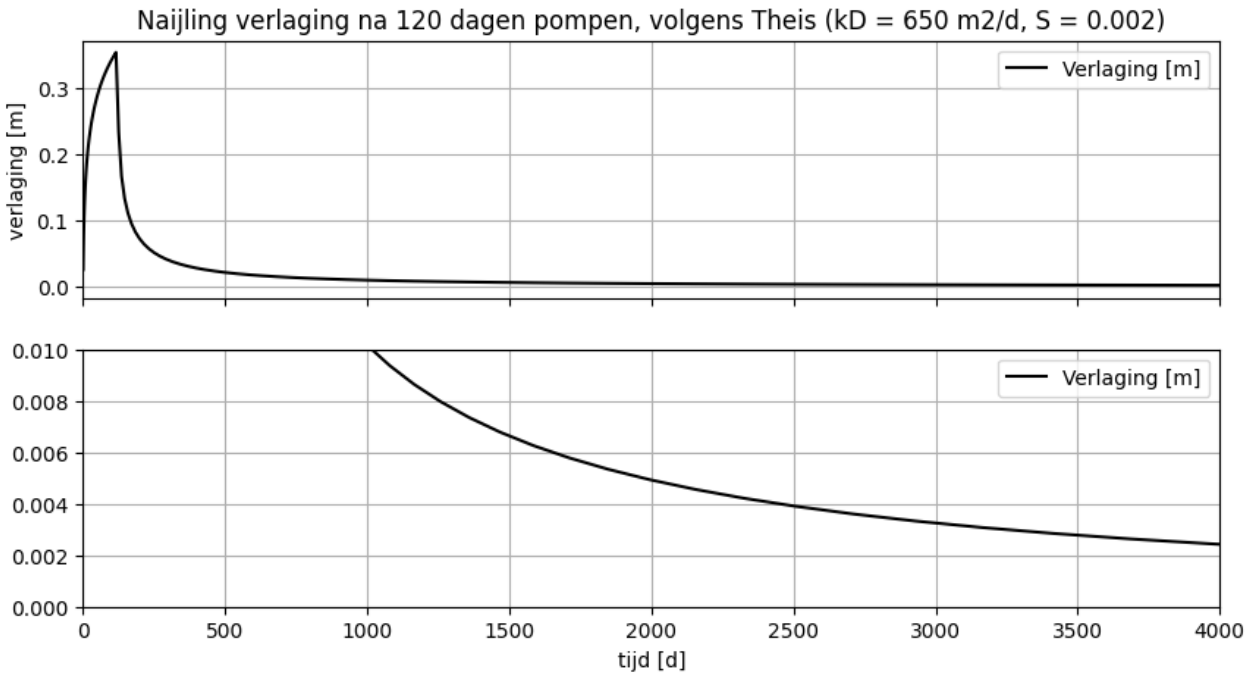
De resterende verlaging is dan gelijk aan

$$s(r,t) = \frac{Q}{4\pi kD}\left[W_t\left(\frac{r^2 S}{4kDt}\right) - W_t\left(\frac{r^2 S}{4kD(t-\Delta t)}\right)\right]$$

Voorbeeld:

In [3]:
```
kD, S = 650, 0.002 # [m2/d]
Dt = 120 # [d] Duur van de onttrekking
r = 1000 # [m]
t = np.logspace(0, 4, 121) # [d]

putten = {
    0: {'Q': +650., 'tStart':   0.},
    1: {'Q': -650., 'tStart': 120.}
}

s = np.zeros_like(t)
for k, put in putten.items():
    Q, tStart = put['Q'], put['tStart']
    u = r ** 2 * S  / (4 * kD * (t[t > tStart] - tStart))
    s[t > tStart] += Q / (4 * np.pi * kD) * Wt(u)

fig, axs = plt.subplots(2, 1, figsize = (10, 5), sharex=True)
for ax in axs:
    ax.grid()
    ax.plot(t, s,  color='k', label='Verlaging [m]')
    ax.legend()
axs[0].set(title='Naijling verlaging na {:.0f} dagen pompen, volgens Theis
           ylabel='verlaging [m]',
            yscale='linear', xscale='linear')
axs[1].set_ylim(0, 0.01)
axs[1].set_xlabel('tijd [d]')
axs[1].set_xlim(0, 4000)
```

Out[3]:
```
(0.0, 4000.0)
```



Het cumulatieve effect van gelijktijdige en in de tijd verschoven onttrekkingen

Bij een groot aantal bemalingen willekeurig in de tijd ontstaat zo een cumulatief effect.

Veronderstel een stad met een oppervlak van 3x3 km, waar op eerste van elke maand op een willekeurige locatie in de stad een bemaling start van 1800 m3/d start, die drie maanden aanhoudt. Dat betekent dat vanaf maand 3 er steeds 3 bemalignen actief zijn elke op een willekeurige locatie in de stad. De bemalingen vinden plaats over een periode van 120 maanden. De periode daarna laat naijling zien. De periode gedurende de beschouwde 120 maanden laat het cumulatieve effect van de bemalingen zien.

De bodemconstantn kD en S zijn zodanig dat bij 1800 m3/d op 10 m afstand na 120 dagfen een verlaging optreedt van 3 m. Dit zijn allemaal redelijke aannamen zoals je die in een stad zou verwachten.

In [4]:
```
kD, S = 650, 0.002 # [m2/d]
Q, duur, r = 1800, 120, 10

print('Verlaging na {} d, bij Q = {:.0f} m3/d op r = {:.1f} m bij kD ={:.0f
    duur, Q, r, kD, S, Q / (4 * np.pi * kD) * Wt(r ** 2 * S / (4 * kD * duu
```

```
Verlaging na 120 d, bij Q = 1800 m3/d op r = 10.0 m bij kD =650 m2/den S =
0.002 m: = 3.02 m
```

De stad met de 100 locaties met een onttreking gedurende drie maanden verdeeld over een periode van 10 jaar.

Elke maand start er een onttrekking op een van de 100 locaties gedurene een periode van 3 maanden. Dit impliceert dat er vanaf maand 3 steeds 3 bemalingen actief zijn op 3 willekeurige locaties van de 100. Het gaat daarom een periode van 43 maanden, waarna alle onttrekkingen zijn gestopt.

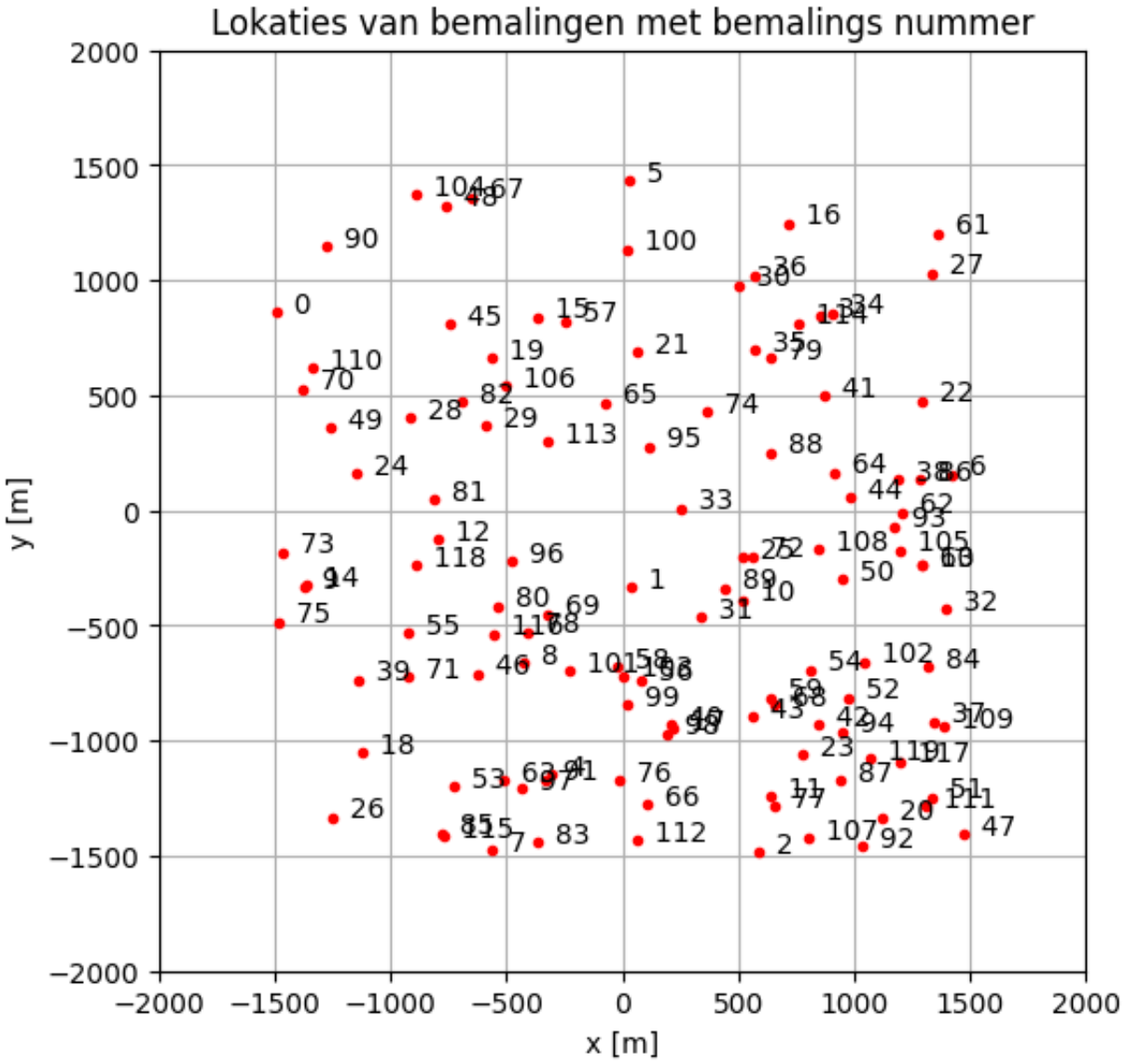
In [5]:
```
# Plot de bemaling op de kaart
tend = 10 * 365

# Kies 120 x,y locaties binnen de stad van 3x3 km.
xy = np.round((np.random.rand(240).reshape(2, 120) - 0.5) * 3000)

# Elke maand start een volgende bemaling
tStart = np.linspace(0, tend, 120).reshape(1, 120)

bemaling = pd.DataFrame(data=np.vstack((xy, tStart)).T, columns=['x', 'y',

bemaling.index.name = 'bemaling'
bemaling['Q']    = -1800 # d
bemaling['duur'] =   90  # d
print(bemaling)
```

bemaling	x	y	tStart	Q	duur
0	-1491.0	861.0	0.000000	-1800	90
1	39.0	-331.0	30.672269	-1800	90
2	585.0	-1484.0	61.344538	-1800	90
3	856.0	842.0	92.016807	-1800	90
4	-306.0	-1146.0	122.689076	-1800	90
...	...	...	...	...	...
115	-773.0	-1416.0	3527.310924	-1800	90
116	-557.0	-536.0	3557.983193	-1800	90
117	1203.0	-1090.0	3588.655462	-1800	90
118	-894.0	-237.0	3619.327731	-1800	90
119	1075.0	-1079.0	3650.000000	-1800	90

[120 rows x 5 columns]

Locaties van de bemalingen

In [6]:
```
ax = newfig('Lokaties van bemalingen met bemalings nummer', 'x [m]', 'y [m]
            xlim=(-2000, 2000), ylim=(-2000, 2000), figsize=(6, 6))
ax.plot(bemaling['x'], bemaling['y'], 'r.', label='locatie bemaling')
for nr in bemaling.index:
    x, y, tstart, Q, duur = bemaling.loc[nr]
    ax.text(x, y, '  {}'.format(nr))
```

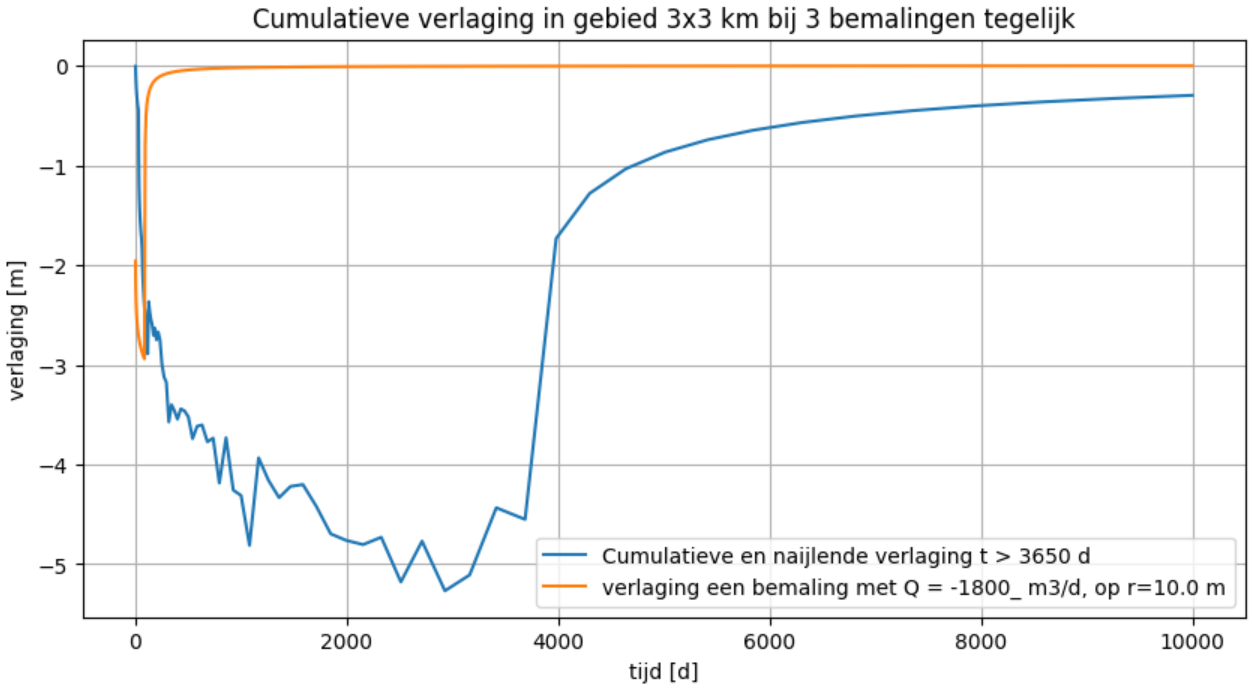


Het cumulatieve effect van de bemalingen

In [7]:

```
x0, y0 = 0., 0. # [m] locatie van de waarneming

ax = newfig('Cumulatieve verlaging in gebied 3x3 km bij 3 bemalingen tegeli
            'tijd [d]', 'verlaging [m]', figsize=(10, 5))

s = np.zeros_like(t)
for k in bemaling.index:
    xw, yw, tstart, Q, duur = bemaling.loc[k]
    r = np.sqrt((xw - x0) ** 2 + (yw - y0) ** 2)

    # Put aan
    u = r ** 2 * S  / (4 * kD * (t[t > tstart] - tstart))
    s[t > tstart] += Q / (4 * np.pi * kD) * Wt(u)

    # Put uit
    u = r ** 2 * S  / (4 * kD * (t[t > tstart + duur] - (tstart + duur)))
    s[t > tstart + duur] -= Q / (4 * np.pi * kD) * Wt(u)

ax.plot(t, s, label='Cumulatieve en naijlende verlaging t > 3650 d')

# Een enkele bemaling 1800 m3/d gedurende 120 d, observatie op r=10 m.
Q, r = -1800, 10

# Put aan op t=0
s = Q / (4 * np.pi * kD) * Wt(r ** 2 * S / (4 * kD * t))

# Put uit op t=90 d
s[t > duur] += -Q / (4 * np.pi * kD) * Wt(r ** 2 * S / (4 * kD * (t[t > duu

ax.plot(t, s, label='verlaging een bemaling met Q = {:.0f}_ m3/d, op r={:.(

ax.legend()
```

Out[7]: <matplotlib.legend.Legend at 0x11fe1aff0>



Formule van Verruijt versus die van Dupuit

De formule van Verruijt gaat ervan uit dat de onttrekking precies gelijk is aan de neerslag binnen de straal waarop de verlaging nul is. Het is wel heel toevallig wanneer de ontrekking en gebiedsrand met vaste stijghoogte met elkaar overeenkomen.

Men kan ook zeggen dat de formule van Verruijt de verlaging berekent binnen een circulair intrekgebied, het gebied waarbinnen al het water naar de put stroomt. Het intrekgebied wordt begrensd door de cirkel waarop er geen verhang is naar de put. Dit is echter niet de grens van de verlaging die de put veroorzaakt, want die wordt bepaald door de afstand tot de rand met gefixeerde stijghoogte.

Volgens Verruijt is in dit geval de volumestroom Q_r bij putonttrekking Q en neerslagoverschot N, gelijk aan

$$Q_r = -\left(h_0^2 - h_w^2 + \frac{N}{2k}(R^2 - r^2)\right)\frac{\pi k}{\ln\frac{r}{R}}$$

Dit kan veel eenvoudiger als volgt worden opgeschreven:

$$Q_r = N\pi r^2 - Q$$

Met als oplossing van Verruijt voor onttrekking uit een pakket met vrije watertafel en vlakke bodem, uit het centrum van een gebied met neerslag :

$$h(r)^2 = H^2 + \frac{N}{2k}(R^2 - r^2) + \frac{Q_0}{\pi k}\ln\frac{r}{R}$$

Het is duidelijk dat de radius waarop het verhang nul is, de grens van het intreggebied van de put, wordt bepaald door

$$Q = N\pi r^2$$

Het zou wel heel toevallig zijn wanneer de onttrekking gelijk is aan de totale neerslag op het circulare gebied. Alleen dan valt het inttrekgebied van de put samen met de gebiedsrand waar $h = H$. Het plaatje bij de uitleg van de formule van Verruijt in de "Handleiding Berekeningsinstrument bemalingen van een bouwput", pag. 9. staat wel $h(0) = H$, de stijghoogte op de rand en er staat R is de invloedsstraal, de afstand van een vaste rand tot aan de put.

Voorbeeld

Een cirkelvormig gebied met neerslag en vaste stijghoogte op de rand met een onttrekking in het centrum.

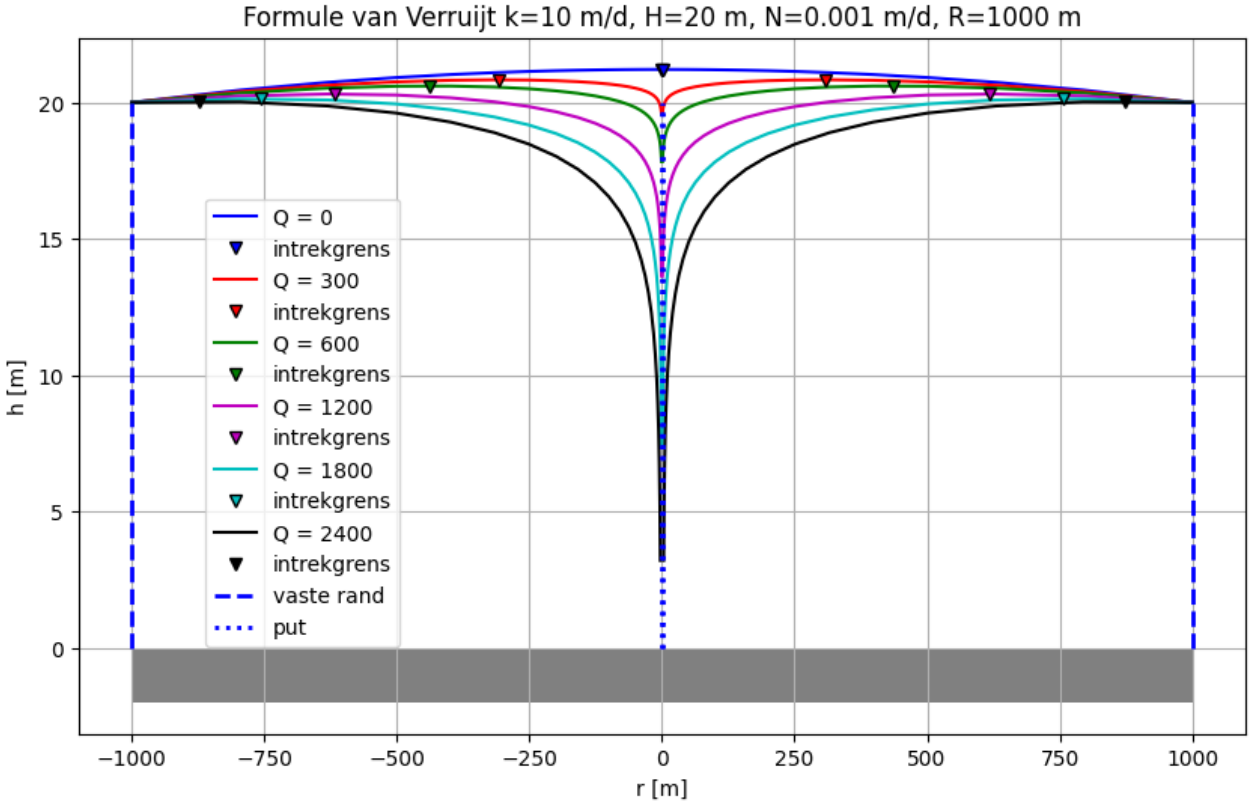
Zonder onttrekking is er alleen sprake van radiaal symmetrische opbolling waarbij de grondwaterstand alleen horizontaal is in het centrum.

Met onttrekking ontstaat er een intrekgebied rond de put met en straal die afhangt van de onttrekking. Op de rand van het inttrekgebied is het verhang gelijk aan nul. De neerslag buiten het intrekgebied stroot nog steeds af naar de vaste rand. Bij toenemende onttrekking neemt de straal van het inttrekgebied eveneens toe. Op het plaatje hieronder is voor een aantal onttrekkingen de verlagingslijn weergegeven en met een driehoekje de straal van het inttrekgebied. De verlaging, de afstand tussen de blauwelijn voor Q=0 en de betreffende gekleurde lijn is alleen nul op de rand. Het gebied met verlaging reikt altijd tot de rand met vaste stijghoogte en dus verder dan het inttrekkgebied. De formule van Verruijt is dus ongeschikt om het gebied met verlaging af te bakenen.

In [8]:

```
# Bodemconstanten, gebiedsradius, voeding, onttrekking en putstraal
k, H, R, N, Q, rw = 10, 20, 1000, 0.001, 1200, 1.


def h(Q, r):
    """Return stijghoogte bij gegeven onttrekking als functie van de afstar

    Gwbruikt de parameters die hierboven zijn gegeven
    """
    return np.sqrt(H ** 2 + N / (2 * k) * (R ** 2 - r ** 2) + Q / (np.pi *


ax = newfig("Formule van Verruijt k={:.0f} m/d, H={:.0f} m, N={:.3g} m/d, R
            k, H, N, R),
            "r [m]", "h [m]",
            figsize=(10, 6))

# Kleuren en debieten
clrs = cycle('brgmck')
Qs =[0.001, 300, 600, 1200, 1800, 2400]

r = np.logspace(0, 3, 31)

for Q in Qs:
    clr = next(clrs)
    # Stijghoogte (links en rechts)
    ax.plot(-r, h(Q, r), color=clr, label='Q = {:.0f}'.format(Q))
    ax.plot(+r, h(Q, r), color=clr, label='')

    # Intrekgebied, radius = rI, links en rechts
    rI = np.sqrt(Q / (np.pi * N))
    ax.plot(-rI, h(Q, rI), ls='none', marker='v', mfc=clr, mec='k', ms=6, l
    ax.plot(+rI, h(Q, rI), ls= 'none', marker='v', mfc=clr, mec='k', ms=6,

# Vaste randen
ax.plot([+R, +R], [0, H], '--', color='blue', lw=2, label='vaste rand')
ax.plot([-R, -R], [0, H], '--', color='blue', lw=2, label='')

# Put
ax.plot([+rw, +rw], [0, H], ':', color='blue', lw=2, label='put')
ax.plot([-rw, -rw], [0, H], ':', color='blue', lw=2, label='')

# Pakket bodem
ax.add_patch(patches.Rectangle((-R, -2), 2 * R, 2, fc='gray'))

leg = ax.legend(loc='lower left', fontsize=10)
leg.set_bbox_to_anchor((0.10, 0.11, 0.3, 0.5), transform=ax.transAxes)
```

```
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/363856301
3.py:10: RuntimeWarning: invalid value encountered in sqrt
  return np.sqrt(H ** 2 + N / (2 * k) * (R ** 2 - r ** 2) + Q / (np.pi * k)
* np.log(r  / R))
```



Conclusie formule van Verruijt

De formule van Verruijt is evenzeer van toepassing als die van Dupuit. Waar Dupuit alleen de verlaging berekent of the natte dikte van de de freatische laag zonder neerslag, geeft de formule van Verruijt deze natte dikte van de freatische laag voor ontrekking met neerslag. De verlaging kan nu worden berekend door $h(Q,r)$af te trekken van de h voor $h(Q=0,r)$. Wat wellicht verwarring geeft als gevolg van de figuur in genoemde handleiding, is dat daar in de figuur de cirkel zonder verhang samenvalt met de cirkel met vaste rand. Dat is echter geen beperking in de formule van Verruijt. Deze neemt binnen de vaste rand netjes neerslag en verlaging samen mee voor de situatie waarin de effectieve laagdikte significant verandert.

De verlaging wordt bij Verruijt verkregen door de grondwaterstand met onttrekking af te trekken van die zonder onttrekking. Bij Dupuit en alle andere oplossingen met vaste laagdikte wordt direct de verlaging berekend. De relatie tussen de berekenigen met vaste laagdikte en die met freatische, ruimtelijke laagdikte volgt direct uit de formule van Verruijt door te schrijven:

$$h^2 - H^2 = (h-H)(h+H) \approx (h-H)2H = 2Hs$$

Dupuit met neerslag:

$$s = \frac{N}{4kH}(R^2 - r^2) + \frac{Q_0}{2\pi kH}\ln\frac{r}{R}$$

Verruijt

$$s = \frac{N}{2k(h+H)}(R^2 - r^2) + \frac{Q_0}{\pi k(h+H)}\ln\frac{r}{R}$$

Bij Verruijt blijft de te berekenen h nog in de noemer staan als je direct de verlaging wilt berekenen. Het verschil tussen beide zit dus in de afwijing van h t.o.v. H. Bij een afwijking van 20% is de fout in de berekening ca. 10%; uiteraard kan direct de formule van Verruijt worden gebruikt, of kan de verlaging iteratief worden berekend met de formule hierboven.

Formule van Blom

Blom heeft in de jaren tachtig van de vorige eeuw een formule ontwikkeld voor de stationaire verlaging die wordt veroorzaakt door een putonttreking in een watervoerend pakket dat wordt gedraineerd door sloten. Door de onttrekking valt een deel van het oppervlaktewater rond de onttrekking droog, waarmee daar ook de drainage wegvalt, die binnen deze radius derhalve kan als voeding kan worden begrepen.

Dus voor $r \leq R$

$$Q = Q_0 - \pi N r^2, \quad r \leq R$$

$$s = \frac{Q_0}{2\pi kD} \ln \frac{r}{R} - \frac{N(R^2 - r^2)}{4kD} + s_R, \quad r \leq R$$

met voor $r = R$

$$Q_R = Q_0 - \pi N R^2$$

en voor $r > r$

$$s = \frac{Q_R}{4\pi kD} \frac{K_0 \frac{r}{\lambda}}{\frac{r}{\lambda} K_1 \frac{r}{\lambda}}$$

met voor $r = R$

$$s_R = \frac{Q_R}{4\pi kD} \frac{K_0 \frac{R}{\lambda}}{\frac{R}{\lambda} K_1 \frac{R}{\lambda}}$$

en dus

$$s = \frac{Q_0}{2\pi kD} \ln \frac{r}{R} - \frac{N(R^2 - r^2)}{4kD} + \frac{Q_R}{4\pi kD} \frac{K_0 \frac{R}{\lambda}}{\frac{R}{\lambda} K_1 \frac{R}{\lambda}}, \quad r \leq R$$

$$s = \frac{Q_R}{4\pi kD} \frac{K_0 \frac{r}{\lambda}}{\frac{r}{\lambda} K_1 \frac{r}{\lambda}} \quad r > R$$

Er is toch iets fundamenteel niet in orde met de formule van Blom. Er wordt een vaste straal aangenomen waarbinnen geen sloten aanwezig zijn en alle neerslagoverschot wordt toegevoegd aan de voeding van het pakket. Daarbuiten is de voeding evenredig met de verlaging, overeenkomstig Hantush. Een voorwaarde is dat die voeding niet meer dan het neerslagoverschot kan zijn. Dat is nu niet ingebakken in de vergelijkingen.

$$N_{s_R} = N_{r=R} = \frac{s_R}{c} = N$$

Als $s_R = Nc$, dan volgt daaruit de radius R, waarbinnen de sloten zijn droog gevallen.

$$Nc = \frac{Q_R}{4\pi kD} \frac{K_0 \frac{R}{\lambda}}{\frac{R}{\lambda} K_1 \frac{R}{\lambda}}$$

$$Nc = \frac{Q_0 - \pi N R^2}{4\pi kD} \frac{K_0 \frac{R}{\lambda}}{\frac{R}{\lambda} K_1 \frac{R}{\lambda}}$$

De waarde voor R is hieruit iteratief te berekenen met de Newton methode, waarvoor de afgeleide nodig is van functie $y(R)$:

$$y(R) = -Nc + \frac{Q_0 - \pi N R^2}{4\pi kD} \frac{K_0 \frac{R}{\lambda}}{\frac{R}{\lambda} K_1 \frac{R}{\lambda}}$$

R is de waarde waarvoor de functie $y(R) = 0$.

Merk op dat

$$\frac{dK_0(r)}{dr} = -K_1 r$$

$$\frac{rK_1(r)}{dr} = -rK_2(r)$$

Dus dat is te doen.

Hoe lang duurt het voordat de verlagingen als stationair kan worden beschouwd?

Bij onttrekking met een put in een oneindig uitgestrekt watervoerend paket ontstaat nooit een stationaire eindsituatie omdat er geen terugkoppeling is tussen verlagen en infiltratie.

Verlagingen kunnen alleen stationair worden wanneer ergens in het watervoernde pakket randen zijn met vaste stijghoogte, danwel er een vaste stijghoogte is in een bovenliggend pakket waarmee hydraulische verbinding bestaat via lek door een semi-doorlatende laag. De essentie is dat door de verlaging een stroming wordt opgewekt die op termijn de onttrekking volledig kan compenseren.

De belangrijkste situatie waarbij stationaire stoming optreedt is meestal die met oppervlaktewater verderop in het gebied of een semi-permanente toplaag waarboven het oppervlaktewaterpeil ruimtelijk wordt beheerd.

Voor een put die onttrekt uit een pakket met semi-gespannen water geldt de formule van Hantush (1955)

$$s(r,t) = \frac{Q}{4\pi kD} W_h\left(u, \rho\right), \quad with \;\; u = \frac{r^2 S}{4kDt}, \quad and \;\; \rho = \frac{r}{\lambda}$$

Wanneer deze verlaging wordt uitgezet op logartmische tijdschaal blijkt er een buigpunt

te zijn halverwege de eindverlaging, waar de verlaging steeds trager naar toe loopt. Het tijdstip van dit buigpunt is een geschikt moment om op praktische wijze de tijd te bepalen waarna de verlaging als stationair mag worden beschouwd.

Het buigpunt ligt op

$$u = \frac{\rho}{2} = \frac{r}{2\lambda}$$

Om het snijpunt van de raaklijn door het buigpunt en de horizontale lijn van de eindverlaging te bepalen, is het mathematisch handiger om direct met de de parameter op de horizontale as te werken

$$\eta = \ln\left(\frac{1}{u}\right) = -\ln u$$

De tangent of the raaklijn door de het buigpunt op de as $1/u$ is

$$\left(\frac{\partial W_h}{\partial \eta}\right) = e^{-\rho}$$

De stationaire eindwaarde is

$$W_h(0,\rho) = 2K_0(\rho)$$

Waar de raaklijn door het buigpunt de lijn met de eindverlaging snijdt is de verlaging halverwege de eindverlaging behorend bij de stationaire situatie. Het punt waarop de verlaging stationair is geworden kan praktisch worden gedefinieerd als het punt waarop de raaklijn door het buigpunt de horizontale lijn met de eindverlaging snijdt.

$$\eta = e^{\rho}K_0(\rho) + \eta_{Bp}, \quad with \quad \eta = -\ln(u)$$

$$\eta_{Bp} = -\ln(u_{Bp}), \quad with \quad u_Bp = \frac{\rho}{2}$$

$$\eta - \eta_Bp = e^{\rho}K_0(\rho)$$

$$\frac{e^{\eta}}{e^{\eta_{Bp}}} = e^{e^{\rho}K_0(\rho)}$$

$$\frac{u_{Bp}}{u} = e^{e^{\rho}K_0(\rho)}$$

Het stijdtip (de u-waarde) waarop de verlaging als stationair mag worden beschouwd volgt dus direct uit de u-waarde van het buigpunt u_{Bp} en $\rho = r/(2\lambda)$:

$$u = u_{Bp}e^{-e^{\rho}K_0(\rho)}$$

In het voorbeeld hieronder wordt de Hantush functie getoond versus $1/u$ op

logaritmische schaal. Het eerste deel van de dubbele buiging start wanneer de invloedsgrens van de onttrekking het waarnemingsput bereikt en het tweede deel wanneer de verlaging wordt afgeremd door de opgewekte lek. De eindverlaging is gelijk aan de stationaire verlaging, zodat, met $t \to \infty,\ u \to 0$:

$$W_h(0, \rho) = 2K_0(\rho)$$

Het buigpunt light op halve hoogte, dus bij $W_h(u_{bp}) = K_0(\rho)$. Voor het buigpunt geldt bovendien

$$u_{bp} = \frac{\rho}{2} = \frac{r}{2\lambda}$$

De berekende buigpunten worden met een dikke stip in de grafiek aangegeven.

Voorts is in de grafieken aangegeven waar de raaklijn door het buigpunt en waar de eindverlaging snijdt (stippelijn met snijpunt).

Het snijpunt van deze raaklijn met de eindverlaging ligt op

$$u = u_{Bp} e^{-e^{\rho} K_0(\rho)}$$

Deze gemakkelijk te berekenen waarde van u is een praktische waarde voor het definieren van het moment waarop de eindverlaging is bereikt kan worden beschouwd.

$$u = \frac{\rho}{2} e^{-e^{\rho} K_0(\rho)}$$

Voor een reële situatie (met het argument in de Theis-formule (geen lek) links staat en de invloed van de lek rechts):

$$u = \frac{r^2 S}{4kDt} = \frac{r}{2\lambda} e^{-e^{\frac{r}{\lambda}} K_0\left(\frac{r}{\lambda}\right)}$$

of

$$u = \frac{r^2 S}{4kDt} = \frac{1}{2} \rho e^{-e^{\rho} K_0(\rho)}$$

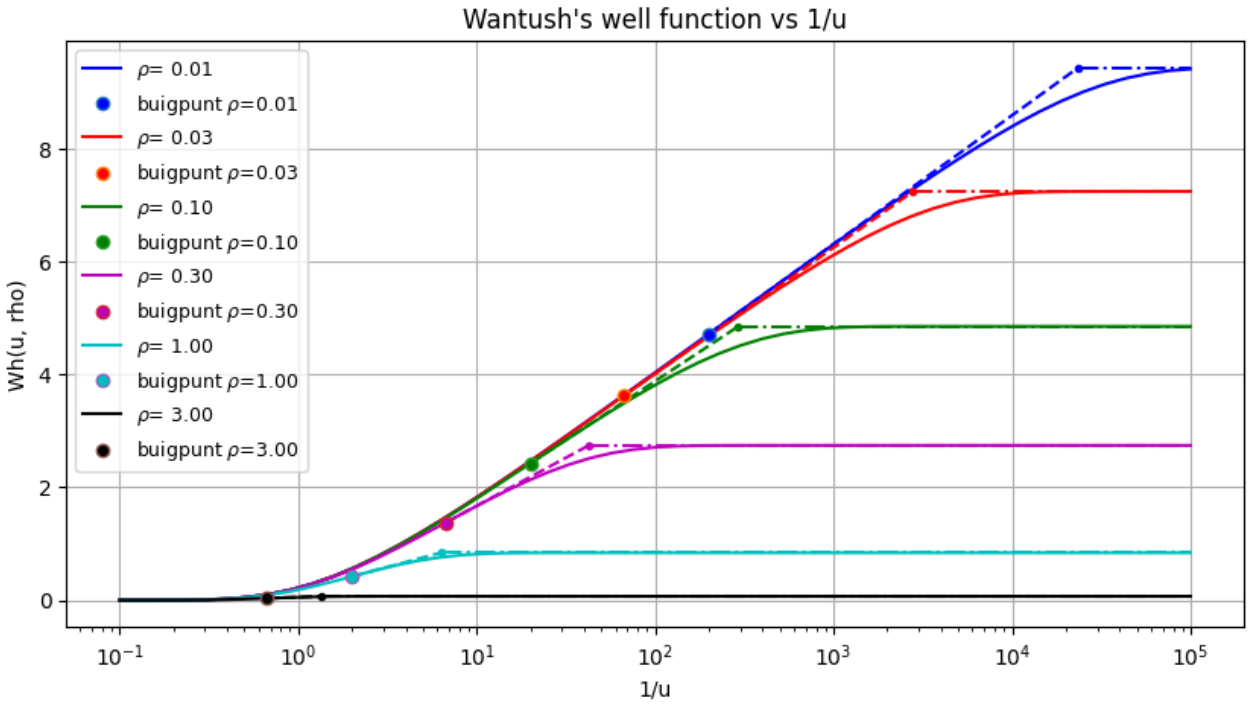
Voorbeeld

In [9]:

```
u = np.logspace(-5, 1, 51)
ax = newfig("Wantush's well function vs 1/u", "1/u", "Wh(u, rho)", xscale='

clrs = cycle('brgmck')

rhos = [0.01, 0.03, 0.1, 0.3, 1, 3]

for rho in rhos:
    clr = next(clrs)

    # Hantush
    ax.plot(1/u, Wh(u, rho), color=clr, label=r'$\rho$= {:.2f}'.format(rho)

    # u-waarde van buigpunt
    uBp = rho  / 2
    eta2 = np.exp(rho) * K0(rho) - np.log(uBp)
    u2 = np.exp(-eta2)

    # Buigpunt
    ax.plot(1 / uBp, Wh(rho / 2, rho), 'o', mfc=clr,  label=r'buigpunt $\rh

    # Het raaklijn door buigpunt --- lijn
    ax.plot([1 / uBp, 1 / u2], [K0(rho), 2 * K0(rho)], '.--', color=clr, la

    # Eindverlaging -.-.- lijng
    ax.plot([1 / u2, 1 / u[0]], [2 * K0(rho), 2 * K0(rho)], '-.', color=clr

ax.legend(loc='upper left', fontsize=9)
```

Out[9]:

```
<matplotlib.legend.Legend at 0x11fd13080>
```



Onvolkomen putten

Een onvolkomen put veroorzaakt op kortere afstanden van de put een afwijking van de verlaging ten opzichte van de situatie met een put met een filter over de gehele dikte van de watervoerende laag. Onvolkomen filters zijn echter de gangbare situatie, met name in dikkere pakketten.

We kunnen de invloed van de onvokomenheid van het putfilter verwerken met een correctie van Hantush. Deze correctie kan gewoon worden meegenomen in de analytische oplossing.

Nog uitwerken (kan eenvoudig met formule hiervoor van Hantush (1956?) of eventueel van Huisman (1970)).

In [ ]:

Vertraagde nalevering (Delayed yield)

Pompproeven die als basis voor aanvragen van vergunningen worden uitgevoerd, zijn vaak zo kort dat alleen de verlaging in het bepompte pakket kan worden bepaald, en geen informatie wordt verkregen van het later nazakken van het freatisch vlak boven de scheidende laag, die optreedt als gevolg van de door het pompten toegenomen lek. De voortoets zou altijd moeten eisen dat dit effect van nazakken wordt meegenomen in de analyse.

$$s = \frac{Q}{4\pi kD} W(u) = \frac{Q}{2\pi kD} K_0\left(\frac{r}{\lambda}\right)$$

$$W(u) = 2K_0\left(\frac{r}{\lambda}\right)$$

$$\ln\left(\frac{2.25kDt}{r^2 S_y}\right) = 2K_0\left(\frac{r}{\lambda}\right)$$

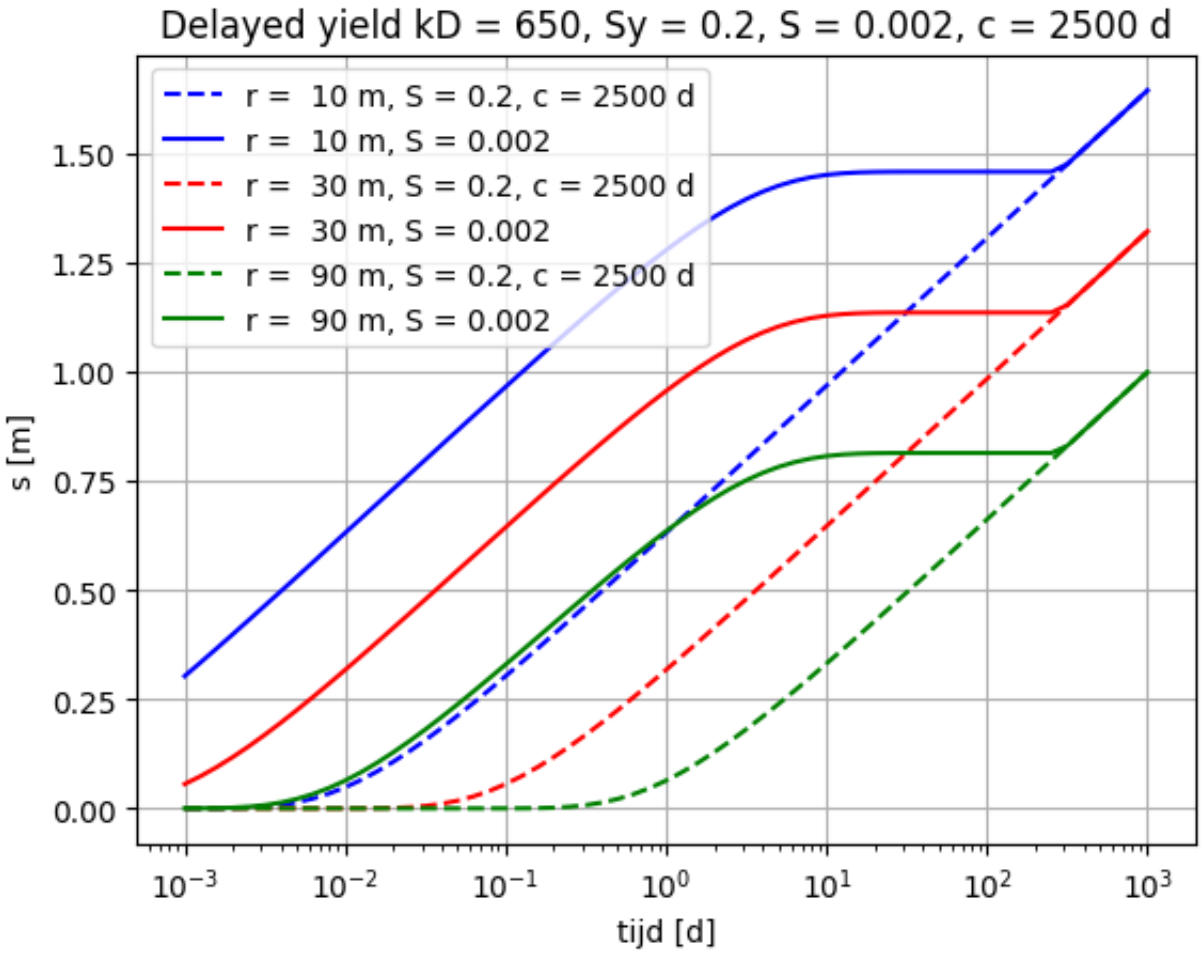
$$t = \frac{r^s S_y}{2.25kD} e^{2K_0\left(\frac{r}{\lambda}\right)}$$

De algemene situatie met vertraagde naverling kan voor het semi-gespannen pakket worden benaderd door de aanvankeijke verlaging volgens Hantush met de elastische bergingscoëfficiënt S, gevolgd door de verlaging volgens Theis met de freatische bergingscoëfficiënt wanneer de verlaging volgens Theis die volgens Hantush inhaalt. Vanaf dat moment zijn de verlaging in het ondiepe en diepe pakket geijk, en lopen samen verder op. In de werkelijkheid ontstaat een vloeiende overgang van de Hantush verlaging naar die volgens Theis, die met een tweelagen model kan worden berekend. Dat is wat ignewikkelder. Voor de praktijk kan de hier gevolgde methode als benadering worden gebruikt. Hij laat in elk geval zien hoe lang men een pompproef doen om zeker te weten dat er in later stadium geen substantiële daling van het freatisch vlak optreedt.

Voorbeeld:

In [10]:

```
kD, Sy,S, c, Q0 = 650, 0.2, 0.002, 2500., 1200
L = np.sqrt(kD * c)

t = np.logspace(-3, 3, 61)
rs = [10, 30, 90]

ax = newfig("Delayed yield kD = {:.0f}, Sy = {:.3g}, S = {:.3g}, c = {:.0f}
            "tijd [d]", "s [m]",xscale='log')
clrs = cycle('brgmck')
for r in rs:
    clr = next(clrs)
    rho = r  / L
    tx = r ** 2 * Sy / (2.25 * kD) * np.exp(2 * K0(rho))
    u1 = r ** 2 * Sy / (4 * kD * t)
    u2 = r ** 2 * S  / (4 * kD * t)
    s1 = Q0 / (4 * np.pi * kD) * Wt(u1)
    s2 = Q0 / (4 * np.pi * kD) * Wh(u2, rho)
    s2[t > tx] = s1[t > tx]
    ax.plot(t, s1, ls='--', color=clr, label='r = {:3.0f} m, S = {}, c = {:
    ax.plot(t, s2, ls= '-', color=clr, label='r = {:3.0f} m, S = {}'.format
ax.legend()
plt.show()
```



Worst case = "Je mag het niet missen, wanneer de voorgenomen ingreep serieuze gevolgen heeft"

Hoe vullen we dat in? Nog uit te werken. Mogelijk niet eenduidig.

Grondwaterbalans. Hoe hiermee omgaan in de passende beoordeling?

Dit gaat onder andere om indirect, via het grondwater, onttrekken aan beken met beperkte afvoer, die in de zomer kwetsbaar zijn. Dit speelt in het algemeen op hogere gronden waar geen externe aanvoer naar het oppervlaktewater kan plaats vinden, dus waarin al het oppervlaktewater afkomstig is van toestromend grondwater.

De onttrekking eigent zich water toe dat anders een andere bestemming zou vinden, zoals voeding van beken en of van de vegetatie. Er zijn derhavle atijd gevolgen van een onttrekking voor de waterbalans van het gebied waarop de ontrekking invloed uitoefent. Men kan berekenen hoeveel indirect via het grondwater aan een naburige beek wordt onttrokken, repectievelijk aan voeding wordt ontnomen. Met name kleinere beken zijn gevoelig voor aanpalende grondwateronttrekkingen. Regionaal kan het cumlatieve effect grote negatieve gevolgen hebben voor de beekafvoer en daarmee voor de ecologie en andere gebruiksdoelen. In de literatuur zijn vele voorbeelden te vinden waarin beken en hele rivieren droog zijn gevallen door onttrekking van grondwater.

GxG kaart. Welke rol kan die spelen bij de passende beoordeling?

Kan hiermee Vlaanderen worden verdeeld in grondwatergebiedstypen: Infiltratiegebied, intermediar en kwelgebied? Zulke kaareten geven aan wat er als natuurlijke variatie in de grondwaterstand beschouwd kan worden.

Er kan voor de beoordeling van vergunningaanvragen rekening worden gehouden met het type grondwatersysteem: infiltratiegebied (hoog gelegen met diepe grondwaterstand, weinig of geen drainagemiddelen of natuurlijke afvoer) en een kwelgebied (relatief laag gelegen ten opzichte van de omgeving, met hoge grondwaterstanden en veel drainagemiddelen), danwel een overgangsgebied tussen deze twee uitersten in. Naar verwachting zullen toekomstige GxG kaarten deze gebiedsindeling weerspiegelen. De gevolgen van een onttrekking in de verschillende zones zullen ook verschillen. Zo zal een onttrekking in een infiltratiegebied ogenschijnlijk nabij nauwelijks invloed hebben maar zal de invloed verderop manifesteren in het droogvallen van koppen van beken, en van sloten in het overgangsgebied. Deze kunnen daar heel gevoelig voor zijn, terwijl het een op een aantonen van deze effecten aan de hand van metingen wordt bemoeilijkt door de grote natuurlije variatie van de afvoer van de beekkoppen en de van nature variënde slootpeilen in de overganszone. De reikwijdte van de invloed van de onttrekking in een infiltratiegebied is in het algemeen groot doordat de randen die de invloed dempen ver weg liggen.

De situatie bij een onttrekking in een kwelgebied met veel oppervlaktewater, dat stevig gevoed wordt door kwel uit de verre omgeving en (dus uit een groot gebied) is omgekeerd. De reikwijdte is door het vele dempende oppervlaktewater in de naaste omgeving beperkt, zowel wat de verlaging betreft als wat betreft de invloed op het oppervlaktewater.

De situatie in de overgangszone ligt hier tussenin. Beken en sloten tegen het hogere gebied aan zijn zeer gevoelig voor de onttrekking, voor die in de richting het kwelgebied is dit juist minder het geval. De invloed van de onttrekking en zijn reikwijdte is daarom asymmetrisch, hij is groter richting het infiltratiegebied en kleiner richting het kwelgebied.

Deze nuances zijn analytisch moeilijk te benaderen. Een model heeft hier echter geen moeite mee. Analytisch zou men de invloedsradius kunnen kiezen afhankelijk van het gebiedstype, af te leide uit de GxG kaarten. Een methode hiervoor zou moeten worden uitgewerkt en getoetst aan een aantal representatieve situaties.

Invloedstraal (radius of influence)

Om de in de tijd toenemende reikwijdte van een onttrekking aan te geven wordt wel gesproken van invloedsgrens of "radius of influence". Deze "radius of influence" bij een continue onttrekking in een zich oneindig uitstrekkend watervoerend pakket zonder lek kan direct worden afgeleid uit de verlaging volgens Theis, die in dit geval van toepassing is. De verlaging volgens Theis kan voor voldoende grote tijd worden benaderd met de vereenvoudigde formule met daarin de logarithme in plaats van de well function:

$$s = \frac{Q}{4\pi kD} \ln\left(\frac{2.25kDt}{r^2 S}\right)$$

Voor het nog niet beïnvloede gebied geldt dat de verlaging nog nul is. Dit is hier (bij benadering) het geval wanneer het argument onder de logaritme gelijk is aan 1:

$$\frac{2.25kDt}{r^2 S} = 1$$

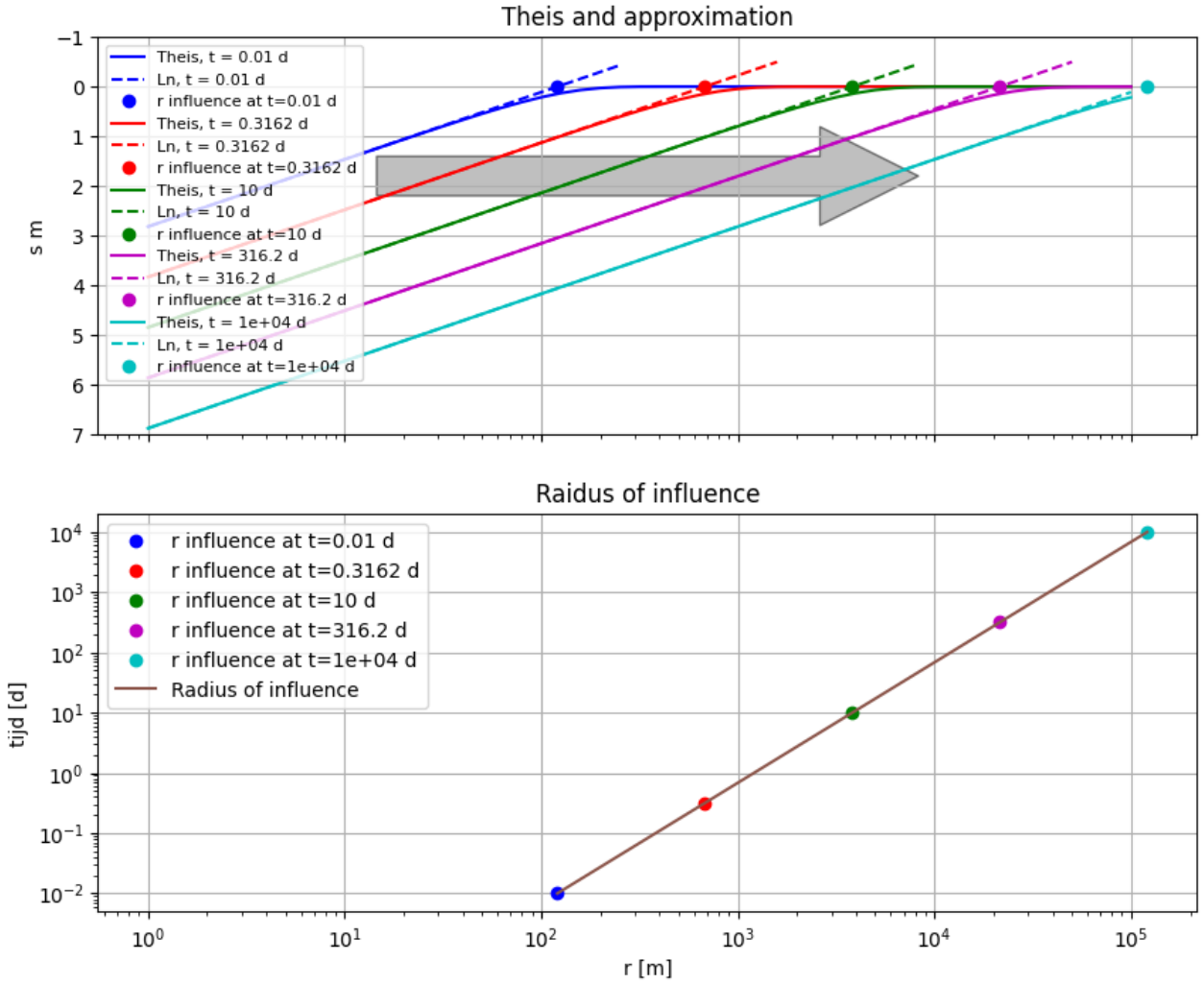
zodat daar een "radius of influence" uit volgt, die gelijk is aan

$$r = \sqrt{\frac{2.25kDt}{S}}$$

Deze formule is een eenvoudige doch uitermate praktisch uitdrukking om de radius te bepalen waarbuiten de invloed van de onttrekking nog niet bemerkbaar is. Het is een benadering, omdat de gebruikte formule van de verlaging een benadering is, maar het is een praktische benadering die op de aangegeven wijze objectief is afgeleid. Hieronder volgt een voorbeeld waarin de werking van deze radius inzichtelijk wordt gemaakt.

In [11]:

```
kD, S, Q0 = 650, 0.001, 2400

fig, axs = plt.subplots(2, 1, figsize=(10, 8), sharex=True )
axs[0].set_title('Theis and approximation')
axs[0].set_ylabel('s m')
axs[0].grid()
axs[0].set_xscale('log')
axs[0].set_ylim(7, -1)
axs[1].set_title('Raidus of influence')
axs[1].set_ylabel('tijd [d]')
axs[1].set_xlabel('r [m]')
axs[1].set_yscale('log')
axs[1].grid()

r = np.logspace(0, 5, 51)
ts = np.logspace(-2, 4, 5)

clrs = cycle('brgmck')
for t in ts:
    clr = next(clrs)
    u = r ** 2 * S  /( 4 * kD * t)
    sTheis = Q0 / (4 * np.pi * kD)* Wt(u)
    sLn    = Q0 / (4 * np.pi * kD) * np.log(2.25 * kD * t / (r ** 2 * S))
    axs[0].plot(r, sTheis, ls='-',  color=clr, label='Theis, t = {:.4g} d'.
    axs[0].plot(r[sLn > -0.5], sLn[sLn > -0.5],    ls='--', color=clr, labe
    rinf = np.sqrt(2.25 * kD * t / S)
    axs[0].plot(rinf, 0, 'o', mfc=clr, mec=clr, label='r influence at t={:.
    axs[1].plot(rinf, t, 'o', mfc=clr, mec=clr, label='r influence at t={:.

arrow = patches.FancyArrowPatch((0.25, 0.65), (0.75,0.65), transform=axs[0]
                                mutation_scale=100, fc='gray', ec='black',
#arrow = patches.FancyArrowPatch((0.25, 0.75), (0.75,0.75), transform=fig.t
axs[0].add_patch(arrow)
axs[0].legend(loc='upper left', fontsize=8)

tt = np.logspace(-2, 4, 61)
rinf = np.sqrt(2.25 * kD * tt / S)
axs[1].plot(rinf, tt, label='Radius of influence')
axs[1].legend()
plt.show()
```



Exacte reikwijdte voor de Theis situatie

Ik denk dat de benadering even goed werkt in de praktijk, maar men kan een invloedsstraal berekenen voor een exacte opgegeven verlaging van bijv. 5 cm. Daarvoor hebben we wel de inverse nodig van $W(u)$, dus de u die hoort bij bekende $W(u)$.

De exacte reikwijdte (voor zover die in de praktijk bepaalbaar is) voor de Theis-situatie voor een gegeven verlagingvan bijvoorbeeld 5 cm, vergt een wat nauwkeuriger berekening.

$$s = \frac{Q}{4\pi kD} W(u)$$

en dus

$$W(u) = 4\pi kD \frac{s}{Q}$$

waarbij s gegeven is net als Q. Om de reikwijdte te berekenen hebben we u nodig

$$u = W^{-1}(u)$$

Voor de inverse van W(..) (de exponential integral) bestaat geen kant en klare analytische uitdrukking. Ergo de inverse moet numeriek worden uitgerekend. Dit kan snel met Newton's methode of nog sneller met Halley's method (zie wikipedia). Echter het is aan te bevelen om in plaats van W(u) ln(W(u) als functie te nemen en ln(u) als argument. We beperken ons vanwege de afleiding dan wel tot Newton's method.

We solve for

$$F(z, A) = ln(W(u)) - ln(A) = 0$$

met $z = ln(u)$, zodat, met $u = e^z$ volgt dat $\frac{de^z}{dz} = e^z = u$

$$F1(z) = \frac{dF(z, A)}{dz}$$

$$\frac{dF(z, A)}{dz} = \frac{d(\ln(W(u)) - \ln(A))}{dz}) = \frac{1}{W(u)} \frac{dW(u)}{du} \frac{du}{dz} = \frac{1}{W(u)} \frac{-e^{-u}}{u} \frac{de^z}{dz} = \frac{1}{W(}$$

$$F2(z) = -\frac{d}{dz}\left(\frac{e^{-u}}{W(u)}\right) = W^{-2}(u)\frac{e^{-u}}{u}\frac{du}{dz}e^{-u} - W^{-1}(u)e^{-u}\frac{du}{dz} = \frac{ue^{-u}}{W(u)} - \frac{e^{-2u}}{W^2(u)}$$

Newtons's method

$$z_{n+1} = z_n - \frac{F(z_n, A)}{F1(z_n)}$$

Finally,

$$u = \ln(z)$$

Halley's method convergeert nog sneller en is gedefineerd als (Wikipedia)

$$z_{n+1} = z_n - \frac{F(z_n, A)}{F1(z_n)}\left[1 - \frac{F(z_n, A)}{F1(z_n)} \times \frac{F2(z_n)}{2F1(z_n)}\right]^{-1}$$

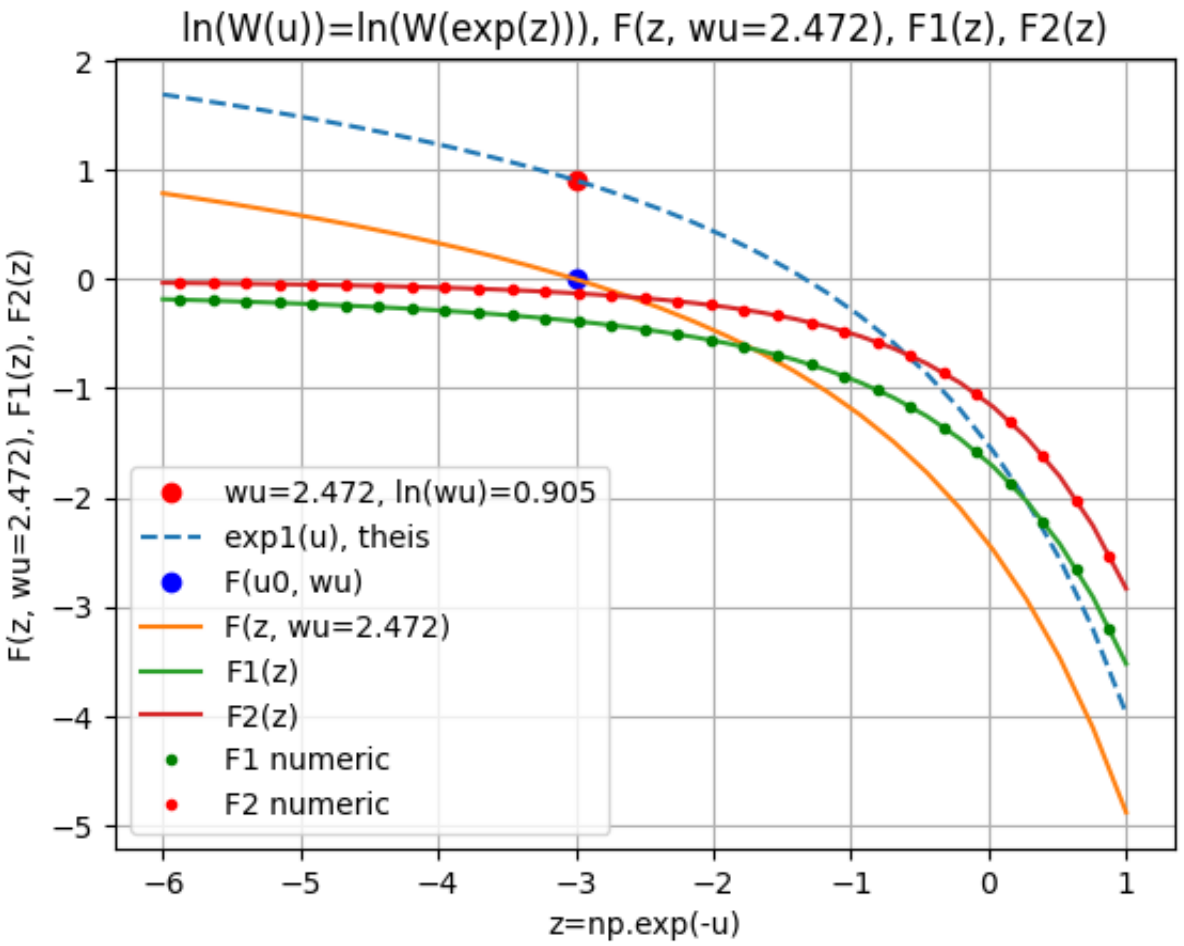
Halley's methode is ingebakken als een optie in de funktie beneden. Voor Haley moet $A = W(u)$ < 20, wat in de praktijk altijd zo is.

Verification of F(z, wu) the derivatives F1(z) and F2(z)

In [12]:

```
def F(z, wu):
    u = np.exp(z)
    return np.log(exp1(u)) - np.log(wu)

def F1(z):
    u = np.exp(z)
    return -np.exp(-u) / exp1(u)

def F2(z):
    u = np.exp(z)
    return u * np.exp(-u) / exp1(u) - np.exp(-2 * u) / exp1(u) ** 2

# The point u0, W(u0)
u0 = np.exp(-3)

z = np.linspace(-6, 1, 30)

fig, ax = plt.subplots()
ax.grid()
ax.set_title(f'ln(W(u))=ln(W(exp(z))), F(z, wu={exp1(u0):.4g}), F1(z), F2(z
ax.set_xlabel('z=np.exp(-u)')
ax.set_ylabel(f'F(z, wu={exp1(u0):.4g}), F1(z), F2(z)')

# The point
ax.plot(np.log(u0), np.log(exp1(u0)), 'ro', label=f"wu={exp1(u0):.4g}, ln(w
# exp1(u)
ax.plot(z, np.log(exp1(np.exp(z))), '--', label="exp1(u), theis")
# F(z, wu)
ax.plot(np.log(u0), F(np.log(u0), exp1(u0)), 'bo', label="F(u0, wu)")
# The point now zero
ax.plot(z, F(z, exp1(u0)), label=f"F(z, wu={exp1(u0):.4g})")
# F1(z)
ax.plot(z, F1(z), label="F1(z)")
# F2(z)
ax.plot(z, F2(z), label="F2(z)")
# F1 numerically computed from F
FF1 = (F(z[ 1:], exp1(u0)) - F( z[:-1], exp1(u0))) / np.diff(z)
# F2 numerically computed from F1
FF2 = (F1(z[1:])           - F1(z[:-1]))           / np.diff(z)

zm = 0.5 * (z[:-1] + z[1:])
# F1 numerically
ax.plot(zm, FF1, 'g.', label="F1 numeric")
#F2 numerically
ax.plot(zm, FF2, 'r.', label="F2 numeric")

ax.legend()
```

Out[12]: <matplotlib.legend.Legend at 0x11fc02ff0>



In [13]:

```
def exp1_inv(A, Halley=False, verbose=False):
    """Return inverse of function scipy.special.exp1 (Theis well function)

    Parameters
    ----------
    A: float
        The W(u) or exp1(u) value for  which u values is desired.
    """

    def F(z, A):
        """Return funcion value to be zeroed.

        Parameters
        ----------
        z: float
            z = ln(u)
        A: float
            A = W(u), given, u is to e determined
        """
        u = np.exp(z)
        return np.log(exp1(u)) - np.log(A)

    def F1(z):
        """Return derivative of F(z, A).
```

```
        Parameters
        ----------
        z: float
            z = ln(u)
        """
        u = np.exp(z)
        return -1 / exp1(u) * np.exp(-u)

    def F2(z):
        """Return second derivative of F(z, A)"""
        u = np.exp(z)
        return u * np.exp(-u) / exp1(u) - np.exp(-2 * u) / exp1(u) ** 2

    assert isinstance(A, float) and A > 1e-5, "A must be positive float lar
    if Halley:
        assert A <=20., "If Halley, A must be < 20."
        if verbose:
            print("Hayley point seeking applied.")
    elif verbose:
        print("Newton's point seekind applied")

    u = 1.0 # Initial value
    z = np.log(u)
    while True:
        N = F(z, A) / F1(z)
        if Halley:
            N = N / (1 - N * F2(z) / (2 * F1(z)))
        dz = N
        if verbose:
            print(z, dz)
        z -= dz
        if abs(dz) < 1e-12 or np.isnan(z):
            break
    return np.exp(z)

# Controleren van het resultaat, krijgen we de A terug als we de met A gevo
As = np.logspace(-5, 2, 8) * 2.
for A in As:
    #u = exp1_inv(A, Halley=True, verbose=True)
    #u = exp1_inv(A, Halley=False, verbose=True)
    u = exp1_inv(A, Halley=False, verbose=False)
    #u = exp1_inv(A, Halley=True, verbose=False)
    print(f"A was {A} --> A = {exp1(u)}, u={u}")
```

```
A was 2e-05 --> A = 1.9999999999999954e-05, u=8.570422533425244
A was 0.0002 --> A = 0.00019999999999999996, u=6.514957070441814
A was 0.002 --> A = 0.002, u=4.530286442484425
A was 0.02 --> A = 0.02000000000000001, u=2.6678509610000907
A was 0.2 --> A = 0.19999999999999998, u=1.0556504654350867
A was 2.0 --> A = 2.0, u=0.08237202962072025
A was 20.0 --> A = 20.0, u=1.1572542497456033e-09
A was 200.0 --> A = 200.0, u=7.770018292116169e-88
```

Toepassing vergelijking van de eenvoudige en de meer complexe berekening van de invloedsstraal

In [14]:

```
# Parameters
kD, S, Q = 600., 0.2, 788.
s = 0.05 # m
t = 10.

# De waarde van W(u) voor de opgegeven kleine verlaging s=0.05 m
wu = 4 * np.pi * kD * s / Q

# De waarde van u voor deze W(u)u
u = exp1_inv(wu)

# Radius of influence
r05 = np.sqrt(4 * u * kD * t / S) # exact 5 cm verlaging
r =   np.sqrt(2.25  * kD * t / S) # Benadering

# Resultaat
print(f"kD={kD}, S={S}, Q={Q}, s={s}, t={t}, wu={wu:.4g}, u={u:.4g}, r05={r
```

```
kD=600.0, S=0.2, Q=788.0, s=0.05, t=10.0, wu=0.4784, u=0.5746, r05=262.6, r
=259.8
```

Berekening van de verlaging, ten gevolge van een willekeurig in de tijd verlopende onttrekking, met convolutie

Convolutie is een algemene method waarmee het dynamische verloop van dde grondwaterstand of verlaging kan worden berekend die volgt uit een in de tijd variërende onttrekking of andere stress zoals neerslagoverschot. Convolutie is in feite een slimme manier van superpositie, waarmee in eenslag een hele tijdreeks kan worden berekend.

De stanaard oplossingen voor het niet-stationaire verloop van de verlaging door een stationaire onttrekking zijn op te vatten als stapresponses wanneer de onttrekking gelijk aan 1 wordt genomen. Bijvoorbeeld de verlaging volgens Theis. De staprespons volgens theis is dan

$$SR = \frac{1}{4\pi kD} W_t(u) = \frac{1}{4\pi kD} \int_u^\infty \frac{e^{-y}}{y} dy, \;\; u = \frac{r^2 S}{4kDt}$$

Hieruit volgt de zogenoemde impulsrespons door differentiatie

$$IR = \frac{\partial}{\partial t} SR = \frac{Q}{4\pi kD} \frac{e^{-u}}{t}$$

De impulsrespons heeft een groot theoretisch belang, maar is voor ons niet nodig. Belangrijker in de praktijk is de staprespons die het gevolg is van een continue onttrekking van 1 m3/d vanaf t=0 en de daaruit afgeleide blokrespons die de reactie is van de grondwaterstand op een plotseling onttrekking van 1 m3/d gedurende exact 1 tijdstap (meestal een dag)

$$BR = \frac{Q}{4\pi kD} \left[W_t\left(u_t\right) - W_t\left(u_{t-dt}\right) \right]$$

De berekening van de verlaging door een willekeurig verlopend debiet geschiedt het gemakkelijkst met de functie `lfilter(...)` uit de module `scipy.signal` .

$$s(r,t) = \frac{1}{4\pi kD} \sum_{i=0}^{\infty} Q(t-\tau_i) BR(\tau_i) = lfilter\left(BR(\tau, 1, Q(t-\tau))\right)$$

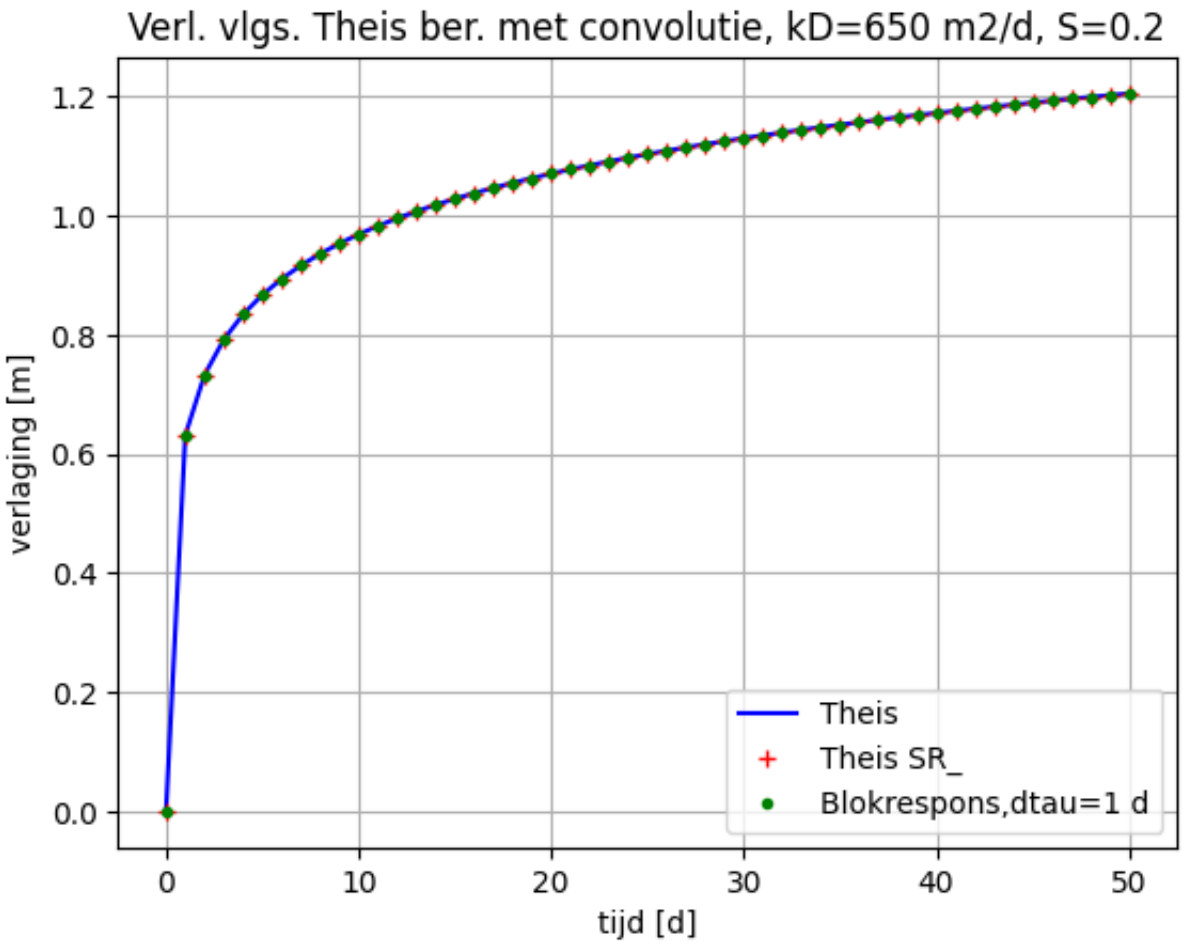
Het werken met de blokrespons is doorgaans het meest effectief. Het is hetzelfde als convolutie.

In [114…

```
kD, S, Q0 = 650, 0.2, 1200

# Piezometer
r = 10

# Tijd
tauEnd, dtau = 50., 1
tau = np.arange(0, tauEnd + dtau, dtau)

# Berekening verlaging Theis met convolutie
u = r ** 2 * S / (4 * kD * tau[1:])

SR = SRtheis(kD, S, r, tau)
BR = BRtheis(kD, S, r, tau)

Q = Q0 * np.ones_like(tau)

sBR = lfilter(BR, 1, Q)  # Convolutie

# Theis jgebruik makend van de formule (= SR, staprespons)
sTh = Q  * SR

title = f"Verl. vlgs. Theis ber. met convolutie, kD={kD:.0f} m2/d, S={S:.3g
ax = newfig(title, "tijd [d]", "verlaging [m]")

clrs = cycle('brgmck')
ax.plot(tau, sTh, color=next(clrs), label='Theis')
ax.plot(tau, sTh, '+', color=next(clrs), label='Theis SR_')
ax.plot(tau, sBR, '.', color=next(clrs), label=f'Blokrespons,dtau={dtau:.3g
ax.legend()
```

```
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/327931767
0.py:75: RuntimeWarning: divide by zero encountered in divide
  u = r ** 2 * S / (4  * T * t)
```

Out[114… <matplotlib.legend.Legend at 0x12b225fa0>



De berekening van de verlaging met convolutie middels de staprespons geeft dus hetzelfde resultaat als de analytische oplossing van Theis. Dit moet ook zo zijn.

Convolutie met de impulsrespons

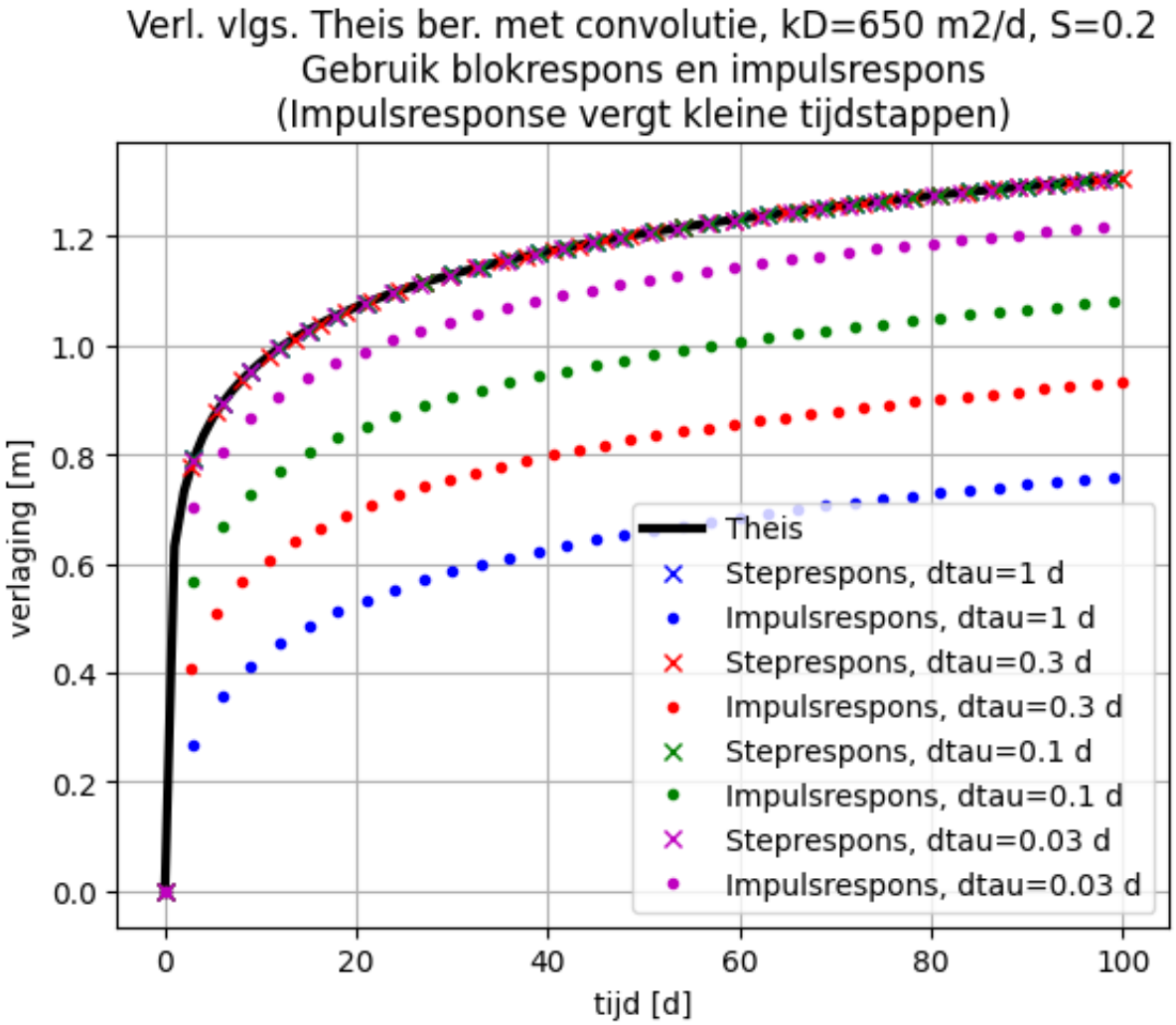
Hieronder wordt Theis opnieuw berekend maar nu met zowel de blokrespons als de impulsrepons. Voor grote tijdstappen blijkt de impulsrespons veel te kleine waarden op te leveren, pas bij tijdstappen kleinder dan 0.03 komen beide berekeningen goed met elkaar overeen.

In de praktijk gebruiken we altijd de blokrespons, want die is exact en niet afhankelijk van de grootte van de tijdstap.

In [142…

```
title = f"Verl. vlgs. Theis ber. met convolutie, kD={kD:.0f} m2/d, S={S:.3g
ax = newfig(title, "tijd [d]", "verlaging [m]")
dtaus = np.array([1, 0.3, 0.1, 0.03])
tmin, tmax = 0, 100

t= np.arange(tmin, tmax, 1.0)
sTheis = np.hstack((0, Q0 /(4 * np.pi * kD) * exp1( r ** 2 * S  / (4 * kD *
ax.plot(t, sTheis, lw=3, color='black', label='Theis')

clrs = cycle('brgmck')
for dtau in dtaus:
    clr = next(clrs)
    tau = np.arange(tmin, tmax, dtau)
    Q = Q0 * np.ones_like(tau)

    n = 3 * int(1 / dtau)

    SR = SRtheis(kD, S, r, tau)
    ax.plot(tau[::n], Q0 * SR[::n], 'x', color=clr, lw=0.25, label=f'Stepre

    # Gebruik de impulsrespons: Werkt alleen voor voldoend kleine stapgroot
    IR = IRtheis(kD, S, r, tau)
    sIR = lfilter(IR, 1, Q)

    ax.plot(tau[::n], sIR[::n], '.', color=clr, label=f'Impulsrespons, dtau

ax.legend()
plt.show()
```

```
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/345383505
2.py:75: RuntimeWarning: divide by zero encountered in divide
  u = r ** 2 * S / (4  * T * t)
```



We kunnen nu, met convolutie met de blokrespons nauwkeurig de verlaging bij elk willekeurig verloop van de onttrekking uittrekenen.

Enkele voorbeelden van convolutie met niet constante onttrekking

1. Willekeurig verloop van de onttrekking
2. Onttrekking neemt lineair af in de tijd, waarbij de onttrekking gehalveerd wordt over een tijdvak T
3. Onttrekking neemt exponentiaal af in de tijd, waarbij de onttrekking halveeft over een tijdvak T

In [ ]:

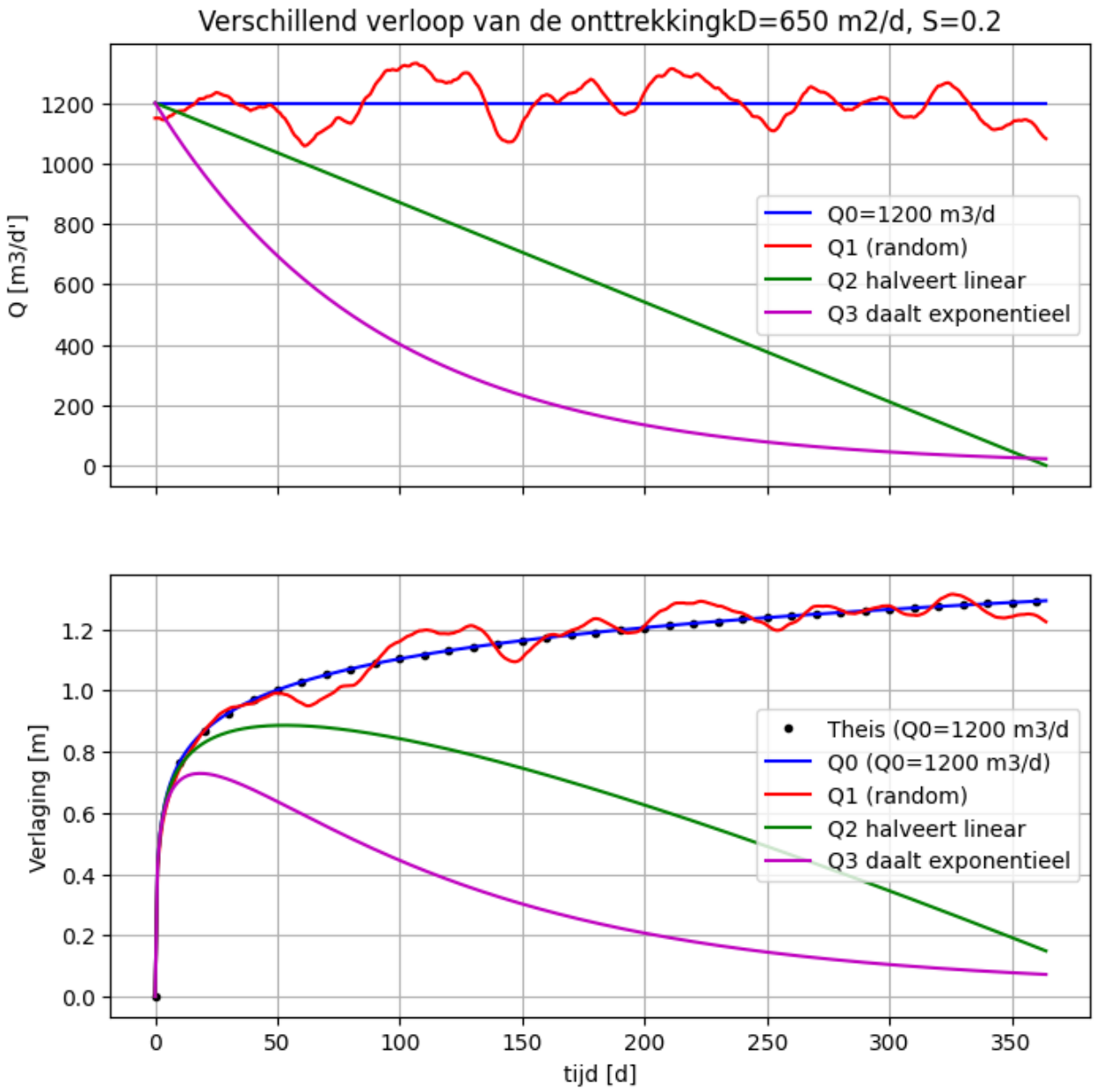
```
kD, S, Q0 = 650, 0.2, 1200

r = 20 # Piezometer

dtau = 1.0
tauEnd = 365
```

```
tau = np.arange(0, tauEnd, dtau)
T = tauEnd / 4
Q  = np.ones_like(tau) * Q0

# Random Q
Q1= Q0 * (np.random.rand(len(tau)) + 0.5)

# Voortschrijdend gemiddelde over 4% van de tijdstappen om een meer vloeie
B = np.ones(int(tauEnd / 25))

# Normaliseer, zodat de gemiddelde Q0 behouden blijft
B /= len(B)

# Vlak het random verloop van de onttrekking uit door
# voortschrijdend gemiddelde naar links en naar rechts te nemen
Q1 = filtfilt(B, 1, Q1, method='gust', irlen=len(B))

# Lineair in de tijd afnemend debiet
Q2 = np.interp(tau, np.array([0., tau[-1]]), np.array([Q0, 0]))

# In de tijd exponentieel afnemend debiet
Q3 = Q0 * np.exp(- tau / T)

u = r ** 2 * S  / (4 * kD * tau[1:])
SR = SRtheis(kD, S, r, tau)
BR = BRtheis(kD, S, r, tau)

sT = Q0 * SR
s0 = lfilter(BR, 1.0, Q)  # Constante onttrekking
s1 = lfilter(BR, 1.0, Q1) # Uitgevlakt willekeurig verlopende onttrekking
s2 = lfilter(BR, 1.0, Q2) # Lineair in de tijd afnemende onttrekking
s3 = lfilter(BR, 1.0, Q3) # Exponentieel in de tijd afnemende onttrekking

# Plots
fig, axs = plt.subplots(2, 1, sharex=True, figsize=(8, 8))
axs[0].set_title(f"Verschillend verloop van de onttrekkingkD={kD:.0f} m2/d,
axs[0].set_ylabel("Q [m3/d']")
axs[1].set_ylabel("Verlaging [m]")
axs[1].set_xlabel("tijd [d]")
axs[0].grid()
axs[1].grid()

clrs = cycle('brgmck')

axs[0].plot(tau,  Q, color=next(clrs), label='Q0={:.0f} m3/d'.format(Q0))
axs[0].plot(tau, Q1, color=next(clrs), label='Q1 (random)')
axs[0].plot(tau, Q2, color=next(clrs), label='Q2 halveert linear')
axs[0].plot(tau, Q3, color=next(clrs), label='Q3 daalt exponentieel')

clrs = cycle('brgmck')

axs[1].plot(tau[::10], sT[::10], 'k.', label='Theis (Q0={} m3/d'.format(Q0)
axs[1].plot(tau, s0, color=next(clrs), label='Q0 (Q0={:.0f} m3/d)'.format(Q
axs[1].plot(tau, s1, color=next(clrs), label='Q1 (random)')
axs[1].plot(tau, s2, color=next(clrs), label='Q2 halveert linear')
axs[1].plot(tau, s3, color=next(clrs), label='Q3 daalt exponentieel')
```

```
axs[0].legend()
axs[1].legend()
```

```
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/345383505
2.py:75: RuntimeWarning: divide by zero encountered in divide
  u = r ** 2 * S / (4  * T * t)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x12b650320>
```



Welke onttrekking is nodig om de gewenste verlaging op een gegeven afstand te handhaven?

Bemalingen hebben als doel een verlaging op de gewenste plek te bereiken om droog te kunnen werken en bouwen. Er zal normaliter een voldoende hoog debiet worden ingezet om in een beperkte tijd de vereiste verlaging te halen. Daarna zal het debiet worden verminderd om de verlaging te behouden tot aan het einde van het project. Zonder deze vermindering van het debiet zal de verlaging verder oplopen. Kunnen we het verloop van het debiet gedurende de gehele duur van de bemaling berekenen?

De analytische formules die ons ter beschikking staan vergen alle een bepaald onttrekkingsdebiet, waarvan de verlaging dan het gevolg is. Er zijn geen analytische formules die de verlaging opleggen en daarbij het in de tijd verlopende bemalingsdebiet uitrekenen. Omgekeerd, het is niet moeilijk om het tijdsverloop van een verlaging te berekenene voor de situatie waarin de onttrekking willekeurig in de tijd varieert; dit kan immers met behulp van convolutie. De omgekeerde situatie is gecomplicdeerder door het gebrek aan een formule die bij gegeven verlaging het bijbehorende verloop van het debiet in de tijd berekent. Om dit toch te brekenen is een iteratief proces een mogelijkheid.

Hieronder wordt een zeer eenvoudige procedure voorgefsteld en uitgwerkt om de gewenste verlaging op geggeven afsand te handhaven. De methode is niet exact maar wel een goede en praktische benadering.

De verlaging in de tijd in de situatie van Theis is

$$s(r,t) = \frac{Q_0}{4\pi kD} W(u), \quad u = \frac{r^2 S}{4kDt}$$

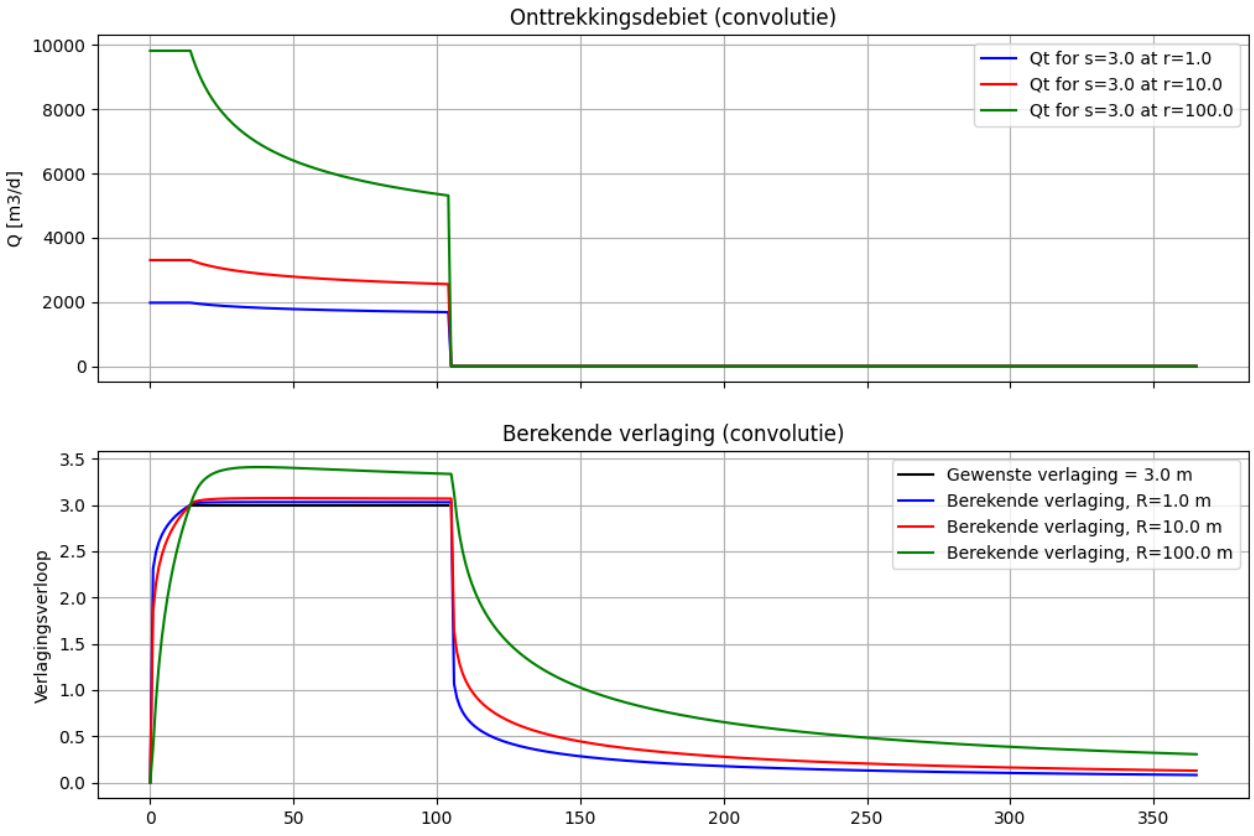
We kunnen dus de $Q(r,t)$ berekenen die de verlaging $s(r,t)$ de gewenste waarde geeft. Om de verlaging op de gewenste waarde te houden moet Q op een bepaalde wijze in de tijd variëren. Hoe we deze Q(r,t) precies kunnen berekenen weten we niet, maar we kunnen de constante Q die nodig is om de verlaging $s(r,t)$t$ en deze in de tijd variërende Q als de gezochte benadering beschouwen, die uiteraard steeds nauwkeuriger wordt naarmate de tijd verder voortschrijdt.

Dit wordt hieronder uitgewerkt en grafisch geverifieerd.

In [285…

```
# Parameters aquifer
kD, S = 600, 0.2

t = np.arange(366.0)
t[0] = 1e-6 # Voorkom deling door nul

# Gewense verlaging and afstand waarop deze moet gelden
s, r =3.0, 20.

# Periode  van gewenste verlaging
t1, t2 = 14, 14 + 90

fig, (ax1, ax2) = plt.subplots(2,1, sharex=True, figsize=(12, 8))
ax1.set_title("Onttrekkingsdebiet (convolutie)")
ax2.set_title("Berekende verlaging (convolutie)")
ax1.set_ylabel("Q [m3/d]")
ax2.set_ylabel("Verlagingsverloop")
ax1.grid(True)
ax2.grid(True)

ax2.plot([t1, t2], [s, s], '-', color='black', label=f'Gewenste verlaging =

rs = [1., 10., 100.]
clrs = cycle('brgmck')
for R in rs:
    # Berekening van Q(r,t)
    clr = next(clrs)

    u = R ** 2 * S / (4 * kD * t[1:]) # Skip t=0
    Q = np.hstack((0, s * (4 * np.pi * kD) / exp1(u)))

    # Initiële periode, opstarten bemaling tot gewenste verlaging
    Q[t <= t1] = Q[t <= t1][-1]

    # Periode nadat bemaling beëndigd is
    Q[t > t2] = 0.

    ax1.plot(t, Q, color=clr, label=f"Qt for s={s} at r={R}")

    # Convolutie: Staprespons en blokrespons
    SR = SRtheis(kD, S, R, t)
    BR = BRtheis(kD, S, R, t)

    # Exacte berekening van de verlaging bij de hierboven berekende onttrel
    st = lfilter(BR, 1, Q)

    ax2.plot(t, st, color=clr, label=f"Berekende verlaging, R={R} m")

ax1.legend()
ax2.legend()
```

Out[285… <matplotlib.legend.Legend at 0x12f2293a0>



Conclusie:

alleen wanneer de radius van de straal waarop de gewenste verlaging moet geleden groot wordt zijn de afwijkingen aanzienlijk. In de meeste gevallen is het verschil verwaarloosbaar voor de praktijk.

De aanloopperiode is die waarin de grond onder de te realiseren bouwput wordt leeggepompt met constant debiet om de bemaling in eerste instantie te realiseren. Dan volgt een periode waarin de verlaging wordt gehandhaafd, waarbij het debiet geleidelijk in de tijd daalt. De laatste tak van de grafiek is de periode na afloop van de bemaling.

Conclusie

We zien dat het berekende verlagingsverloop bij de berekende in de tijd verlopende onttrekking maar zeer gering afwijkt van de gewenste constante verlaging op de gegeven afsfand van het hart van de bouwput. We zien ook dat de onttrekking in de tijd moet afnemen om de verlaging op het gewenste niveau te handhaven. De afname blijkt met 25% af te nemen over de 90 dagen die hier zijn aangenomen als duur van de bemaling.

Analytische oplossingen voor verloop van onttrekking bij constante verlaging

Er bestaa1 een formule voor het debiet tussen twee ringen wanneer het peil (de verlaging) op de binnenste ring op $t = 0$ met een vaste waarde verandert (Zie Carslaw & Jaeger (1959)). Er is ook een rapport van de USGS (Bennet & Patten (1964)) waarin dit vraagstuk wordt behandeld en analytisch wordt opgelost. Echter de formule bevat zeer slecht convergerende besselfuncties is daardoor nauwelijks te integreren. Mijn conclusie is dat de hier voorgestelde benadering veel praktischer is en nauwkeurig genoeg. Onnauwkeurigheden door onvoldoende kennis van de bodemconstanten of door variërend debiet of minder goed afgesteld debiet of door neerslag tijdens de bemaling zullen groter zijn dan de fout die met de hiervoor uitgewerkte berekeningsmethode wordt gemaakt.

Bennett, GD, Patten EP (1964) Groudnwater hydraulics. Constant-head pumping test of a muliaquifer well to determine characteristics of individual aquifers.USGS Water-Supply Paper 1536-G, Washington 1962.

$$G(\beta) = \frac{4\beta}{\pi} \int_{x=0}^{\infty} x e^{-\beta x^2} \left[\frac{\pi}{2} + \arctan \frac{K_0(x)}{J_0(x)} \right] dx$$

Dit zou de onttrekking in de tijd moeten zijn bij constante verlaging op afstand r

$$Q(t) = 2\pi k D \, H \, G \left(\frac{kDt}{r^2 S} \right)$$

Ferris, JG, Knowles DB, Brrown RH and Stallman RW (1962) Theory of aquifer tests. US Geologial Survey Water Supply Paper 1536-E

Dit is een iets eerder document met in essentie dezelfde oplossing.

Op pagina 110 en volgende staat:

$$Q = 2\pi T s_w G(\alpha)$$

$$G(\alpha) = \frac{4\alpha}{\pi} \int_{0}^{\infty} x e^{-\alpha x^2} \left[\frac{\pi}{2} + \tan^{-1} \frac{Y_0(x)}{J_0(x)} \right] dx$$

$$\alpha = \frac{Tt}{r^2 S}$$

$$Q = 2\pi T s_w G(\alpha)$$

$$G(u) = \frac{1}{\pi u} \int_{0}^{\infty} x e^{-\frac{x^2}{4u}} \left[\frac{\pi}{2} + \tan^{-1} \frac{Y_0(x)}{J_0(x)} \right] dx$$

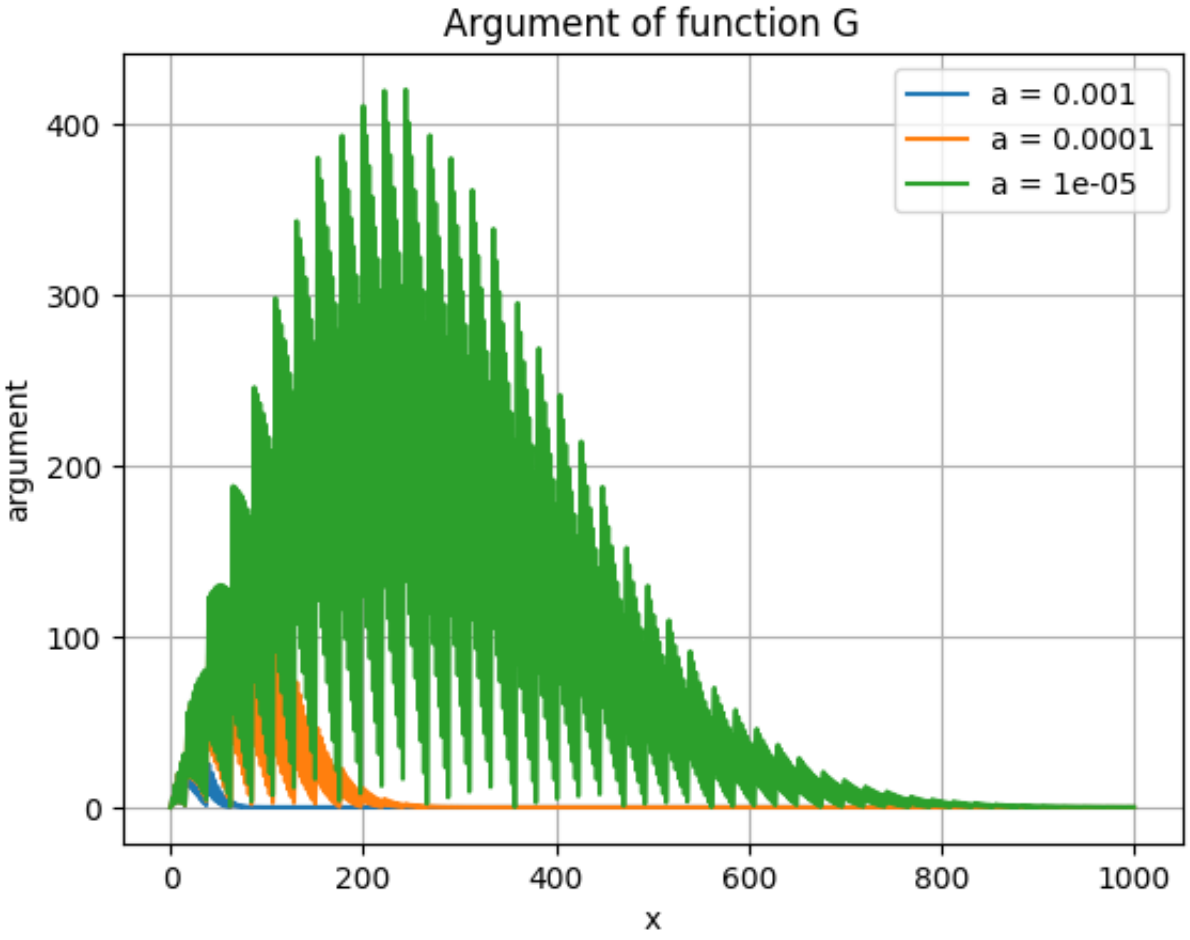
$$\alpha = \frac{4kDt}{r^2 S} = \frac{1}{4u}$$

De functies G blijken nauwelijks te integreren. Hieronder wordt nog het argument van de functie G getoond, waaruit het complexe gedrag van de functie blijkt.

Er wordt verder geen aandacht aan deze oplossing besteed.

In [19]:
```
def argG(a, x):
    return x * np.exp(-a * x ** 2) * (np.pi / 2 + np.arctan(Y0(x)/J0(x)))

x =np.linspace(0, 1000, 1000)
ax = newfig("Argument of function G", "x", "argument", yscale='linear')
for a in 10. ** -np.array([3, 4, 5]):
    ax.plot(x, argG(a, x), label=f"a = {a}")
ax.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x120a4fef0>
```



Bruggeman (1999) Solution 223.02. Flow outside a cylinder of radius R with given head change at t=0.

Bruggeman (1999) geeft ook een oplossing voor de stroming buiten een cilinder met radius R voor de situatie dat de stijghoogte op de rand van de cilinder op $t = 0$ plotseling verandert met een vaste waarde h:

$$\phi(r,t) = h \left\{ 1 - \frac{2}{\pi} \int_0^\infty \frac{1}{u} h(u,r) \exp\left(-\frac{u^2 t}{\beta^2 R^2} \right) du \right\}$$

terwijl

$$h(u,r) = \frac{J_0(u) Y_0(\frac{r}{R}u) - Y_0(u) J_0(\frac{r}{R}u)}{J_0^2(u) + Y_0^2(u)}$$

Dit beschrijft de verandering van de stijghoogte of verlaging als functie van r en t. We willen echter een relatie voor het debiet op afstand $r = R$.

$$Q|_{r=R} = -2\pi r T \frac{\partial \phi(r,t)}{\partial r}|_{r=R}$$

partial differentiëren naar r

$$\frac{\partial h(u,r)}{\partial r}|_{r=R} = \frac{u}{R} \frac{Y_0(u) J_1(u) - J_0(u) Y_1(u)}{J_0^2(u) + Y_0^2(u)} = \frac{u}{R} g(u)$$

met

$$g(u) = \frac{Y_0(u) J_1(u) - J_0(u) Y_1(u)}{J_0^2(u) + Y_0^2(u)}$$

zodat

$$Q_{r=R}(t) = -2\pi R T \; h \left\{ -\frac{2}{\pi} \int_0^\infty \frac{1}{u} \frac{u}{R} g(u) \exp\left(-\frac{u^2 t}{\beta^2 R^2} \right) du \right\}$$

zodat tenslotte

$$Q_{r=R}(t) = 4T \; h \int_0^\infty g(u) \exp\left(-\frac{u^2 t}{\beta^2 R^2} \right) du$$

Voorbeeld

In [287…

```
def fixedHeadQttimMl(T=600, S=0.2, D=20., rw=10, sr=-3, tmin=0, tmax=300):
    """Return ttim model with well of radius rw in which head is fixed for

    Parameters
    ----------
    T: float
        Transmissivity
    S: float
        storage coefficient
    D: float
        Thickness of the aquifer
    rw: float
        well radius
    sr: float
        fixed drawdown
    tmin, float
        minimum time
    tmax: float
        maximum time
    N: int
        number of time points
    """
    ml = ttim.ModelMaq(kaq=T / D, z=[0, -D], Saq= S / D, tmin=tmin, tmax=tm
    _ = ttim.HeadWell(ml, rw=r, tsandh=[(1.0, -3)], label='well')
    ml.solve()
    return ml
```

In [288…

```
T, S, D, R, sr, tmin, tmax = 600, 0.2, 20., 10., -3, 1., 300
beta2 = S / T

# Get the model
ml = fixedHeadQttimMl(T=T, S=S, D=D, rw=R, sr=sr, tmin=tmin, tmax=tmax)

# time for presentation
t =np.linspace(tmin, tmax, 5000)

# ttim computed discharge of this well
Qttim = np.round(ml.elementdict['well'].discharge(t)[0], 3)
httim = np.round(ml.head(R, 0, t)[0], 3)

valid = np.logical_not(np.logical_or(np.isnan(Qttim), np.isnan(httim)))
lastNaN = np.where(np.logical_not(valid))[0][-1] + 1
Qttim[:lastNaN] = 0.
httim[:lastNaN] = 0.
```

```
self.neq  1
solution complete
Warning, some of the times are smaller than tmin after
a change in boundary condition. nans are substituted
Warning, some of the times are smaller than tmin after
a change in boundary condition. nans are substituted
```

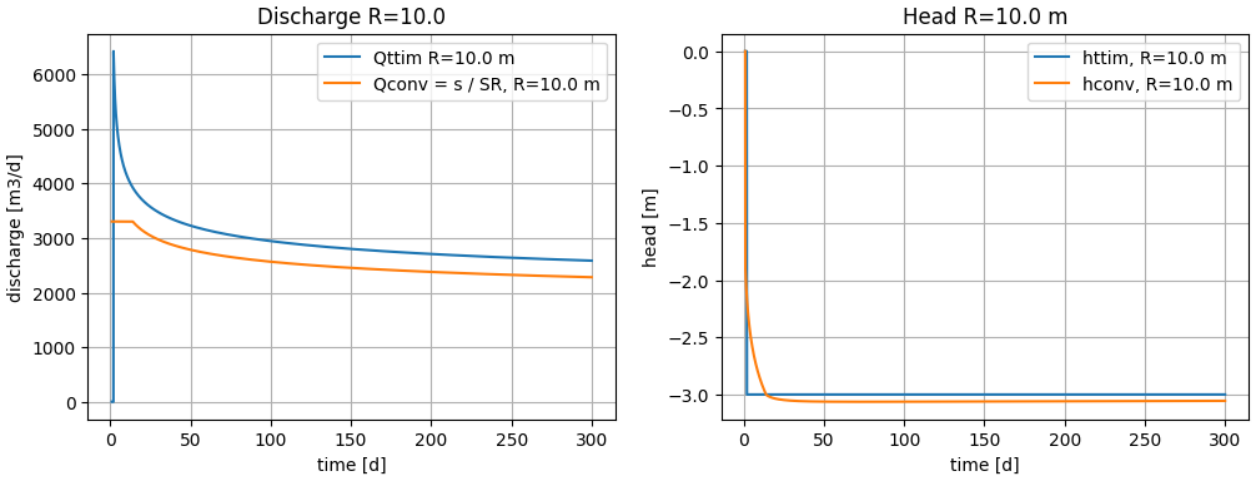
In [289…

```
# Using convolution with the Q from the Theis equation

SR = SRtheis(T, S, R, t)
BR = BRtheis(T, S, R, t)

t1 = 14
Qconv = sr / SR
Qconv[t <= t1] = Qconv[t <= t1][-1]
hconv = lfilter(BR, 1, Qconv)

hconv

fig, (ax2, ax1) = plt.subplots(1, 2, sharex=True, figsize=(12, 4))
ax1.set_title(f"Head R={R} m")
ax2.set_title(f"Discharge R={R}")
ax1.set_xlabel('time [d]')
ax2.set_xlabel('time [d]')
ax1.set_ylabel('head [m]')
ax2.set_ylabel('discharge [m3/d]')
ax1.grid(True)
ax2.grid(True)
ax1.plot(t, httim, label=f'httim, R={R} m')
ax1.plot(t, hconv, label=f'hconv, R={R} m')
ax2.plot(t, Qttim, label=f"Qttim R={R} m")
ax2.plot(t, -Qconv, label=f'Qconv = s / SR, R={R} m')
ax1.legend()
ax2.legend()
```

```
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/163001744
8.py:7: RuntimeWarning: divide by zero encountered in divide
  Qconv = sr / SR
```

Out[289…

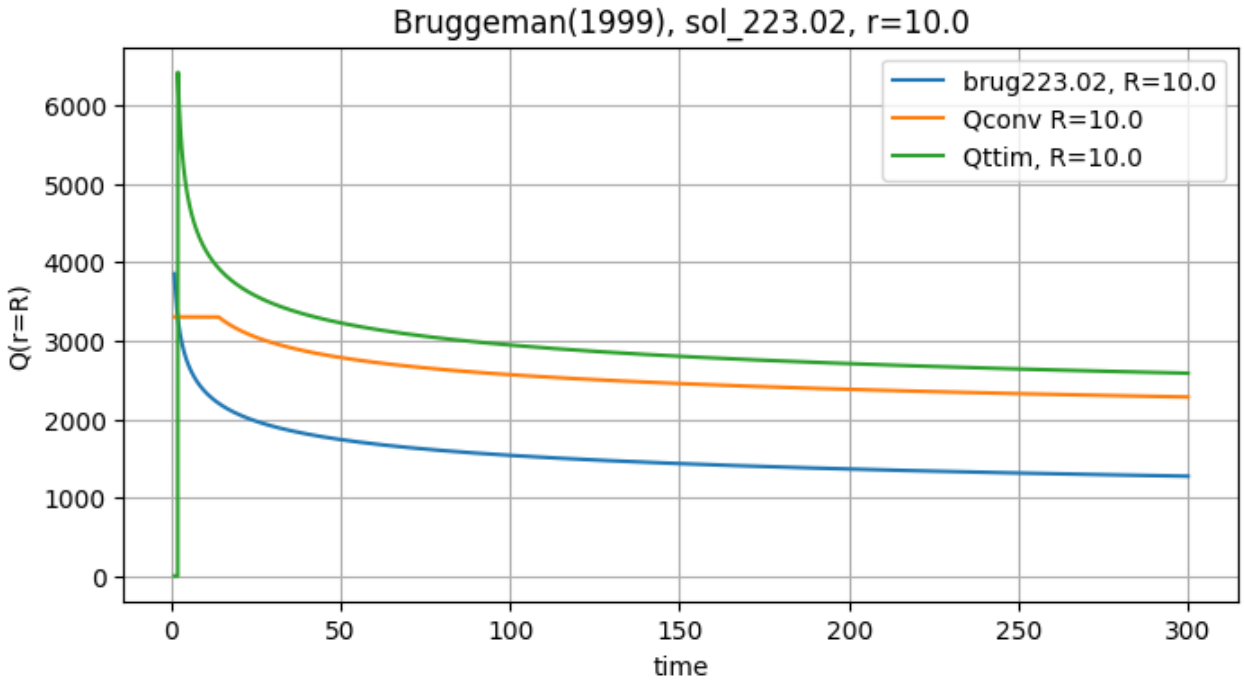
```
<matplotlib.legend.Legend at 0x12f346870>
```



In [292…

```
# Functies
def brug223(T, S, R, h, t):
    """Return Q of solution Bruggeman(1999) 223.02"""
    def g(u):
        return (Y0(u) * J1(u) - J0(u) * Y1(u)) / (J0(u) ** 2 + Y0(u) ** 2)

    def fexp(u, t, R, beta2):
        return np.exp(-u ** 2 * t / (beta2 * R ** 2))

    u = np.logspace(-5, 0, 1000)

    F = g(u) * fexp(u, t, R, S / T)
    Integral = np.sum(0.5 * (F[:-1] + F[1:]) * np.diff(u))
    return 4 * T * h * Integral

# Tonen onderdelen van de oplossing
fig, ax = plt.subplots(figsize=(8, 4))

ax.set_title("Bruggeman(1999) (solution 223), flow Q")
ax.grid(True)

ax.set_title(f'Bruggeman(1999), sol_223.02, r={R}')
ax.set_xlabel('time')
ax.set_ylabel('Q(r=R)')
ax.grid(True)

# Bereken de oplossing (numeriek integreren Simpson regel)
Qbrug = []
for t_ in t:
    Qbrug.append(brug223(T, S, R, h, t_))
Qbrug = np.array(Qbrug)

# Ter vergfelijking eerder uitgewerkte eenvoudige oplossing
ax.plot(t,  Qbrug, label=f'brug223.02, R={R}')
ax.plot(t, -Qconv, label=f"Qconv R={R}")
ax.plot(t,  Qttim, label=f"Qttim, R={R}")
ax.legend()
```

Out[292… <matplotlib.legend.Legend at 0x12f1eeb70>



Hier verschilt de exacte oplossing 223.02 volgens Bruggeman(1999) sterk van de hiervoor uitgewerkte eenvoudige oplossing. Beide moeten zekerheidshalve worden vergeleken met de numerieke oplossing volgens ttim.

Gebruik van TTIM

TTIM is een open source API voor het analytische modelleren van grondwaterstroming in meerdere lagen. Uitgangspunt is een initiële situaite met de stijghoogten overal gelijk aan nul, die zich aanpast onder invloed van stresses die met analytische elementen worden opgelegd. Zulke analytische elementen zijn putten, "line sinks", "line doublets" en een "circular area sink", een cirkelvorming element met opgelegd neerslagoverschot (voeding). De line sinks kunnen aan elkaar worden geregen tot complexe sloot- beek en rivierconfiguraties of lijnonttrekkingen, met gegeven onttrekking per lengteeenheid of met geegven stijghoogte die al dan niet via een weerstand het grondwater in de aquifers beïnvloedt. De line doublets kunnen worden gebruikt om weerstand tegen horizontale stroming aan te brengen, zoals damwanden en gesloten randen, waarbij het grondwater aan een zijde van de dipool al dan niet via een weerstand communiceert met die aan de andere zijde. Lek tussen aquifers is automatisch besloten in de oplossing omdat het meerlagensysteem tussen elk van de lagen een weerstand krijgt opgelegd.

Voor de specifieke onttreking σ_i [L^3/T/L] = [L^2/T] uit ene line-sink in laag i geldt

$$\sigma_i = w(h_i - h_{ls})/c$$

met w [L] de breedte van de line-sink, en c [T] de weerstand ervan.

De stroming door (loodrecht op) een doublet is

$$q_n = (h^- - h^+)/c$$

Voor putten geldt dat de onttrekking uit laag i, Q_i gelijk is aan

$$Q_i = 2\pi r_w H_i (h_i - h_w)/c$$

met r_w de radius van de put, c de intredeweerstand van de put, h_w de stijghoogte in de put en h_i die in laag i aan de buitenzijde van de put.

Het element van type `Well` heeft een opgelegd debiet die van type `HeadWell` een opgelegde stijghoogt in de put. De onttrekking per laag wordt berekend op basis van het doorlaatvermogen van de lagen en de stijghoogte ter hoogte van elke laag.

Wat moeilijker verloopt is het ruimtelijk aanpassen van het doorlaatvermogen of de berginngscoëfficiënten van de watervoerende lagen of van de weerstand tussen de lagen. Theoretisch is dit wel mogelijk, het programma MLAEM had deze mogelijkheid bijvoorbeeld 20 jaar geleden al, maar TTIM heeft dat nog niet. TTIM is mathematisch wellicht complexer dan het oudere MLAEM omdat TTIM direct analytische meerlagensystemen implementeert, terwijl MLAEM deze systemen element per element opbouwt, en daarmee veel complexer is wat betreft de invoer.

De mogelijkheden van ttim zijn dus enigszins beperkt. Tegelijkertijd is het een zeer krachtig greedschap voor veel situaties.

Put met constant debiet (Vergelijk Theis met ttim uitkomst (example 1 in Readthedocs ttim))

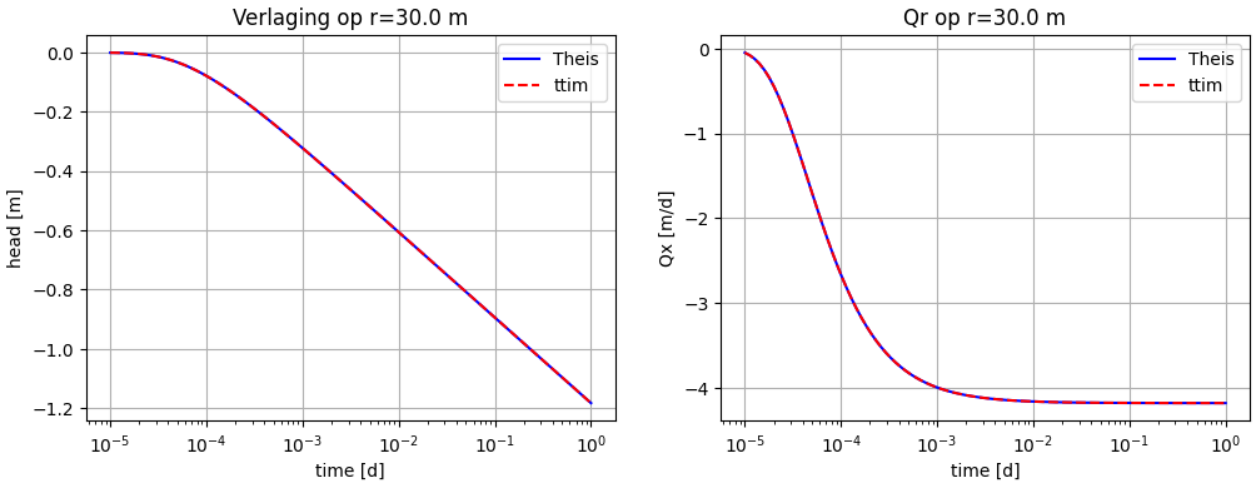
In [21]:
```
def theis(r, t, T, S, Q=-1200.0):
    """Return head change due to Theis"""
    u = r ** 2 * S / (4 * T * t)
    h = -Q / (4 * np.pi * T) * exp1(u)
    return h

def theisQr(r, t, T, S, Q=-1200.):
    """Return specifiec discharge change due to Theis m2/d]"""
    u = r ** 2 * S / (4 * T * t)
    qr = -Q / (2 * np.pi) * np.exp(-u) / r
    return qr
```

In [22]:
```
T, S, t, r, Q = 500., 1e-4, np.logspace(-5., 0., 100), 30., 788.

ml = ttim.ModelMaq(kaq=25., z=[0, -20], Saq= S/ 20., tmin=1e-5, tmax=1)
w = ttim.Well(ml, tsandQ=[(0, Q)], rw=1e-2)
ml.solve()
h=ml.head(r, 0, t)
Qx, Qy = ml.disvec(r, 0, t)
```

```
self.neq  1
solution complete
```

In [23]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, figsize=(12, 4))

ax1.set_title(f"Verlaging op r={r} m")
ax1.set_xlabel('time [d]')
ax1.set_ylabel('head [m]')
ax1.grid(True)
ax1.set_xscale('log')

ax1.plot(t, theis(r, t, T, S, Q), 'b', label='Theis')
ax1.plot(t, h[0], "r--", label='ttim')
ax1.legend()

ax2.set_title(f"Qr op r={r} m")
ax2.set_xlabel("time [d]")
ax2.set_ylabel('Qx [m/d]')

ax2.plot(t, theisQr(r, t, T, S, Q), 'b', label='Theis')
ax2.plot(t, Qx[0], 'r--', label='ttim')
ax2.grid(True)
ax2.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x1216e7560>



Voorbeeld 2

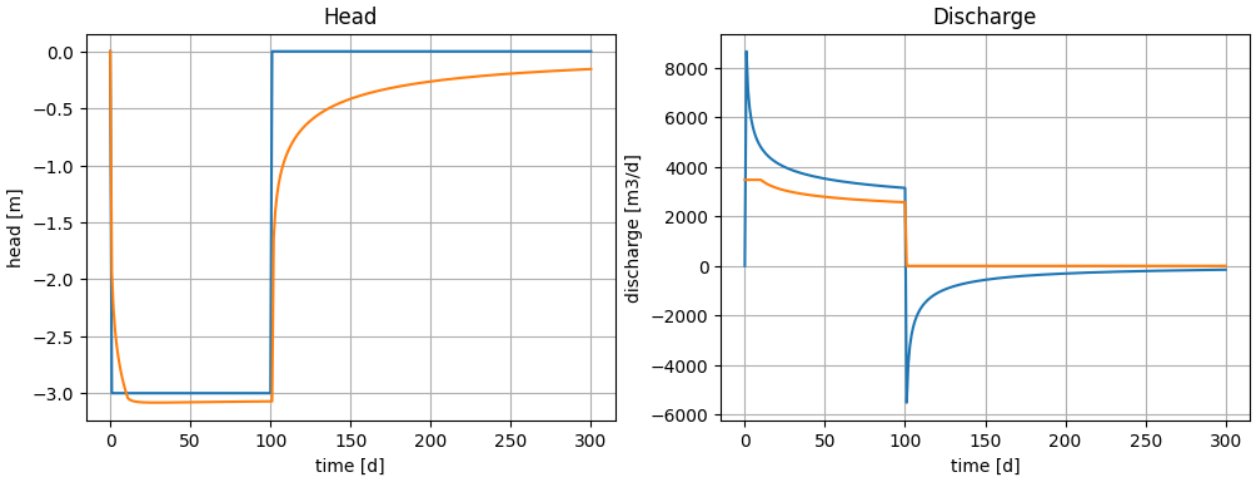
Put met readius r met gegeven verlaging op deze radius

In [24]:

```
T, S, r, hr = 600, 0.2, 10.0, -3.0

tmin, tmax = 0, 300.
t =np.linspace(tmin, tmax, int(tmax - tmin + 1))
SR = np.hstack((0, theis(r, t[1:], T, S, Q=1.0)))
BR = np.hstack((0, SR[1:] -  SR[:-1]))

s = -3.0
Q = s / SR
Q[:10] = Q[10]
Q[t > 100] = 0.

ml = ttim.ModelMaq(kaq=25., z=[0, -20], Saq= S, tmin=1.0, tmax=tmax)
w = ttim.HeadWell(ml, rw=r, tsandh=[(0, -3), (100, 0)])
ml.solve()

fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True, figsize=(12, 4))
ax1.set_title("Head")
ax2.set_title("Discharge")
ax1.set_xlabel('time [d]')
ax2.set_xlabel('time [d]')
ax1.set_ylabel('head [m]')
ax2.set_ylabel('discharge [m3/d]')
ax1.grid(True)
ax2.grid(True)
ax1.plot(t, np.round(ml.head(r, 0, t)[0], 3), label=f'head at r={r} m')
ax1.plot(t, lfilter(BR, 1, Q), label="convolutie met Q")
ax2.plot(t, w.discharge(t)[0], label="discharge")
ax2.plot(t, Q, label='Q = s / SR')
```

```
self.neq  1
solution complete
/var/folders/90/m51x_b713y561gzh2kzy18d00000gq/T/ipykernel_89637/184339465
3.py:9: RuntimeWarning: divide by zero encountered in divide
  Q = s / SR
```

Out[24]: `[<matplotlib.lines.Line2D at 0x121aad4c0>]`



TTIM omgaan met gebiedsweerstand of slotenpatroon

Het beken- en slotenpatroon in een gebied is een uiting van het grondwatersysteem. Een hooggelegen gebied is normaliter een infiltratiegebied met weinig open waterlopen. Het omgekeerde geldt voor en relatie laag gelegen gebied, dat normaliter wordt gekenmerkt door kwel met veel drainagemiddelen. En dan zijn er overgangsgebieden die tussen beide in liggen. Een hoog gelegen gebied heeft hierdoor een hoge drainageweerstand en een laag gelegen gebied een kleine, terwijl die in een overgangsgebied ruimtelijk verloopt. De drainageweerstand is de gemiddlde grondwaterstand h [L] ten opzichte van het peil in het oppervlaktwater h_{ow} gedeeld door de gemiddelde grondwateraanvulling N [L/T]

$$c_{dr} = \frac{\bar{h} - h_{ow}}{N}$$

of

$$N = \frac{\bar{h} - h_{ow}}{c_{dr}}$$

De drainageweerstand kan uitgedrukt worden in slootbreedte w, de slootafstand L en slootbodemweerstand c_{sl} [L/T]

$$c_{sl} = \frac{w}{L} c_{dr}$$

of

$$c_{dr} = \frac{L}{w} c_{sl}$$

In ttim kunnen we de gewenste gebieds of drainageweerstand krijgen door de slootafstand te kiezen. Nemen we de slootbodemweerstand gelijk aan $c_{sl} = 1$ d en de slootbreedte $w = 1$ m, dan volgt $c_{dr} = L$. Let op dat de dimensie van L nu dagen is geworden, wat verwarrend kan zijn. Beter is het om w en c_{sl} eenvoudig in de formules te laten staan, dan is verwarring onmogelijk:

$$c_{dr} = L \frac{c_{sl}}{w}$$

Door parallelle sloten in het model op te nemen met een gegeven breedte en weerstand is het mogelijk om een gebiedsweerstand te simuleren. Men kan eenvoudig de slootbreedte w gelijk aan 1 m nemen en de sloot weerstand gelijk aan $L\gamma$ om de gewenste gebiedsweerstand in het model te creëren en deze indien gewenst ruimtelijke te variëren.

Uiteraard kunnen waterlopen ook worden ingelezen uit een geografisch informatie systeem of een andere database en worden omgezet in de analytische elementen waar ttim mee werkt. De drainageweerstand is daar dan de resultante van. Bij een grotere dichtheid aan open waterlopen is het veelal gemakkelijker om met de wat abstracte drainageweerstand te werken. Het slotenpatroon kan dan in een model worden vervangen door een verticale gebiedsweerstand. Omgekeerd kan een verticale gebiedsweerstand wroden gesimuleerd in ttim door het plaatsen van een aantal lijnelementen op beperkte onderlinge afstand, zoals hiervoor is uitgelegd. Hiermee kunnen situaties worden geschematiseerd in ttim waarin het ene gebiedstype geleidelijk of abrupt overgaat in het andere, iets dat veelvuldig voorkomt in de praktijk.

In het voorbeeld hierna wordt een dergelijke, ruitmelijk variërende drainageweerstand gesimuleerd door middel van een aantal parallelle sloten. We gebruiken hiervoor het analytische element `HeadLineSinkString` , dat bestaat uit een aantal aan elkaar geregen `HeadLineSink` s, LineSinks dus met opgelegde stijghoogte die een bodemweerstand en een breedte hebben.

Voorbeeld

Stel de drainageweerstand ten oosten van de put, in een hydrologisch overgangsgebied verloopt van 1000 d af naar 100 d. We kunnen dit modelleren met een aantal evenwijdige `HeadLineSinkStrings` (sloten) in N-Z richting ten oosten van de put, met een weerstand die van sloot tot sloot verschilt. We leggen in dit voorbeeld de sloten op 200 m uit elkaar, dus $L = 200$ m en passen de slootweerstand $c_{sl} = L/w$ aan zodat de gewenste in de ruimte verlopende drainageweerstand wordt gesimuleerd. We kiezen gemakshalve $w = 1$ m.

In [25]:

```
# Drainageweerstand ter plaatse van de 5 sloten
cdr = np.logspace(3, 1, 5) # drainageweerstand verloopt hoge naar lage waa
cdr = 1000. * np.ones_like(cdr) # drainageweerstand gebied met weinig slot
cdr = 100. * np.ones_like(cdr) # drainageweerstand gebied met veel sloten
cdr = 10. * np.ones_like(cdr)  # uiterst kleine drainageweerstand

# Ligging van de put
xw, yw = 0., 0.

# Slootafstand
L = 200

# Ligging van de 5 sloten
xsl = np.arange(200., 1000. + 1, 200.)

# Slootbreedte
w = 1.0

# Slootbodem weerstand
csl = w / L * cdr

# y-coordinaten van de 20 slootstukken die samen een sloot vormen
# De lijnstukken mogen niet te lang zijn, willen we een vlak verloop langs
y = np.linspace(-1000, 1000, 21)

# Definieer de sloten
sloten = {}
for i, (x, c) in enumerate(zip(xsl, csl)):
    name = f'sloot{i}'
    xy = np.vstack((x * np.ones_like(y), y)).T
    sloten[name] = {'xy': xy, 'res': c, 'wh': w}
```

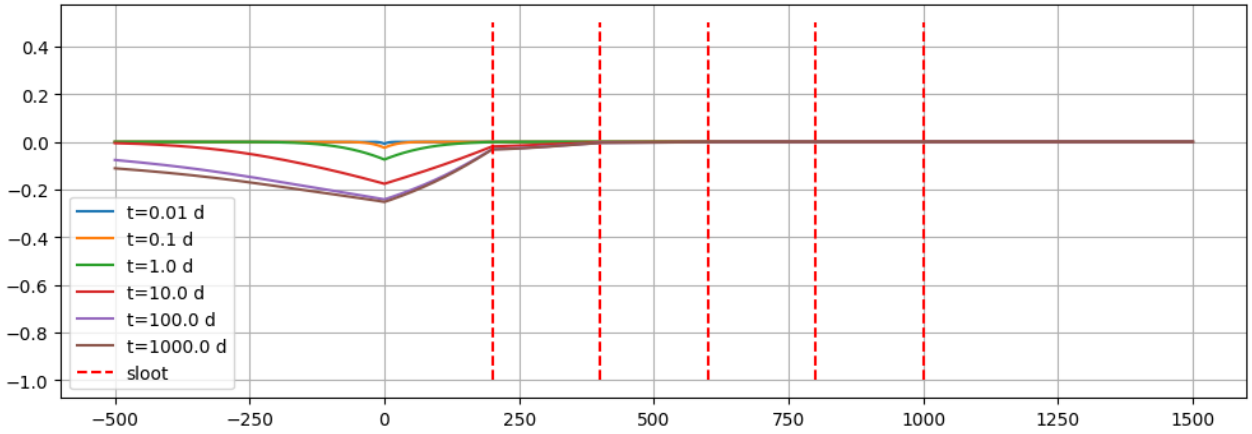
ttim model

In [26]:
```
# Aquifer parameters
T, S, t, Q = 500., 0.2, np.logspace(-5., 0., 100), 788.
rw = 0.2

# Instantiatie van het ttim model
ml = ttim.ModelMaq(kaq=T / 20., z=[0, -20], Saq= S / 20., tmin=0.001, tmax=

# Voeg well element toe
#well = ttim.Well(ml, xw=0., yw=0., tsandQ=[(0, Q)], rw=rw, layers=0, label

# Voog LineSink toe
XL = np.linspace(-200, 200, 9)
YL = np.linspace(-200, 200, 9)
xy = np.vstack((XL, YL)).T

lsink = ttim.LineSinkDitchString(ml, xy=xy, tsandQ=[(0, Q)], res=0., wh=1.(

# Voeg de "sloten" toe
for name, sl in sloten.items():
    # Alle in laag 0, met gegeven stijghoogte 0 vanaf t=0
    sl['sloot'] = ttim.HeadLineSinkString(ml, xy=sl['xy'], tsandh=(0, 0), l

# Bepaal de oplossing van het internse stelsel vergelijkingen
ml.solve()
```

```
self.neq  108
solution complete
```

Stijghoogte langs een doorsnede, in dit geval van west naar oost door de put.

In [27]:
```
times = np.array([0.01, 0.1, 1., 10., 100., 1000.])

fig, ax = plt.subplots(figsize=(12, 4))
ax.grid()

# Kies punten langs de hoirzontale lijn
xL = np.linspace(-500, 1500, 501)

# Bijbehorende y-coordinaten
yL = np.zeros_like(x)

# Bereken de stijghoogte langs de lijn voor alle punten en tijden
h = ml.headalongline(xL, yL, times, layers=0)

labels = [f't={t} d' for t in times]
ax.plot(xL, h[0].T, label=labels)

ax.vlines(xsl, -1., 0.5, colors='r', linestyles='--', label='sloot')

ax.legend()

plt.show()
```



Toon resultaten in contourplot

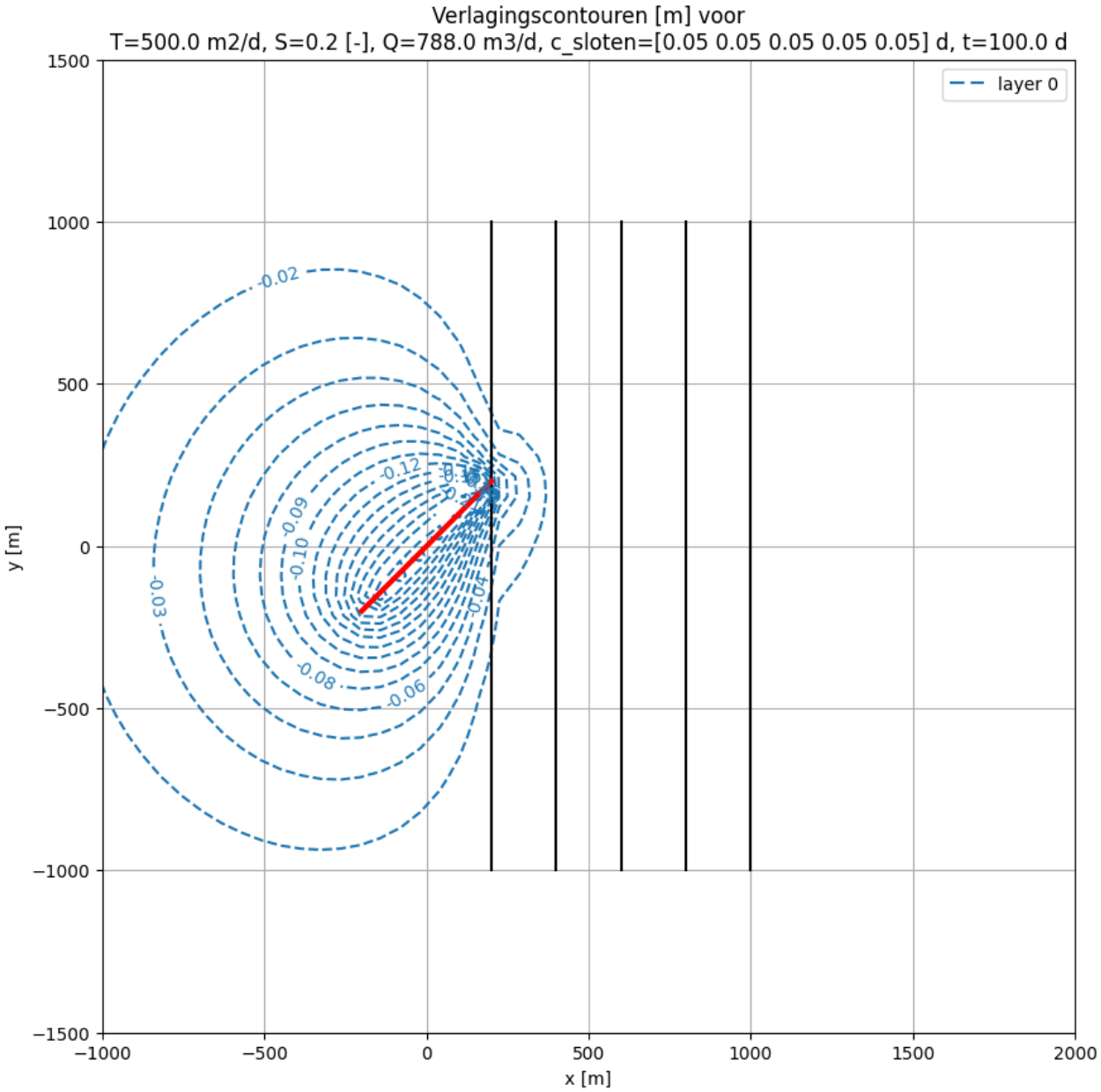
Het berekenen van de punten voor de contourplot kost veel tijd. In dit geval ca. 1m13.5 s. Dus vergt wat geduld.

De weerstand van de sloten neemt van west naar oost af om het overgangsgebied tussen het hoge gebied links en het lage gebied rechts te modelleren.

In [28]:

```
# Window voor contour plot
win = (-1000, 2000, -1500, 1500)


# Contouring, layout=True impliceert dat de elementen ook worden getekend.
t = 100.
ml.contour(win=win, ngr=50, t=100, layers=0, levels=20, layout=True, decima

ax = plt.gca()
ax.set_title(f"Verlagingscontouren [m] voor\nT={T} m2/d, S={S} [-], Q={Q} m
ax.set_xlabel('x [m]')
ax.set_ylabel('y [m]')
ax.grid()
ax.plot(XL, YL, 'r', lw=3, label='lsink')
```

Out[28]: [<matplotlib.lines.Line2D at 0x121853d40>]



TTIM: Same but packed in a few funtions for more easily computing scenarios

The following few cells do the same same as above, but now uses some functions to more easily compute scenarios.

In [29]:

```
def add_dtiches(ml, x1=200., L=200., n=5., csl=None, w=1):
    """Return ditches as a dict.

    The diches are transformed into `ttim.HeadLineSinkString` and added tot

    Parameters
    ----------
    ml:
        ttim model
```

```
    x1: float
        x of left most ditch
    L: float
        distance between ditches
    n: int
        number of ditches
    csl: float or list
        drainage resistance at each of the ditches
    w: float
        ditch width

    """
    xsl = np.arange(x1, x1 + n * L, L)

    if np.isscalar(csl):
        csl = csl * np.ones_like(xsl)

    assert csl is not None, f"cdr must be list of drainage resistances of l
    assert len(csl) == len(xsl), f"cdr must be a list of len(xsl) = {len(xs

    y = np.linspace(-1000, 1000, 21) # ycoordinaten van de slootstukken die

    # Define the ditches / canals
    ditches = {}
    for i, (x, c) in enumerate(zip(xsl, csl)):
        name = f'ditch{i}'
        xy = np.vstack((x * np.ones_like(y), y)).T
        ditches[name] = {
            'name': name,
            'xy': xy,
            'res': csl,
            'wh': w,
            # Define ttim.HeadLineSinkString and add to model
            'dtich': ttim.HeadLineSinkString(ml,
                xy=xy,
                tsandh=(0, 0),
                res=c,
                wh=w,
                layers=0,
                label=name)
        }
    return ditches


def add_lsinkditch(ml, XL=[-100., 100.], YL=[-100., 100.], Q=788, csl=0, w=
    """Return linesink as dict.

    Parameters
    ----------
    XL: list or np.array
        x-coordinafes of exracting canal (LineSinkDitchString element)
    YL: list of floats or np.array
        y-coordinates of the same
    Q: float
        total extraction from t=0
    c: float
        bottom resistance of ditches
```

```
    w: float
        bottom width
    """
    name = 'lsink'
    xy = np.vstack((XL, YL)).T
    lsink = {'name': name,
             'XL': XL,
             'YL': YL,
             'Q': Q,
             'res': csl,
             'wh': w,
             'lsink': ttim.LineSinkDitchString(ml,
                   xy=xy, tsandQ=[(0, Q)], res=c, wh=w, layers=0, label=na
    }
    return lsink


def plot_contours(ml, win=(-1000, 2000, -1500, 1500), t=100, lsink=None):
    """Return contour plot with default number of levels of layer 0, with l

    Parameters
    ----------
    ml: ttim model object
        the model
    win: tuple of 4 floats
        x1, x2, y1, y2 of computation window
    t: float
        time of the contours plotted
    lsink: dict
        dictionary with the ditch properties for plotting in different colo
    """

    ml.contour(win=win, ngr=50, t=t, layers=0, levels=20, layout=True, deci
    return plt.gca()
```

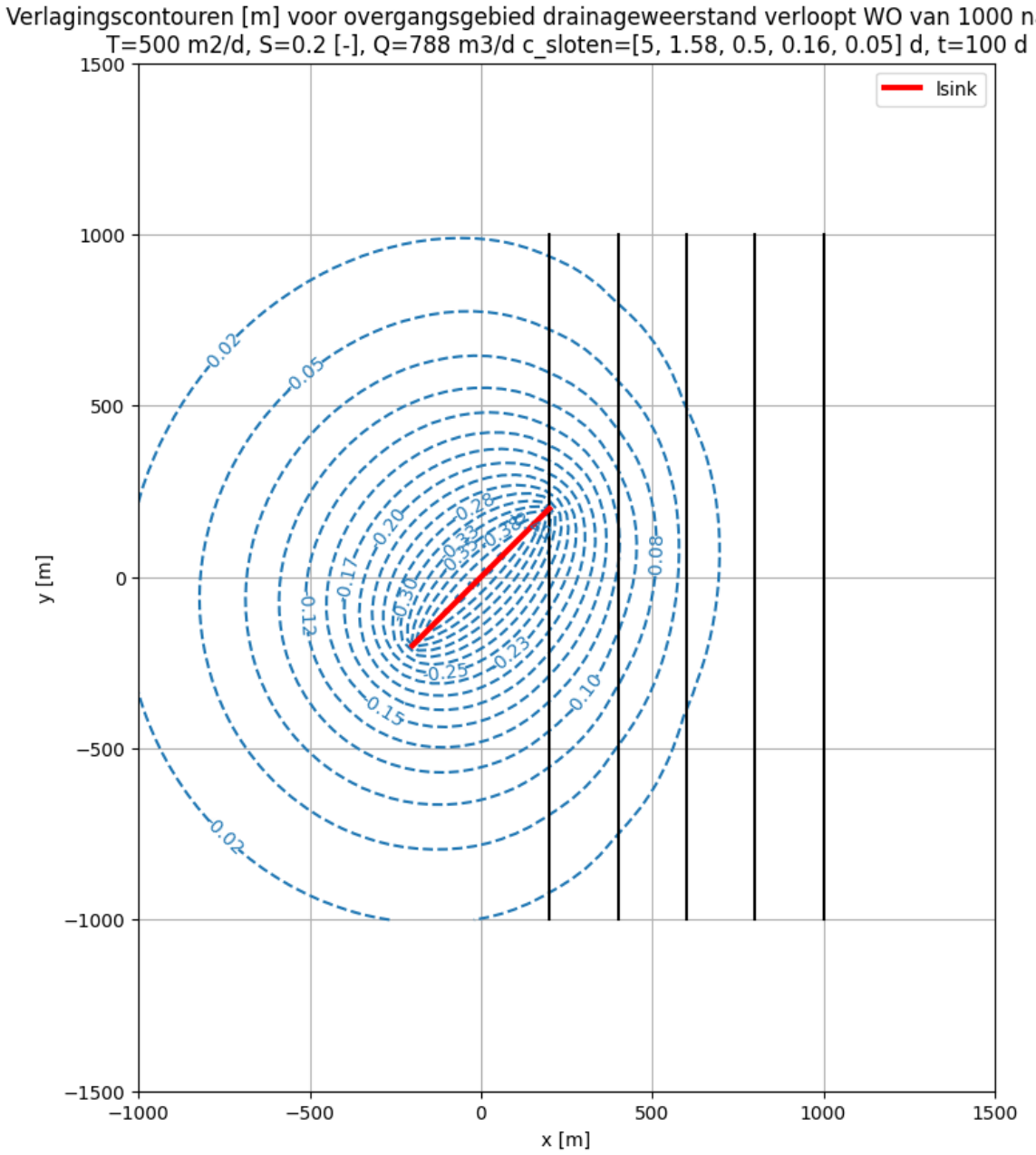
The actual drainage scenario choice and contour plotting

In [30]:

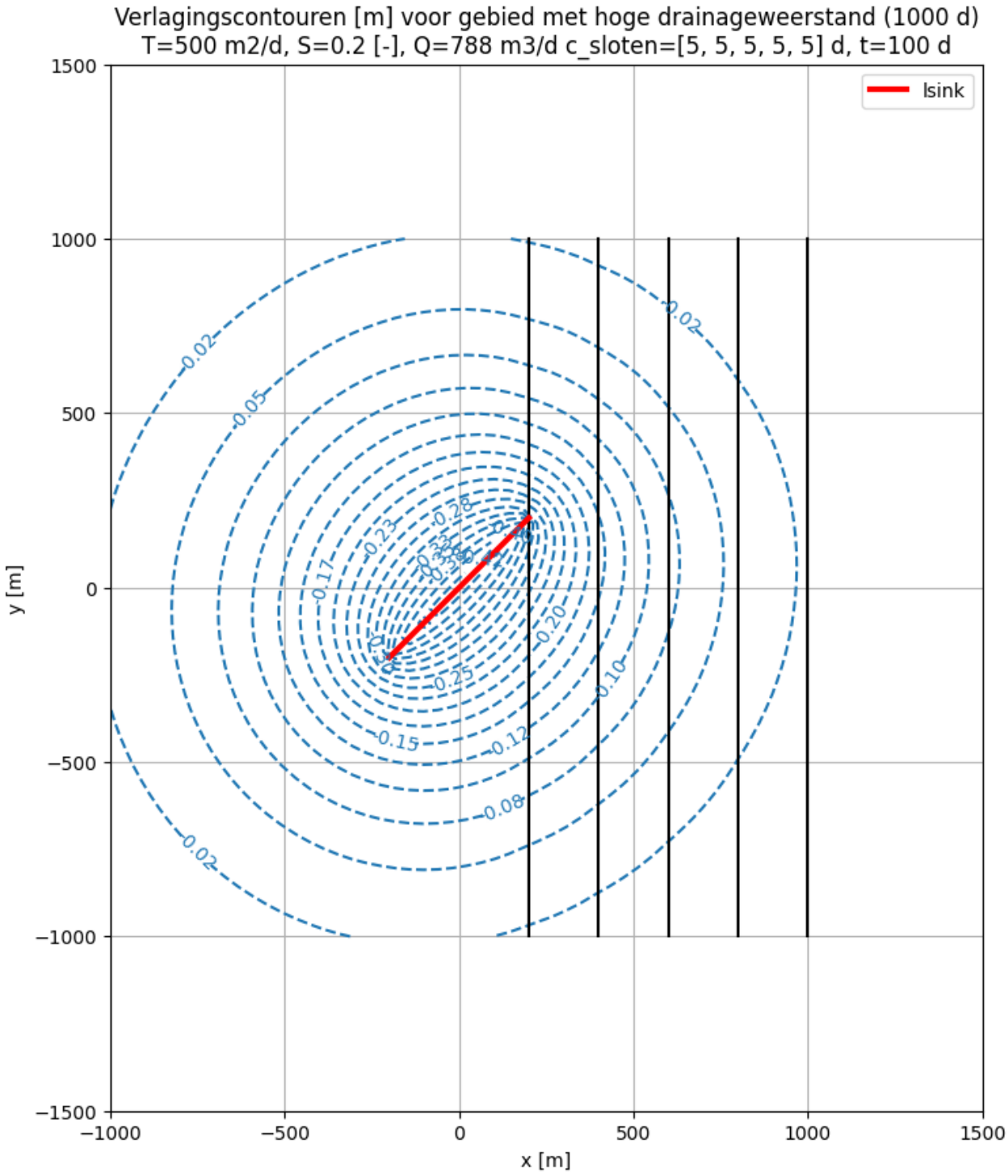
```
# Aquifer parameters
T, S, t, Q = 500., 0.2, np.logspace(-5., 0., 100), 788.
rw = 0.2
L, w = 200., 1.0

# Drainageweerstand ter plaatse van de 5 sloten
cdr_area = np.logspace(3, 1, 5) # drainageweerstand verloopt hoge naar lage

cdrs = {'overgangsgebied': {'cdr': cdr_area, 'area_type': 'overgangsgebied
        'hoog':      {'cdr': 1000. * np.ones_like(cdr_area), 'area_type':
        'laag':      {'cdr':  100. * np.ones_like(cdr_area), 'area_type':
        'zeer laag': {'cdr':   10. * np.ones_like(cdr_area), 'area_type':
}

for k, cdr in cdrs.items():
    # Instantiatie van het ttim model
    print(f"Gebieds type: {cdr['area_type']}.")
    ml = ttim.ModelMaq(kaq=T / 20., z=[0, -20], Saq= S / 20., tmin=0.001, t

    csl = np.round(cdr['cdr'] * w / L, 2)

    # Ditches simulating areal drainage resistance
    ditches = add_dtiches(ml, x1=200., L=L, n=5., csl=csl, w=w)

    # Extracting canal coordinates
    XL = np.linspace(-200, 200, 9)
    YL = np.linspace(-200, 200, 9)
    lsink = add_lsinkditch(ml, XL=XL, YL=YL, Q=Q)

    # Contour time
    t = 100.

    ml.solve()

    print("Computing the points for contouring takes a lot of time (order 1
    computation_window = (-1000, 1000, -1000, 1000)
    ax = plot_contours(ml, win=computation_window, t=t, lsink=lsink)

    # Extras for the plot
    ax.set_title(f"Verlagingscontouren [m] voor {cdr['area_type']}\n" +
                 f"T={T:.0f} m2/d, S={S} [-], Q={Q:.0f} m3/d" +
                f" c_sloten=[{', '.join([f'{c:.3g}' for c in csl])}] d, t={
    ax.set_xlabel('x [m]')
    ax.set_ylabel('y [m]')
    ax.grid()
    ax.plot(lsink['XL'], lsink['YL'], 'r', lw=3, label=lsink['name'])
    ax.set_xlim(-1000., 1500.)
    ax.set_ylim(-1500., 1500.)
    ax.set_aspect(1.0)
    ax.legend()
    plt.show()
```

```
Gebieds type: overgangsgebied drainageweerstand verloopt WO van 1000 naar 1
0 d.
self.neq  108
solution complete
```

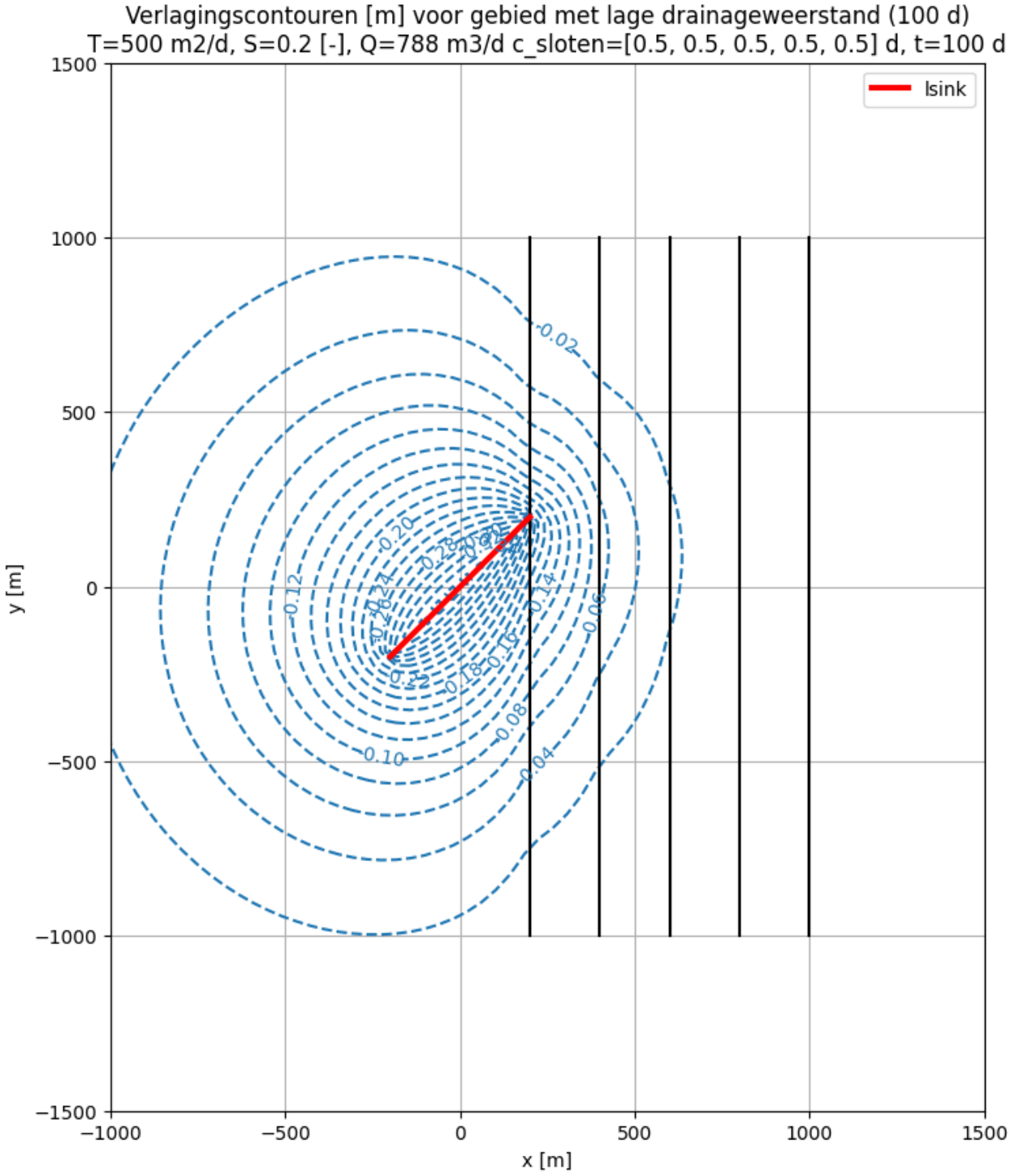
```
Computing the points for contouring takes a lot of time (order 1m30s), so p
lease be patient!
```



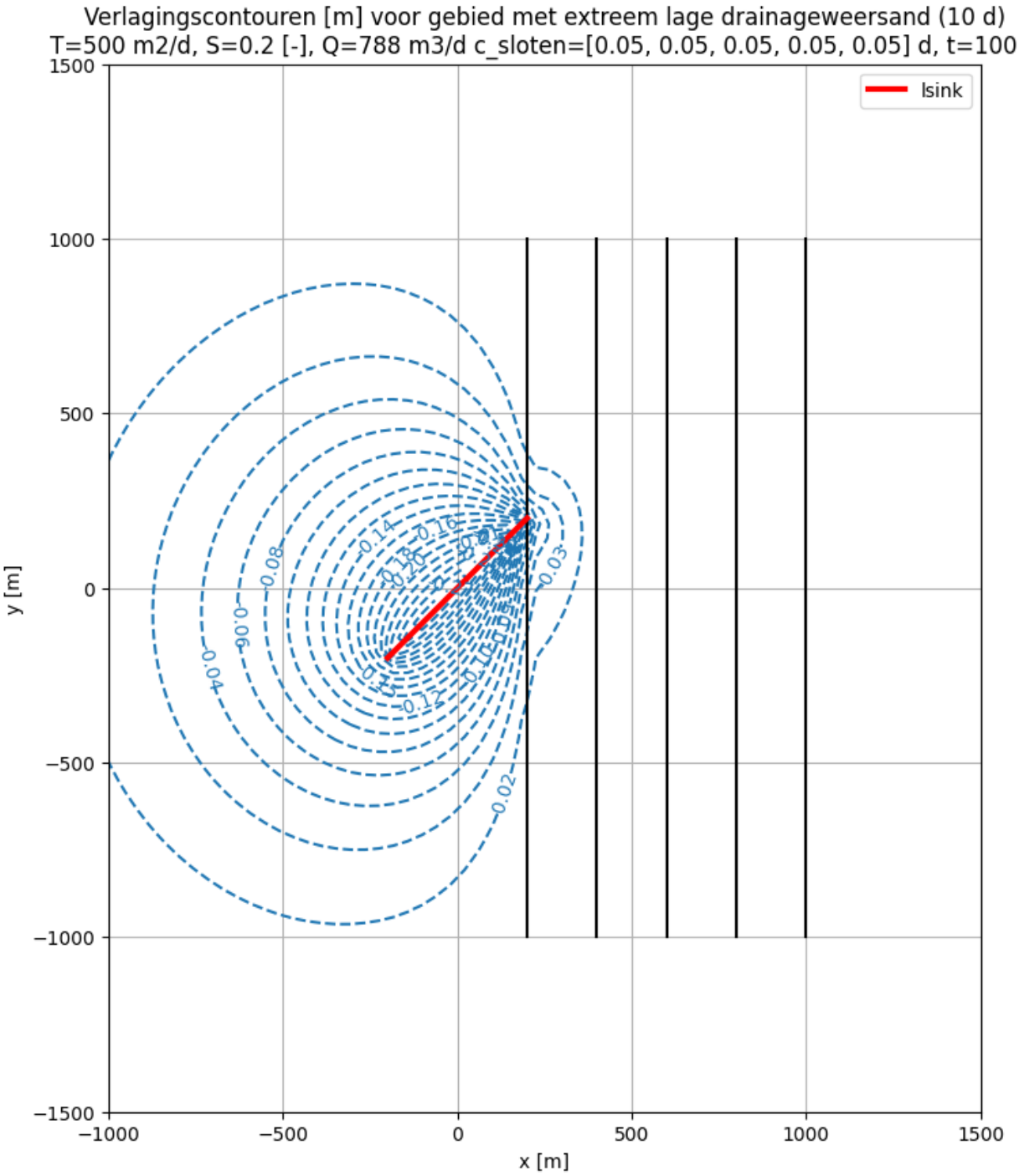

```
Gebieds type: gebied met hoge drainageweerstand (1000 d).
self.neq  108
solution complete
Computing the points for contouring takes a lot of time (order 1m30s), so p
lease be patient!
```




```
Gebieds type: gebied met lage drainageweerstand (100 d).
self.neq  108
solution complete
Computing the points for contouring takes a lot of time (order 1m30s), so p
lease be patient!
```




```
Gebieds type: gebied met extreem lage drainageweersand (10 d).
self.neq  108
solution complete
Computing the points for contouring takes a lot of time (order 1m30s), so p
lease be patient!
```


Verlagingscontouren [m] voor gebied met extreem lage drainageweersand (10 d)
T=500 m2/d, S=0.2 [-], Q=788 m3/d c_sloten=[0.05, 0.05, 0.05, 0.05, 0.05] d, t=100 d
lsink
y [m]
x [m]
1500
1000
500
0
−500
−1000
−1500
−1000
−500
0
500
1000
1500
0.02
0.03
0.04
0.06
0.08
0.10
0.12
0.14
0.16