alpha_c

Entrée [100]:
```
reset
```

Entrée [1]:
```
%pylab inline
from scipy.integrate import solve_ivp
import scipy.special as sc
```

```
Populating the interactive namespace from numpy and matplotlib
```

1d Case

In one-dimension, $\psi = (v, u)$ solves the equation

$$\begin{cases} v' = -\left((\lambda + m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right) u \\ u' = \left((\lambda - m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right) v \end{cases}$$

The quantity $\alpha(\lambda, p)$ is the Lp norm of V, with $V = (|u|^2 + |v|^2)^{\frac{1}{p-1}}$.

Entrée [2]:
```
# solves the equation with initial condition (v0, 0)
def solve_with_v0_d1(v0, args):

    # We solve the equation on the interval tt
    tt, m, λ, p = args["tt"], args["m"], args["λ"], args["p"]

    def f1(t, y):
        v, u = y
        V = (u**2 + v**2)**(1/(p-1))
        dv = - ((λ+m) + V)*u
        du = ((λ-m) + V)*v
        return array([dv, du])

    a, b = tt[0], tt[-1]
    sol = solve_ivp(f1, t_span=[a, b], y0 = [v0, 0], t_eval=tt, rto
    return sol.y

# We find the best initial condition with a dichotomy
def find_psi_ode_d1(args):
    vmin, vmax = args["vmin"], args["vmax"]
    vmin0, vmax0 = vmin, vmax
    for n in range(30):
        v0 = (vmin + vmax)/2
        psi = solve_with_v0_d1(v0, args)
        v, u = psi

        # if good precision
        if norm (vmax - vmin) < 1e-7:
            print("λ = {:3}, p = {:.4f}, vmin = {:.4f}, vmax = {:.4
            return psi, v0

        # else dichotomy
        if min(v) < 1e-3: # the solution becomes negative: v0 is to
            vmin, vmax = vmin, v0
        else:
            vmin, vmax = v0, vmax
    print("problem")
    return psi, v0

def alpha_ode_d1(args, psi = None):
    if psi == None:
        psi, _ = find_psi_ode_d1(args)
    v, u = psi

    p, tt = args["p"], args["tt"]
    eps = tt[1] - tt[0]
    V = (v**2 + u**2)**(1/(p-1))

    return (2*sum(V**p)*eps)**(1/p) #the Lp norm of V # 2 factor fo
```

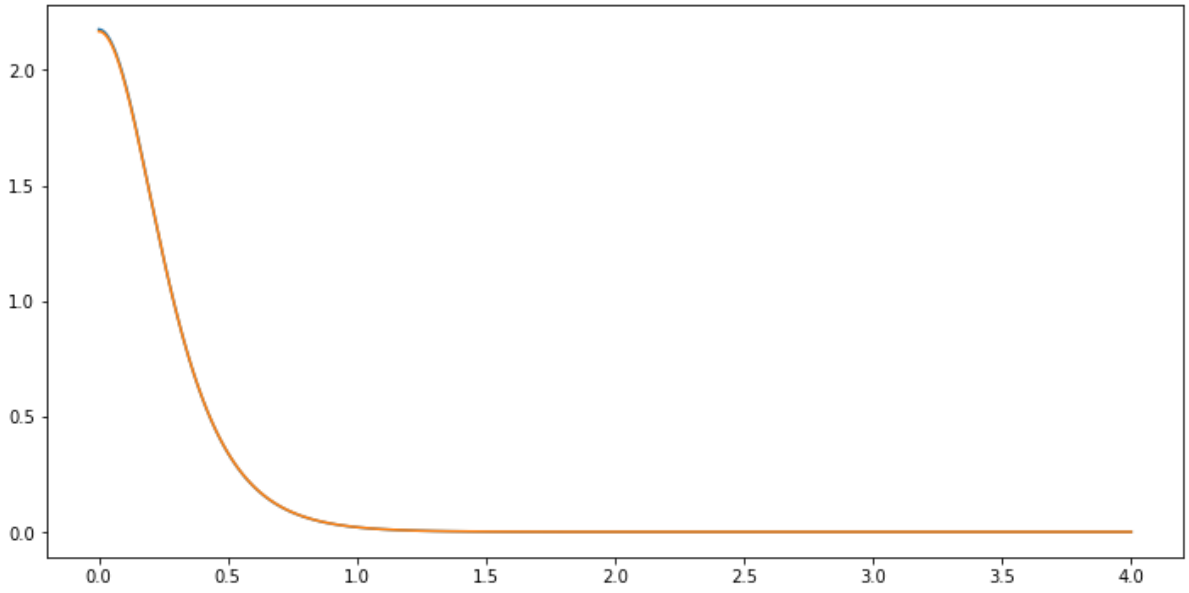
Entrée [3]:
```
# Theoretical solutions
def get_Vth_d1(args):
    tt, m, λ, p = args["tt"], args["m"], args["λ"], args["p"]

    if λ > -m:
        A = p/(p-1)*(m**2 - λ**2)
        B = 2/(p-1)*sqrt(m**2 - λ**2)
        return A/(m*cosh(B*tt) + λ)

    else:
        C = 2*m/(p-1)
        return C*p/(1 + C**2 * tt**2)

def alpha_th_d1(args):
    λ, p, m = args["λ"], args["p"], args["m"]

    if λ > -m:
        z0 = (m-λ)/(m+λ)
        A = p**p*(m + λ)**(p-1)/(p-1)**(p-1)*z0**(p-1/2)*sc.beta(1/
        return A**(1/p)

    if λ == -m:
        A = ( (p**p) * ((2*m)/(p-1))**(p-1) )*sc.beta(1/2, p-1/2)
        return A**(1/p)
```

Entrée [4]:
```
# test

# parameters
Nb = int(1e4)
tt = linspace(0, 4, Nb)
m = 1
λ = 0.5
p = 1.3
vmin, vmax = 0.1, 5
eps = tt[1] - tt[0]

args = {  "tt": tt,    "m": m,    "λ": λ,    "p": p,    "vmin": vmi

# ode values
psi, v0 = find_psi_ode_d1(args)
v, u = psi
Vode = (abs(v)**2 + abs(u)**2)**(1/(p-1))
alpha_ode = alpha_ode_d1(args)

# theoritical values
Vth = get_Vth_d1(args)
alpha_th = alpha_th_d1(args)

print("(Check V) must be 0:", norm(Vode - Vth))
print("(Check alpha) must be 0:", abs(alpha_ode - alpha_th))
```

```
λ = 0.5, p = 1.3000, vmin = 0.1000, vmax = 5.0000, v0 = 1.1236
λ = 0.5, p = 1.3000, vmin = 0.1000, vmax = 5.0000, v0 = 1.1236
(Check V) must be 0: 0.15383657062570855
(Check alpha) must be 0: 0.0034294158995393786
```

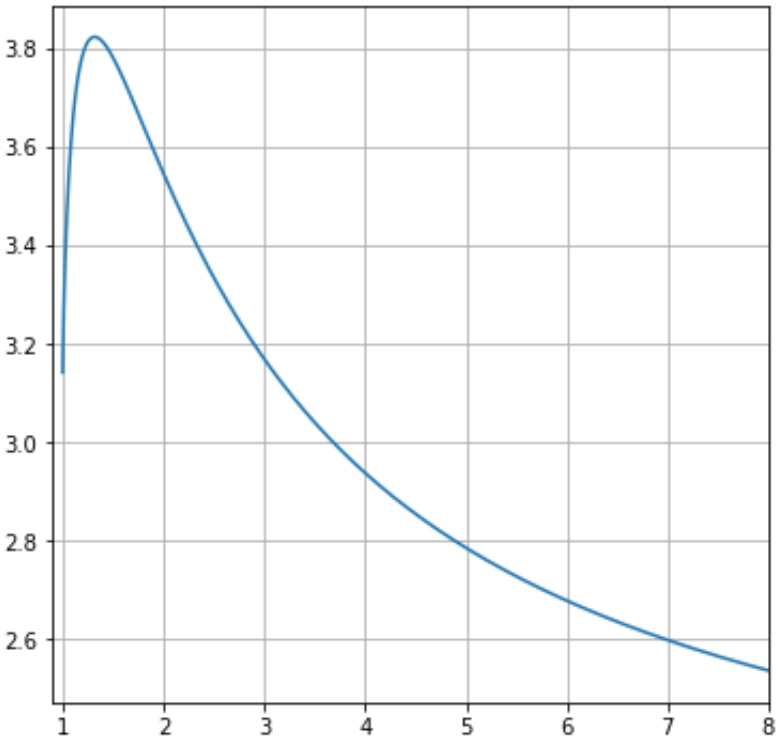
Entrée [5]:
```
figsize(12, 6)

plot(tt, Vode)
plot(tt, Vth)
print("alpha = ", alpha_th)
```

```
alpha =  1.3338553731317149
```



Entrée [6]:
```
### Plot \alpha_c

pp = linspace(1+1e-6, 8, 2000)
alphac = []

for p in pp:
    args = {"m": 1,    "λ": -1,    "p": p}
    alphac.append(alpha_th_d1(args))

alpha0 = []

for p in pp:
    args = {"m": 1,    "λ": 0,    "p": p}
    alpha0.append(alpha_th_d1(args))
```

Entrée [10]:
```
figsize(6, 6)
plot(pp, alphac)
#plot(pp, alpha0)
xlim(0.9,8)

grid()

savefig('../alphac_d1.png', dpi=600, bbox_inches='tight')
```



Entrée [11]:
```
pp[argmax(alphac)]
```

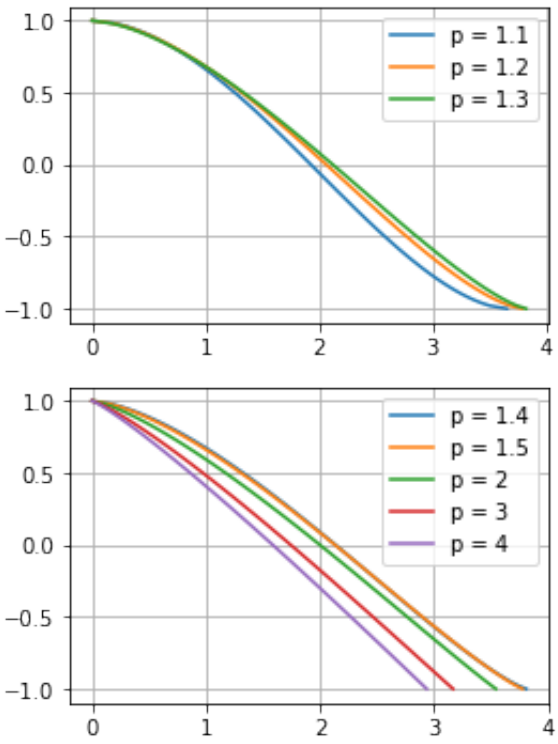
Out[11]:
```
1.318660284142071
```

Plot of $\alpha \mapsto \Lambda(\alpha, p)$ for several values of p

Entrée [12]:

```
pp1 = [1.1, 1.2, 1.3]
pp2 = [1.4, 1.5, 2, 3, 4]
λλ = linspace(-1, 1 -1e-6, 100)

Np1, Np2, Nλ = len(pp1), len(pp2), len(λλ)

alpha_record1 = zeros((Np1, Nλ))
alpha_record2 = zeros((Np2, Nλ))

for ip in range(Np1):
    for iλ in range(Nλ):
        p, λ = pp1[ip], λλ[iλ]
        args = {"m": 1,    "λ": λ,    "p": p}
        alpha_record1[ip, iλ] = alpha_th_d1(args)

for ip in range(Np2):
    for iλ in range(Nλ):
        p, λ = pp2[ip], λλ[iλ]
        args = {"m": 1,    "λ": λ,    "p": p}
        alpha_record2[ip, iλ] = alpha_th_d1(args)
```

Entrée [13]:
```
figsize(4, 6)
subplot(211)
for ip in range(Np1):
    p = pp1[ip]
    plot(alpha_record1[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

subplot(212)
for ip in range(Np2):
    p = pp2[ip]
    plot(alpha_record2[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

savefig('../Lambda_d1.png', dpi = 600, bbox_inches='tight')
```



2d case

The solution in the $n \in \mathbb{Z}$ space solves the equation (we write $\psi(r) = (v(r)\exp(in\theta), iu(r)\exp i(n+1)\theta))$

$$\begin{cases} v' = \frac{n}{r}v - \left((\lambda + m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)u \\ u' = -\frac{n+1}{r}u + \left((\lambda - m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)v \end{cases}$$

We focus on the two cases $n = 0$ and $n = -1$, respectively

$$(n=0) \quad \begin{cases} v' = -\left((\lambda + m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)u \\ u' = -\frac{1}{r}u + \left((\lambda - m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)v \end{cases}$$

and

$$(n=1) \quad \begin{cases} v' = -\frac{1}{r}v - \left((\lambda + m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)u \\ u' = \left((\lambda - m) + (|u|^2 + |v|^2)^{\frac{1}{p-1}}\right)v \end{cases}$$

In the case $n = 0$, we must have $u(0) = 0$. We notice that the function v is decreasing, hence positive. This gives a dichotomy criterion to find the initial condition for $v(0)$.

In the case $n = -1$, we must have $v(0) = 0$, and, assuming v and u positive, $v' < 0$. Our criterion for $u(0) > 0$ is therefore to have $v < 0$ everywhere

Entrée [14]:

```
# solves the equation with initial condition (x0, 0) if n==0, and w
def solve_with_x0_d2(x0, args):

    # We solve the equation on the interval tt
    tt, m, λ, p, n = args["tt"], args["m"], args["λ"], args["p"], a

    def f2(t, y):
        v, u = y
        V = (u**2 + v**2)**(1/(p-1))
        dv = n/t*v - ((λ + m) + V)*u
        du = -(n+1)/t*u + ((λ - m) + V)*v
        return array([dv, du])

    a, b = tt[0], tt[-1]
    if n==0:
        sol = solve_ivp(f2, t_span=[a, b], y0 = [x0, 0], t_eval=tt,
    if n==-1:
        sol = solve_ivp(f2, t_span=[a, b], y0 = [0, x0], t_eval=tt,
    return sol.y

# We find the best initial condition with a dichotomy
def find_psi_ode_d2(args):
    xmin, xmax = args["xmin"], args["xmax"]
    λ, p, n = args["λ"], args["p"], args["n"]
    xmin0, xmax0 = xmin, xmax

    for _ in range(100):
        x0 = (xmin + xmax)/2
        psi = solve_with_x0_d2(x0, args)
        v, u = psi

        # if good precision
        if norm (xmax - xmin) < 1e-7:
            print("λ = {:3}, p = {:.4f}, n = {}, xmin = {:.4f}, xma
            return psi, x0

        # else dichotomy
        if n == 0:
            if min(v) < 1e-8: # the solution becomes negative: v0 i
                xmin, xmax = xmin, x0
            else:
                xmin, xmax = x0, xmax
        if n ==-1:
            if max(v) > 1e-8: # in the n=-1 case, expect v to be ne
                xmin, xmax = xmin, x0
            else:
                xmin, xmax = x0, xmax
    print("problem")
    return psi, x0
```

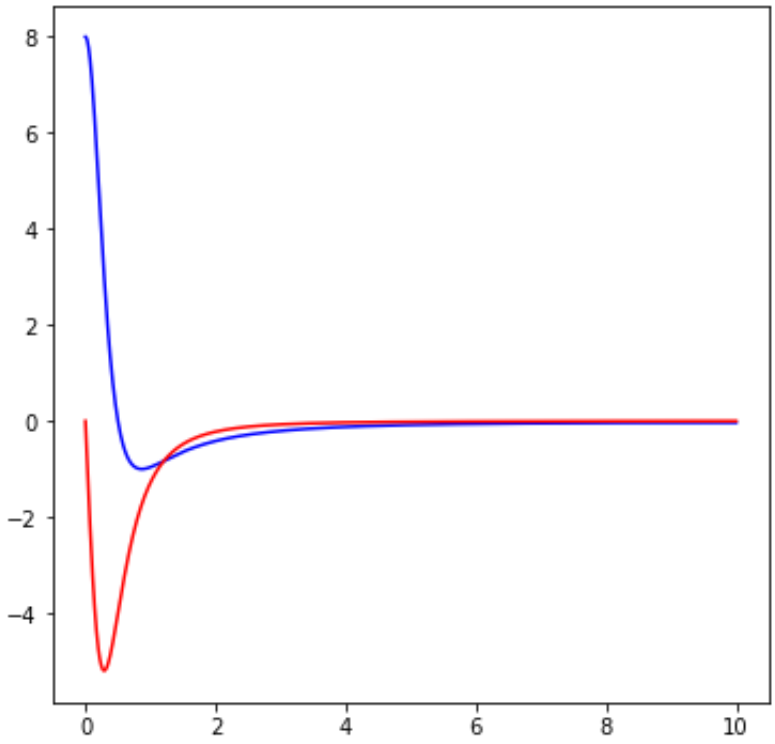
Entrée [22]:
```
def alpha_d2(psi, args):
    v, u = psi

    p, tt = args["p"], args["tt"]
    eps = tt[1] - tt[0]
    V = (v**2 + u**2)**(1/(p-1))
    integral = 2*pi*sum(tt*V**p)*eps
    return (integral)**(1/p)
```

Entrée [38]:
```
# test

Nb = 1000
tt = linspace(1e-6, 10, Nb)
m = 1
λ = -1
p = 3
xmin, xmax = 0, 50

# we expect that the alpha with n = 0 is greater (update : smaller
for n in [0, -1]:
    print("\n n = ", n)
    args = {"m": m, "λ": λ, "p": p, "tt": tt, "xmin": xmin, "xmax":
    psi, x0 = find_psi_ode_d2(args)
    v, u = psi
    Vode = (abs(v)**2 + abs(u)**2)**(1/(p-1))
    print("alpha = ", alpha_d2(psi, args) )
```

```
 n =  0
λ =  -1, p = 3.0000, n = 0, xmin = 0.0000, xmax = 50.0000, x0 = 6.
0001
alpha =  4.393618268506922

 n =  -1
λ =  -1, p = 3.0000, n = -1, xmin = 0.0000, xmax = 50.0000, x0 = 8
.0003
alpha =  5.858217563943089
```

Entrée [24]:
```
print("n = ", n)
v, u = psi
plot(tt, u, 'b')
plot(tt, v, 'r')

print("u[-1] = {}, v[-1] = {}".format(u[-1], v[-1]))
```

```
n =  -1
u[-1] = -0.03287051811316601, v[-1] = -2.5254246246869953e-07
```



Computation of \alpha_c for $d = 2$

Entrée [32]:
```

```

```
# Case n = 0
# ### Plot \alpha_c
Nb = 1000
tt = linspace(1e-6, 10, Nb)
xmin, xmax = 0, 50

pp_n0 = linspace(2+0.1, 9, 100)
alphac2_n0 = []
psi_record_n0 = []

for p in pp_n0:
    args = {"m": 1, "λ": -1, "p": p, "tt": tt, "n":0, "xmin":xmin, "
    psi, x0 = find_psi_ode_d2(args)
    psi_record_n0.append(psi)
    alphac2_n0.append(alpha_d2(psi, args))
    xmin, xmax = maximum(0, x0-2), x0+2
```

```
λ =  -1, p = 2.1000, n = 0, xmin = 0.0000, xmax = 50.0000, x0 = 7.
8130
λ =  -1, p = 2.1697, n = 0, xmin = 5.8130, xmax = 9.8130, x0 = 6.6
581
λ =  -1, p = 2.2394, n = 0, xmin = 4.6581, xmax = 8.6581, x0 = 6.1
420
λ =  -1, p = 2.3091, n = 0, xmin = 4.1420, xmax = 8.1420, x0 = 5.8
709
λ =  -1, p = 2.3788, n = 0, xmin = 3.8709, xmax = 7.8709, x0 = 5.7
234
λ =  -1, p = 2.4485, n = 0, xmin = 3.7234, xmax = 7.7234, x0 = 5.6
482
λ =  -1, p = 2.5182, n = 0, xmin = 3.6482, xmax = 7.6482, x0 = 5.6
198
λ =  -1, p = 2.5879, n = 0, xmin = 3.6198, xmax = 7.6198, x0 = 5.6
239
λ =  -1, p = 2.6576, n = 0, xmin = 3.6239, xmax = 7.6239, x0 = 5.6
519
λ =  -1, p = 2.7273, n = 0, xmin = 3.6519, xmax = 7.6519, x0 = 5.6
983
λ =  -1, p = 2.7970, n = 0, xmin = 3.6983, xmax = 7.6983, x0 = 5.7
594
λ =  -1, p = 2.8667, n = 0, xmin = 3.7594, xmax = 7.7594, x0 = 5.8
325
λ =  -1, p = 2.9364, n = 0, xmin = 3.8325, xmax = 7.8325, x0 = 5.9
160
λ =  -1, p = 3.0061, n = 0, xmin = 3.9160, xmax = 7.9160, x0 = 6.0
084
λ =  -1, p = 3.0758, n = 0, xmin = 4.0084, xmax = 8.0084, x0 = 6.1
088
λ =  -1, p = 3.1455, n = 0, xmin = 4.1088, xmax = 8.1088, x0 = 6.2
164
λ =  -1, p = 3.2152, n = 0, xmin = 4.2164, xmax = 8.2164, x0 = 6.3
306
λ =  -1, p = 3.2848, n = 0, xmin = 4.3306, xmax = 8.3306, x0 = 6.4
```

```
511
λ =  -1, p = 3.3545, n = 0, xmin = 4.4511, xmax = 8.4511, x0 = 6.5
774
λ =  -1, p = 3.4242, n = 0, xmin = 4.5774, xmax = 8.5774, x0 = 6.7
094
λ =  -1, p = 3.4939, n = 0, xmin = 4.7094, xmax = 8.7094, x0 = 6.8
468
λ =  -1, p = 3.5636, n = 0, xmin = 4.8468, xmax = 8.8468, x0 = 6.9
895
λ =  -1, p = 3.6333, n = 0, xmin = 4.9895, xmax = 8.9895, x0 = 7.1
375
λ =  -1, p = 3.7030, n = 0, xmin = 5.1375, xmax = 9.1375, x0 = 7.2
906
λ =  -1, p = 3.7727, n = 0, xmin = 5.2906, xmax = 9.2906, x0 = 7.4
488
λ =  -1, p = 3.8424, n = 0, xmin = 5.4488, xmax = 9.4488, x0 = 7.6
121
λ =  -1, p = 3.9121, n = 0, xmin = 5.6121, xmax = 9.6121, x0 = 7.7
804
λ =  -1, p = 3.9818, n = 0, xmin = 5.7804, xmax = 9.7804, x0 = 7.9
539
λ =  -1, p = 4.0515, n = 0, xmin = 5.9539, xmax = 9.9539, x0 = 8.1
326
λ =  -1, p = 4.1212, n = 0, xmin = 6.1326, xmax = 10.1326, x0 = 8.
3164
λ =  -1, p = 4.1909, n = 0, xmin = 6.3164, xmax = 10.3164, x0 = 8.
5054
λ =  -1, p = 4.2606, n = 0, xmin = 6.5054, xmax = 10.5054, x0 = 8.
6997
λ =  -1, p = 4.3303, n = 0, xmin = 6.6997, xmax = 10.6997, x0 = 8.
8995
λ =  -1, p = 4.4000, n = 0, xmin = 6.8995, xmax = 10.8995, x0 = 9.
1046
λ =  -1, p = 4.4697, n = 0, xmin = 7.1046, xmax = 11.1046, x0 = 9.
3154
λ =  -1, p = 4.5394, n = 0, xmin = 7.3154, xmax = 11.3154, x0 = 9.
5317
λ =  -1, p = 4.6091, n = 0, xmin = 7.5317, xmax = 11.5317, x0 = 9.
7538
λ =  -1, p = 4.6788, n = 0, xmin = 7.7538, xmax = 11.7538, x0 = 9.
9817
λ =  -1, p = 4.7485, n = 0, xmin = 7.9817, xmax = 11.9817, x0 = 10
.2156
λ =  -1, p = 4.8182, n = 0, xmin = 8.2156, xmax = 12.2156, x0 = 10
.4556
λ =  -1, p = 4.8879, n = 0, xmin = 8.4556, xmax = 12.4556, x0 = 10
.7018
λ =  -1, p = 4.9576, n = 0, xmin = 8.7018, xmax = 12.7018, x0 = 10
.9543
λ =  -1, p = 5.0273, n = 0, xmin = 8.9543, xmax = 12.9543, x0 = 11
.2132
λ =  -1, p = 5.0970, n = 0, xmin = 9.2132, xmax = 13.2132, x0 = 11
.4788
λ =  -1, p = 5.1667, n = 0, xmin = 9.4788, xmax = 13.4788, x0 = 11
```

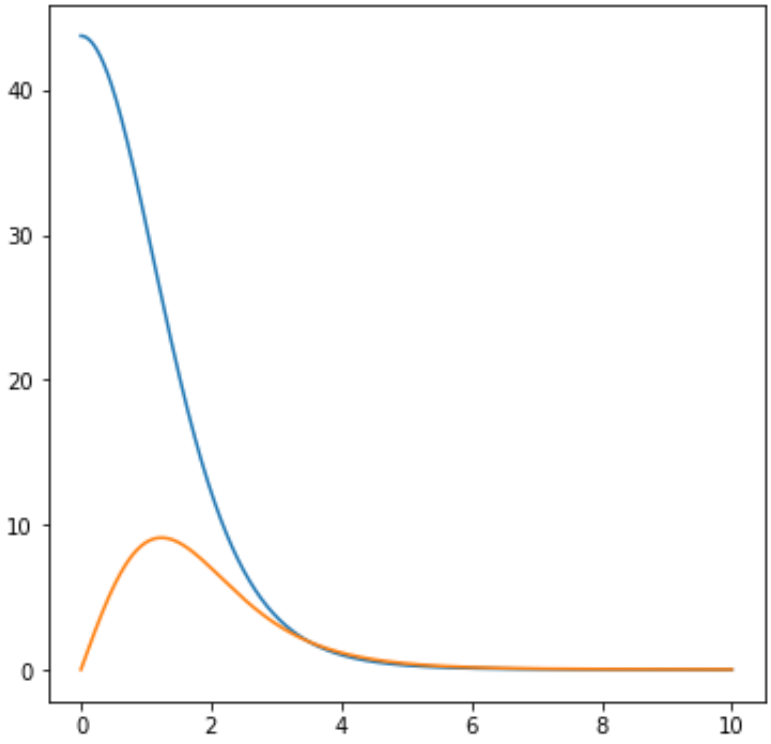
```
.7511
λ =  -1, p = 5.2364, n = 0, xmin = 9.7511, xmax = 13.7511, x0 = 12
.0304
λ =  -1, p = 5.3061, n = 0, xmin = 10.0304, xmax = 14.0304, x0 = 1
2.3166
λ =  -1, p = 5.3758, n = 0, xmin = 10.3166, xmax = 14.3166, x0 = 1
2.6101
λ =  -1, p = 5.4455, n = 0, xmin = 10.6101, xmax = 14.6101, x0 = 1
2.9110
λ =  -1, p = 5.5152, n = 0, xmin = 10.9110, xmax = 14.9110, x0 = 1
3.2194
λ =  -1, p = 5.5848, n = 0, xmin = 11.2194, xmax = 15.2194, x0 = 1
3.5355
λ =  -1, p = 5.6545, n = 0, xmin = 11.5355, xmax = 15.5355, x0 = 1
3.8595
λ =  -1, p = 5.7242, n = 0, xmin = 11.8595, xmax = 15.8595, x0 = 1
4.1916
λ =  -1, p = 5.7939, n = 0, xmin = 12.1916, xmax = 16.1916, x0 = 1
4.5319
λ =  -1, p = 5.8636, n = 0, xmin = 12.5319, xmax = 16.5319, x0 = 1
4.8808
λ =  -1, p = 5.9333, n = 0, xmin = 12.8808, xmax = 16.8808, x0 = 1
5.2382
λ =  -1, p = 6.0030, n = 0, xmin = 13.2382, xmax = 17.2382, x0 = 1
5.6046
λ =  -1, p = 6.0727, n = 0, xmin = 13.6046, xmax = 17.6046, x0 = 1
5.9800
λ =  -1, p = 6.1424, n = 0, xmin = 13.9800, xmax = 17.9800, x0 = 1
6.3647
λ =  -1, p = 6.2121, n = 0, xmin = 14.3647, xmax = 18.3647, x0 = 1
6.7590
λ =  -1, p = 6.2818, n = 0, xmin = 14.7590, xmax = 18.7590, x0 = 1
7.1629
λ =  -1, p = 6.3515, n = 0, xmin = 15.1629, xmax = 19.1629, x0 = 1
7.5769
λ =  -1, p = 6.4212, n = 0, xmin = 15.5769, xmax = 19.5769, x0 = 1
8.0010
λ =  -1, p = 6.4909, n = 0, xmin = 16.0010, xmax = 20.0010, x0 = 1
8.4356
λ =  -1, p = 6.5606, n = 0, xmin = 16.4356, xmax = 20.4356, x0 = 1
8.8810
λ =  -1, p = 6.6303, n = 0, xmin = 16.8810, xmax = 20.8810, x0 = 1
9.3373
λ =  -1, p = 6.7000, n = 0, xmin = 17.3373, xmax = 21.3373, x0 = 1
9.8048
λ =  -1, p = 6.7697, n = 0, xmin = 17.8048, xmax = 21.8048, x0 = 2
0.2838
λ =  -1, p = 6.8394, n = 0, xmin = 18.2838, xmax = 22.2838, x0 = 2
0.7746
λ =  -1, p = 6.9091, n = 0, xmin = 18.7746, xmax = 22.7746, x0 = 2
1.2775
λ =  -1, p = 6.9788, n = 0, xmin = 19.2775, xmax = 23.2775, x0 = 2
1.7927
λ =  -1, p = 7.0485, n = 0, xmin = 19.7927, xmax = 23.7927, x0 = 2
```

```
2.3206
λ =  -1, p = 7.1182, n = 0, xmin = 20.3206, xmax = 24.3206, x0 = 2
2.8614
λ =  -1, p = 7.1879, n = 0, xmin = 20.8614, xmax = 24.8614, x0 = 2
3.4156
λ =  -1, p = 7.2576, n = 0, xmin = 21.4156, xmax = 25.4156, x0 = 2
3.9833
λ =  -1, p = 7.3273, n = 0, xmin = 21.9833, xmax = 25.9833, x0 = 2
4.5649
λ =  -1, p = 7.3970, n = 0, xmin = 22.5649, xmax = 26.5649, x0 = 2
5.1608
λ =  -1, p = 7.4667, n = 0, xmin = 23.1608, xmax = 27.1608, x0 = 2
5.7714
λ =  -1, p = 7.5364, n = 0, xmin = 23.7714, xmax = 27.7714, x0 = 2
6.3969
λ =  -1, p = 7.6061, n = 0, xmin = 24.3969, xmax = 28.3969, x0 = 2
7.0377
λ =  -1, p = 7.6758, n = 0, xmin = 25.0377, xmax = 29.0377, x0 = 2
7.6942
λ =  -1, p = 7.7455, n = 0, xmin = 25.6942, xmax = 29.6942, x0 = 2
8.3668
λ =  -1, p = 7.8152, n = 0, xmin = 26.3668, xmax = 30.3668, x0 = 2
9.0559
λ =  -1, p = 7.8848, n = 0, xmin = 27.0559, xmax = 31.0559, x0 = 2
9.7619
λ =  -1, p = 7.9545, n = 0, xmin = 27.7619, xmax = 31.7619, x0 = 3
0.4852
λ =  -1, p = 8.0242, n = 0, xmin = 28.4852, xmax = 32.4852, x0 = 3
1.2262
λ =  -1, p = 8.0939, n = 0, xmin = 29.2262, xmax = 33.2262, x0 = 3
1.9853
λ =  -1, p = 8.1636, n = 0, xmin = 29.9853, xmax = 33.9853, x0 = 3
2.7630
λ =  -1, p = 8.2333, n = 0, xmin = 30.7630, xmax = 34.7630, x0 = 3
3.5598
λ =  -1, p = 8.3030, n = 0, xmin = 31.5598, xmax = 35.5598, x0 = 3
4.3760
λ =  -1, p = 8.3727, n = 0, xmin = 32.3760, xmax = 36.3760, x0 = 3
5.2122
λ =  -1, p = 8.4424, n = 0, xmin = 33.2122, xmax = 37.2122, x0 = 3
6.0689
λ =  -1, p = 8.5121, n = 0, xmin = 34.0689, xmax = 38.0689, x0 = 3
6.9466
λ =  -1, p = 8.5818, n = 0, xmin = 34.9466, xmax = 38.9466, x0 = 3
7.8457
λ =  -1, p = 8.6515, n = 0, xmin = 35.8457, xmax = 39.8457, x0 = 3
8.7669
λ =  -1, p = 8.7212, n = 0, xmin = 36.7669, xmax = 40.7669, x0 = 3
9.7106
λ =  -1, p = 8.7909, n = 0, xmin = 37.7106, xmax = 41.7106, x0 = 4
0.6773
λ =  -1, p = 8.8606, n = 0, xmin = 38.6773, xmax = 42.6773, x0 = 4
1.6677
λ =  -1, p = 8.9303, n = 0, xmin = 39.6677, xmax = 43.6677, x0 = 4
```

```
2.6824
λ =  -1, p = 9.0000, n = 0, xmin = 40.6824, xmax = 44.6824, x0 = 4
3.7218
```

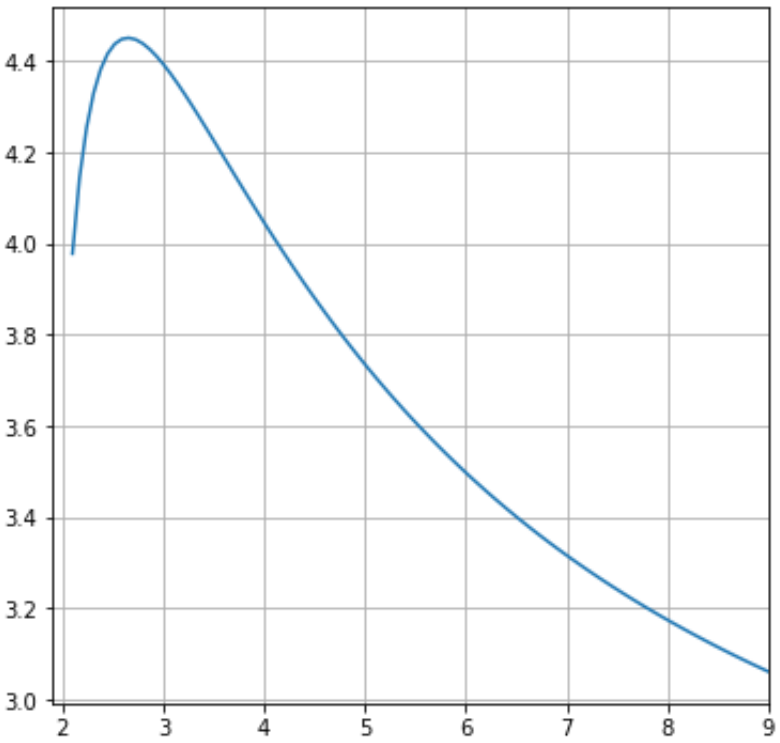
Entrée [33]:
```
v,u = psi_record_n0[-1]
plot(tt, v)
plot(tt, u)
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x11e1ae7c0>]
```



Entrée [34]:
```
figsize(6, 6)
plot(pp_n0, alphac2_n0)
xlim(1.9,9)

grid()

savefig('../alphac_d2_n0.png', dpi = 600, bbox_inches='tight')
print("maximum at p = {}".format(pp_n0[argmax(alphac2_n0)] ))
```

```
maximum at p = 2.6575757575757577
```



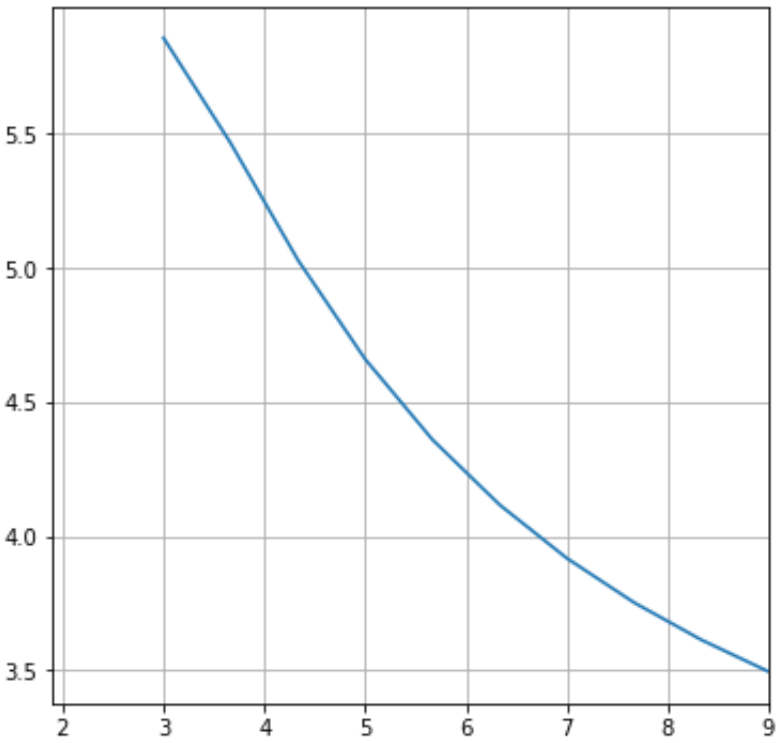
Entrée [ ]:

Entrée [36]:
```
# Case n = -1
# ### Plot \alpha_c
Nb = 10000
tt = linspace(1e-6, 10, Nb)
xmin, xmax = 0, 50

pp_nm1 = linspace(3, 9, 10)
alphac2_nm1 = []
psi_record_nm1 = []

for p in pp_nm1:
    args = {"m": 1, "λ": -1, "p": p, "tt": tt, "n":-1, "xmin":xmin,
    psi, x0 = find_psi_ode_d2(args)
    psi_record_nm1.append(psi)
    alphac2_nm1.append(alpha_d2(psi, args))
    xmin, xmax = maximum(0, x0-3), x0+5
```

```
λ =  -1, p = 3.0000, n = -1, xmin = 0.0000, xmax = 50.0000, x0 = 8
.0003
λ =  -1, p = 3.6667, n = -1, xmin = 5.0003, xmax = 13.0003, x0 = 7
.6967
λ =  -1, p = 4.3333, n = -1, xmin = 4.6967, xmax = 12.6967, x0 = 8
.0923
λ =  -1, p = 5.0000, n = -1, xmin = 5.0923, xmax = 13.0923, x0 = 8
.8836
λ =  -1, p = 5.6667, n = -1, xmin = 5.8836, xmax = 13.8836, x0 = 1
0.0105
λ =  -1, p = 6.3333, n = -1, xmin = 7.0105, xmax = 15.0105, x0 = 1
1.4818
λ =  -1, p = 7.0000, n = -1, xmin = 8.4818, xmax = 16.4818, x0 = 1
3.3398
λ =  -1, p = 7.6667, n = -1, xmin = 10.3398, xmax = 18.3398, x0 =
15.6511
λ =  -1, p = 8.3333, n = -1, xmin = 12.6511, xmax = 20.6511, x0 =
18.5050
λ =  -1, p = 9.0000, n = -1, xmin = 15.5050, xmax = 23.5050, x0 =
22.0162
```

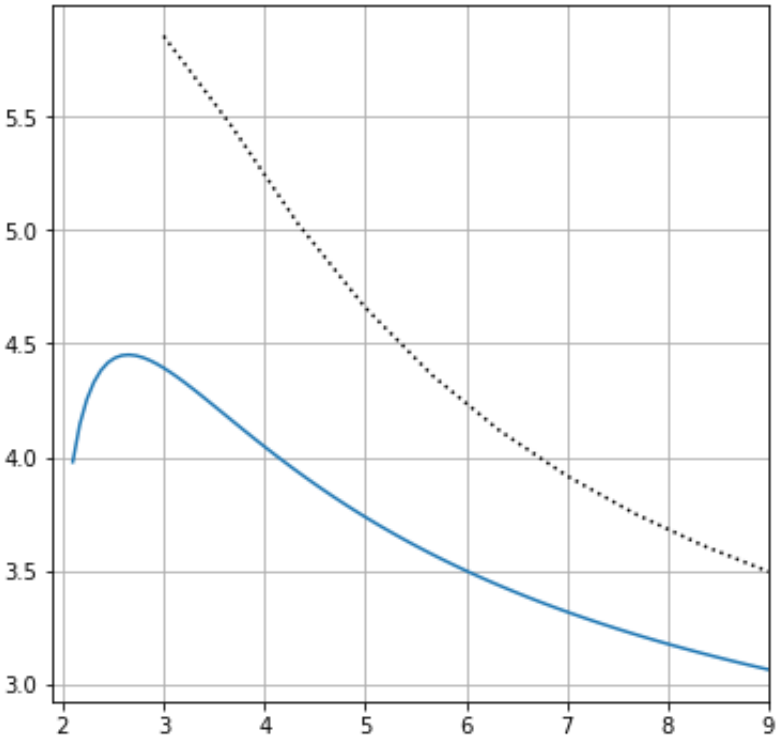
Entrée [39]:
```
figsize(6, 6)
plot(pp_nm1, alphac2_nm1)
xlim(1.9,9)

grid()

savefig('../alphac_d2_nm1.png', dpi = 600, bbox_inches='tight')
print("maximum at p = {}".format(pp_nm1[argmax(alphac2_nm1)] ))
```

```
maximum at p = 3.0
```



Entrée [40]:
```
figsize(6, 6)
plot(pp_n0, alphac2_n0)
plot(pp_nm1, alphac2_nm1, ':k')
xlim(1.9,9)

grid()

#savefig('../alphac_d2_nm1.png', bbox_inches='tight')
print("maximum at p = {}".format(pp_nm1[argmax(alphac2_nm1)] ))
```

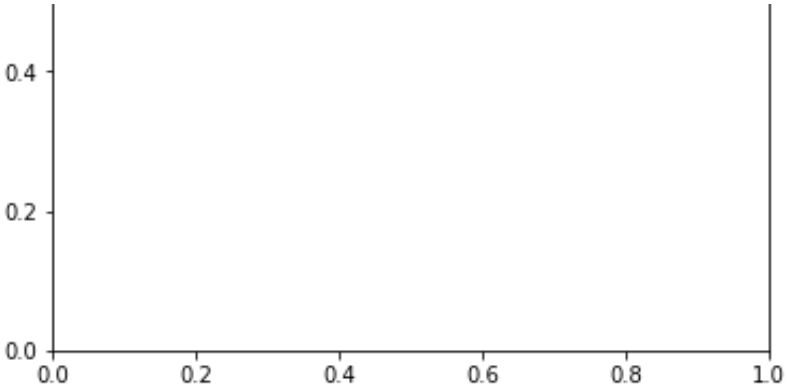
```
maximum at p = 3.0
```



Entrée [41]:
```
ind = -1

v,u = psi_record_nm1[ind]
p = pp[ind]

print("p = ", p)
plot(tt, v)
plot(tt, u)
```

```
p =  8.0

---------------------------------------------------------------------
---------
ValueError                                Traceback (most recent c
all last)
<ipython-input-41-ab088b92b845> in <module>
      5
      6 print("p = ", p)
----> 7 plot(tt, v)
```

```
      8 plot(tt, u)

/usr/local/lib/python3.9/site-packages/matplotlib/pyplot.py in plo
t(scalex, scaley, data, *args, **kwargs)
   2838 @_copy_docstring_and_deprecators(Axes.plot)
   2839 def plot(*args, scalex=True, scaley=True, data=None, **kwa
rgs):
-> 2840     return gca().plot(
   2841         *args, scalex=scalex, scaley=scaley,
   2842         **({"data": data} if data is not None else {}), **
kwargs)

/usr/local/lib/python3.9/site-packages/matplotlib/axes/_axes.py in
plot(self, scalex, scaley, data, *args, **kwargs)
   1741         """
   1742         kwargs = cbook.normalize_kwargs(kwargs, mlines.Lin
e2D)
-> 1743         lines = [*self._get_lines(*args, data=data, **kwar
gs)]
   1744         for line in lines:
   1745             self.add_line(line)

/usr/local/lib/python3.9/site-packages/matplotlib/axes/_base.py in
__call__(self, data, *args, **kwargs)
    271                 this += args[0],
    272                 args = args[1:]
--> 273             yield from self._plot_args(this, kwargs)
    274
    275     def get_next_color(self):

/usr/local/lib/python3.9/site-packages/matplotlib/axes/_base.py in
_plot_args(self, tup, kwargs)
    397
    398         if x.shape[0] != y.shape[0]:
--> 399             raise ValueError(f"x and y must have same firs
t dimension, but "
    400                              f"have shapes {x.shape} and {
y.shape}")
    401         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shape
s (1000,) and (10000,)
```

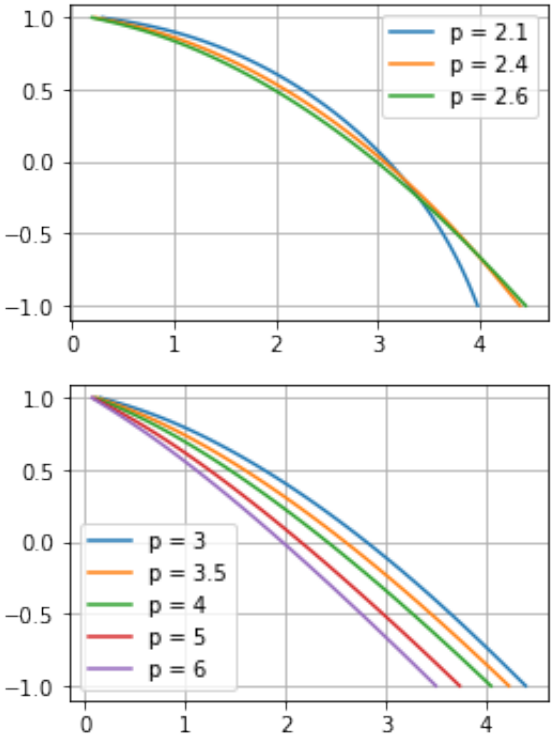


Computation of Λ for several values of p

Entrée [47]:
```
pp1 = [2.1, 2.4, 2.6]
pp2 = [3, 3.5, 4, 5, 6]
λλ = linspace(-1 + 1e-6, 1 -1e-6, 100)

Nb = 1000
tt = linspace(1e-6, 10, Nb)

Np1, Np2, Nλ = len(pp1), len(pp2), len(λλ)

alpha2_record1 = zeros((Np1, Nλ))
alpha2_record2 = zeros((Np2, Nλ))

for ip in range(Np1):
    xmin, xmax = 0, 50
    for iλ in range(Nλ):
        p, λ = pp1[ip], λλ[iλ]
        args = {"m": 1, "λ": λ, "p": p, "tt": tt, "n":0, "xmin":xmi

        psi, x0 = find_psi_ode_d2(args)
        alpha2_record1[ip, iλ] = alpha_d2(psi, args)
        xmin, xmax = maximum(0, x0-2), x0+2

for ip in range(Np2):
    xmin, xmax = 0, 50
    for iλ in range(Nλ):
        p, λ = pp2[ip], λλ[iλ]
        args = {"m": 1, "λ": λ,    "p": p, "tt": tt, "n":0, "xmin":

        psi, x0 = find_psi_ode_d2(args)
        alpha2_record2[ip, iλ] = alpha_d2(psi, args)
        xmin, xmax = maximum(0, x0-2), x0+2
```

```
λ = -0.999999, p = 2.1000, n = 0, xmin = 0.0000, xmax = 50.0000, x
0 = 7.8129
λ = -0.979797, p = 2.1000, n = 0, xmin = 5.8129, xmax = 9.8129, x0
= 7.6271
λ = -0.959595, p = 2.1000, n = 0, xmin = 5.6271, xmax = 9.6271, x0
= 7.4562
λ = -0.9393929999999999, p = 2.1000, n = 0, xmin = 5.4562, xmax =
9.4562, x0 = 7.2943
λ = -0.919191, p = 2.1000, n = 0, xmin = 5.2943, xmax = 9.2943, x0
= 7.1391
λ = -0.898989, p = 2.1000, n = 0, xmin = 5.1391, xmax = 9.1391, x0
= 6.9894
λ = -0.878787, p = 2.1000, n = 0, xmin = 4.9894, xmax = 8.9894, x0
= 6.8445
λ = -0.8585849999999999, p = 2.1000, n = 0, xmin = 4.8445, xmax =
8.8445, x0 = 6.7039
λ = -0.838383, p = 2.1000, n = 0, xmin = 4.7039, xmax = 8.7039, x0
= 6.5673
λ = -0.818181, p = 2.1000, n = 0, xmin = 4.5673, xmax = 8.5673, x0
```

Entrée [48]:
```
figsize(4, 6)
subplot(211)
for ip in range(Np1):
    p = pp1[ip]
    plot(alpha2_record1[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

subplot(212)
for ip in range(Np2):
    p = pp2[ip]
    plot(alpha2_record2[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

savefig('../Lambda_d2.png', dpi = 600, bbox_inches='tight')
```



Comparison n = 0 and n = -1

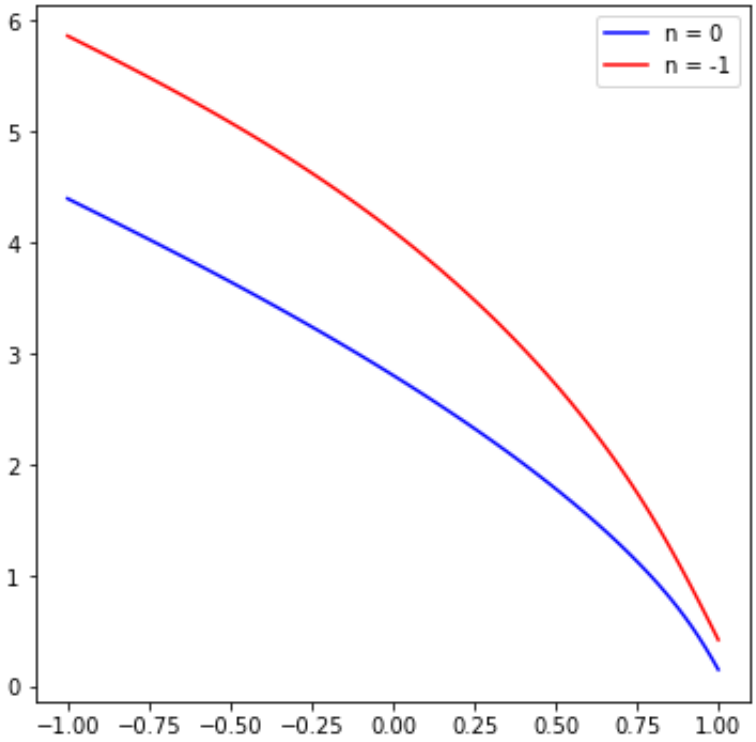
Entrée [168]:
```
p = 3
λλ = linspace(-1, 1 -1e-6, 100)
Nλ =  len(λλ)

Nb = 1000
tt = linspace(1e-6, 10, Nb)

alpha_record = zeros((2, Nλ)) # n = 0 and n = -1

for n in [0, -1]:
    xmin, xmax = 0, 50
    for iλ in range(Nλ):
        λ = λλ[iλ]
        args = {"m": 1, "λ": λ,    "p": p, "tt": tt, "n":n, "xmin":

        psi, x0 = find_psi_ode_d2(args)
        alpha_record[abs(n), iλ] = alpha_d2(psi, args)
        xmin, xmax = max(0, x0-2), x0+2
```

```
λ = -1.0, p = 3.0000, n = 0, xmin = 0.0000, xmax = 50.0000, x0 = 6
.0001
λ = -0.9797979898989899, p = 3.0000, n = 0, xmin = 4.0001, xmax =
8.0001, x0 = 5.9225
λ = -0.9595959797979798, p = 3.0000, n = 0, xmin = 3.9225, xmax =
7.9225, x0 = 5.8455
λ = -0.9393939696969698, p = 3.0000, n = 0, xmin = 3.8455, xmax =
7.8455, x0 = 5.7691
λ = -0.9191919595959596, p = 3.0000, n = 0, xmin = 3.7691, xmax =
7.7691, x0 = 5.6933
λ = -0.8989899494949495, p = 3.0000, n = 0, xmin = 3.6933, xmax =
7.6933, x0 = 5.6179
λ = -0.8787879393939394, p = 3.0000, n = 0, xmin = 3.6179, xmax =
7.6179, x0 = 5.5431
λ = -0.8585859292929293, p = 3.0000, n = 0, xmin = 3.5431, xmax =
7.5431, x0 = 5.4687
λ = -0.8383839191919192, p = 3.0000, n = 0, xmin = 3.4687, xmax =
7.4687, x0 = 5.3948
λ = -0.818181909090909, p = 3.0000, n = 0, xmin = 3.3948, xmax = 7
```

Entrée [169]:
```
print("alpha enfonction de λ pour n = 0 et n = -1")
plot(λλ, alpha_record[0, :], 'b', label="n = 0")
plot(λλ, alpha_record[1, :], 'r', label="n = -1")
legend()
```

```
alpha enfonction de λ pour n = 0 et n = -1
```

Out[169]:
```
<matplotlib.legend.Legend at 0x124f01ca0>
```



3D case

Entrée [89]:
```
# solves the equation with initial condition (x0, 0)
def solve_with_x0_d3(x0, args):

    # We solve the equation on the interval tt
    tt, m, λ, p, kappa = args["tt"], args["m"], args["λ"], args["p"

    def f3(t, y):
        v, u = y
        V = (u**2 + v**2)**(1/(p-1))
        dv = +(kappa-1)/t*u - ((λ + m) + V)*u
        du = -(kappa+1)/t*u + ((λ - m) + V)*v
        return array([dv, du])

    a, b = tt[0], tt[-1]

    if kappa==1:
        sol = solve_ivp(f3, t_span=[a, b], y0 = [x0, 0], t_eval=tt,
    #if n==-1:
    #    sol = solve_ivp(f3, t_span=[a, b], y0 = [0, x0], t_eval=tt
```

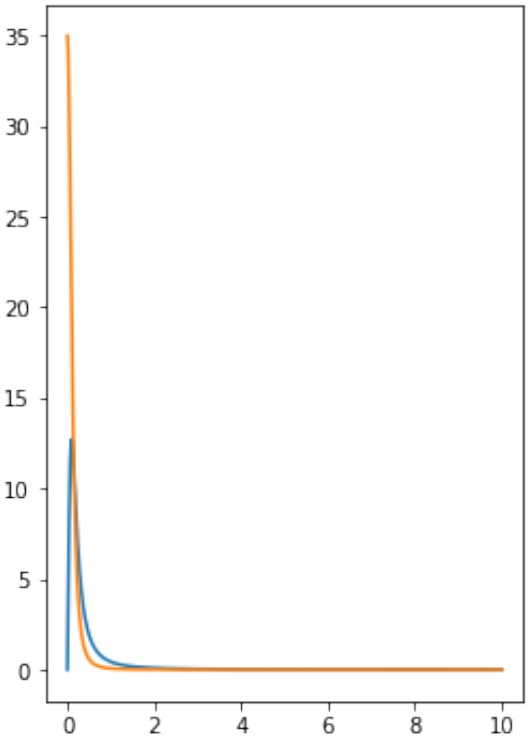
```
    return sol.y

# We find the best initial condition with a dichotomy
def find_psi_ode_d3(args):
    xmin, xmax = args["xmin"], args["xmax"]
    λ, p = args["λ"], args["p"]
    xmin0, xmax0 = xmin, xmax

    for n in range(100):
        x0 = (xmin + xmax)/2
        psi = solve_with_x0_d3(x0, args)
        v, u = psi

        # if good precision
        if norm (xmax - xmin) < 1e-7:
            print("λ = {:3}, p = {:.4f}, xmin = {:.4f}, xmax = {:.4
            return psi, x0

        # else dichotomy
        if min(v) < 1e-8: # the solution becomes negative: x0 is to
            xmin, xmax = xmin, x0
        else:
            xmin, xmax = x0, xmax
    print("problem")
    return psi, x0

def alpha_d3(psi, args):
    v, u = psi

    p, tt = args["p"], args["tt"]
    eps = tt[1] - tt[0]
    V = (v**2 + u**2)**(1/(p-1))
    integral = 4*pi*sum(tt**2*V**p)*eps
    return (integral)**(1/p)
```

Entrée [90]:
```
# test

Nb = 2000
tt = linspace(1e-6, 10, Nb)
m = 1
λ = -1
p = 3.3
xmin, xmax = 0, 200
kappa = 1

args = {
    "tt": tt,
    "m": m,
    "λ": λ,
    "p": p,
    "xmin": xmin,
    "xmax": xmax,
    "kappa": kappa
}

psi, x0 = find_psi_ode_d3(args)
v, u = psi

Vode = (abs(v)**2 + abs(u)**2)**(1/(p-1))
print("alpha = ", alpha_d3(psi, args) )
```

```
λ =  -1, p = 3.3000, xmin = 0.0000, xmax = 200.0000, x0 = 34.9776
alpha =  4.787741818934142
```

Entrée [91]:
```
v, u = psi
plot(tt, u)
plot(tt, v)
u[-1]
```

Out[91]: 0.0024207010390108 58



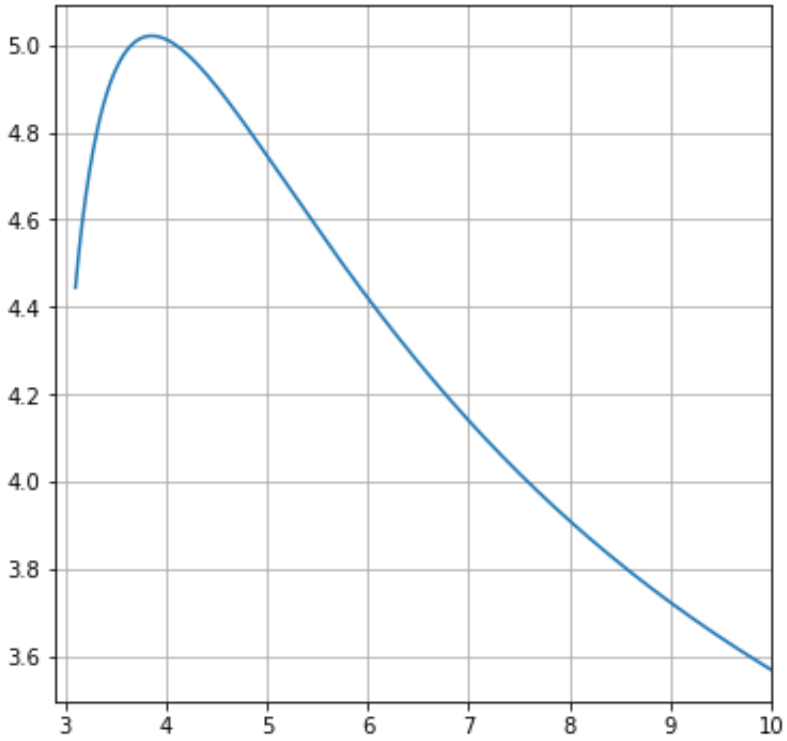
Computation of \alpha_c for $d = 3$

Entrée [92]:
```
### Plot \alpha_c
Nb = 1000
tt = linspace(1e-6, 10, Nb)
xmin, xmax = 0, 200

pp = linspace(3+0.1, 10, 200)
alphac3 = []
psi_record = []

for p in pp:

    args = {"m": 1, "λ": -1,  "p": p, "tt": tt, "kappa":1, "xmin":xm
    psi, x0 = find_psi_ode_d3(args)
    psi_record.append(psi)
    alphac3.append(alpha_d3(psi, args))
    xmin, xmax = maximum(0, x0-20), x0+10
```

```
λ =  -1, p = 3.1000, xmin = 0.0000, xmax = 200.0000, x0 = 76.2104
λ =  -1, p = 3.1347, xmin = 56.2104, xmax = 86.2104, x0 = 60.2917
λ =  -1, p = 3.1693, xmin = 40.2917, xmax = 70.2917, x0 = 50.8671
λ =  -1, p = 3.2040, xmin = 30.8671, xmax = 60.8671, x0 = 44.6448
λ =  -1, p = 3.2387, xmin = 24.6448, xmax = 54.6448, x0 = 40.2396
λ =  -1, p = 3.2734, xmin = 20.2396, xmax = 50.2396, x0 = 36.9664
λ =  -1, p = 3.3080, xmin = 16.9664, xmax = 46.9664, x0 = 34.4470
λ =  -1, p = 3.3427, xmin = 14.4470, xmax = 44.4470, x0 = 32.4553
λ =  -1, p = 3.3774, xmin = 12.4553, xmax = 42.4553, x0 = 30.8480
λ =  -1, p = 3.4121, xmin = 10.8480, xmax = 40.8480, x0 = 29.5297
λ =  -1, p = 3.4467, xmin = 9.5297, xmax = 39.5297, x0 = 28.4341
λ =  -1, p = 3.4814, xmin = 8.4341, xmax = 38.4341, x0 = 27.5143
λ =  -1, p = 3.5161, xmin = 7.5143, xmax = 37.5143, x0 = 26.7355
λ =  -1, p = 3.5508, xmin = 6.7355, xmax = 36.7355, x0 = 26.0719
λ =  -1, p = 3.5854, xmin = 6.0719, xmax = 36.0719, x0 = 25.5037
λ =  -1, p = 3.6201, xmin = 5.5037, xmax = 35.5037, x0 = 25.0153
λ =  -1, p = 3.6548, xmin = 5.0153, xmax = 35.0153, x0 = 24.5945
λ =  -1, p = 3.6894, xmin = 4.5945, xmax = 34.5945, x0 = 24.2317
λ =  -1, p = 3.7241, xmin = 4.2317, xmax = 34.2317, x0 = 23.9188
```

Entrée [94]:
```
figsize(6, 6)
plot(pp, alphac3)
#plot(pp, alpha0)
xlim(2.9,10)

grid()

savefig('../alphac_d3.png', dpi = 600, bbox_inches='tight')
print("maximum at p = {}".format(pp[argmax(alphac3)] ))
```

```
maximum at p = 3.862814070351759
```

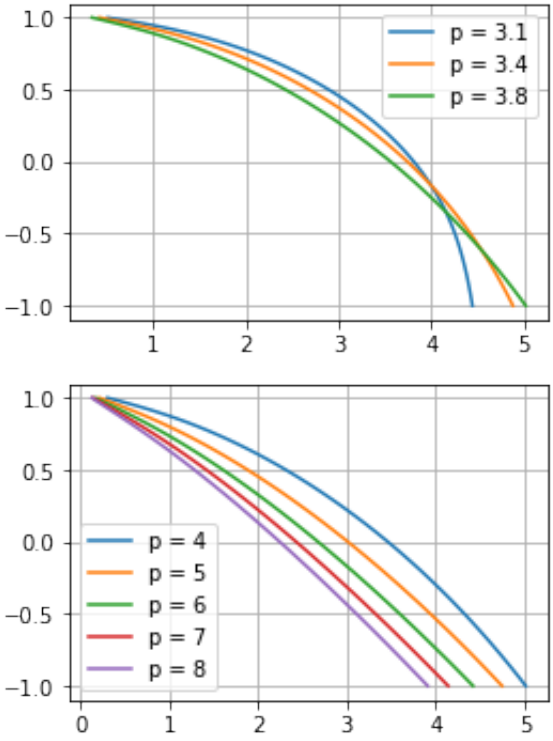


Entrée [99]:

```
pp1 = [3.1, 3.4, 3.8]
pp2 = [4, 5, 6, 7, 8]
λλ = linspace(-1+1e-6, 1 -1e-6, 100)

Nb = 1000
tt = linspace(1e-6, 10, Nb)

Np1, Np2, Nλ = len(pp1), len(pp2), len(λλ)

alpha3_record1 = zeros((Np1, Nλ))
alpha3_record2 = zeros((Np2, Nλ))

for ip in range(Np1):
    xmin, xmax = 0, 500
    for iλ in range(Nλ):
        p, λ = pp1[ip], λλ[iλ]
        args = {"m": 1, "λ": λ,    "p": p, "tt": tt, "kappa":1, "xm

        psi, x0 = find_psi_ode_d3(args)
        alpha3_record1[ip, iλ] = alpha_d3(psi, args)
        xmin, xmax = maximum(0, x0-10), x0+10

for ip in range(Np2):
    xmin, xmax = 0, 500
    for iλ in range(Nλ):
        p, λ = pp2[ip], λλ[iλ]
        args = {"m": 1, "λ": λ,    "p": p, "tt": tt, "kappa":1, "xm

        psi, x0 = find_psi_ode_d3(args)
        alpha3_record2[ip, iλ] = alpha_d3(psi, args)
        xmin, xmax = maximum(0, x0-10), x0+10
```

```
λ = 0.6767669999999998, p = 3.1000, xmin = 0.0000, xmax = 11.6534,
x0 = 1.5319
λ = 0.6969689999999998, p = 3.1000, xmin = 0.0000, xmax = 11.5319,
x0 = 1.4138
λ = 0.7171709999999999, p = 3.1000, xmin = 0.0000, xmax = 11.4138,
x0 = 1.2990
λ = 0.7373729999999997, p = 3.1000, xmin = 0.0000, xmax = 11.2990,
x0 = 1.1874
λ = 0.7575749999999998, p = 3.1000, xmin = 0.0000, xmax = 11.1874,
x0 = 1.0788
λ = 0.7777769999999998, p = 3.1000, xmin = 0.0000, xmax = 11.0788,
x0 = 0.9733
λ = 0.7979789999999999, p = 3.1000, xmin = 0.0000, xmax = 10.9733,
x0 = 0.8707
λ = 0.8181809999999999, p = 3.1000, xmin = 0.0000, xmax = 10.8707,
x0 = 0.7710
λ = 0.8383829999999998, p = 3.1000, xmin = 0.0000, xmax = 10.7710,
x0 = 0.6743
λ = 0.8585849999999998, p = 3.1000, xmin = 0.0000, xmax = 10.6743,
x0 = 0.5806
```

Entrée [100]:
```
figsize(4, 6)
subplot(211)
for ip in range(Np1):
    p = pp1[ip]
    plot(alpha3_record1[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

subplot(212)
for ip in range(Np2):
    p = pp2[ip]
    plot(alpha3_record2[ip, :], λλ, label='p = {}'.format(p))
legend()
grid()

savefig('../Lambda_d3.png', dpi = 600, bbox_inches='tight')
```



Entrée [ ]:

Entrée [ ]:

Birman_Schwinger_2d

Entrée [134]:
```
reset
```

```
Nothing done.
```

Entrée [1]:
```
%pylab inline
import os
import scipy.sparse.linalg as LA
import scipy.optimize as optimize
```

```
Populating the interactive namespace from numpy and matplotlib
```

2d Case ¶

In 2d, the Dirac operator is

$$D_m = \begin{pmatrix} m & \partial_x - i\partial_y \\ -\partial_x - i\partial_y & -m \end{pmatrix}.$$

We want to find the optimal of

$$\inf\left\{\langle\phi, K_W(\lambda)\phi\rangle, \quad \|\phi\|_{L^2} = 1, \quad \|W\|_{L^p} = 1\right\}.$$

with $1 < p \leq \infty$ and

$$K_W(\lambda) := \sqrt{W}R_0(\lambda)\sqrt{W}, \quad \text{with} \quad R_0(\lambda) = (D_m - \lambda)^{-1} = \frac{1}{-\Delta + m^2 - \lambda^2}(D_m +$$

We compute the Dirac operator (and its inverse), in Fourier space. Writing (with Python convention)

$$f(x) = \sum_{k=-M}^{M} f_k \exp(-ik\frac{2\pi}{2a}x),$$

we access the coefficients v_k with `fft.fft(v)` (and some shifts), and the operator $(-i\partial_x)$ becomes the multiplication by $-\frac{\pi}{a}k$.

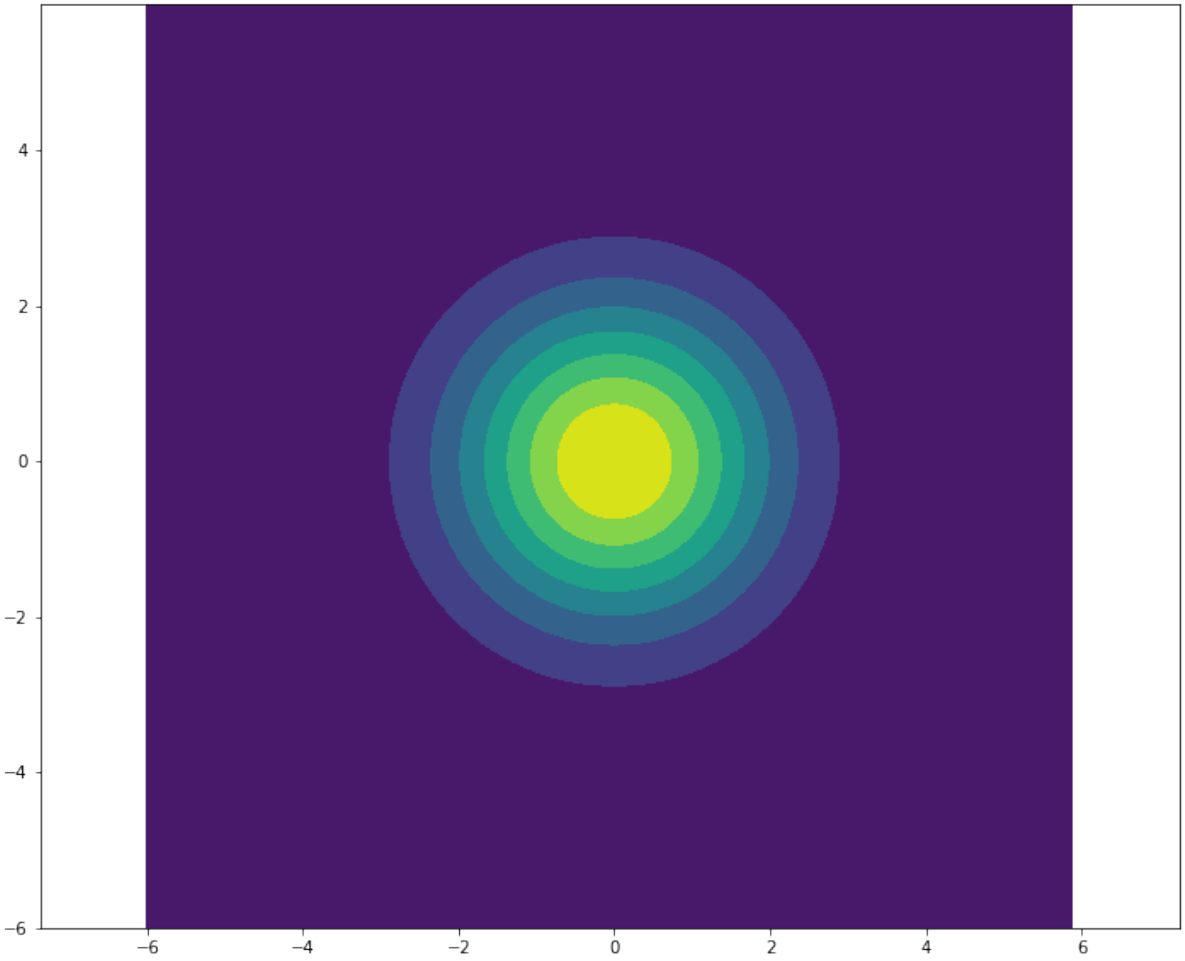
Entrée [2]:
```
## The grid to compute the potential
a = 6                          # we compute quantities on the interval
Nb = 100                        # number of discretization points. Must
xx = linspace(-a, a, Nb+1)     # The 1d grid
xx = xx[:Nb]
eps = xx[1] - xx[0]            # the step

xgrid, ygrid = meshgrid(xx, xx)

up, down = range(Nb**2), range(Nb**2, 2*Nb**2)
```

Entrée [3]:

```
m = 1 # the mass for Dirac

########################################
def get_Dirac_inverse(Nb, λ = 0):
    # returns the operator (Dirac - λ)^{-1}
    # λ is between -1, and 1

    KF = pi/a* roll( arange(-Nb//2+1, Nb//2+1), Nb//2 +1) # Fourier
    Kx, Ky = meshgrid(KF, KF)

    def mult_Dirac_m1(w):

        w_up, w_down = reshape(w[up], (Nb, Nb)), reshape(w[down], (N
        w_up_fft, w_down_fft = fft.fft2(w_up), fft.fft2(w_down)

        res_up_fft = (m + λ)*w_up_fft + (1j*Kx - Ky)*w_down_fft
        res_down_fft = (-1j*Kx - Ky)*w_up_fft - (m - λ)*w_down_fft

        res_up_fft *= 1/(Kx**2 + Ky**2 + m**2 - λ**2)
        res_down_fft *= 1/(Kx**2 + Ky**2 + m**2 - λ**2)

        res = zeros(2*Nb**2, dtype='complex')
        res[up] = reshape( fft.ifft2(res_up_fft) , Nb**2 )
        res[down] = reshape( fft.ifft2(res_down_fft) , Nb**2)

        return res

    return LA.LinearOperator((2*Nb**2, 2*Nb**2), matvec = mult_Dira
```

Entrée [4]:
```
figsize(12, 10)
V0 = 2*exp(-(xgrid**2 + ygrid**2)/4)
contourf(xx, xx, V0)
axis('equal')
```

Out[4]: (-6.0, 5.879999999999999, -6.0, 5.879999999999999)



We compute the infimum problem with a self-consistent loop. We set ϕ_n the highest normalized eigenvector of $K_{W_n}(\lambda)$, and set

$$W_{n+1} = |\phi_n|^{2/p}.$$

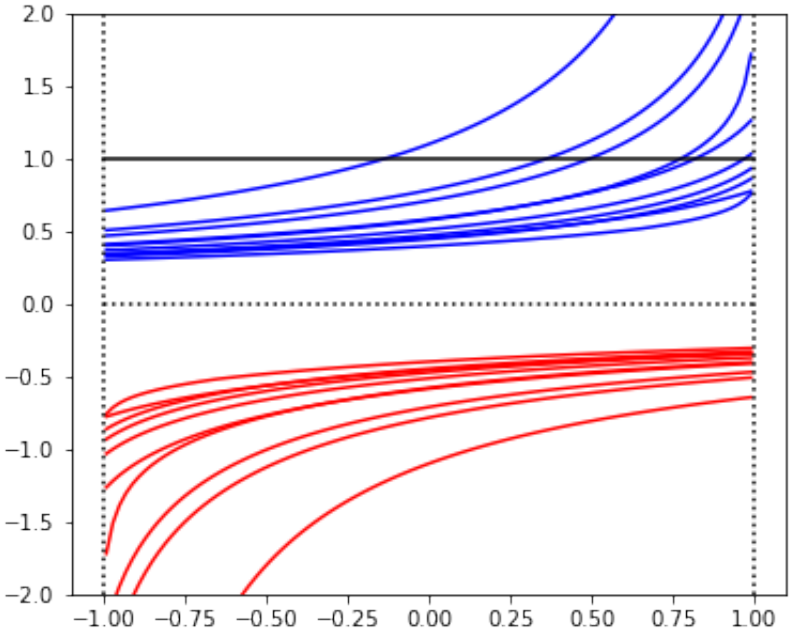
Entrée [5]:
```
def get_K(λ):
    Dirac_m1 = get_Dirac_inverse(Nb, λ)
    W = V0
    def mult_op(phi):
        sqrtW = zeros(2*Nb**2)
        sqrtW[up] = reshape(W**(1/2), Nb**2)
        sqrtW[down] = reshape(W**(1/2), Nb**2)

        phi1 = sqrtW*phi
        phi2 = Dirac_m1@phi1
        phi3 = sqrtW*phi2
        return phi3

    return LA.LinearOperator((2*Nb**2, 2*Nb**2), matvec = mult_op)
```

Entrée [6]:
```
# test cell
K = get_K(0.5)
shape(K)

eigval_pos, _ = LA.eigsh(K, k=10, which='LA')
eigval_neg, _ = LA.eigsh(K, k=10, which='SA')

print("positive eigenvalues = ", eigval_pos)
print("negative eigenvalues = ", eigval_neg)
```

```
positive eigenvalues =  [1.80252724 1.12633277 1.00589081 0.774950
75 0.76738202 0.65970116
 0.60084385 0.5631593  0.562614   0.48210895]
negative eigenvalues =  [-0.80655061 -0.61262709 -0.56161326 -0.47
967589 -0.47055514 -0.42696759
 -0.39631453 -0.38752154 -0.37619894 -0.34335281]
```

Entrée [7]:
```
# Compute eigenvalues on a grid
Nλ = 100
λλ = linspace(-0.99, 0.99, Nλ)

Neigs = 10
eig_pos = zeros((Nλ, Neigs))
eig_neg = zeros((Nλ, Neigs))

for iλ in range(Nλ):
    λ = λλ[iλ]
    K = get_K(λ)
    eig_pos[iλ, :], _ = LA.eigsh(K, k=Neigs, which='LA')
    eig_neg[iλ, :], _ = LA.eigsh(K, k=Neigs, which='SA')
```

Entrée [9]:
```
figsize(6, 5)
plot(λλ, eig_pos, 'b')
plot(λλ, eig_neg, 'r')

plot([-1, 1], [1, 1], 'k')

plot([1, 1], [-2, 2], ':k')
plot([-1, -1], [-2, 2], ':k')
plot([-1, 1], [0, 0], ':k')

ylim([-2, 2])

savefig("../spectrum_K_Dirac.png", dpi = 600, bbox_inches='tight')
```



Same computation with Laplace operator

Entrée [10]:

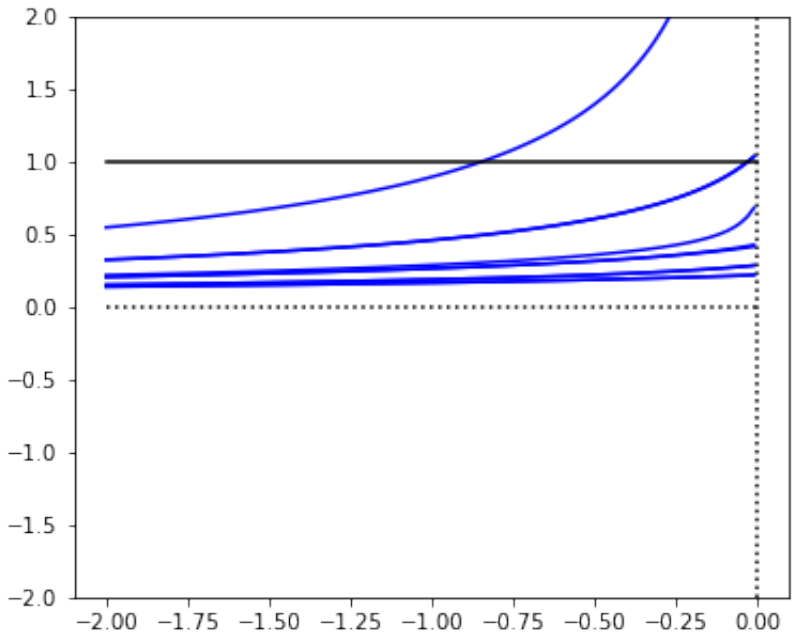
```
########################################
def get_Laplacian_inverse(Nb, λ = -1):
    # returns the operator ((-\Delta) - λ)^{-1}
    # λ is negative

    KF = pi/a* roll( arange(-Nb//2+1, Nb//2+1), Nb//2 +1) # Fourier
    Kx, Ky = meshgrid(KF, KF)

    def mult_Laplace_m1(w):
        w_fft = fft.fft2( reshape(w, (Nb, Nb)) )
        res_fft = w_fft/(Kx**2 + Ky**2 - λ)
        res = reshape( fft.ifft2(res_fft) , Nb**2 )
        return res

    return LA.LinearOperator((Nb**2, Nb**2), matvec = mult_Laplace_
```

Entrée [11]:

```
def get_tildeK(λ):
    Laplace_m1 = get_Laplacian_inverse(Nb, λ)
    W = V0
    def mult_op(phi):
        sqrtW = reshape(W**(1/2), Nb**2)

        phi1 = sqrtW*phi
        phi2 = Laplace_m1@phi1
        phi3 = sqrtW*phi2
        return phi3

    return LA.LinearOperator( (Nb**2, Nb**2), matvec = mult_op)
```

Entrée [12]:
```
# Compute eigenvalues on a grid
Nλ = 100
λλ = linspace(-2, -0.01, Nλ)

Neigs = 10
eigval = zeros((Nλ, Neigs))

for iλ in range(Nλ):
    λ = λλ[iλ]
    tildeK = get_tildeK(λ)
    eig, _ =  LA.eigsh(tildeK, k=Neigs, which='LA')
    eigval[iλ, :] = sort(eig)
```

Entrée [13]:
```
figsize(6, 5)
plot(λλ, eigval, 'b')

plot([-2, 0], [1, 1], 'k')
plot([0, 0], [-2, 2], ':k')
plot([-2, 0], [0, 0], ':k')

ylim([-2, 2])
savefig("../spectrum_K_Laplace.png", dpi = 600, bbox_inches='tight'
```



Entrée [ ]:

Keller_2d

Entrée [118]:
```
reset
```

Entrée [1]:
```
%pylab inline
import os
import scipy.sparse.linalg as LA
import scipy.optimize as optimize
```

```
Populating the interactive namespace from numpy and matplotlib
```

2d Case

In 2d, the Dirac operator is

$$D_m = \begin{pmatrix} m & \partial_x - i\partial_y \\ -\partial_x - i\partial_y & -m \end{pmatrix}.$$

We want to find the optimal of

$$\inf\left\{\langle\phi, K_W(\lambda)\phi\rangle, \quad \|\phi\|_{L^2} = 1, \quad \|W\|_{L^p} = 1\right\}.$$

with $1 < p \leq \infty$ and

$$K_W(\lambda) := \sqrt{W}R_0(\lambda)\sqrt{W}, \quad \text{with} \quad R_0(\lambda) = (D_m - \lambda)^{-1} = \frac{1}{-\Delta + m^2 - \lambda^2}(D_m +$$

Entrée [2]:
```
## The grid to compute the Lp integrals
a = 6                          # we compute quantities on the interval
Nb = 100                        # number of discretization points. Must
xx = linspace(-a, a, Nb+1)     # The 1d grid
xx = xx[:Nb]
eps = xx[1] - xx[0]            # the step

xgrid, ygrid = meshgrid(xx, xx)

up, down = range(Nb**2), range(Nb**2, 2*Nb**2)

## The Lp-norm
def Lp_norm(f, p):
    if len(f) == 2*Nb**2:
        # If f is a 2*Nb vector, the first Nb^2 entries are for spi
        absv = sqrt(abs(f[up])**2 + abs(f[down])**2)
        integral = sum(absv**p)*eps**2
        return integral**(1/p)
    else: # for the potential, non spins
        integral = sum(abs(f)**p )*eps**2
        return integral**(1/p)
```

We compute the Dirac operator (and its inverse), in Fourier space. Writing (with Python convention)

$$f(x) = \sum_{k=-M}^{M} f_k \exp(-ik\frac{2\pi}{2a}x),$$

we access the coefficients v_k with `fft.fft(v)` (and some shifts), and the operator $(-i\partial_x)$ becomes the multiplication by $-\frac{\pi}{a}k$.

Entrée [3]:

```
m = 1 # the mass

#######################################
def get_Dirac(Nb, λ = 0):
    # returns the operator (Dirac - λ)

    K = pi/a* roll( arange(-Nb//2+1, Nb//2+1), Nb//2 +1) # Fourier
    Kx, Ky = meshgrid(K, K)

    def mult_Dirac(w):
        w_up, w_down = reshape(w[up], (Nb, Nb)), reshape(w[down], (
        w_up_fft, w_down_fft = fft.fft2(w_up), fft.fft2(w_down)

        res_up_fft = (m - λ)*w_up_fft + (1j*Kx - Ky)*w_down_fft
        res_down_fft = (-1j*Kx - Ky)*w_up_fft - (m + λ)*w_down_fft

        res = zeros(2*Nb**2, dtype='complex')
        res[up] = reshape( fft.ifft2(res_up_fft) , Nb**2 )
        res[down] = reshape( fft.ifft2(res_down_fft) , Nb**2)

        return res
    return LA.LinearOperator((2*Nb**2, 2*Nb**2), matvec = mult_Dira

#######################################
def get_Dirac_inverse(Nb, λ = 0):
    # returns the operator (Dirac - λ)^{-1}

    K = pi/a* roll( arange(-Nb//2+1, Nb//2+1), Nb//2 +1) # Fourier
    Kx, Ky = meshgrid(K, K)

    def mult_Dirac_m1(w):

        w_up, w_down = reshape(w[up], (Nb, Nb)), reshape(w[down], (
        w_up_fft, w_down_fft = fft.fft2(w_up), fft.fft2(w_down)

        res_up_fft = (m + λ)*w_up_fft + (1j*Kx - Ky)*w_down_fft
        res_down_fft = (-1j*Kx - Ky)*w_up_fft - (m - λ)*w_down_fft

        res_up_fft *= 1/(Kx**2 + Ky**2 + m**2 - λ**2)
        res_down_fft *= 1/(Kx**2 + Ky**2 + m**2 - λ**2)

        res = zeros(2*Nb**2, dtype='complex')
        res[up] = reshape( fft.ifft2(res_up_fft) , Nb**2 )
        res[down] = reshape( fft.ifft2(res_down_fft) , Nb**2)

        return res

    return LA.LinearOperator((2*Nb**2, 2*Nb**2), matvec = mult_Dira
```

Entrée [4]:
```
##################################
# Check cell

Dirac, Dirac_m1 = get_Dirac(Nb), get_Dirac_inverse(Nb)

print("Check self-adjointness")
u = rand(2*Nb**2) + 1j*rand(2*Nb**2)
v = rand(2*Nb**2) + 1j*rand(2*Nb**2)
print("\t must be 0 : ", norm ( dot(conj(u),Dirac@v) - dot(conj(Dir

print("Check invertibility")
u = rand(2*Nb**2) + 1j*rand(2*Nb**2)
print("\t must be 0 : ", norm ( Dirac_m1@(Dirac@u) - u ) )
```

```
Check self-adjointness
         must be 0 :  1.5463081705019526e-12
Check invertibility
         must be 0 :  5.371017150109813e-14
```

We compute the infimum problem with a self-consistent loop. We set ϕ_n the highest normalized eigenvector of $K_{W_n}(\lambda)$, and set

$$W_{n+1} = |\phi_n|^{2/p}.$$

Entrée [5]:
```
def get_W(phi, p):
    phi_up, phi_down = reshape(phi[up], (Nb, Nb)) , reshape(phi[dow
    W = (abs(phi_up)**2 + abs(phi_down)**2)**(1/p)
    # Shift to center
    imax = argmax(W)
    i, j = imax//Nb, mod(imax, Nb)
    W1 = roll( W, -i + Nb//2, axis=0)
    W2 = roll(W1, -j + Nb//2, axis=1)
    return W2

def get_KW(W, Dirac_m1=Dirac_m1):
    def mult_op(phi):
        sqrtW = zeros(2*Nb**2)
        sqrtW[up] = reshape(W**(1/2), Nb**2)
        sqrtW[down] = reshape(W**(1/2), Nb**2)

        phi1 = sqrtW*phi
        phi2 = Dirac_m1@phi1
        phi3 = sqrtW*phi2
        return phi3

    return LA.LinearOperator((2*Nb**2, 2*Nb**2), matvec = mult_op)

def energy(KW, phi):
    return real( dot(conj(phi), KW@phi)*eps**2 )
```

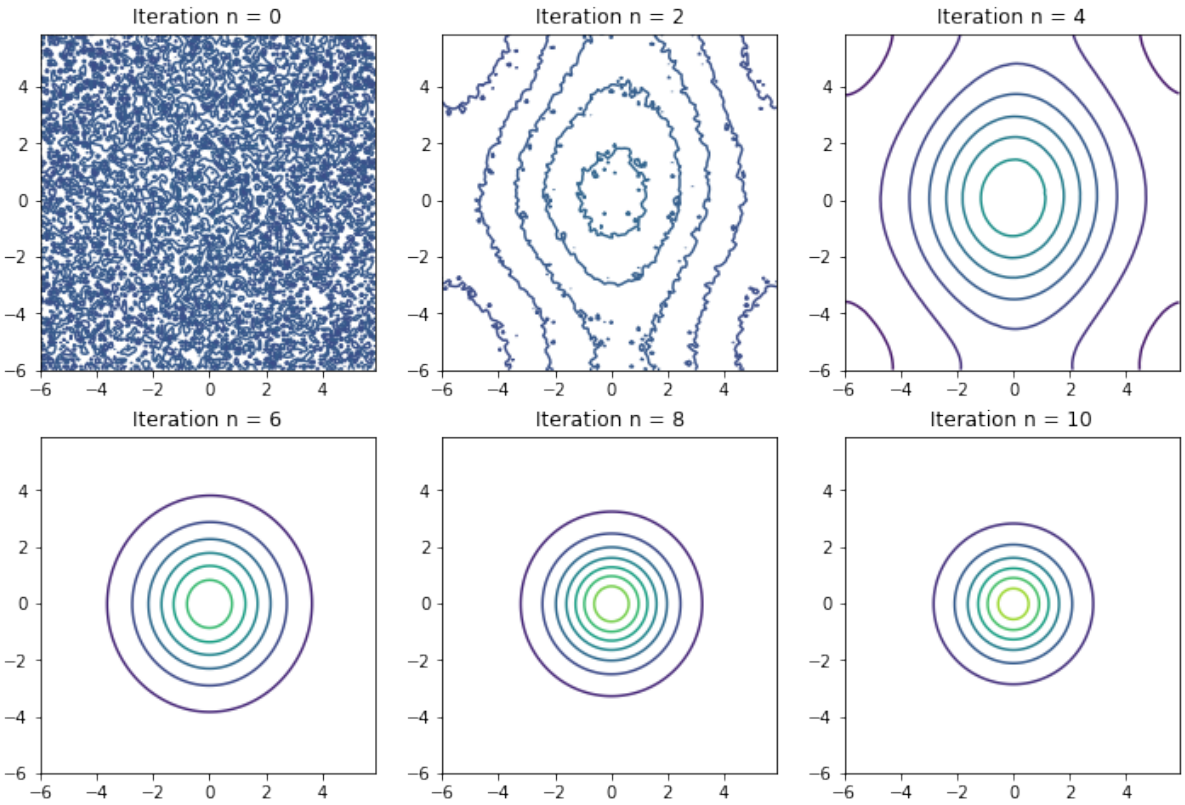
Entrée [6]:

```
def get_optimalW(λ, p):

    Wrecord = []

    print("\n\nComputation for p = {} and λ = {}".format(p, λ))

    # Initialization vector
    phi0 = zeros(2*Nb**2, dtype='complex')
    phi0 = rand(2*Nb**2) + 1j*rand(2*Nb**2)
    phi0 = phi0/Lp_norm(phi0, 2)
    W0 = get_W(phi0, p)

    # Parameters of the loop
    Niter = 100    # maximal number of iterations
    tol = 1e-7      # tolerance

    ##################################
    Dirac_m1 = get_Dirac_inverse(Nb, λ)

    ##################################
    # Main loop
    En, Wn, phin = -1, W0, phi0

    for n in range(Niter):
        KWn = get_KW(Wn, Dirac_m1)
        eigval, eigvec = LA.eigsh(KWn, k=1, which='LA') #largest ei

        phinp1, Enp1 = eigvec[:,0], eigval[0]
        phinp1 = phinp1/Lp_norm(phinp1, 2)

        if Enp1 < En:
            print("problem, the energy is not increasing...")
            break

        Wnp1 = get_W(phinp1, p)
        if norm(Wnp1 - Wn) < tol:
            break

        if mod(n, 10) == 0: #print
            print("\tIteration n = {:4}, Energy = {}, norm = {}".fo

        En, Wn, phin = Enp1, Wnp1, phinp1
        Wrecord.append(Wn)

    phistar, Wstar, tau = phin, Wn, Enp1
    print("\n\tDone. Number of iterations = {}, Energy = {:.4}, tau
    return Wstar, Wrecord
```

Entrée [7]:
```
## Test cell
p = 3
λ = 0.5

Wstar, Wrecord = get_optimalW(λ, p)
```

```
Computation for p = 3 and λ = 0.5
        Iteration n =    0, Energy = -1, norm = 2.0407521101002972
        Iteration n =   10, Energy = 0.5590547759064945, norm = 0.
356313505698372
        Iteration n =   20, Energy = 0.5613230029283961, norm = 0.
005301816543182837
        Iteration n =   30, Energy = 0.5613236174687144, norm = 8.
604889318319096e-05
        Iteration n =   40, Energy = 0.5613236176319182, norm = 1.
4015348692814811e-06

        Done. Number of iterations = 47, Energy = 0.5613, tau = 0.
5613
```

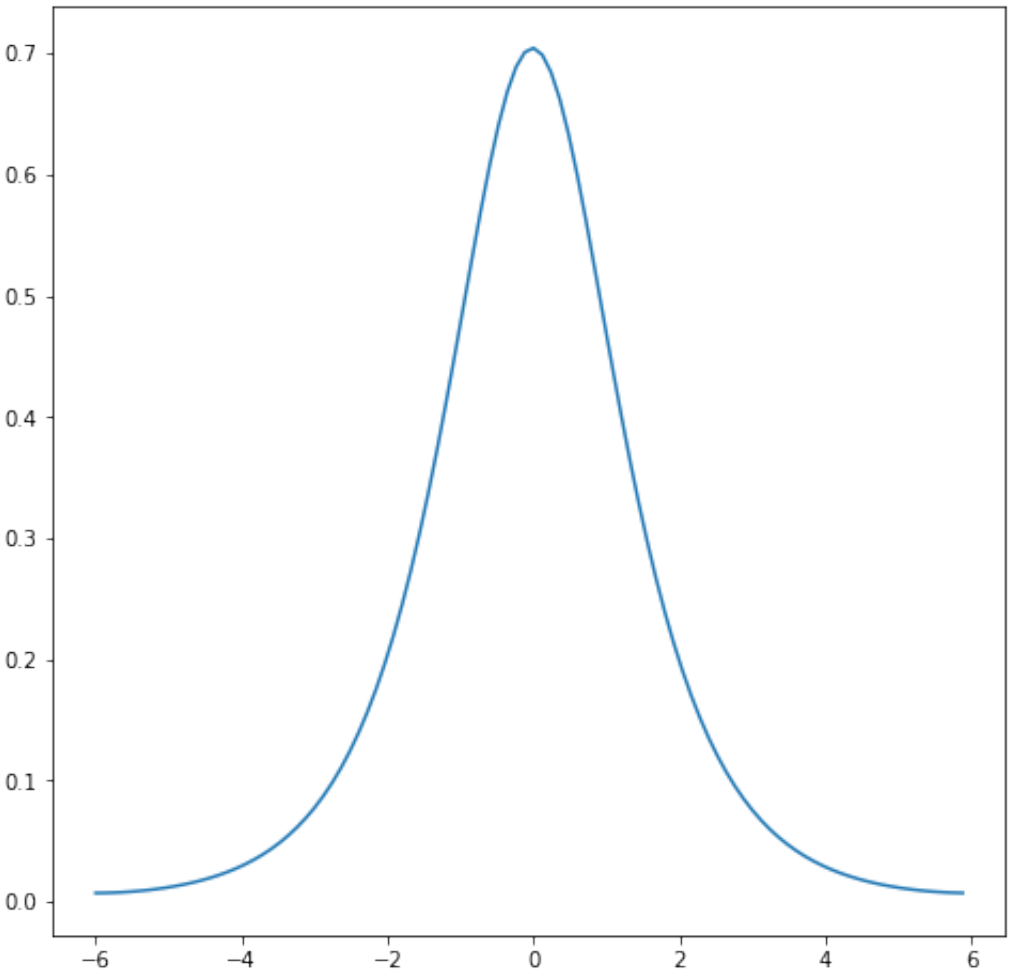
Entrée [8]:
```
figsize(12, 8)

for i in range(6):
    subplot(2,3,i+1)
    snapshot =  2*i
    W = Wrecord[snapshot]

    contour(xx, xx, W, vmin=0, vmax = 0.7)
    #colorbar()
    axis('equal')
    title("Iteration n = {}".format(snapshot))

savefig("Wn_movie.png", dpi = 600, bbox_inches = 'tight')
```



Entrée [9]:
```
# the radial part
print("should be 0: ", xx[Nb//2]) # check the middle point
plot(xx, Wstar[:, Nb//2])
```

```
should be 0:  0.0
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x1214933a0>]
```



We compute dtheta of a function with the formula

$$\partial_\theta = -r\sin\theta\cdot\partial_x f + r\cos\theta\cdot\partial_y f = -y\cdot\partial_x f + x\cdot\partial_y f$$

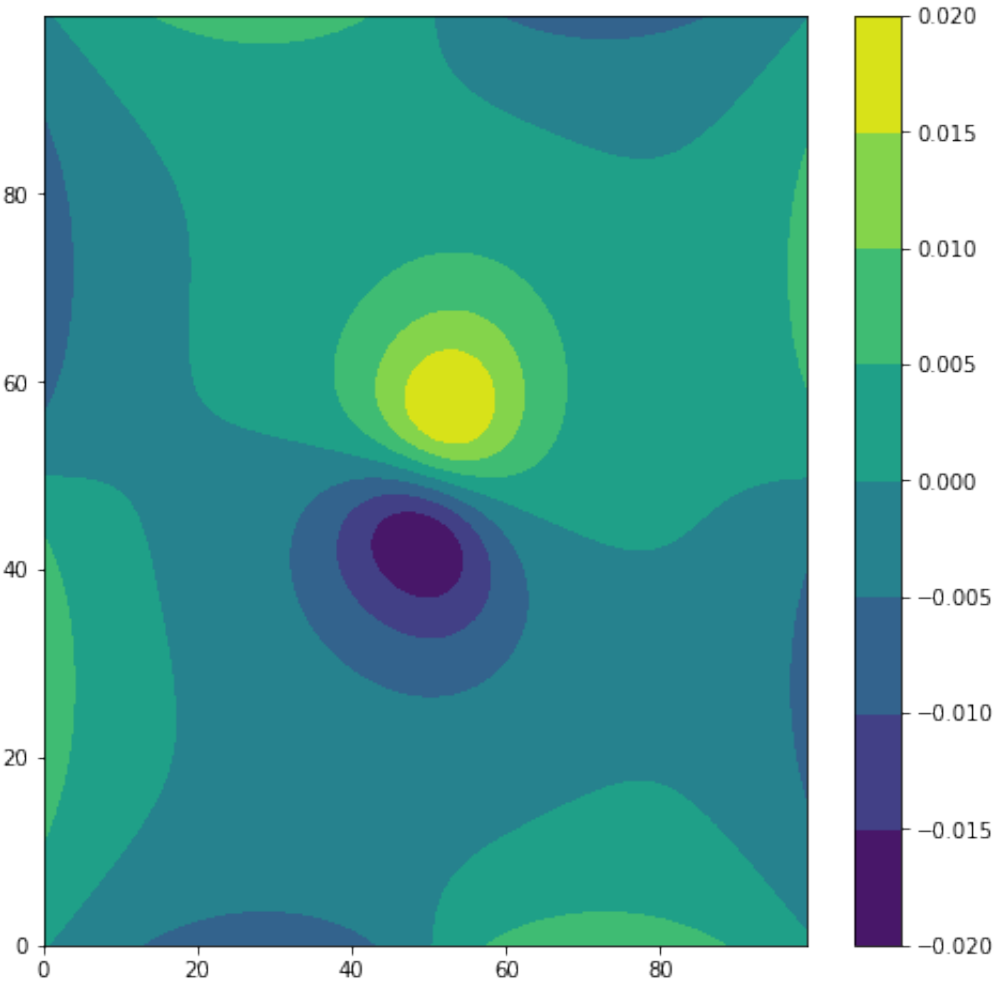
Entrée [16]:
```
def dtheta(F):
    dxF = (roll(F, 1, axis=1) - roll(F, -1, axis=1))/eps/2
    dyF = (roll(F, 1, axis=0) - roll(F, -1, axis=0))/eps/2
    return -ygrid*dxF + xgrid*dyF
```

Entrée [18]:
```
dtheta_W = dtheta(Wstar)
Test = exp(-xgrid**2 - ygrid**2)

Lp_norm(dtheta_W, p)
#Lp_norm(dtheta(Test), p)
```

Out[18]: 0.031073398968935694

Entrée [20]:
```
contourf(dtheta_W)
colorbar()
```

Out[20]: <matplotlib.colorbar.Colorbar at 0x1220f1460>



Check radiality in a range

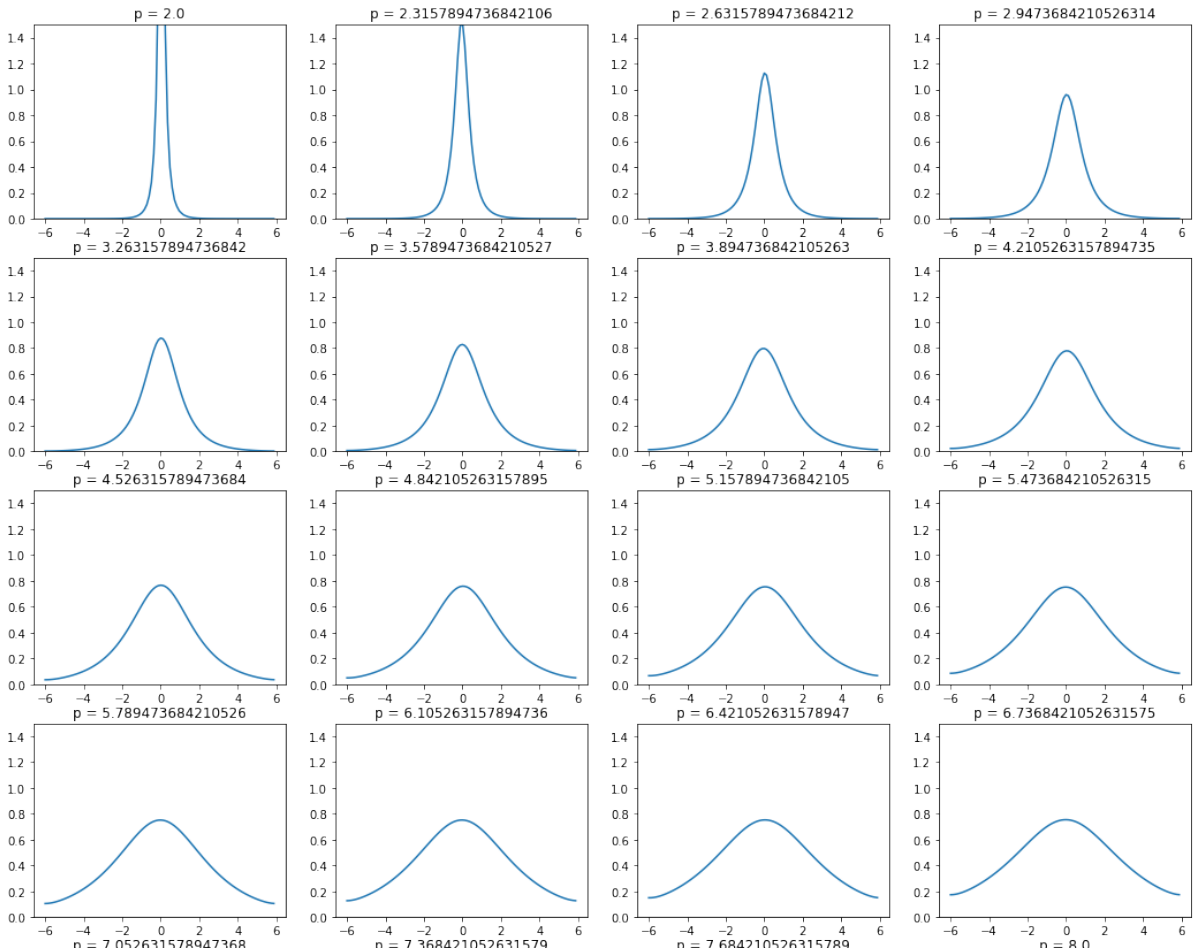
Entrée [21]:
```
# The computations are done with the parameters
# a = 6                        # we compute quantities on the interva
# Nb = 200                      # number of discretization points. Mu
# xx = linspace(-a, a, Nb+1)    # The 1d grid
# xx = xx[:Nb]
# eps = xx[1] - xx[0]           # the step

ll = [0.9, 0.5, 0, -0.5, -0.9]
pp = linspace(2, 8, 20)

for λ in ll:
    for p in pp:
        Wstar = get_optimalW(λ, p)

        data = {"λ": λ, "p": p, "Wstar": Wstar, "a": a, "Nb": Nb}
        np.save("data/data_p{:d}_l{:d}.npy".format(int(100*p), int(
```

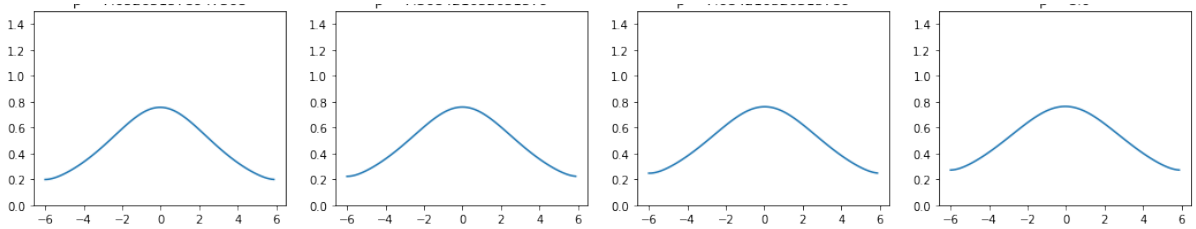
```
Computation for p = 2.0 and λ = 0.9
        Iteration n =    0, Energy = -1, norm = 0.9290807759848871
        Iteration n =   10, Energy = 0.8336341112196622, norm = 0.
09296957159762986
        Iteration n =   20, Energy = 0.8502252622754524, norm = 0.
637723662430845
        Iteration n =   30, Energy = 0.9508935786143167, norm = 0.
20703546312823623
        Iteration n =   40, Energy = 0.9552309352893624, norm = 0.
017711854973982987
        Iteration n =   50, Energy = 0.9552629578823114, norm = 0.
001482550788082295
        Iteration n =   60, Energy = 0.9552631822205534, norm = 0.
00012368222946888824
        Iteration n =   70, Energy = 0.9552631837818608, norm = 1.
0314918022317574e-05
        Iteration n =   80, Energy = 0.9552631837927209, norm = 8.
```

Entrée [29]:
```

```

```
figsize(18, 18)

Np, Nλ = len(pp), len(ll)

iλ = 2
λ = ll[iλ]
print("λ = ", λ)

error = []

for ip in range(Np):
    λ, p = ll[iλ], pp[ip]
    name = "data/data_p{:d}_l{:d}.npy".format(int(100*p), int(10*λ))

    data = np.load(name, allow_pickle=True).item()

    Wstar = data["Wstar"]
    subplot(5,4,ip+1)

    plot(xx, Wstar[:, Nb//2])
    title("p = {}".format(p))
    ylim(0, 1.5)

    error.append( Lp_norm(dtheta(Wstar), p) )
```

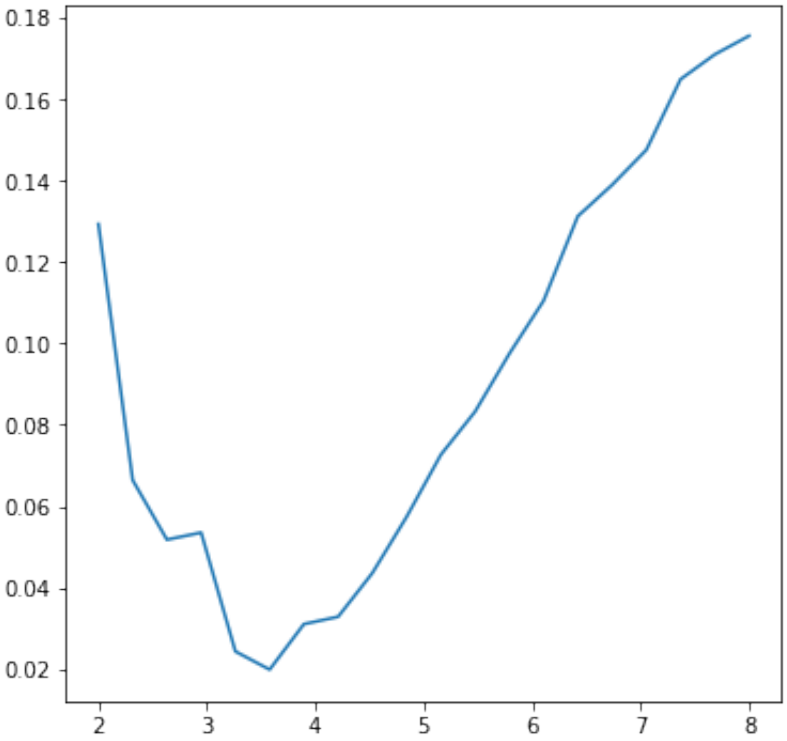
```
λ =  0
```





Entrée [30]:
```
figsize(6, 6)
plot(pp, error)
```

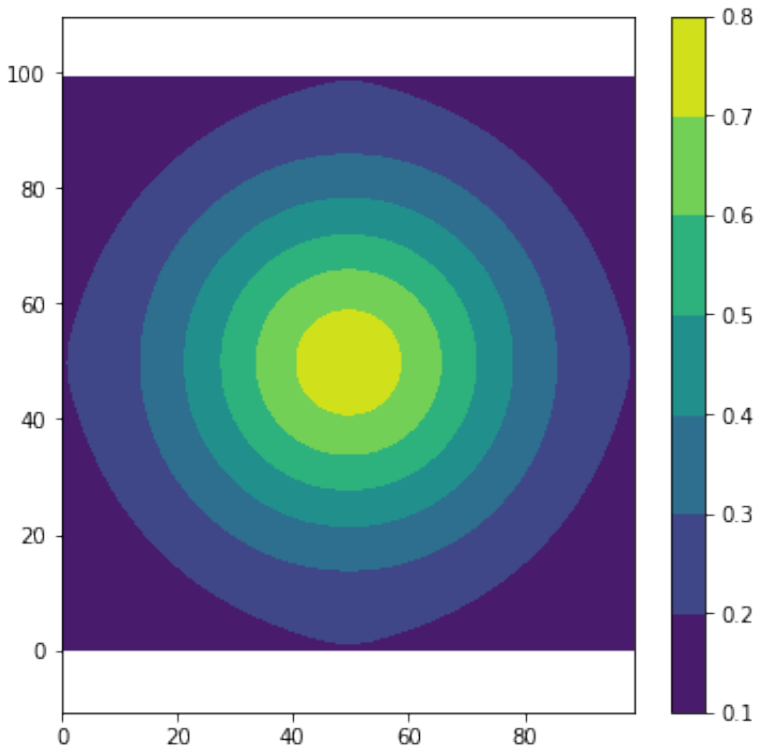
Out[30]:
```
[<matplotlib.lines.Line2D at 0x1225ce490>]
```



Entrée [31]:
```
iλ = 2
ip = 16

λ, p = ll[iλ], pp[ip]
print(" p = {}, λ = {} ".format(p, λ))
name = "data/data_p{:d}_l{:d}.npy".format(int(100*p), int(10*λ))
data = np.load(name, allow_pickle=True).item()
Wstar = data["Wstar"]

contourf(Wstar)
colorbar()
axis('equal')
```

```
 p = 7.052631578947368, λ = 0
```

Out[31]:
```
(0.0, 99.0, 0.0, 99.0)
```



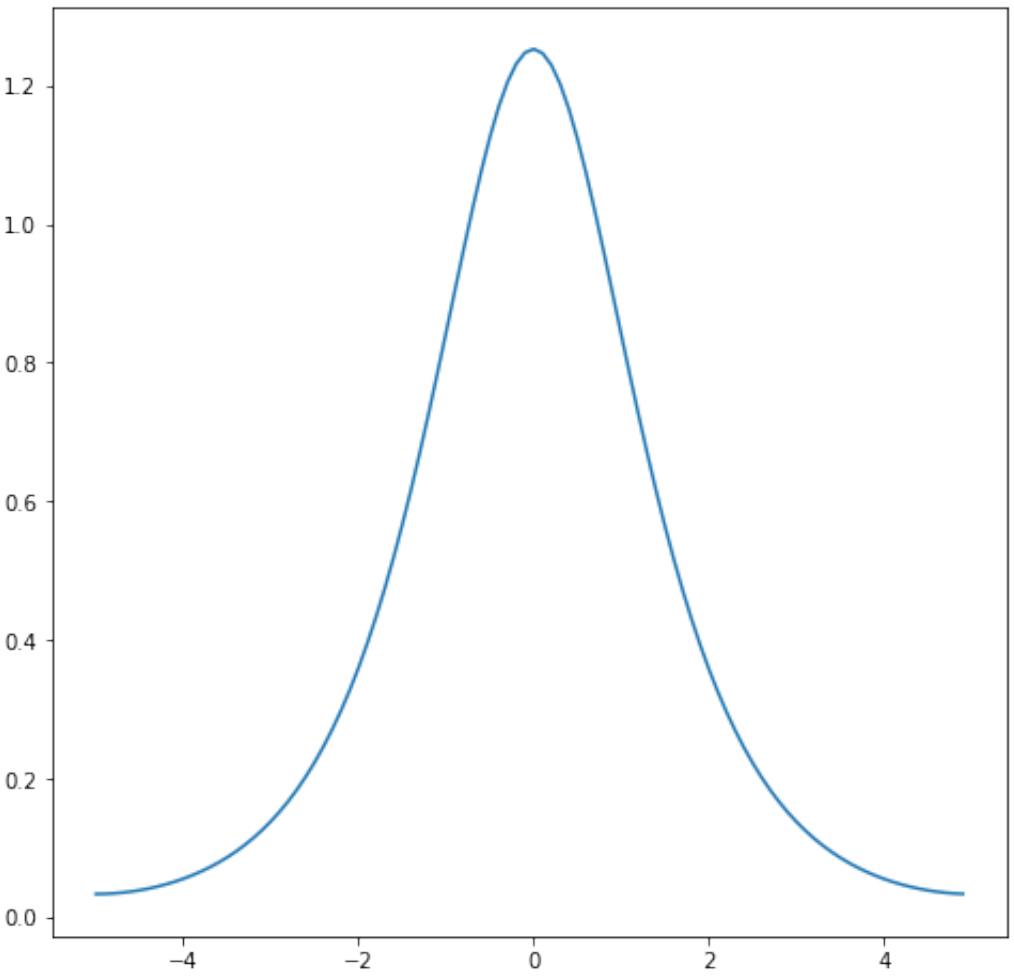
Eigenfunctions

Entrée [144]:
```
# Check the corresponding eigenfunction

psi_up = tau**((1-p)/2)*Wstar**(-1/2)*reshape(phistar[up], (Nb, Nb)
psi_down = tau**((1-p)/2)*Wstar**(-1/2)*reshape(phistar[down], (Nb,

psi = zeros(2*Nb**2, dtype='complex')
psi[up] = reshape(psi_up, Nb**2)
psi[down] = reshape(psi_down, Nb**2)

V = (abs(psi_up)**2 + abs(psi_down)**2)**(1/(p-1))

V2 = zeros(2*Nb**2, dtype='complex')
V2[up] = reshape(V, Nb**2)
V2[down] = reshape(V, Nb**2)

print("should be 0:" , norm(Dirac@psi - V2*psi))
```

```
should be 0: 5.060123772771483e-07
```

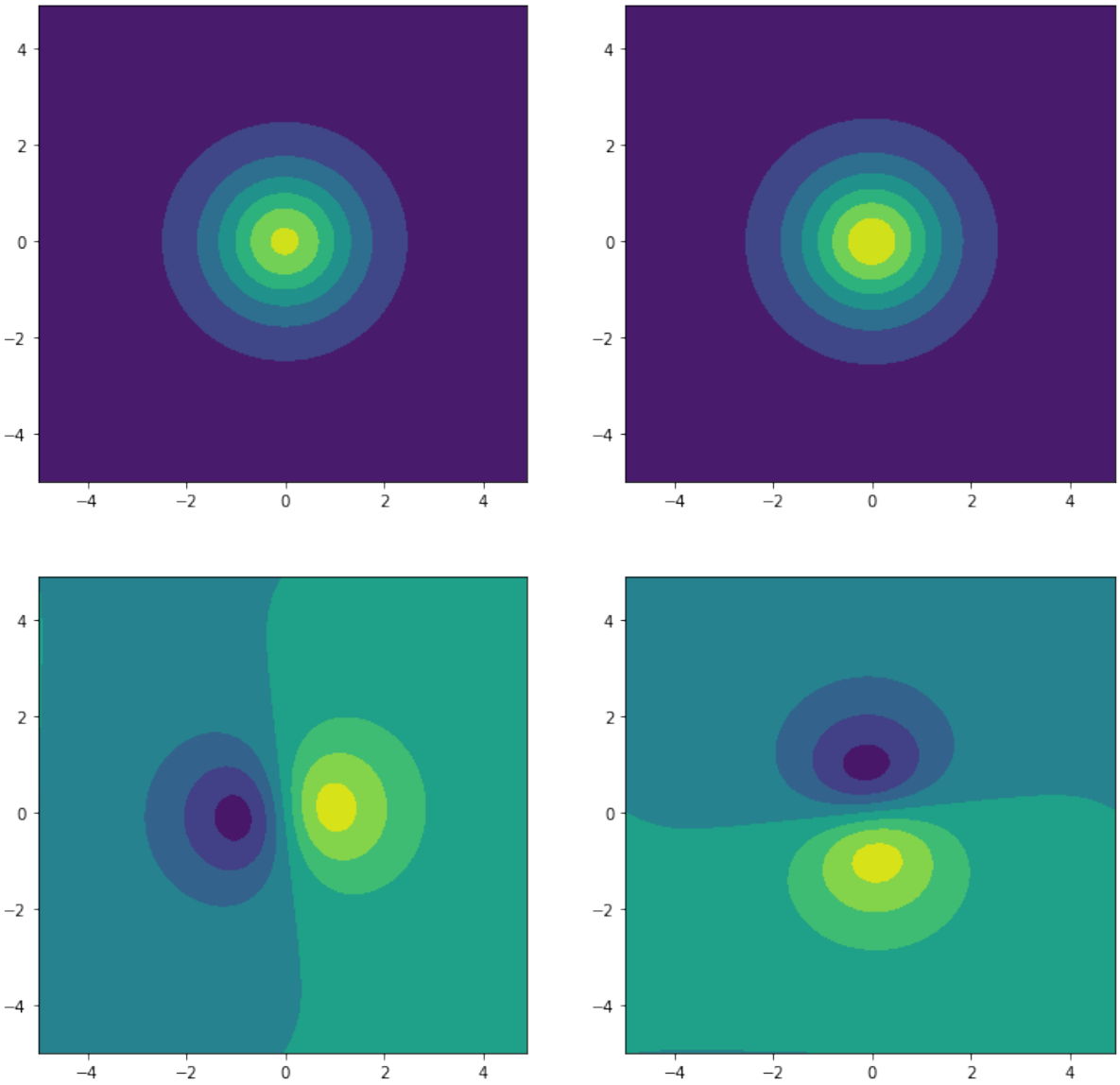
Entrée [145]:
```
# the radial part
plot(xx, V[:, Nb//2])
```

Out[145]:
```
[<matplotlib.lines.Line2D at 0x122a9e9d0>]
```



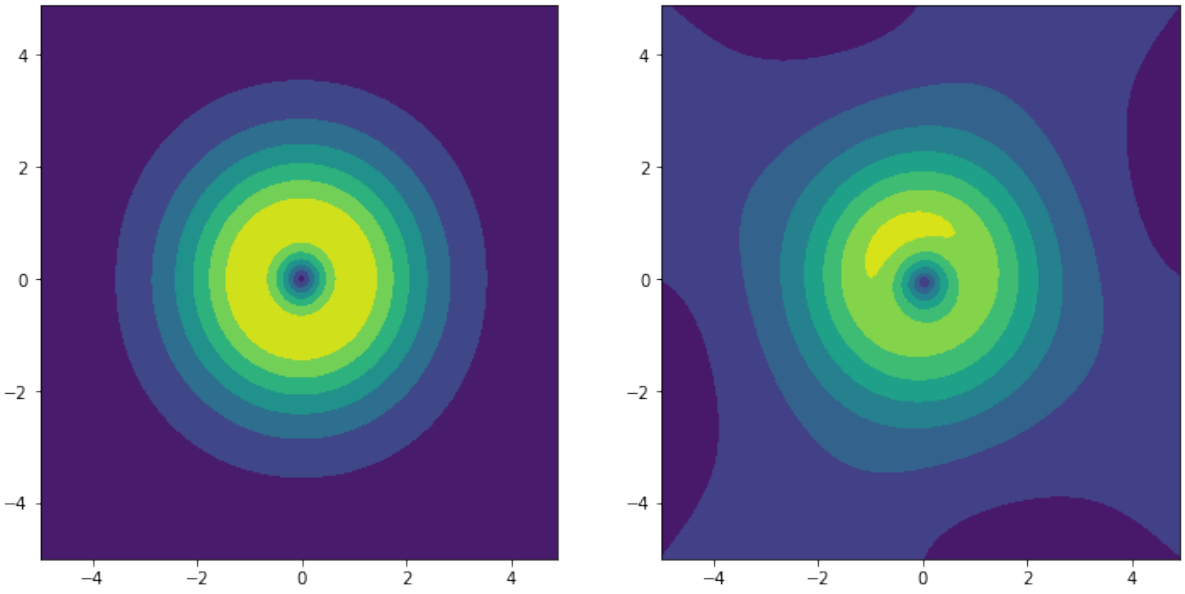
Entrée [146]:
```
figsize(12, 12)
subplot(221)
contourf(xx, xx, real(psi_up))
subplot(222)
contourf(xx, xx, imag(psi_up))

subplot(223)
contourf(xx, xx, real(psi_down))
subplot(224)
contourf(xx, xx, imag(psi_down))
```

Out[146]: <matplotlib.contour.QuadContourSet at 0x1232922b0>



Entrée [147]:
```
figsize(12, 6)
# check radial * exp ?
theta = arctan2(ygrid, xgrid)
rad_down = psi_down*exp(1j*theta)

subplot(121)
contourf(xx, xx, real(rad_down))
subplot(122)
contourf(xx, xx, imag(rad_down))
```

Out[147]:
```
<matplotlib.contour.QuadContourSet at 0x123385730>
```



Type *Markdown* and LaTeX: α^2