Functies onderzoeken - Numworks

Pythagoras jaargang 62, nummer 3, januari 2023

1 Functies tekenen

We gaan functies tekenen en onderzoeken. In de opeenvolgende paragrafen komen diverse functies voorbij. Daarbij hebben we ons laten inspireren door het eindexamen VWO wiskunde B, 2022 tijdvak 1 van mei 2022.

1.1 Inhoud

1.2 1. Functie onderzoek

Functieonderzoek is een belangrijk onderdeel van het VWO eindexamen Wiskunde B. Het is een onderwerp dat minder vaak ter sprake komt in Pythagoras. We maken hier een uitzondering. We laten zien wat je kunt doen (ter ondersteuning) met Python. Verwacht geen wonderen. Python sec kan niet de afgeleide berekenen. En de beredenering, die moet je echt zelf doen. Als toegift toveren we enkele fraaie plaatjes tevoorschijn.

1.3 2. Functies tekenen in Python op de GR van Numworks

Het tekenen van de functies gebeurt met matplotlib. Dat is vrij technisch. Hieronder worden enkele details uitgelegd die wij gebruiken. Voor andere details moet je kijken op internet. Omdat het geheugen van de GR beperkt is en uiteraard vooral bestemd is voor de 'gewone' berekeningen met je GR, is maar een beperkte functionaliteit van de bibliotheken beschikbaar. Bovendien is een belangrijke bibliotheek, Numpy, helemaal niet aanwezig.

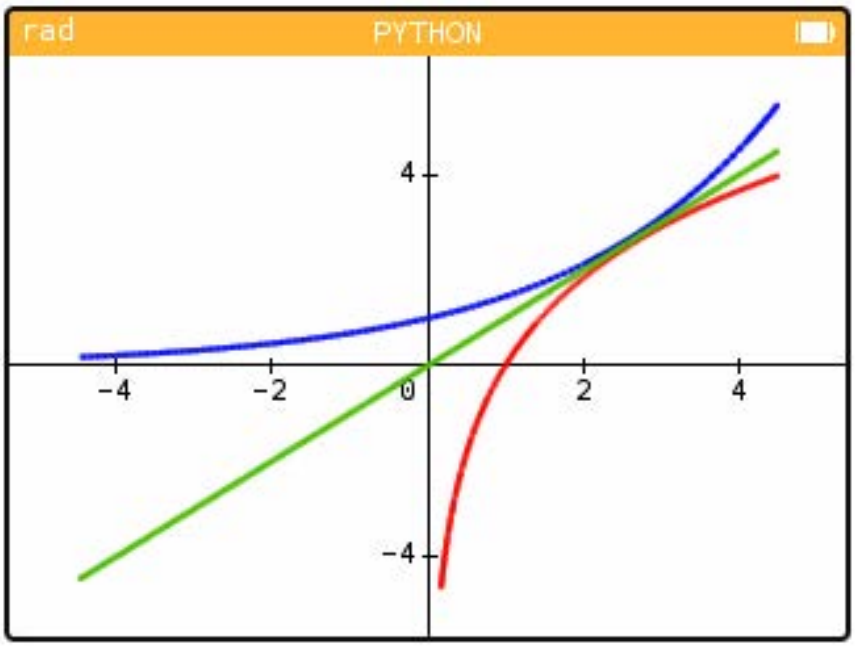
- Numpy heeft een procedure linspace die een lijst van x-waarden genereert, gegeven X_0 en X_1. De derde variabele is NP, het aantal punten dat wordt getekend. Deze functionaliteit

hebben we hier ook nodig, maar we kunnen Numpy dus niet gebruiken. Dat gaat vrij eenvoudig. In plaats van x = linspace(X0, X1, NP) zetten we x = [X0 + i*(X1-X0)/(NP-1) for i in range(NP)]. Dit geeft precies dezelfde lijst van x-waarden. Als je vaker zo'n lijstje nodig hebt, is het handig om de functie linspace eerst zelf te definiëren.
- Numpy zorgt er ook voor dat je bewerkingen in één keer op een hele lijst kunt uitvoeren. Ook die functionaliteit zullen we nu zelf moeten inbouwen. Dit valt ook reuze mee. In plaats van y = wiskundige_bewerking(x, p) kun je schrijven y = [wiskundige_bewerking(xi, p) for xi in x]
- In het scherm van de GR is maar één plot beschikbaar, dus de functies figure en subplot zijn niet aanwezig.
- Kleuren worden, behalve met de enkele letters zoals 'r', 'g' en 'b', net iets anders ingevoerd dan in standaard Python: je schrijft niet c = (0, 0, 0), maar je geeft een vector met 3 zogeheten RGB-waarden mee: [r, g, b]. De waarden lopen van 0 tot 255: rood is bijvoorbeeld [255, 0, 0] en cyaan is [0, 255, 255]. Voor N = 100 kun je met c = [255*i*/*N*, *0*, *255*(1-i/N)] langzaam van rood naar blauw gaan in 100 stappen.
- Ook belangrijk is dat het werkgeheugen van de GR echt beperkt is. Dat betekent onder meer dat je minder grafieken tegelijk kunt tekenen, of minder precies. In de programmatuur staat dat gemarkeerd als commentaar op de betreffende regel.

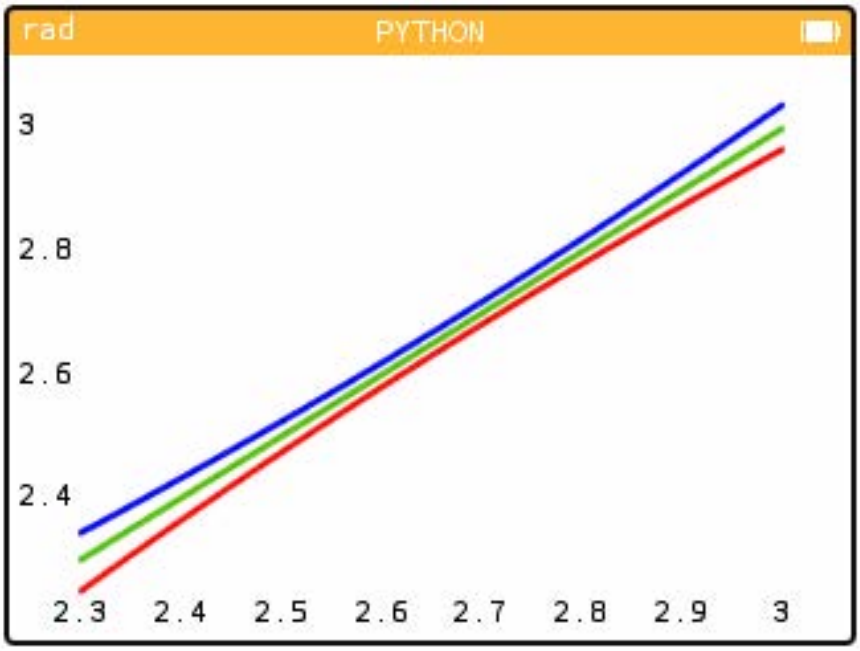
1.4 3. Gebruik maken van je grafische rekenmachine (GR)

Pythagoras gaat de komende afleveringen samenwerken met bouwers van grafische rekenmachines. Momenteel is dat naast die van NumWorks de Texas Instruments TI-84 Plus CE-T (Python Edition).

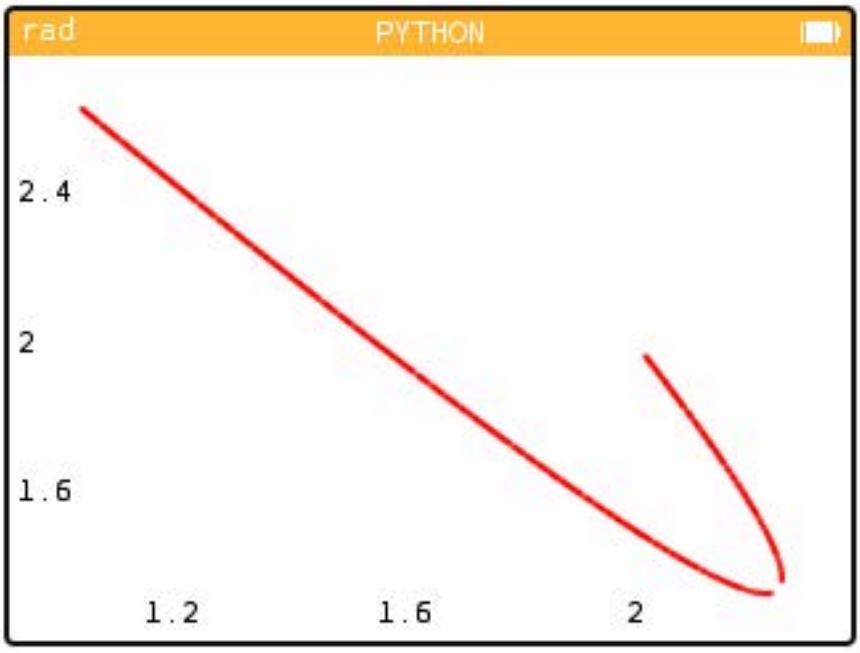
[3]:
```
# 4. Natuurlijke logaritme (Opgave 1-3)

import matplotlib.pyplot as plt
import math

PN = 100
PU = 1/PN
X0 = 17*PU
X1 = 4.5
x1 = [X0+i*(X1-X0)/(PN-1) for i in range(PN)]
x2 = [-X1+i*(X1+X1)/(PN-1) for i in range(PN)]
#p = np.exp(1) # Hiermee raken de krommen
p = 2.65 # Hiermee raken de krommen net niet

f1 = [p*math.log(xi) for xi in x1]
f2 = [math.exp(xi/p) for xi in x2]
f3 = x2
plt.axis()
plt.plot(x1,f1, 'r')
plt.plot(x2,f2, 'b')
plt.plot(x2,f3, 'g')
plt.show()
```



[12]:
```
# 5. Natuurlijke logaritme (Opgave 1-3)
# Uitvergroting rondom het raakpunt.

import matplotlib.pyplot as plt
import math

PN = 100
PU = 1/PN
X0 = 2.3
X1 = 3
x1 = [X0+i*(X1-X0)/(PN-1) for i in range(PN)]
x2 = [X0+i*(X1-X0)/(PN-1) for i in range(PN)]
#p = np.exp(1) # Hiermee raken de krommen
p = 2.7 # Hiermee raken de krommen net niet

f1 = [p*math.log(xi) for xi in x1]
f2 = [math.exp(xi/p) for xi in x2]
f3 = x2
plt.plot(x1,f1, 'r')
plt.plot(x2,f2, 'b')
plt.plot(x2,f3, 'g')
plt.axis()
plt.show()
```

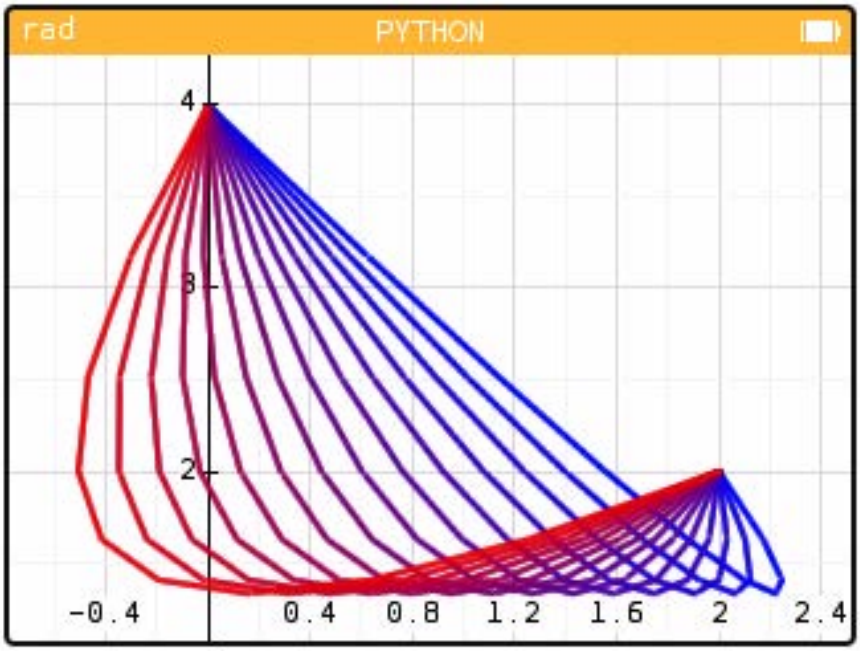


[18]:
```
# 6. Bezier kromme (Opgave 7)
# x(t) = -4t^2+6t
# y(t) = 6t^2-8t+4

import matplotlib.pyplot as plt
import math

PN = 100
T0 = 0.2
T1 = 1
Ax = 0
Ay = 4
Bx = 2
By = 2
Cx = 3
Cy = 0

t = [T0 + i*(T1-T0)/PN for i in range(PN)]
x = [(1-ti)**2*Ax + ti**2*Bx + 2*ti*(1-ti)*Cx for ti in t]
y = [(1-ti)**2*Ay + ti**2*By + 2*ti*(1-ti)*Cy for ti in t]
plt.plot(x,y, 'r')
plt.show()
```



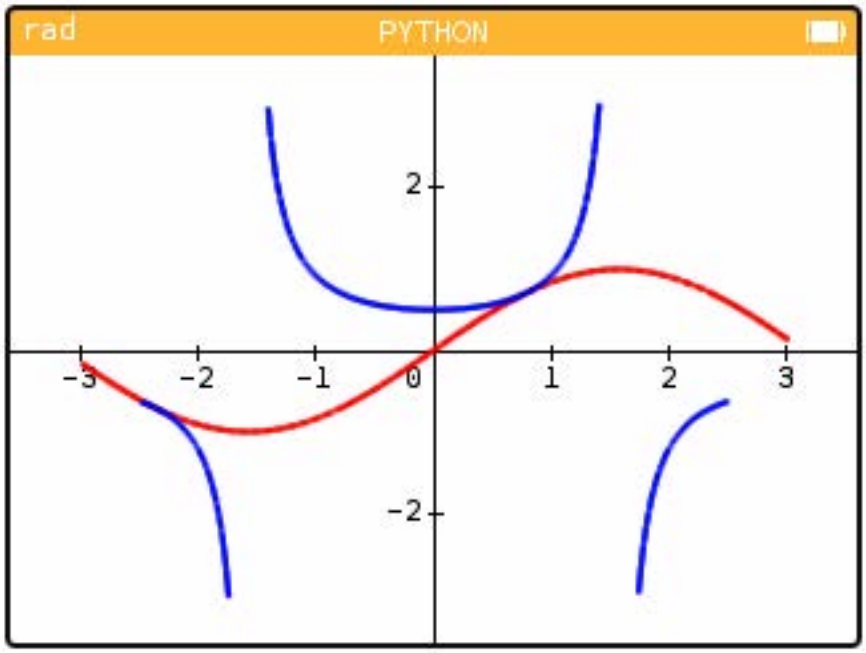
[22]:
```
#7. Bezier kromme (Opgave 7)
# x(t) = -4t^2+6t
# y(t) = 6t^2-8t+4
# Je kunt zelf eens proberen om het punt C een andere baan te laten beschrijven.

import matplotlib.pyplot as plt
import math

PN = 10 # -- met meerdere krommen kun je niet PN=100 nemen
T0 = 0
T1 = 1
Ax = 0
Ay = 4
Bx = 2
By = 2
t = [T0+i*(T1-T0)/(PN-1) for i in range(PN)]
CRange = 14 # -- meer dan 14 krommen passen niet

for i in range(CRange):
    Cx = 3 - 5*i/CRange
    Cy = 0
    x = [(1-ti)**2*Ax + ti**2*Bx + 2*ti*(1-ti)*Cx for ti in t]
    y = [(1-ti)**2*Ay + ti**2*By + 2*ti*(1-ti)*Cy for ti in t]
    plt.plot(x,y,c=(i/CRange, 0, (1-i/CRange)))

plt.grid()
```

```
plt.axis('auto')
plt.show()
```



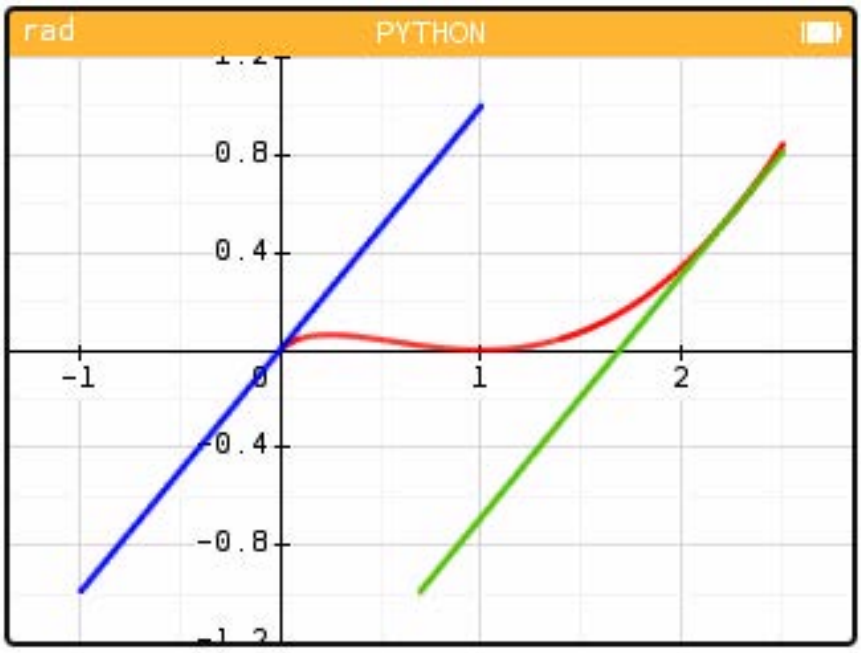
[30]:
```
# 8. Gebroken sinus Kromme (Opgave 8)
# Let op het was nog wel een gepuzzel om bij de polen (plaatsen waar de functie
# niet bestaat) een gebiedje te definieren waar geen functiewaarden worden
# berekend.

import matplotlib.pyplot as plt
import math

def linspace(A,B,N):
    delta=(B-A)/(N-1)
    l=[A+i*delta for i in range(N)]
    return l

Pi2 = math.pi/2

PN = 60 # -- honderd punten is net te veel voor het geheugen
PUP = 1/6
PU = 1/1000
X0 = -3
X1 = 3
x1 = linspace(X0, X1, PN)
x2 = linspace(X0, -Pi2-PUP, PN)
x3 = linspace(-Pi2+PUP, -PU, PN)
```

```
x4 = linspace(PU, Pi2-PUP, PN)
x5 = linspace(Pi2+PUP, X1, PN)

f1 = [math.sin(xi) for xi in x1]
f2 = [math.sin(xi)/math.sin(2*xi) for xi in x2]
f3 = [math.sin(xi)/math.sin(2*xi) for xi in x3]
f4 = [math.sin(xi)/math.sin(2*xi) for xi in x4]
f5 = [math.sin(xi)/math.sin(2*xi) for xi in x5]

plt.plot(x1,f1, 'r')
plt.plot(x2,f2, 'b')
plt.plot(x3,f3, 'b')
plt.plot(x4,f4, 'b')
plt.plot(x5,f5, 'b')
plt.axis('auto')
plt.show()
```



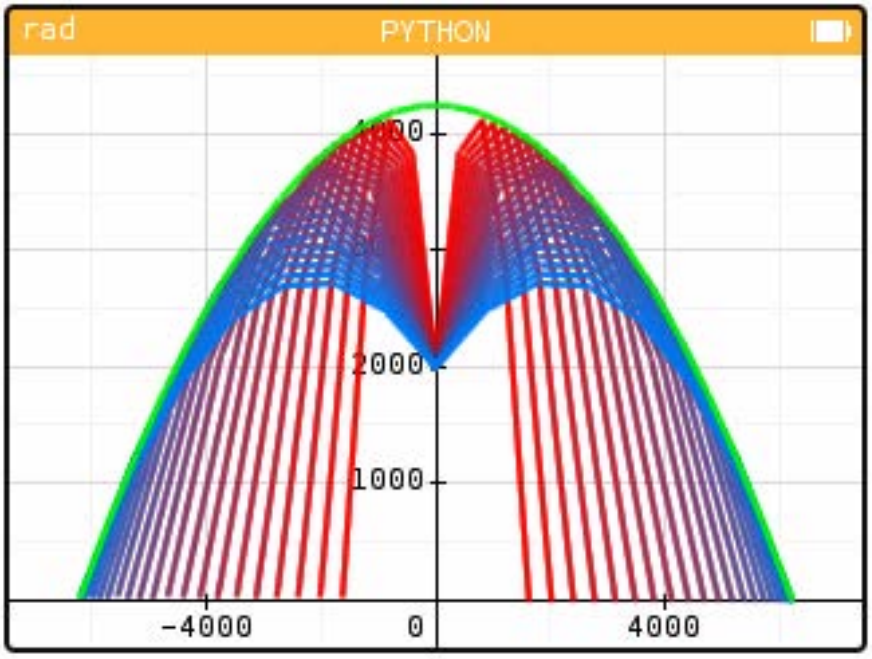
[25]:
```
# 9. Evenwijdige raaklijnen. Opgave 9 - 11
# De berekening die ten grondslag ligt voor de tekening is hier niet
# terug te vinden. Kijk daarvoor bij de uitwerkingen van het eindexamen.

import matplotlib.pyplot as plt
import math

def linspace(A, B, N):
    delta = (B-A)/(N-1)
```

```
    l = [A+i*delta for i in range(N)]
    return l

PN = 100 # -- 1000 punten per lijn is echt te veel voor de GR
PU = 1/PN
X0 = 0
X1 = 2.5
x = linspace(X0, X1, PN)
x2 = linspace(-1, X1-1.5, PN)
x3 = linspace(11/16, X1, PN)

f = [xi*xi+xi-2*xi*math.sqrt(xi) for xi in x]
g = x2
h = [xi-27/16 for xi in x3]

plt.plot(x,f, 'r')
plt.plot(x2,g, 'b')
plt.plot(x3,h, 'g')
plt.axis('auto')
plt.show()
```



[36]:
```
# 10. Vulkaan, Opgave 12 - 14
# ta = tan(alpha)
# y = -(1 + ta^2)/x^2*9000 + ta*x + 2000
#
# Hier zijn in een tekening meerdere functies met varierende alpha
```

```
# getekend.

import matplotlib.pyplot as plt
import math

Pi = math.pi

RNG = 21 # -- bij meer dan 21 'lavabommen' past het niet in het geheugen
BND = 3.9
RST = 1/2-1/BND
XX = 9000 * 4250
X = math.sqrt(XX)

for ii in range(RNG+1):
    i = RNG-ii
    alpha = (1/(BND*RNG) * (i-4) + RST) * Pi
    ta = math.tan(alpha)
    a = -(1 + ta**2)/9000
    b = ta
    c = 2000
    disc = b*b - 4*a*c
    worteldisc = math.sqrt(disc)
    X0 = (-b+worteldisc)/(2*a)
    X1 = (-b-worteldisc)/(2*a)
    PN = min(8,max(round((X1-X0)/1000),5))
    xm = [-X1 + i*X1/(PN-1.) for i in range(PN)]
    fm = [a*xi**2 - b*xi + c for xi in xm]
    col = [255*i/RNG, 255*(0.5-i/(2*RNG)), 255*(1-i/RNG)]
    plt.plot(xm, fm,col)
    xp = [X1*i/(PN-1) for i in range(PN)]
    fp = [a*xi**2 + b*xi + c for xi in xp]
    plt.plot(xp,fp,col)

PN = round(X/240) # -- het aantal punten per lijn niet te groot zijn, dus X␣
 ↪delen door een groter getal
x = [-X + i*2*X/(PN-1) for i in range(PN)]
f = [-xi**2/9000 + 4250 for xi in x]
plt.plot(x,f,[0,255,0])
plt.grid()
plt.show()
```



[24]:
```
# 11. Scheve asymptoot, Opgave 15

import matplotlib.pyplot as plt
import math

PN = 20 # -- het aantal punten per lijn kan niet te groot zijn, 500 is veel te␣
 ↪veel
PU = 1/PN
X0 = 0.3
X1 = 6
p0 = 0.6
p1 = 3.7
RNG = 23 # -- het aantal raaklijnen kan niet te groot zijn, 150 is te veel
X0 = 0

for i in range(RNG+1):
    p = p0 + i*(p1-p0)/RNG
    fp = p + 2/p
    rcp = 1-2/p**2
    cnt = fp - p*rcp
    X2 = -cnt/rcp
    if rcp < 0:
        XX0 = X0
        XX1 = min(X1,X2)
    else:
        XX0 = max(X0,X2)
```

```
        XX1 = X1

    x = [XX0 + (XX1-XX0)*i/(PN-1) for i in range(PN)]
    h = [rcp*xi + cnt for xi in x]
    plt.plot(x,h, [0, 255*(1-i/RNG), 255*i/RNG])

X0 = 0.3
X1 = 6
PN = 50
x = [X0 + (X1-X0)*i/(PN-1) for i in range(PN)]
f = [xi + 2/xi for xi in x]
plt.plot(x,f,'r')
plt.grid()
plt.show()
```