

Trefethen p28 to p40.

This notebook showcases the third fifteen problems in Trefethen's classic book *Spectral Methods in MATLAB*. These problems have been ported to Python by Orlando Camargo Rodríguez.

Program 28 : Eigenmodes of the laplacian on the disk

Note: the "cheb" shown here is the cheb of Rodríguez, which is distinct from that of CPraveen.

In [2]:

```
1 from numpy import *
2 from numpy import matlib
3
4 def cheb(N):
5     # CHEB compute D = differentiation matrix, x = Chebyshev grid
6     D = []
7     x = []
8     if N==0:
9         D = 0.0
10        x = 1.0
11    else:
12        i = arange(0,N+1)
13        x = cos( pi*i/N )
14        c = ones(N+1)
15        c[ 0] = 2.0
16        c[-1] = 2.0
17        c = c*( -1 )** ( arange(0,N+1) )
18        X = matlib repmat(x,N+1,1).transpose()
19        dX = X - X.transpose()
20        C = zeros((N+1,N+1))
21        for i in range(N+1):
22            for j in range(N+1):
23                C[i,j] = c[i]*1.0/c[j]
24        D = C/( dX + eye(N+1) ) # off-diagonal entries
25        S = sum( D, axis = 1 )
26        D = D - diag(S) # diagonal entries
27    return D,x
```

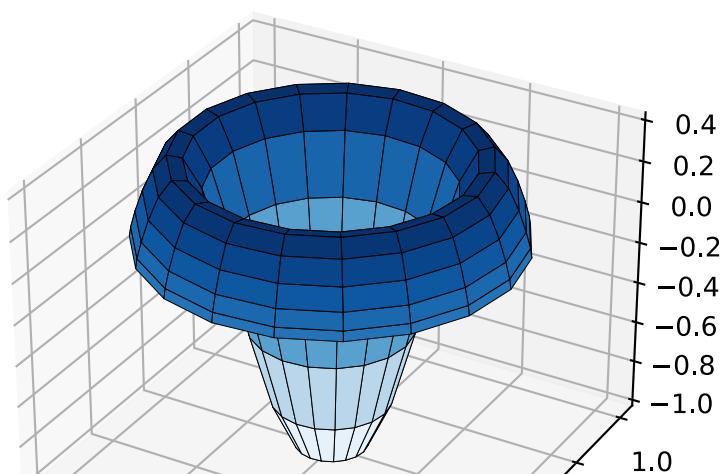
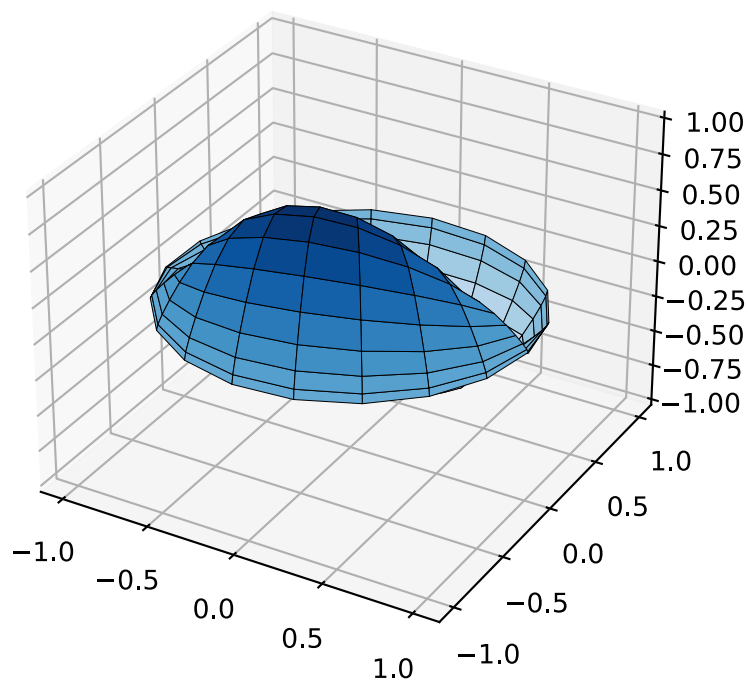
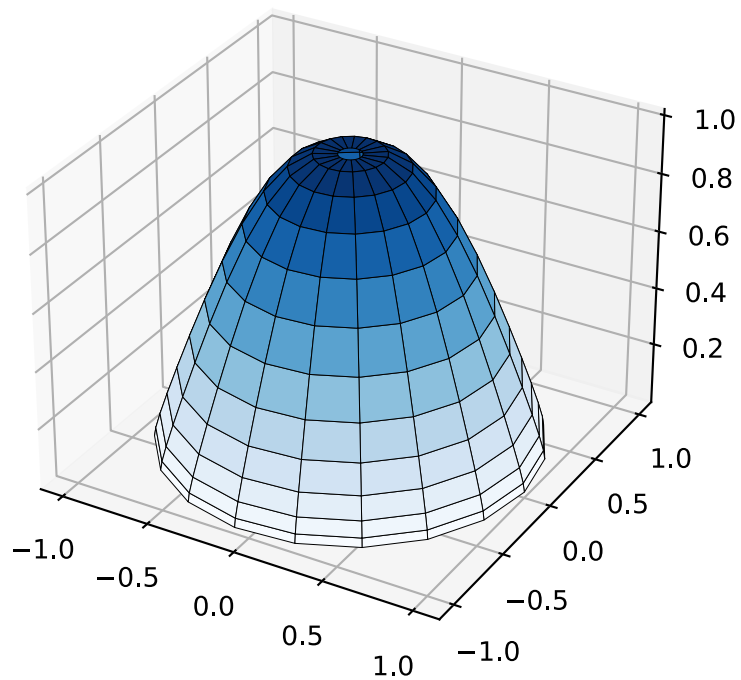
In [39]:

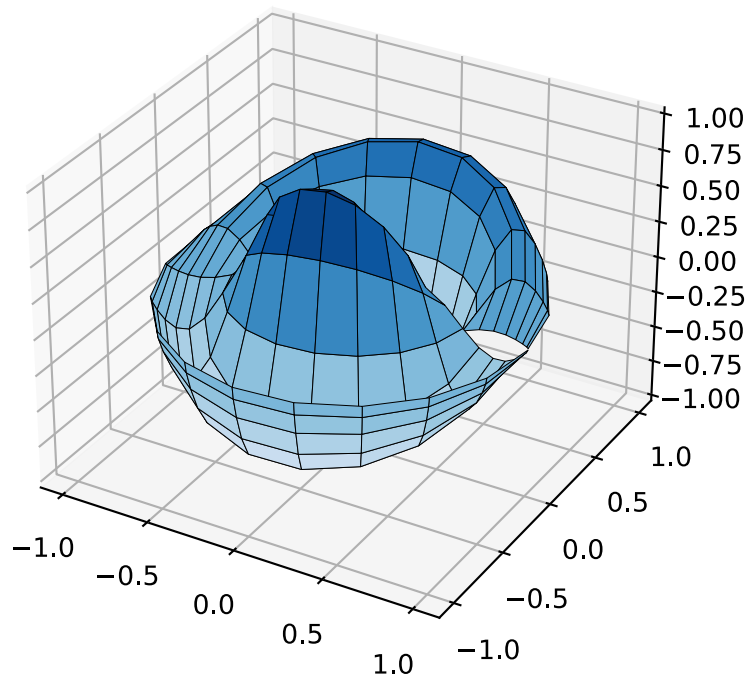
```
1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6 from mpl_toolkits.mplot3d import axes3d
7 %config InlineBackend.figure_formats = ['svg']
8
9 # eigenmodes of Laplacian on the disk
10
11 # r coordinate, ranging from -1 to 1 (N must be odd):
12 N = 25
13 #N2 = (N-1)/2 alternate statement of variable avoids error
14 N2 = 12
15 D,r = cheb(N)
16 DD = matmul(D,D)
17 D1 = DD[1:N2+1,1:N2+1]
18 E1 = D[1:N2+1,1:N2+1]
19 i = arange(-2,-N2-2,-1)
20 D2 = DD[1:N2+1,i]
21 E2 = D[1:N2+1,i]
22
23 # t = theta coordinate, ranging from 0 to 2*pi (M must be even):
24 M = 20
25 dt = 2*pi/M
26 t = dt*arange(1,M+1)
27 #M2 = alternate statement of variable avoids error
28 M2 = 10
29 c = zeros(1)
30 c[0] = -pi**2/(3*dt**2) - 1.0/6.0
31 c = append(c, 0.5*(-1)**arange(2,M+1)/sin( 0.5*dt*arange(1,M) )**2 )
32 D2t = linalg.toeplitz(c)
33
34 # Laplacian in polar coordinates:
35 R = diag( 1.0/r[1:N2+1] )
36 Z = zeros((M2,M2))
37 I = eye(M2)
38 RR = matmul(R,R)
39 ZI= hstack((Z,I))
40 IZ= hstack((I,Z))
41 ZIIZ = vstack((ZI,IZ))
42 M1 = D1 + matmul(R,E1)
43 M2 = D2 + matmul(R,E2)
```

```

44 L = kron( M1, eye(M) ) + kron( M2, ZIIZ ) + kron( RR, D2t )
45
46 # Compute eigenmodes:
47 Lam,V = linalg.eig(-L)
48 ii = argsort( Lam )
49 Lam = Lam[ii]
50 V = V[:,ii]
51 index = [0,2,5,9]
52 Vaux = V[:,index]
53
54 # Plot eigenmodes with nodal lines underneath:
55 tau = linspace(0,2*pi,M)
56 rr,tt = meshgrid( r[0:N2+1], tau )
57 xx = rr*cos( tt )
58 yy = rr*sin( tt )
59 uu = zeros((M,N2+1))
60 for i in range(4):
61     fig = figure(i+1)
62     ax = fig.add_subplot(111, projection='3d')
63     u = reshape( Vaux[:,i], (N2,M) )
64     u = u.transpose()
65     uu[:,0:-1] = u
66     uu[:, -1] = u[:, -1]
67     uv = reshape(uu, -1)
68     uu = uu/linalg.norm(uv,inf)
69     #ax.plot_surface(xx, yy, uu, color='b', linewidth=0.9)
70     ax.plot_surface(xx, yy, uu, cmap=cm.Blues, edgecolor='black', linewidth=0.2)
71 show()

```





Program 29 : solve Poisson equation on the unit disk

In [40]:

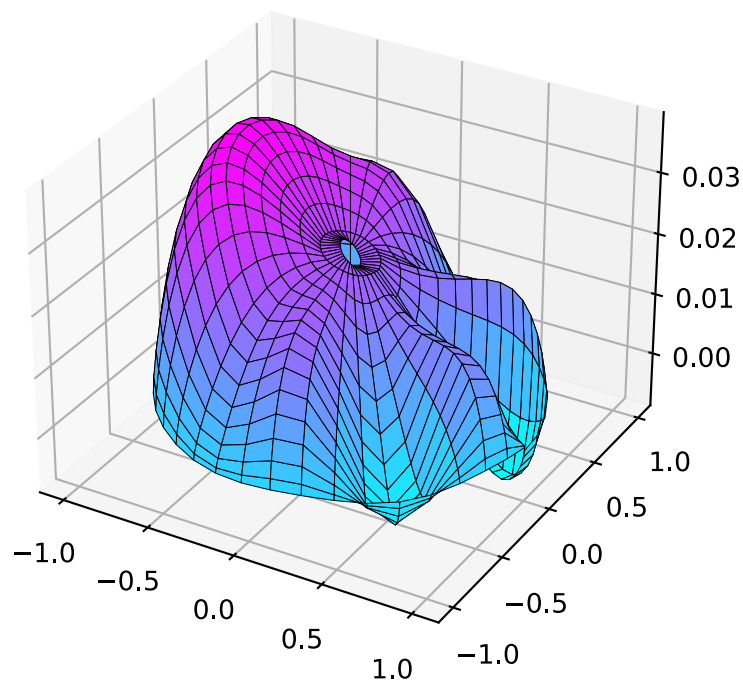
```

1  #from cheb import *
2  from numpy import *
3  from scipy import *
4  from scipy import linalg
5  from matplotlib.pyplot import *
6  from mpl_toolkits.mplot3d import axes3d
7  %config InlineBackend.figure_formats = ['svg']
8
9  # solve Poisson equation on the unit disk
10
11 # Laplacian in polar coordinates:
12 N = 31
13 [D,r] = cheb(N)
14 #N2 = (N-1)/2 alternate declaration of variable avoids error
15 N2 = 15
16 DD = matmul(D,D)
17 D1 = DD[1:N2+1,1:N2+1]
18 E1 = D[1:N2+1,1:N2+1]
19 i = arange(-2,-N2-2,-1)
20 D2 = DD[1:N2+1,i]
21 E2 = D[1:N2+1,i]
22
23 M = 40
24 dt = 2*pi/M
25 t = dt*arange(1,M+1)
26 #M2 = M/2 alternate declaration of variable avoids error
27 M2=20
28 c = zeros(1)
29 c[0] = -pi**2/(3*dt**2) - 1.0/6.0
30 c = append(c, 0.5*(-1)**arange(2,M+1)/sin( 0.5*dt*arange(1,M) )**2 )
31 D2t = linalg.toeplitz(c)
32
33 # Laplacian in polar coordinates:
34 R = diag( 1.0/r[1:N2+1] )
35 Z = zeros((M2,M2))
36 I = eye(M2)
37 RR = matmul(R,R)
38 ZI= hstack((Z,I))
39 IZ= hstack((I,Z))
40 ZIIZ = vstack((ZI,IZ))
41 M1 = D1 + matmul(R,E1)
42 M2 = D2 + matmul(R,E2)
43 L = kron( M1, eye(M) ) + kron( M2, ZIIZ ) + kron( RR, D2t )
44
45 # Right-hand side and solution for u:
46 rr,tt = meshgrid( r[1:N2+1], t )
47 rrr = reshape(rr.transpose(),-1)
48 ttr = reshape(tt.transpose(),-1)
49 f = -rrr**2*sin( 0.5*ttr )**4 + sin( 6*ttr )*cos( 0.5*ttr )**2
50 u = linalg.solve(L,f) # u = L\f
51
52 # Reshape results onto 2D grid and plot them:
53 u = reshape(u,(N2,M))
54 u = u.transpose()
55 uu = zeros((M,N2+1))
56 uu[:,0:-1] = u
57 uu[:, -1] = u[:, -1]
58 rr,tt = meshgrid( r[0:N2+1],linspace(0,2*pi,M) )
59 xx = rr*cos( tt )
60 yy = rr*sin( tt )

```

```
61 fig = figure(1)
62 ax = fig.add_subplot(111, projection='3d')
63 #ax.plot_wireframe(xx, yy, uu, color='b', linewidth=0.9)
64
65 ax.plot_surface(xx, yy, uu, cmap=cm.cool, edgecolor='black', linewidth=0.2)
66 title('Poisson equation on the unit disk', fontsize=18)
67 show()
68
```

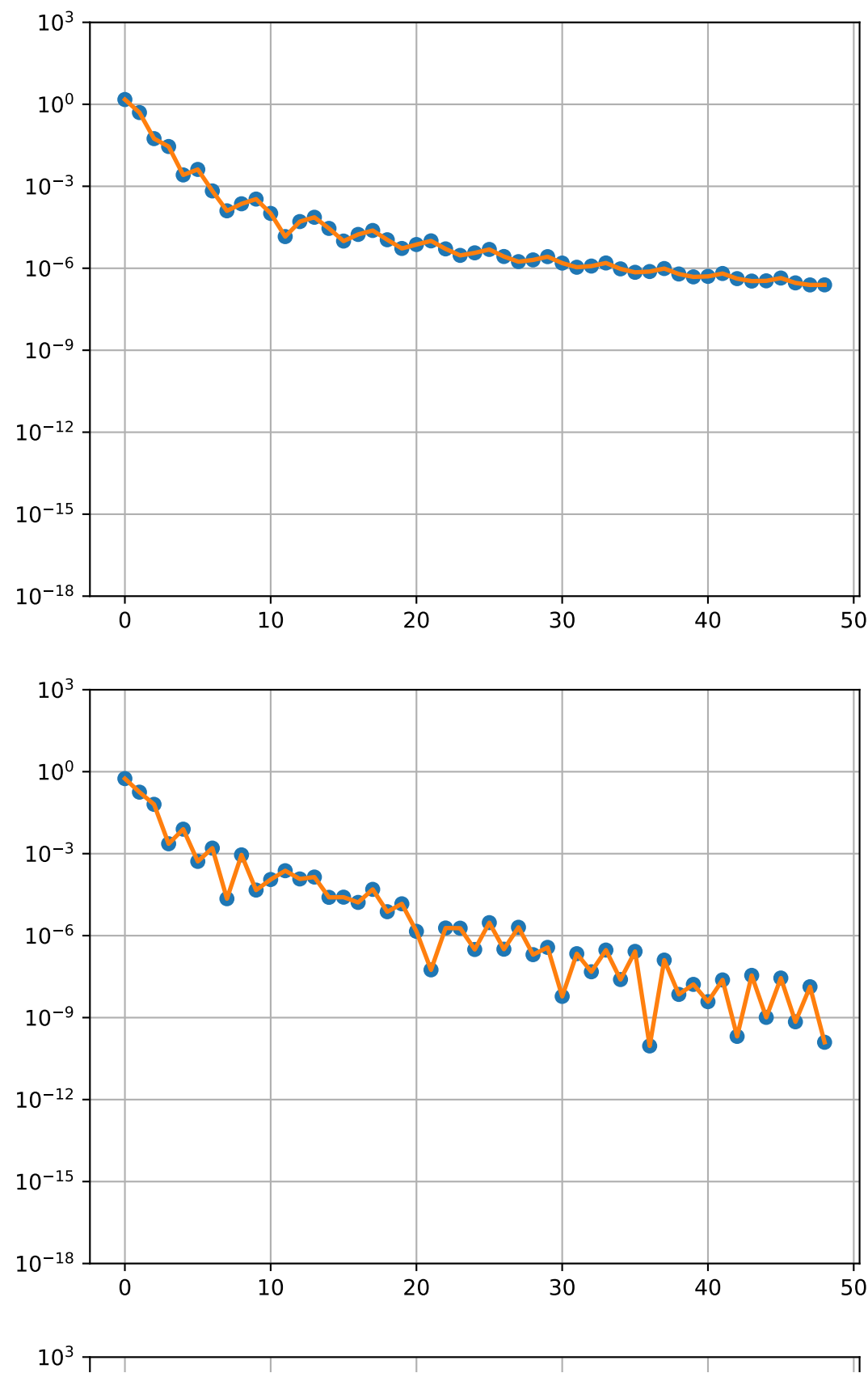
^Poisson equation on the unit disk

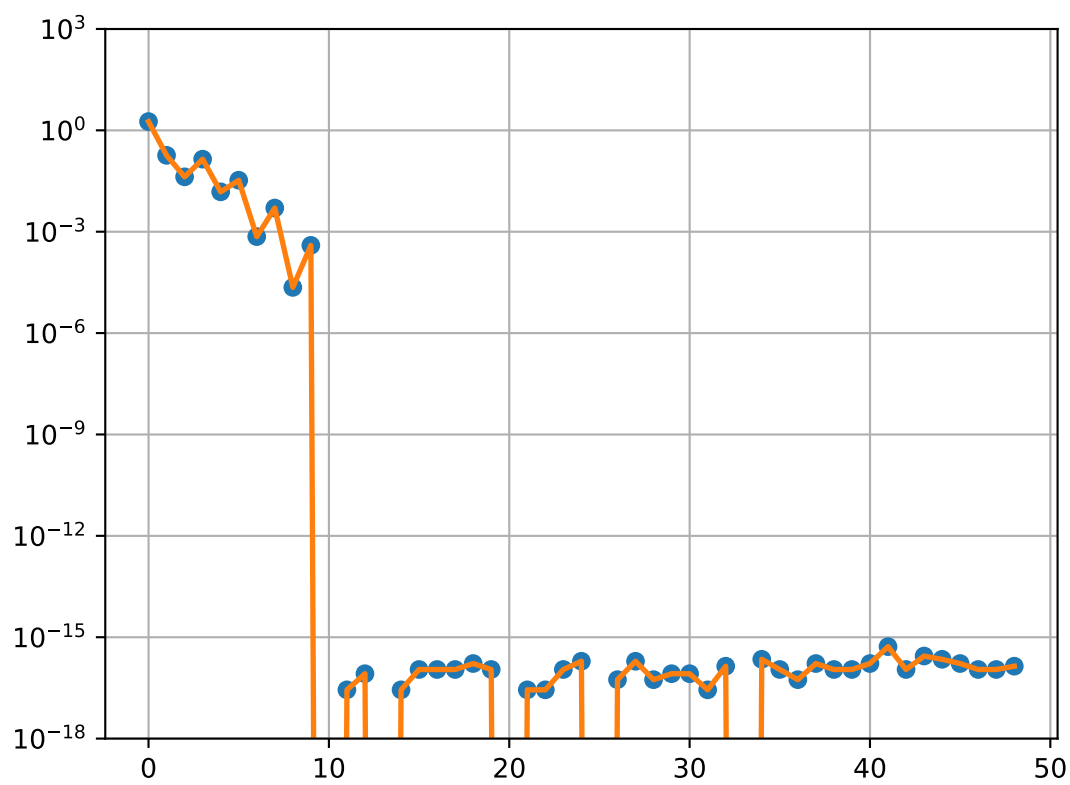


Program 30 : spectral integration, ODE-Style

In [19]:

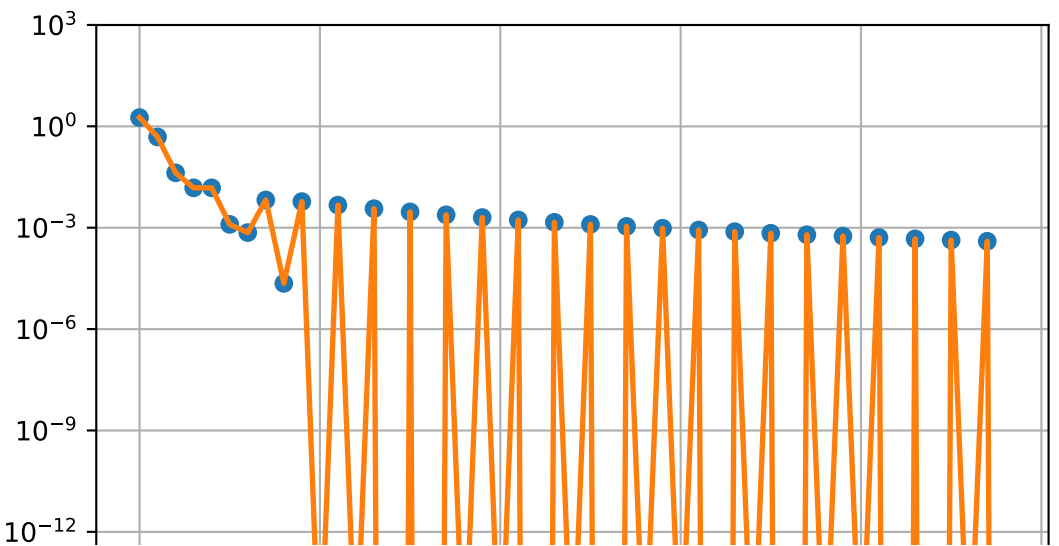
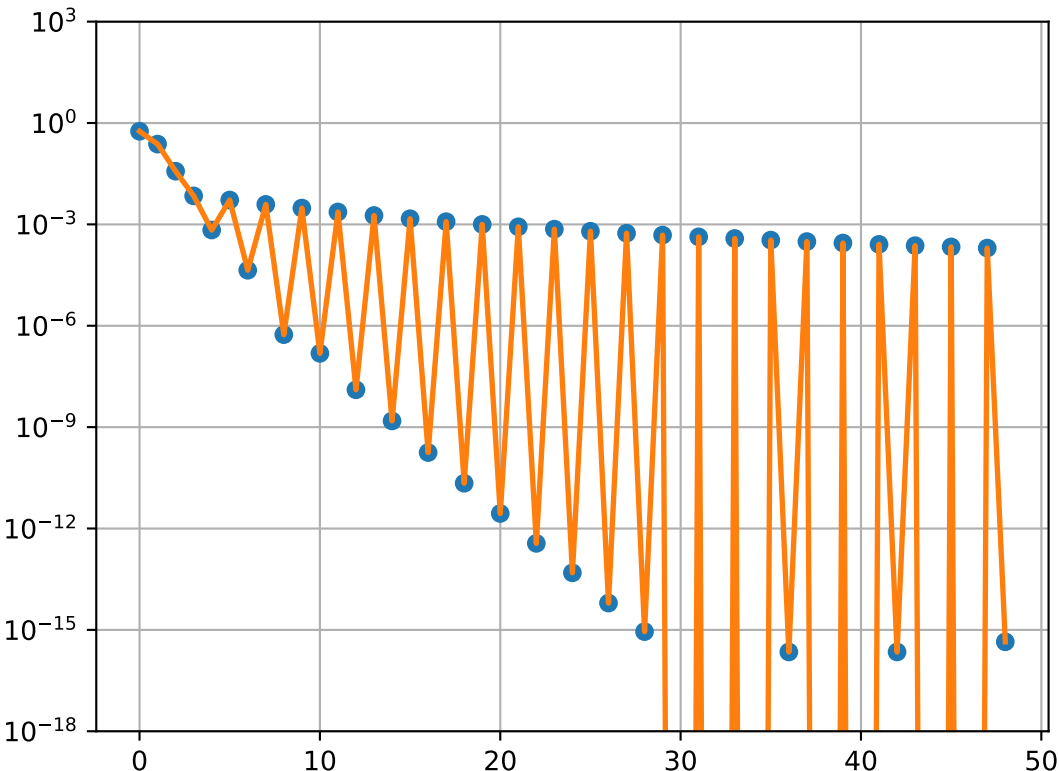
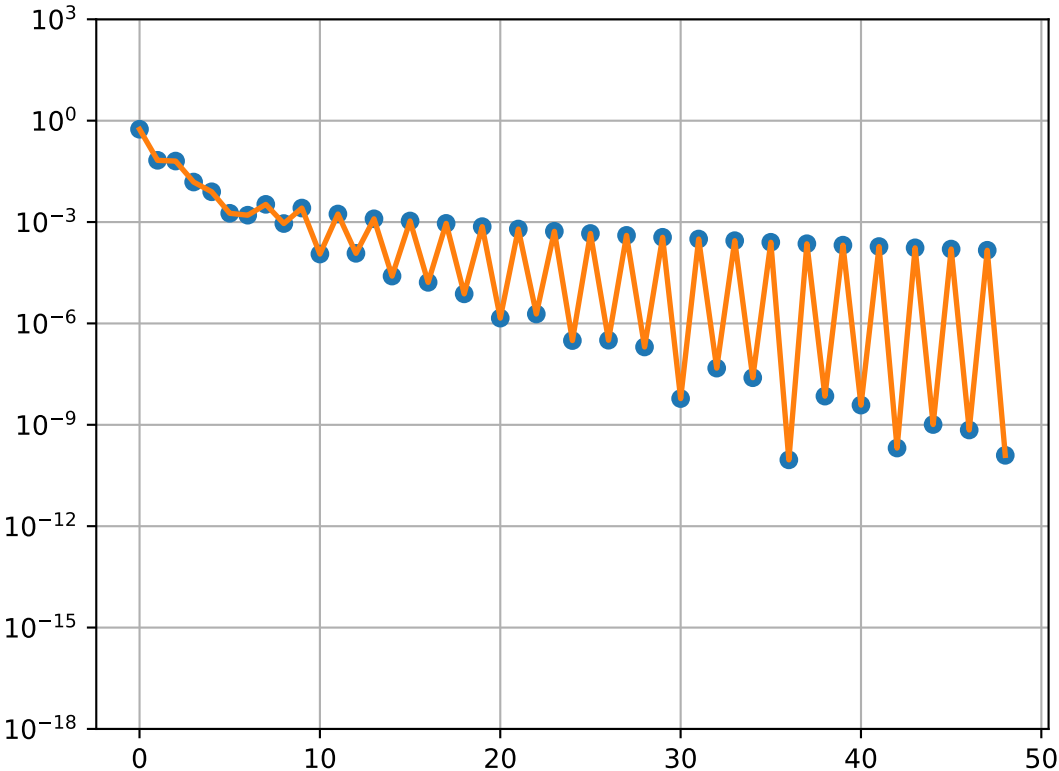
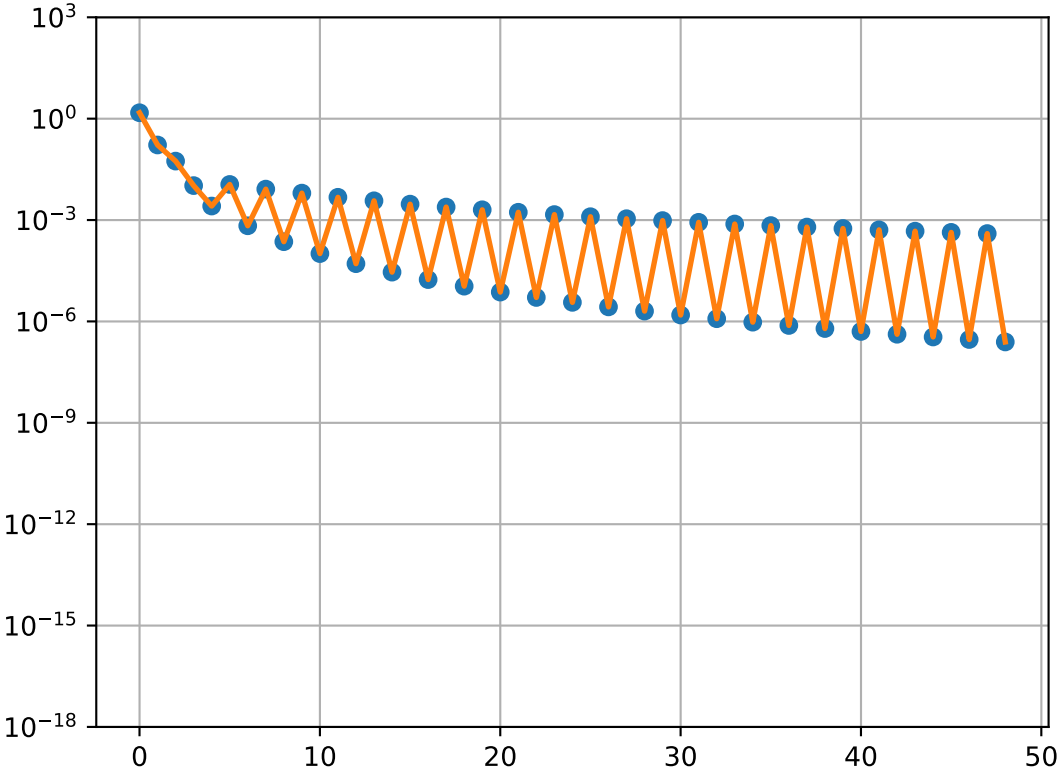
```
1  #from cheb import *
2  from numpy import *
3  from scipy import *
4  from scipy import linalg
5  from scipy import special
6  from matplotlib.pyplot import *
7  from mpl_toolkits.mplot3d import axes3d
8
9  # spectral integration, ODE style
10
11 # Computation: various values of N, four functions:
12
13 Nmax = 50
14 E = zeros((4,Nmax-1))
15 for N in range(1,Nmax):
16     i = arange(0,N)
17     D,x = cheb(N)
18     x = x[0:N]
19     DN = D[0:N,0:N]
20     Di = linalg.inv( DN )
21     w = Di[0,:]
22     f = abs(x)**3
23     E[0,N-1] = abs( dot(w,f) - 0.5)
24     f = exp( -x**(-2) )
25     E[1,N-1] = abs( dot(w,f) - 2*( exp(-1) + sqrt(pi)*(special.erf(1) -1 ) ) )
26     f = 1.0/( 1 + x**2 )
27     E[2,N-1] = abs( dot(w,f) - 0.5*pi )
28     f = x**10
29     E[3,N-1] = abs( dot(w,f) - 2.0/11.0 )
30
31 # Plot results:
32 for iplot in range(4):
33     figure(iplot+1)
34     semilogy(E[iplot,:]+ 1e-100,'o')
35     plot(      E[iplot,:]+ 1e-100, linewidth=2 )
36     ylim(1e-18, 1e3)
37     grid(True)
38 show()
39
40
```





Program 30c : More spectral integration, ODE-style

```
In [20]: 1 # CLENCURT nodes x (Chebyshev points) and weights w
2 #           for Clenshaw-Curtis quadrature
3 from numpy import *
4
5 def clencurt(N):
6     theta = pi*arange(0,N+1)/N
7     x = cos(theta)
8     w = zeros(N+1)
9     ii = arange(1,N)
10    v = ones(N-1)
11    if mod(N,2) == 0:
12        w[0] = 1.0/(N**2-1)
13        w[-1] = w[0]
14        for k in arange(1,N/2):
15            v = v - 2*cos( 2*k*theta[ii] )/( 4*k**2 - 1 )
16            v = v - cos( N*theta[ii] )/(N**2 - 1 )
17    else:
18        w[0] = 1.0/N**2
19        w[-1] = w[0]
20        for k in arange(1,(N-1)/2+1):
21            v = v - 2*cos( 2*k*theta[ii] )/( 4*k**2 - 1 )
22    w[ii] = 2*v/N
23    return x,w
24
25
In [21]: 1 #from clencurt import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from scipy import special
6 from matplotlib.pyplot import *
7 from mpl_toolkits.mplot3d import axes3d
8
9 # spectral integration, ODE style
10
11 # Computation: various values of N, four functions:
12
13 Nmax = 50
14 E = zeros((4,Nmax-1))
15 for N in range(1,Nmax):
16     i = arange(0,N)
17     x,w = clencurt(N)
18     f = abs(x)**3
19     E[0,N-1] = abs( dot(w,f) - 0.5 )
20     f = exp( -x**(-2) )
21     E[1,N-1] = abs( dot(w,f) - 2*( exp(-1) + sqrt(pi)*(special.erf(1) -1 ) ) )
22     f = 1.0/( 1 + x**2 )
23     E[2,N-1] = abs( dot(w,f) - 0.5*pi )
24     f = x**10
25     E[3,N-1] = abs( dot(w,f) - 2.0/11.0 )
26
27 # Plot results:
28 for iplot in range(4):
29     figure(iplot+1)
30     semilogy(E[iplot,:], + 1e-100, 'o')
31     plot( E[iplot,:], + 1e-100, linewidth=2 )
32     ylim(1e-18, 1e3)
33     grid(True)
34 show()
```



In []:

```
1
2
3
4
5
6
7
8
9
10
11
12
```

Program 30g : Still more spectral integration, ODE-style

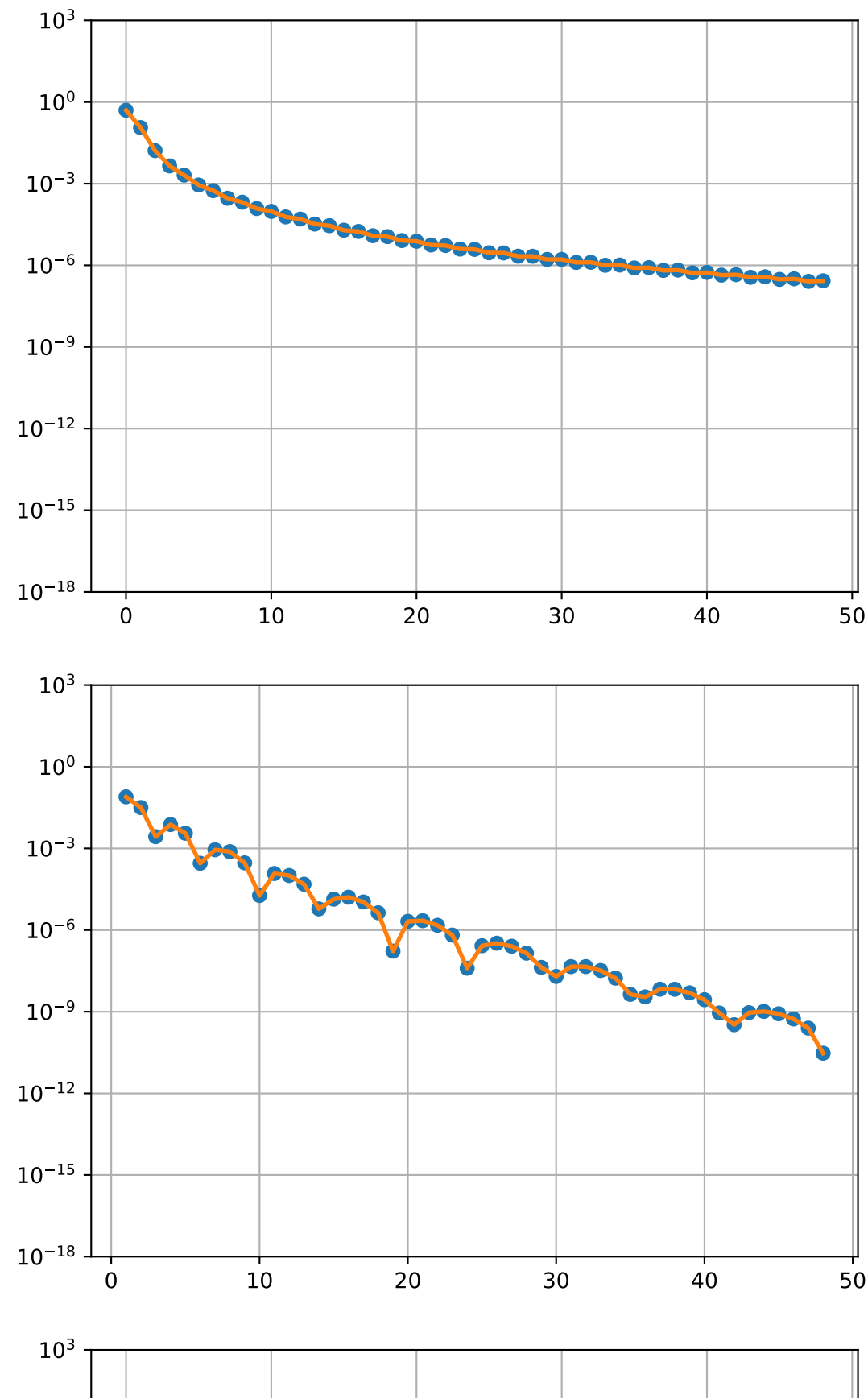
In [22]:

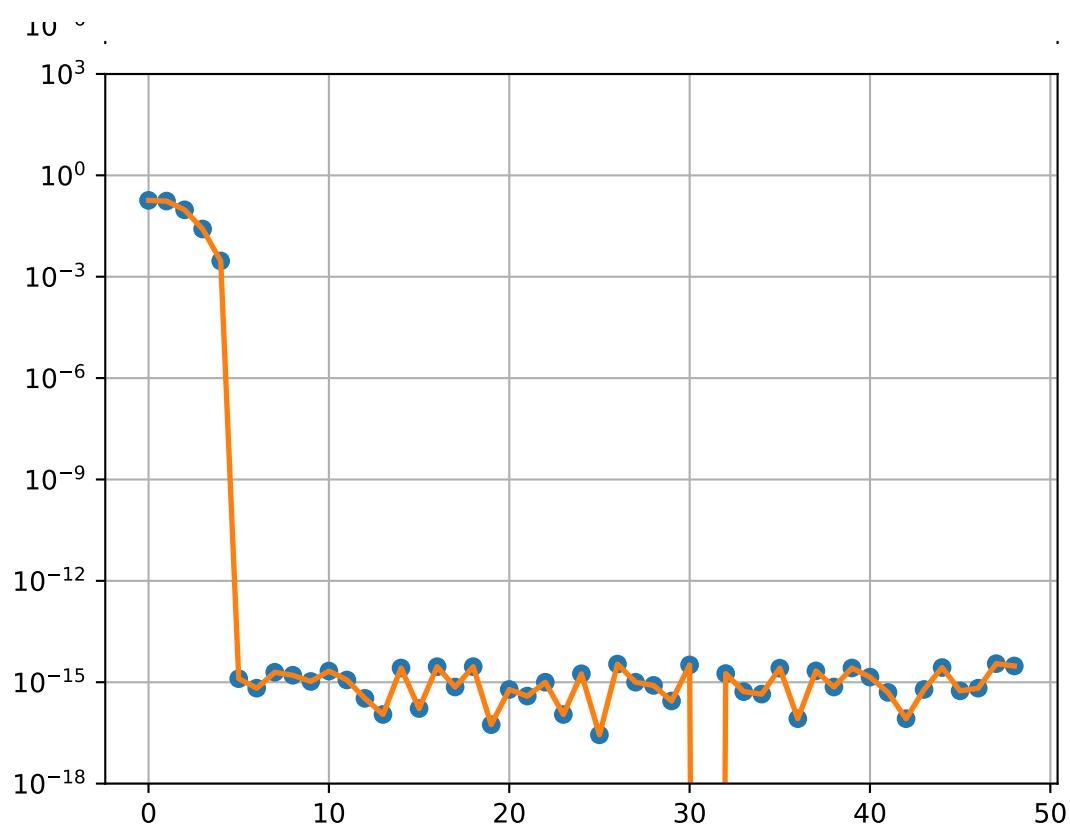
```
1 from numpy import *
2 from scipy import *
3 from scipy import linalg
4 # GAUSS nodes x (Legendre points) and weights w
5 #     for Gauss quadrature
6
7 def gauss(N):
8     beta = 0.5/sqrt( 1.0 - ( 2.0*arange(1,N) )**(-2) )
9     T = diag(beta,1) + diag(beta,-1)
10    x,V = linalg.eig(T)
11    i = argsort( x )
12    x = x[i]
13    w = 2*V[0,i]**2
14    return x,w
15
16
```



```
In [24]: 1 #from gauss import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from scipy import special
6 from matplotlib.pyplot import *
7 from mpl_toolkits.mplot3d import axes3d
8
9 # spectral integration, ODE style
10
11 # Computation: various values of N, four functions:
12
13 Nmax = 50
14 E = zeros((4,Nmax-1))
15 for N in range(1,Nmax):
16     x,w = gauss(N)
17     f = abs(x)**3
18     E[0,N-1] = abs( dot(w,f) - 0.5)
19     f = exp( -x**(-2) )
20     E[1,N-1] = abs( dot(w,f) - 2*( exp(-1) + sqrt(pi)*(special.erf(1) -1 ) ) )
21     f = 1.0/( 1 + x**2 )
22     E[2,N-1] = abs( dot(w,f) - 0.5*pi )
23     f = x**10
24     E[3,N-1] = abs( dot(w,f) - 2.0/11.0 )
25
26 # Plot results:
27 for iplot in range(4):
28     figure(iplot+1)
29     semilogy(E[iplot,:]+ 1e-100,'o')
30     plot(      E[iplot,:]+ 1e-100, linewidth=2 )
31     ylim(1e-18, 1e3)
32     grid(True)
33 show()
34
35
```

C:\Users\gary\AppData\Local\Temp\ipykernel_6208\3930596895.py:19: RuntimeWarning: invalid value encountered in power
f = exp(-x**(-2))

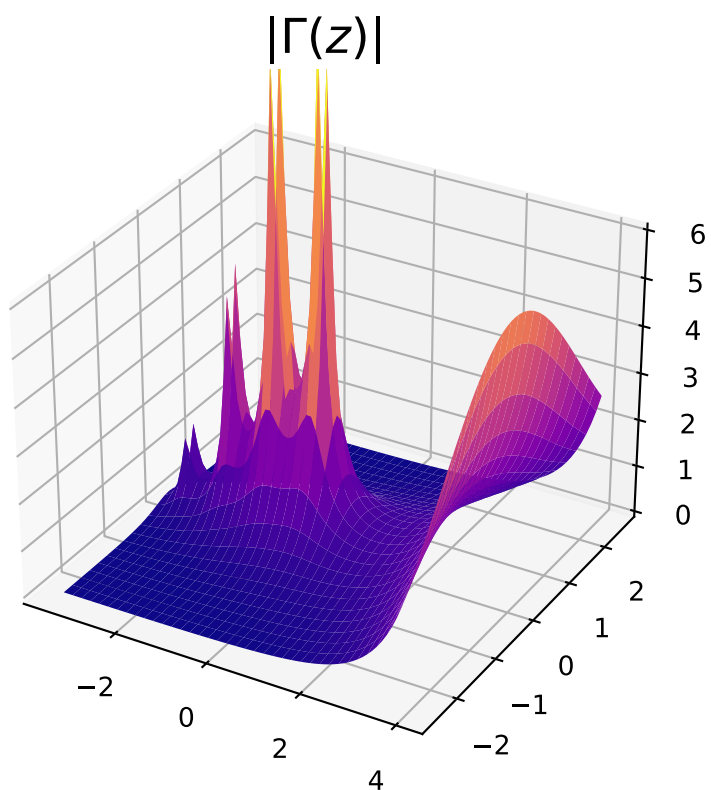




Program 31 : gamma function via complex integral, trapezoid rule

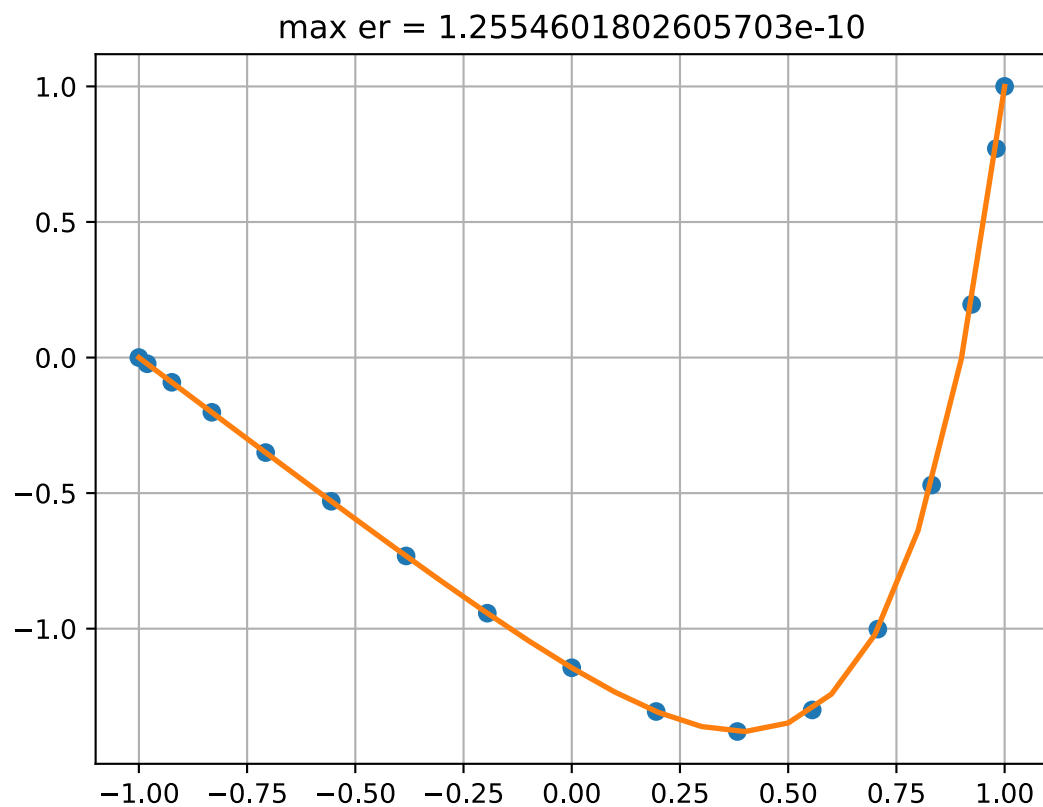
In [41]:

```
1 from numpy import *
2 from scipy import *
3 from matplotlib.pyplot import *
4 from mpl_toolkits.mplot3d import axes3d
5 %config InlineBackend.figure_formats = ['svg']
6
7 # gamma function via complex integral, trapezoid rule
8 N = 70
9 i = arange(0.5,N)
10 theta = -pi + (2*pi/N)*i
11 c = -11.0 # center of circle of integration
12 r = 16.0 # radius of circle of integration
13 x = arange(-3.5,4.1,0.1)
14 y = arange(-2.5,2.6,0.1)
15 xx,yy = meshgrid(x,y)
16 zz = xx + 1j*yy
17 gaminv = 0
18 for i in range(N):
19     t = c + r*exp(1j*theta[i])
20     gaminv = gaminv + exp(t)*t**(-zz)*(t-c)
21 gaminv = gaminv/N
22 gam = 1.0/gaminv
23 fig = figure(1)
24 ax = fig.add_subplot(111, projection='3d')
25 ax.plot_surface(xx, yy, abs(gam), cmap=cm.plasma, linewidth = 0.9)
26 ax.set_zlim(0,6)
27 title(r'$|\Gamma(z)|$', fontsize=18)
28 grid(True)
29 show()
30
31
32
```



Program 32 : solve $u_{xx} = \exp(4x)$, $u(-1)=0$, $u(1)=1$ (compare p13.m)

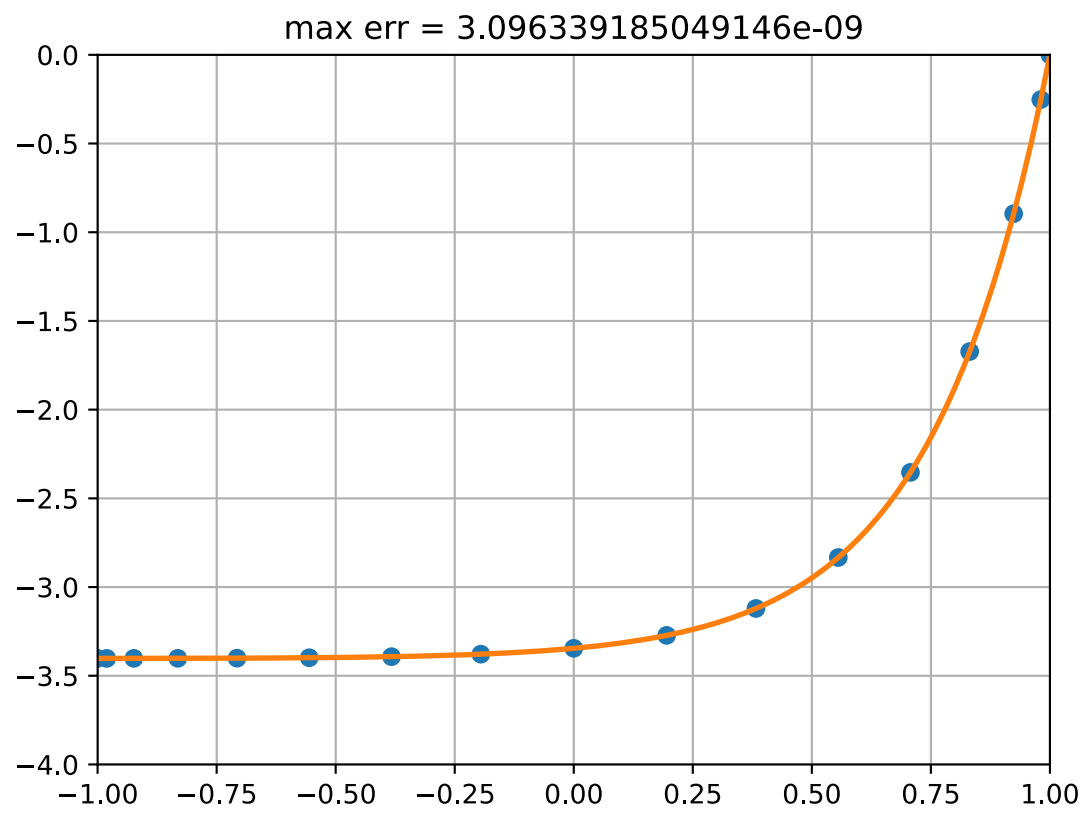
```
In [27]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6
7 # solve  $u_{xx} = \exp(4x)$ ,  $u(-1)=0$ ,  $u(1)=1$  (compare p13.m)
8
9 N = 16
10 D,x = cheb(N)
11 DD = matmul(D,D)
12 D2 = DD[1:N,1:N]
13 f = exp(4*x[1:N])
14 u = linalg.solve(D2,f) #  $u = D2 \backslash f$ ;
15 ux = 0
16 ux = append(ux,u)
17 ux = append(ux,0) + 0.5*(x+1)
18 xx = arange(-1,1.1,0.1)
19 uxx = polyval(polyfit(x,ux,N),xx)
20 exact = (exp(4*xx) - xx*sinh(4) - cosh(4))/16.0 + 0.5*(xx + 1)
21 figure(1)
22 plot(x,ux,'o')
23 plot(xx,uxx,linewidth=2)
24 thetitle = 'max er = ' + str(linalg.norm(uxx-exact,inf))
25 title( thetitle )
26 grid(True)
27 show()
28
```



Program 33 : solve linear BVP $u_{xx} = \exp(4x)$, $u'(-1)=u(1)=0$

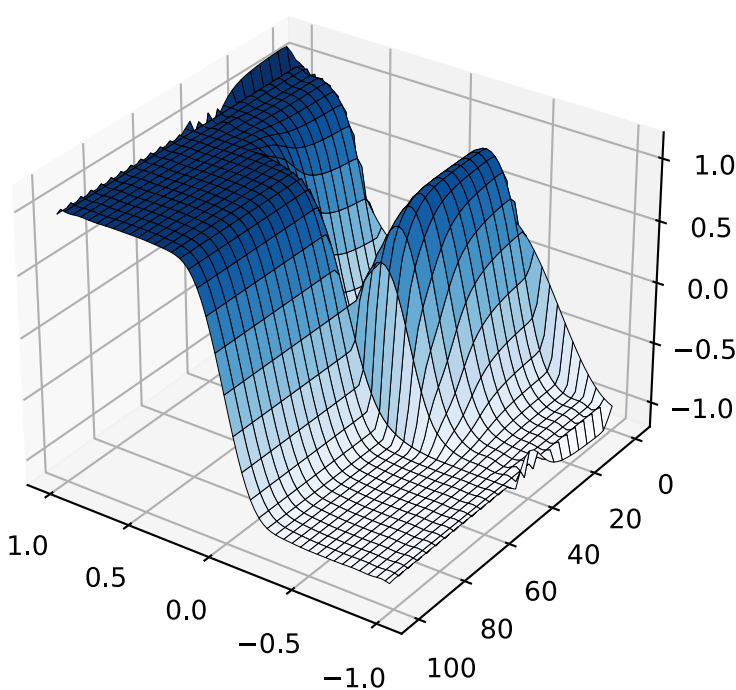
```
In [28]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6
7 # solve linear BVP  $u_{xx} = \exp(4x)$ ,  $u'(-1)=u(1)=0$ 
8
9 N = 16
10 D,x = cheb(N)
11 DD = matmul(D,D)
12 DD[-1,:] = D[-1,:] # Neumann condition at  $x = -1$ 
13 D2 = DD[1:,1:]
14 f = exp( 4*x[1:] )
15 f[-1] = 0.0
16 u = linalg.solve(D2,f) #  $u = D2 \backslash [f;0]$ ;
17 ux = 0
18 ux = append(ux,u)
19 xx = arange(-1,1.01,0.01)
20 uxx = polyval(polyfit(x,ux,N),xx)
21 exact = ( exp(4*xx) - 4*exp(-4)*(xx-1) - exp(4) )/16.0
22 maxerr = linalg.norm(uxx-exact,inf)
23 thetitle = 'max err = ' + str(maxerr)
24 figure(1)
25 plot(x,ux,'o')
```

```
26 plot(xx,uxx,linewidth=2)
27 #plot(xx,exact,linewidth=2)
28 title( thetitle )
29 xlim(-1,1)
30 ylim(-4,0)
31 grid(True)
32 show()
33
```



Program 34 : Allen-Cahn eq. $u_t = \epsilon u_{xx} + u - u^3$, $u(-1) = -1$, $u(1) = 1$

```
In [38]: 1 #from cheb import *
2 %config InlineBackend.figure_formats = ['svg']
3 from numpy import *
4 from scipy import *
5 from scipy import linalg
6 from matplotlib.pyplot import *
7 from mpl_toolkits.mplot3d import axes3d
8
9 # Allen-Cahn eq.  $u_t = \epsilon u_{xx} + u - u^3$ ,  $u(-1)=-1$ ,  $u(1)=1$ 
10
11 # Differentiation matrix and initial data:
12 N = 20
13 [D,x] = cheb(N)
14 D2 = matmul(D,D) # use full-size matrix
15 D2[:, 0] = 0.0 # for convenience
16 D2[:, -1] = 0.0
17 eps = 0.01
18 dt = min([.01, 50*N**(-4)/eps])
19 t = 0
20 v = 0.53*x + 0.47*sin( -1.5*pi*x )
21
22 # Solve PDE by Euler formula and plot results:
23 tmax = 100
24 tplot = 2.0
25 nplots = int(tmax/tplot)
26 plotgap = int(tplot/dt )
27 dt = tplot/plotgap
28 xx = arange(-1,1.025,0.025)
29 vv = polyval(polyfit(x,v,N),xx)
30 plotdata = vstack((vv,zeros((nplots,xx.size))))
31 tdata = t
32 for i in range(nplots):
33     for n in range(plotgap):
34         t = t + dt
35         v = v + dt*( eps*D2.dot(v-x) + v - v**3 ) # Euler
36         vv = polyval(polyfit(x,v,N),xx)
37         plotdata[i+1,:] = vv
38         tdata = append(tdata, t)
39 fig = figure(1)
40 ax = fig.add_subplot(111, projection='3d')
41 ax.view_init(elev=30, azim=125, roll=0)
42 XX,YY = meshgrid(xx,tdata)
43 ax.plot_surface(XX, YY, plotdata, cmap=cm.Blues, edgecolor='black', linewidth = 0.2)
44
45 show()
46
47
```



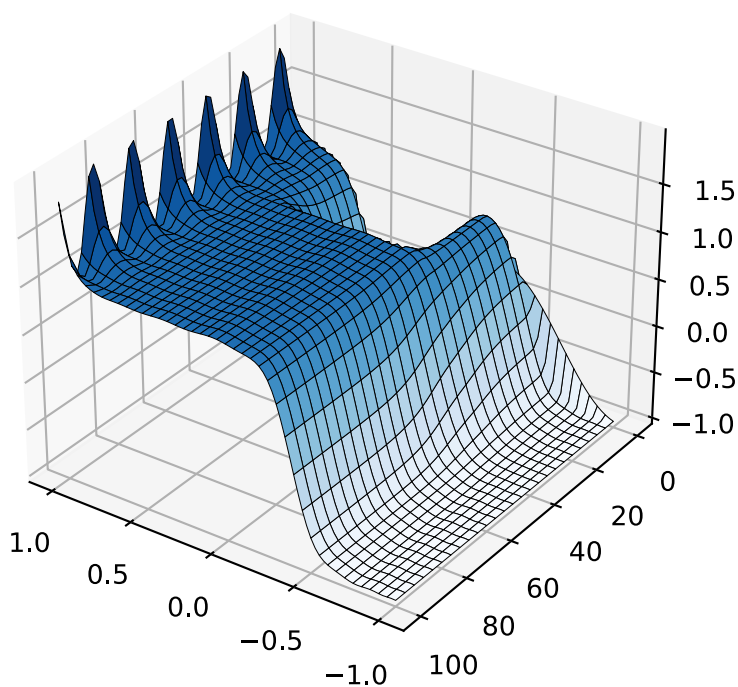
Program 35 : Allen-Cahn eq. with boundary condition imposed explicitly ("method (II)")

```
In [42]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6 from mpl_toolkits.mplot3d import axes3d
7
8 # Allen-Cahn eq. with boundary condition
9 # imposed explicitly ("method (II)")
10
11 # Differentiation matrix and initial data:
12 N = 20
13 [D,x] = cheb(N)
14 D2 = matmul(D,D) # use full-size matrix
```

```

15 eps = 0.01
16 dt = min([.01,50*N**(-4)/eps])
17 t = 0
18 v = 0.53*x + 0.47*sin( -1.5*pi*x )
19
20 # Solve PDE by Euler formula and plot results:
21 tmax = 100
22 tplot = 2.0
23 nplots = int(tmax/tplot)
24 plotgap = int(tplot/dt )
25 dt = tplot/plotgap
26 xx = arange(-1,1.025,0.025)
27 vv = polyval(polyfit(x,v,N),xx)
28 plotdata = vstack((vv,zeros((nplots,xx.size))))
29 tdata = t
30 for i in range(nplots):
31     for n in range(plotgap):
32         t = t + dt
33         v = v + dt*( eps*D2.dot(v-x) + v - v**3 ) # Euler
34         v[ 0] = 1.0 + sin( t/5.0 )**2
35         v[-1] = -1.0
36         vv = polyval(polyfit(x,v,N),xx)
37         plotdata[i+1,:] = vv
38         tdata = append(tdata, t)
39 fig = figure(1)
40 ax = fig.add_subplot(111, projection='3d')
41 ax.view_init(elev=30, azimuth=125, roll=0)
42 XX,YY = meshgrid(xx,tdata)
43 ax.plot_surface(XX, YY, plotdata, cmap=cm.Blues, edgecolor='black', linewidth = 0.2)
44 show()
45

```



Program 36 : Laplace eq. on $[-1,1] \times [-1,1]$ with nonzero BCs

```

In [43]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import interpolate
5 from scipy import linalg
6 from matplotlib.pyplot import *
7 from mpl_toolkits.mplot3d import axes3d
8 %config InlineBackend.figure_formats = ['svg']
9
10 # Laplace eq. on [-1,1]x[-1,1] with nonzero BCs
11
12 # Set up grid and 2D Laplacian, boundary points included:
13 N = 24
14 D,x = cheb(N)
15 y = x
16 xx,yy = meshgrid(x,y)
17 xxr = reshape(xx.transpose(),-1)
18 yyr = reshape(yy.transpose(),-1)
19 D2 = matmul(D,D)
20 I = eye(N+1)
21 L = kron(I,D2) + kron(D2,I)
22
23 # Impose boundary conditions by replacing appropriate rows of L:
24 bw = where( ( abs(xxr) == 1 ) | ( abs(yyr) == 1 ) )
25 b = bw[0]
26 n = b.size
27 L[b,:] = zeros((4*N,(N+1)**2))
28 for i in range(n):
29     for j in range(n):

```



```

30         if i == j:
31             L[b[i],b[j]] = 1.0
32         else:
33             L[b[i],b[j]] = 0.0
34
35     yyri = zeros(n); i = where( yyr[b] == 1 ); yyri[ i[0] ] = 1.0
36     xxri = zeros(n); i = where( xxr[b] < 0 ); xxri[ i[0] ] = xxr[i[0]]
37     XXri = zeros(n); i = where( xxr[b] == 1 ); XXri[ i[0] ] = 1.0
38     rhs = zeros((N+1)**2)
39     rhs[b] = yyri*xxri*sin( pi*xxr[b] )**4 + 0.2*XXri*sin( 3*pi*yyr[b] )
40
41     # Solve Laplace equation, reshape to 2D, and plot:
42     u = linalg.solve(L,rhs) # u = L\rhs;
43     uu = reshape(u,(N+1,N+1))
44     x3 = arange(-1,1.04,0.04)
45     y3 = arange(-1,1.04,0.04)
46     [xxx,yyy] = meshgrid( x3 , y3 )
47     interpolator = interpolate.interp2d(x,y,uu,'cubic')
48     uuu = interpolator(x3,y3)
49     fig = figure(1)
50     ax = fig.add_subplot(111, projection='3d')
51     ax.plot_surface(xxx, yyy, uuu, cmap=cm.turbo, linewidth = 0.9)
52     show()
53     # uuu = interp2(xx,yy,uu,xxx,yyy,'cubic');
54     # subplot('position',[.1 .4 .8 .5])
55     # mesh(xxx,yyy,uuu), colormap(1e-6*[1 1 1]);
56     # axis([-1 1 -1 1 -.2 1]), view(-20,45)
57     # text(0,.8,.4,sprintf('u(0,0) = #12.10f',uu(N/2+1,N/2+1)))
58

```

C:\Users\gary\AppData\Local\Temp\ipykernel_10688\1864508514.py:47: DeprecationWarning: `interp2d` is deprecated!

`interp2d` is deprecated in SciPy 1.10 and will be removed in SciPy 1.12.0.

For legacy code, nearly bug-for-bug compatible replacements are
`RectBivariateSpline` on regular grids, and `bisplrep`/`bisplev` for
scattered 2D data.

In new code, for regular grids use `RegularGridInterpolator` instead.
For scattered data, prefer `LinearNDInterpolator` or
`CloughTocher2DInterpolator`.

For more details see

`<https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff>`

```

interpolator = interpolate.interp2d(x,y,uu,'cubic')

```

C:\Users\gary\AppData\Local\Temp\ipykernel_10688\1864508514.py:48: DeprecationWarning: `interp2d` is deprecated!

`interp2d` is deprecated in SciPy 1.10 and will be removed in SciPy 1.12.0.

For legacy code, nearly bug-for-bug compatible replacements are
`RectBivariateSpline` on regular grids, and `bisplrep`/`bisplev` for
scattered 2D data.

In new code, for regular grids use `RegularGridInterpolator` instead.
For scattered data, prefer `LinearNDInterpolator` or
`CloughTocher2DInterpolator`.

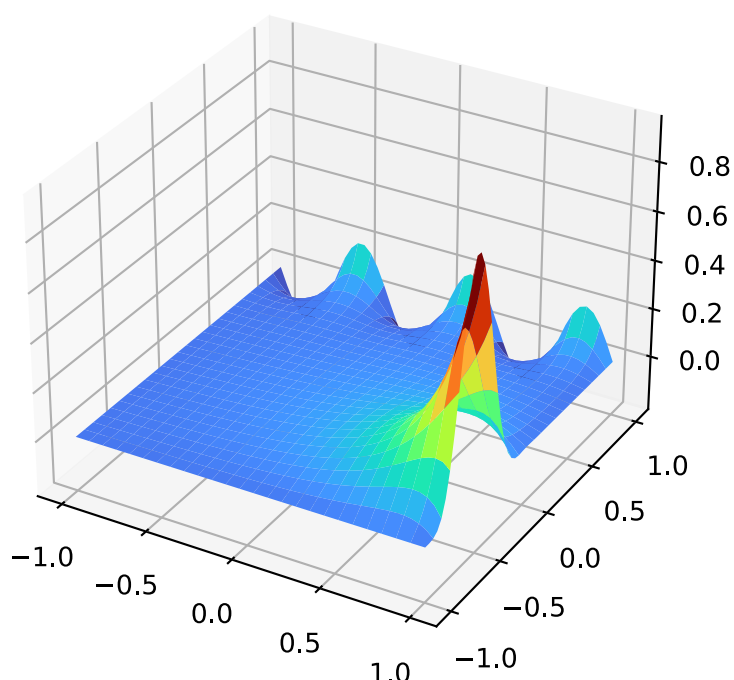
For more details see

`<https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff>`

```

uuu = interpolator(x3,y3)

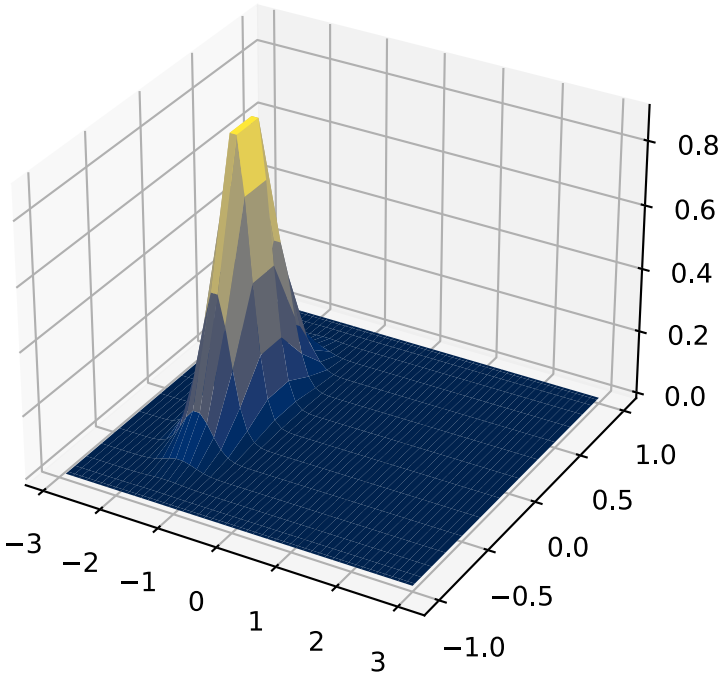
```

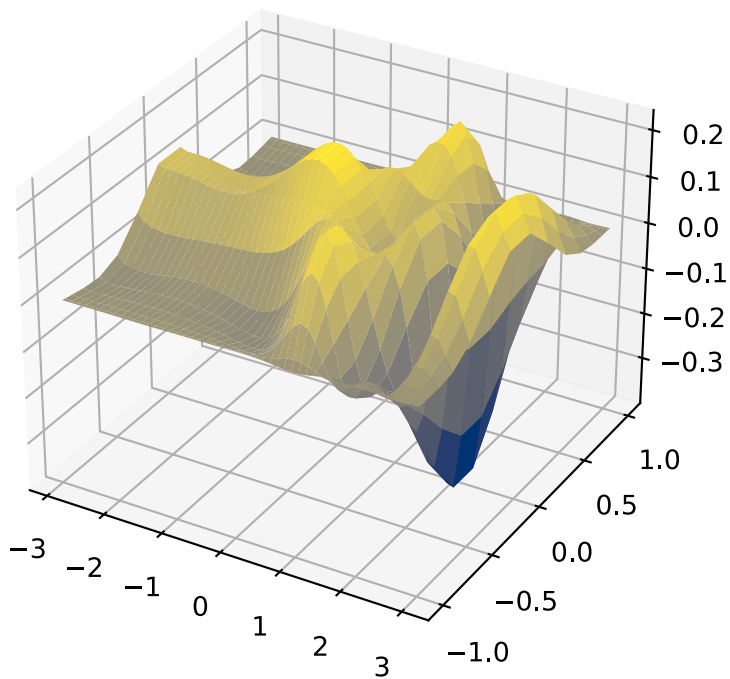
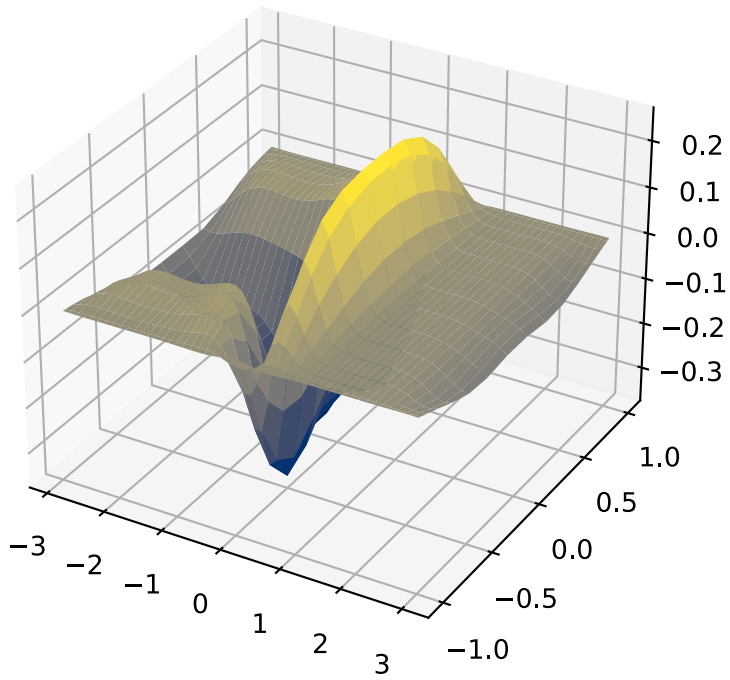


Program 37 : "wave tank" with Neumann BCs for $|y|=1$

In [44]:

```
1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6 from mpl_toolkits.mplot3d import axes3d
7 %config InlineBackend.figure_formats = ['svg']
8
9 # "wave tank" with Neumann BCs for |y|=1
10
11 # x variable in [-A,A], Fourier:
12 A = 3.0
13 Nx = 50
14 dx = 2*A/Nx
15 x = -A + dx*arange(1,Nx+1)
16 c = -1.0/(3.0*(dx/A)**2) - 1.0/6.0
17 c2 = 0.5*(-1)**( arange(2,Nx+1) )/sin( (pi*dx/A)*( 0.5*arange(1,Nx) ) )**2
18 c = append(c, c2)
19 D2x = (pi/A)**2*linalg.toeplitz(c)
20
21 # y variable in [-1,1], Chebyshev:
22 Ny = 15
23 Dy,y = cheb(Ny)
24 D2y = matmul(Dy,Dy)
25 DY = zeros((2,2))
26 DY[0,0] = Dy[ 0, 0]
27 DY[0,1] = Dy[ 0,-1]
28 DY[1,0] = Dy[-1, 0]
29 DY[1,1] = Dy[-1,-1]
30 FY = zeros((2,Ny-1))
31 FY[0,:] = Dy[ 0,1:Ny]
32 FY[1,:] = Dy[-1,1:Ny]
33 BC = linalg.solve(DY,FY)
34
35 # Grid and initial data:
36 xx,yy = meshgrid(x,y)
37 vv = exp(- 8*( ( xx + 1.5)**2 + yy**2 ) )
38 dt = 5.0/( Nx + Ny**2 )
39 vvold = exp( -8*( ( xx + dt + 1.5 )**2 + yy**2 ) )
40
41 # Time-stepping by leap frog formula:
42 plotgap = int( 2.0/dt )
43 dt = 2.0/plotgap
44 j = 0
45 for n in range( 2*plotgap+1 ):
46     t = n*dt
47     if mod( n + .5, plotgap ) < 1:
48         j = j + 1
49         fig = figure(j)
50         ax = fig.add_subplot(111, projection='3d')
51         ax.plot_surface(xx, yy, vv, cmap=cm.cividis, linewidth = 0.9)
52         vvnew = 2*vv - vvold + dt**2*( matmul(vv,D2x) + matmul(D2y,vv) )
53         vvold = vv
54         vv = vvnew
55         # Neumann BCs for |y| = 1
56         prod = matmul(BC,vv[1:Ny,:])
57         vv[ 0,:] = prod[0,:]
58         vv[-1,:] = prod[1,:]
59 show()
60
61
62
63
64
65
66
67
68
```





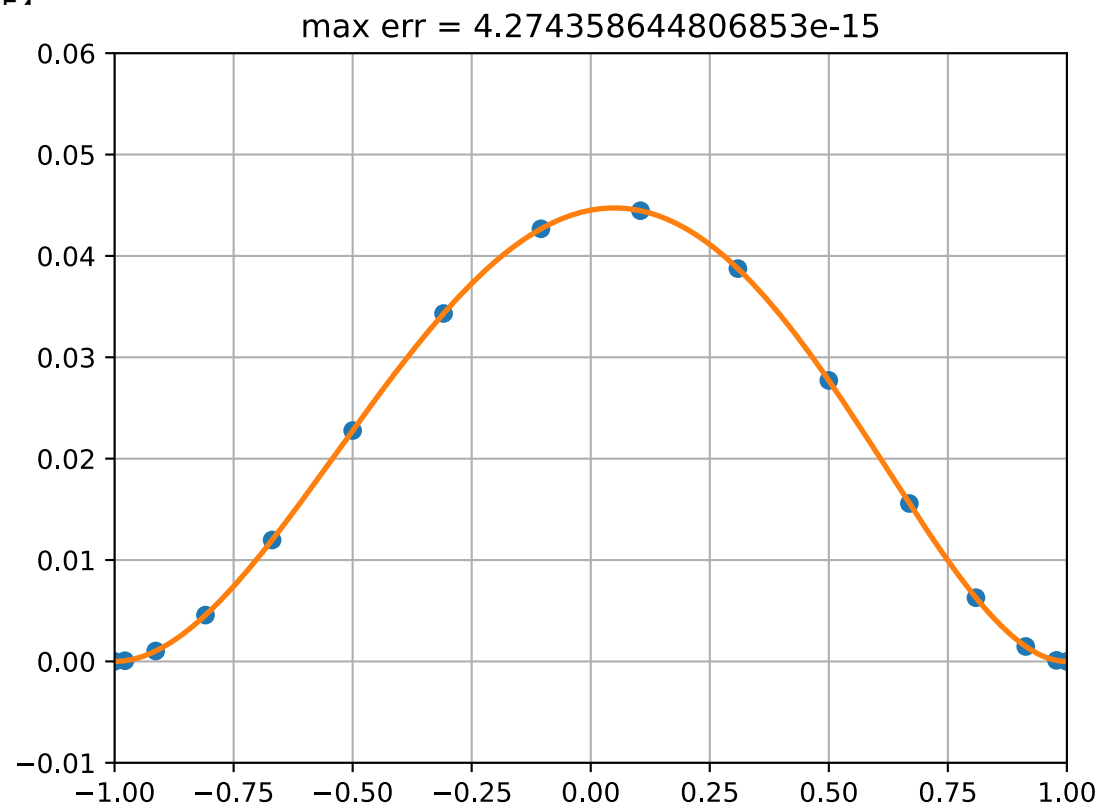
Program 38 : solve $u_{xxxx} = \exp(x)$, $u(-1)=u(1)=u'(-1)=u'(1)=0$

```
In [40]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6
7 # solve  $u_{xxxx} = \exp(x)$ ,  $u(-1)=u(1)=u'(-1)=u'(1)=0$ 
8
9 # Construct discrete biharmonic operator:
10 N = 15
11 D,x = cheb(N)
12 c = 0
13 c = append(c, 1.0/(1 - x[1:N]**2) )
14 c = append(c, 0)
15 S = diag(c)
16 D2 = matmul(D,D)
17 D3 = matmul(D2,D)
18 D4 = matmul(D2,D2)
19 C1 = diag(1.0 - x**2)
20 C2 = diag(x)
21 M = matmul(C1,D4) - matmul(8*C2,D3) - 12*D2
22 D4 = matmul(M,S)
23 Div = D4[1:N,1:N]
24
25 # Solve boundary-value problem and plot result:
26 f = exp(x[1:N])
27 u = linalg.solve(Div,f) #  $u = D4 \backslash f$ ;
28 ux = 0
29 ux = append(ux,u)
30 ux = append(ux,0)
31 xx = arange(-1,1.01,0.01)
32 uu = (1 - xx**2)*polyval(polyfit(x,matmul(S,ux),N),xx)
33
34 # Determine exact solution and print maximum error:
35 A = array([[1, -1, 1, -1],[0, 1, -2, 3],[1, 1, 1, 1],[0, 1, 2, 3]])
36 V = vander(xx)
```

```

37 V2 = V[:, -1:-5:-1]
38 e = array([-1, -1, 1, 1])
39 e = exp(e)
40 c = linalg.solve(A,e) # c = A\exp([-1 -1 1 1]');
41 exact = exp(xx) - matmul(V2,c)
42 maxerr = linalg.norm(uu-exact,inf)
43 thetitle = 'max err = ' + str(maxerr)
44
45 figure(1)
46 plot(x,ux,'o')
47 plot(xx,uu,linewidth=2)
48 title( thetitle )
49 xlim(-1,1)
50 ylim(-0.01,0.06)
51 grid(True)
52 show()
53

```



Program 39 : eigenmodes of biharmonic on a square with clamped BCs

```

In [41]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import interpolate
5 from scipy import linalg
6 from matplotlib.pyplot import *
7
8 # eigenmodes of biharmonic on a square with clamped BCs
9
10 # Construct spectral approximation to biharmonic operator:
11 N = 17
12 D,x = cheb(N)
13 c = 0
14 c = append(c, 1.0/( 1 - x[1:N]**2 ) )
15 c = append( c , 0 )
16 S = diag( c )
17 D2 = matmul(D ,D )
18 D3 = matmul(D2,D )
19 D4 = matmul(D2,D2)
20 C1 = diag( 1.0 - x**2 )
21 C2 = diag( x )
22 M = matmul(C1,D4) - matmul(8*C2,D3) - 12*D2
23 D4 = matmul(M,S)
24 Dii = D2[1:N,1:N]
25 Div = D4[1:N,1:N]
26
27 # D4 = (diag(1-x.^2)*D^4 - 8*diag(x)*D^3 - 12*D^2)*S;
28 # D4 = D4(2:N,2:N);
29 I = eye(N-1)
30 L = kron(I,Div) + kron(Div,I) + 2.0*matmul( kron(Dii,I) , kron(I,Dii) )
31
32 # Find and plot 25 eigenmodes:
33 Lam,V = linalg.eig(-L)
34 rLam = -real( Lam )
35 ii = argsort( rLam )
36 rLam = rLam[ii]; rLam = sqrt( rLam/rLam[0] )
37 V = V[:,ii]
38 xx,yy = meshgrid(x,x)
39 x2 = arange(-1,1.01,0.01)
40 xxx,yyy = meshgrid(x2,x2)
41 for i in range(25):
42     uu = zeros((N+1,N+1))

```

```

43 uu[1:N,1:N] = reshape(real(V[:,i]),(N-1,N-1))
44 figure(i+1)
45 interpolator = interpolate.interp2d(x,x,uu,'cubic')
46 uuu = interpolator(x2,x2)
47 contour(x2,x2,uuu)
48 grid(True)
49 show()
50

```

C:\Users\gary\AppData\Local\Temp\ipykernel_6208\868330203.py:45: DeprecationWarning: `interp2d` is deprecated!
`interp2d` is deprecated in SciPy 1.10 and will be removed in SciPy 1.12.0.

For legacy code, nearly bug-for-bug compatible replacements are
`RectBivariateSpline` on regular grids, and `bisplrep`/`bisplev` for
scattered 2D data.

In new code, for regular grids use `RegularGridInterpolator` instead.
For scattered data, prefer `LinearNDInterpolator` or
`CloughTocher2DInterpolator`.

For more details see
[https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff`](https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff)

```

interpolator = interpolate.interp2d(x,x,uu,'cubic')
C:\Users\gary\AppData\Local\Temp\ipykernel_6208\868330203.py:46: DeprecationWarning: `inter
p2d` is deprecated!
`interp2d` is deprecated in SciPy 1.10 and will be removed in SciPy 1.12.0.

```

For legacy code, nearly bug-for-bug compatible replacements are
`RectBivariateSpline` on regular grids, and `bisplrep`/`bisplev` for
scattered 2D data.

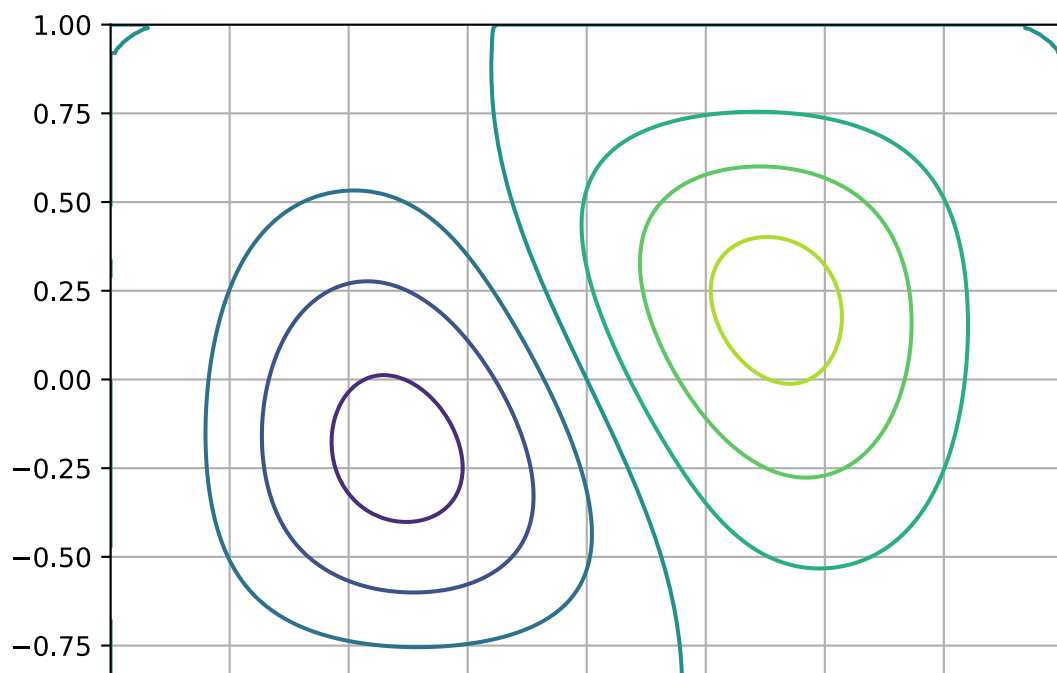
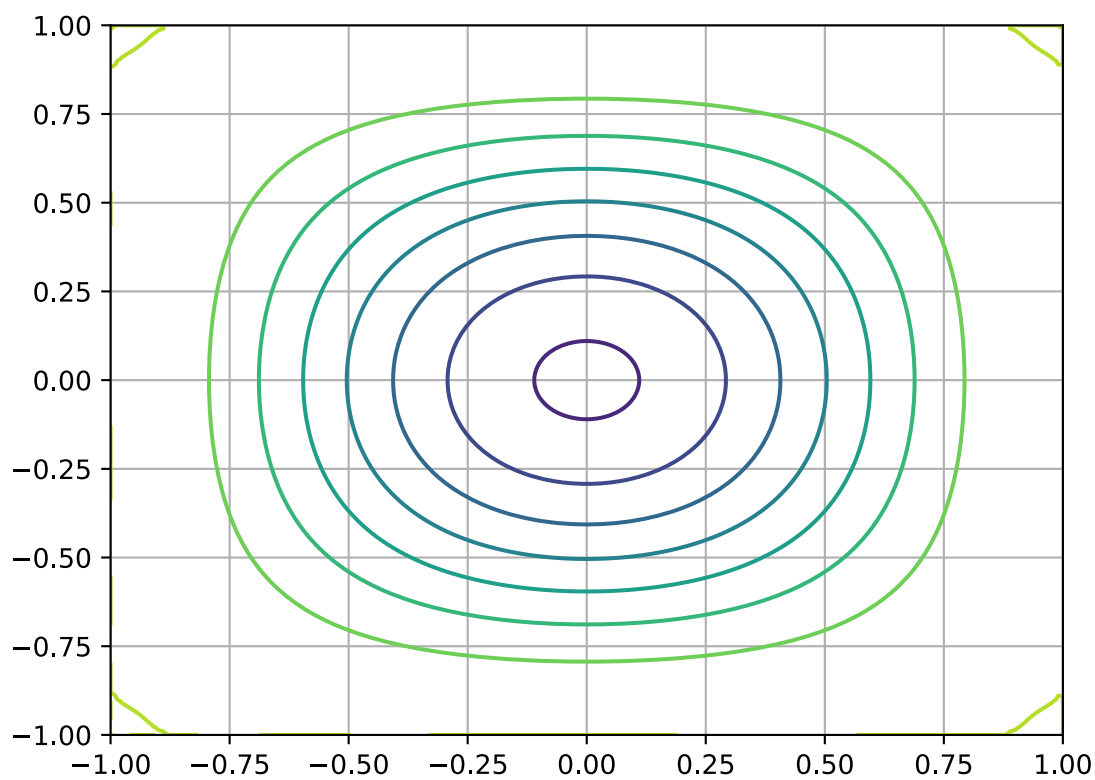
In new code, for regular grids use `RegularGridInterpolator` instead.
For scattered data, prefer `LinearNDInterpolator` or
`CloughTocher2DInterpolator`.

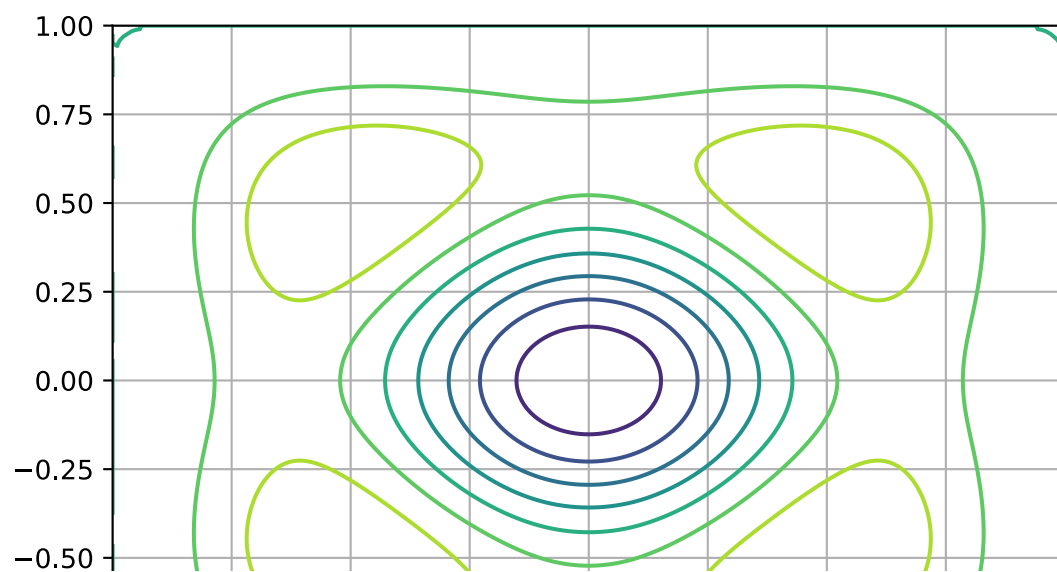
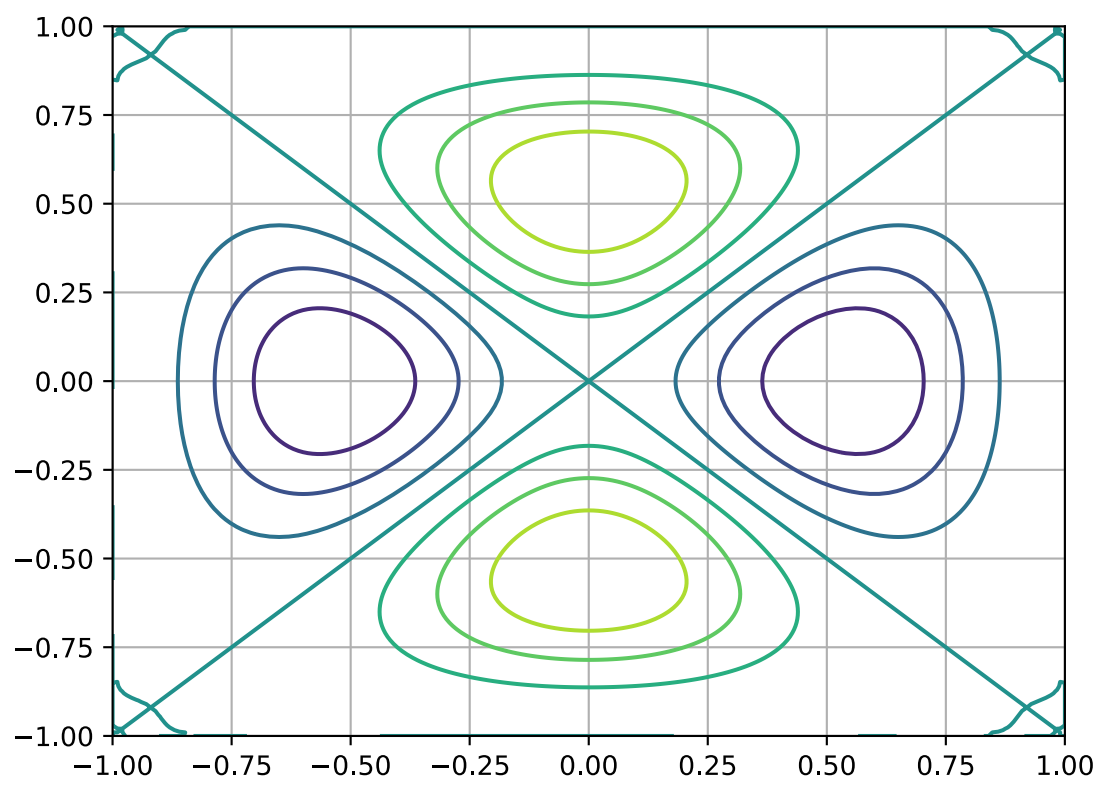
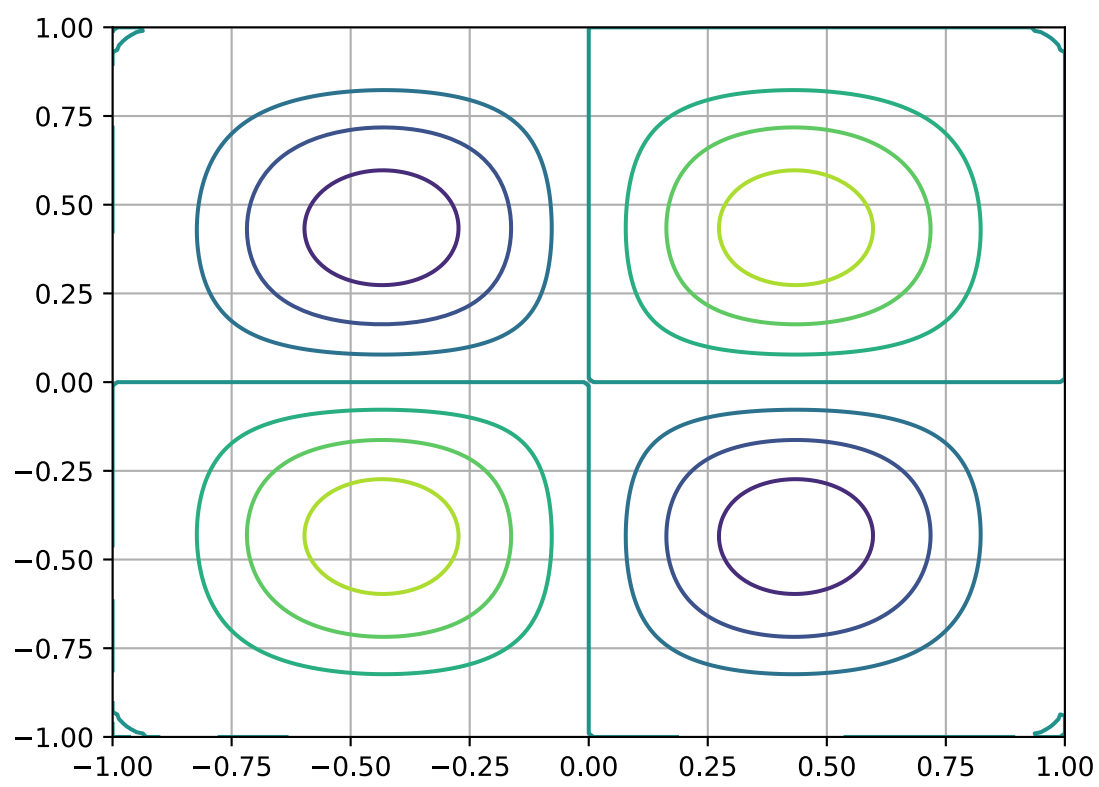
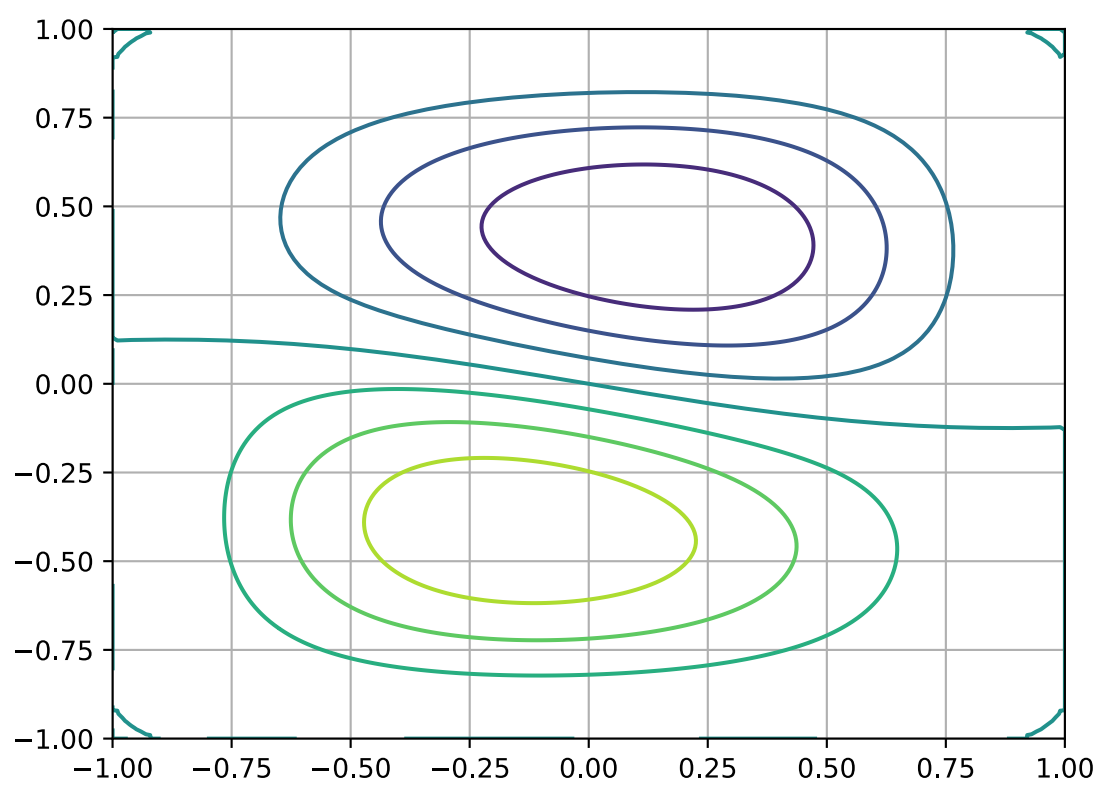
For more details see
[https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff`](https://gist.github.com/ev-br/8544371b40f414b7eaf3fe6217209bff)

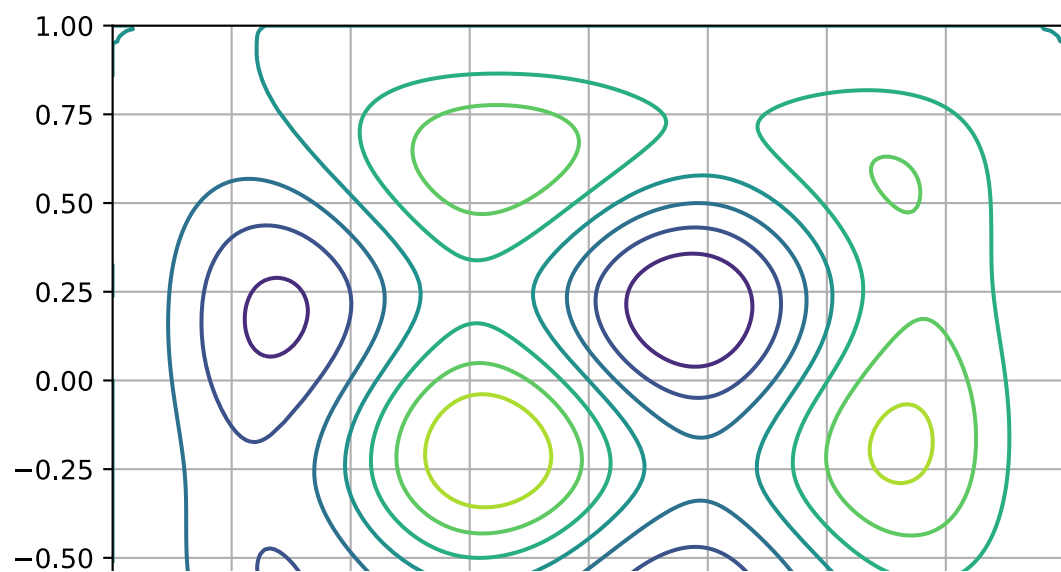
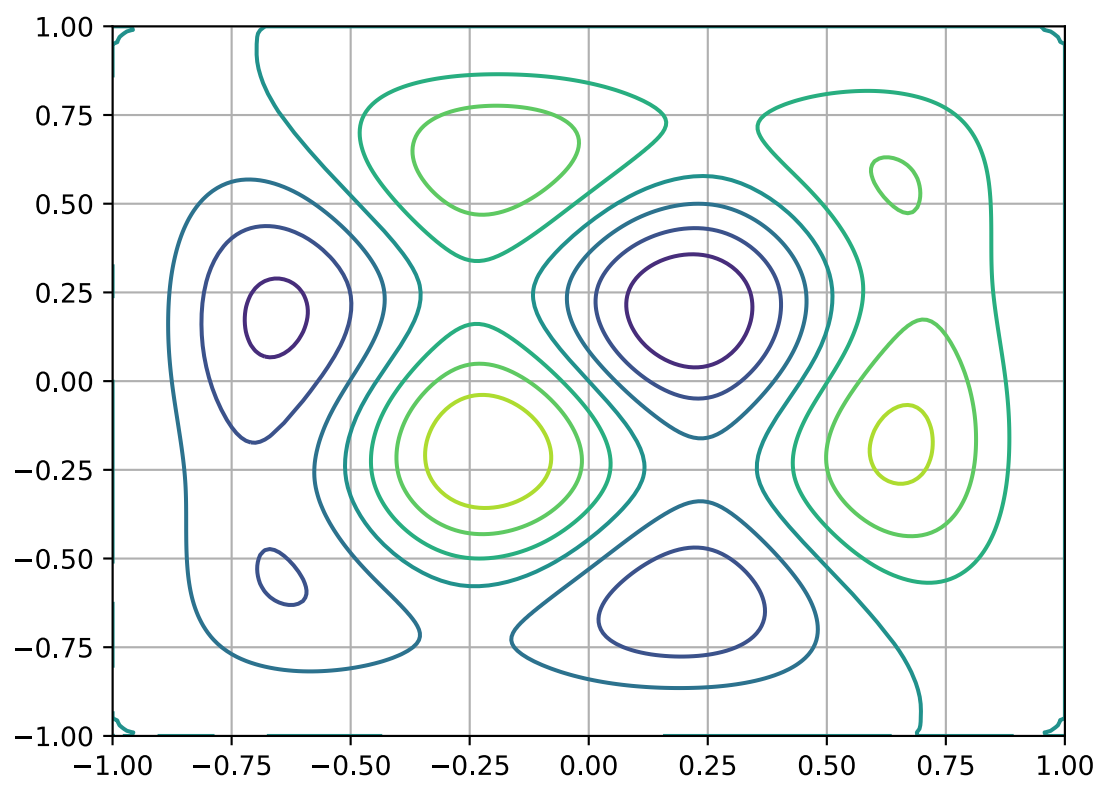
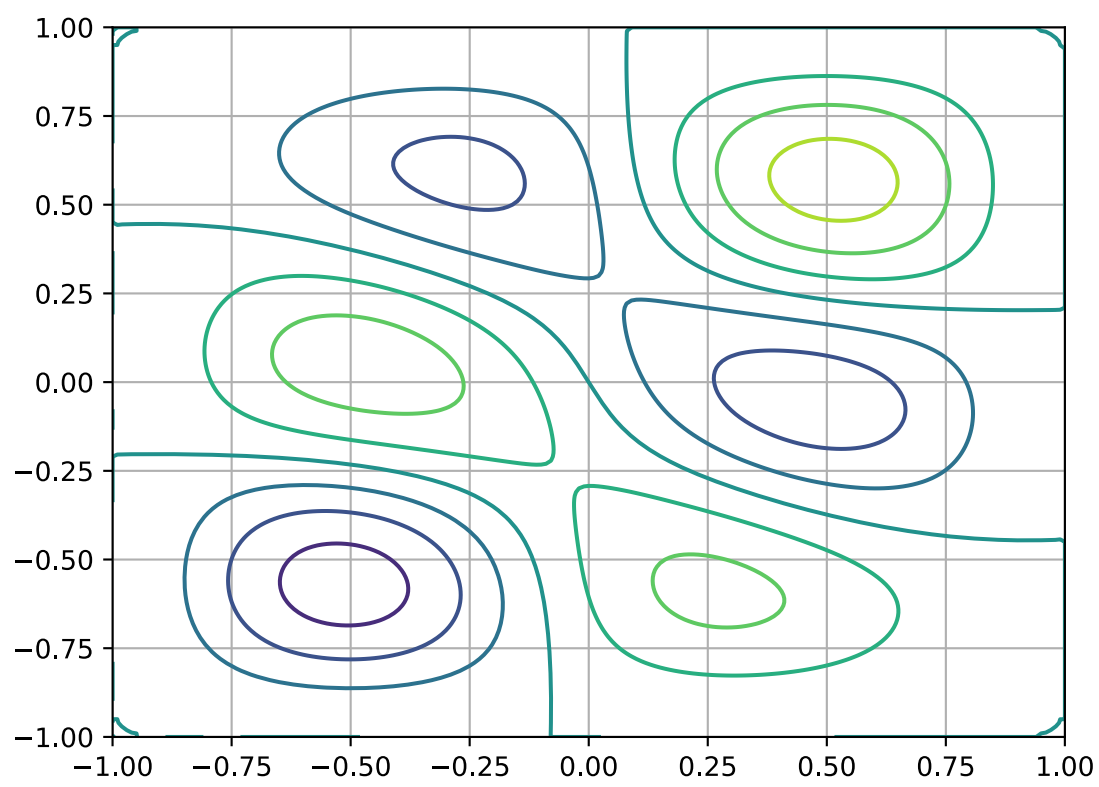
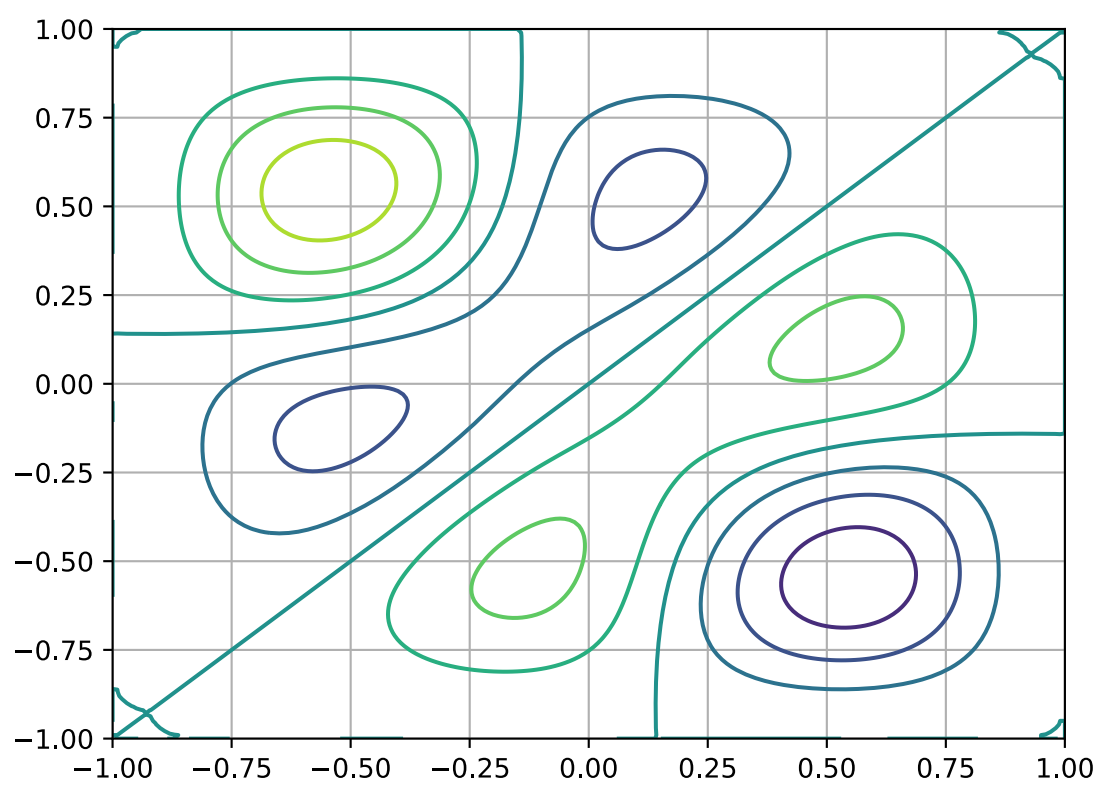
```

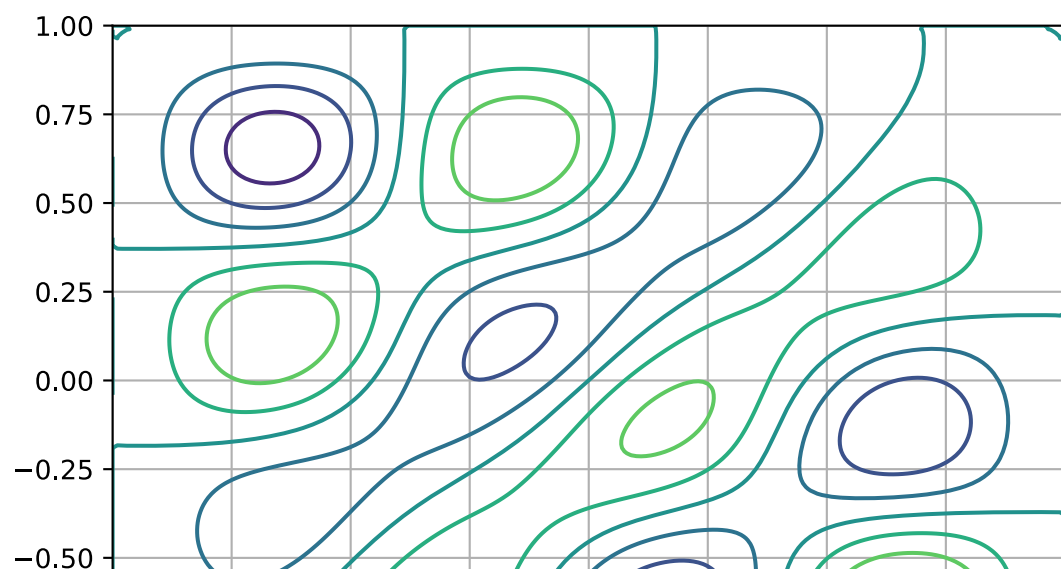
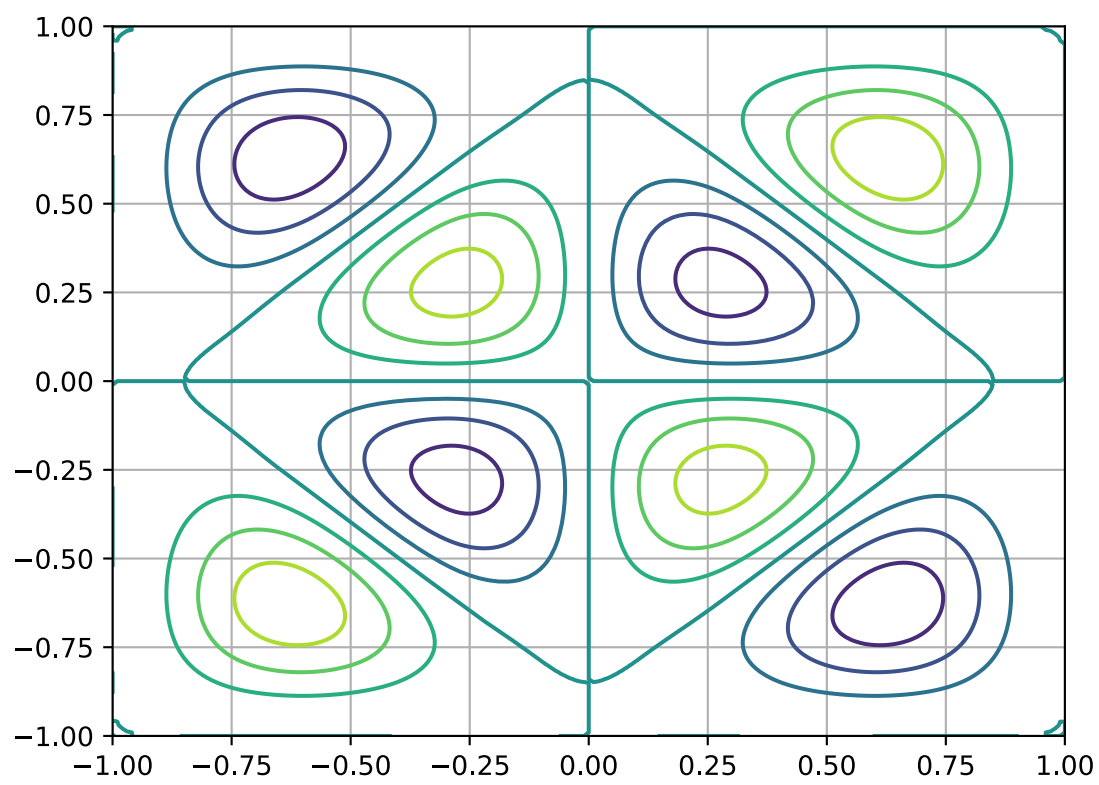
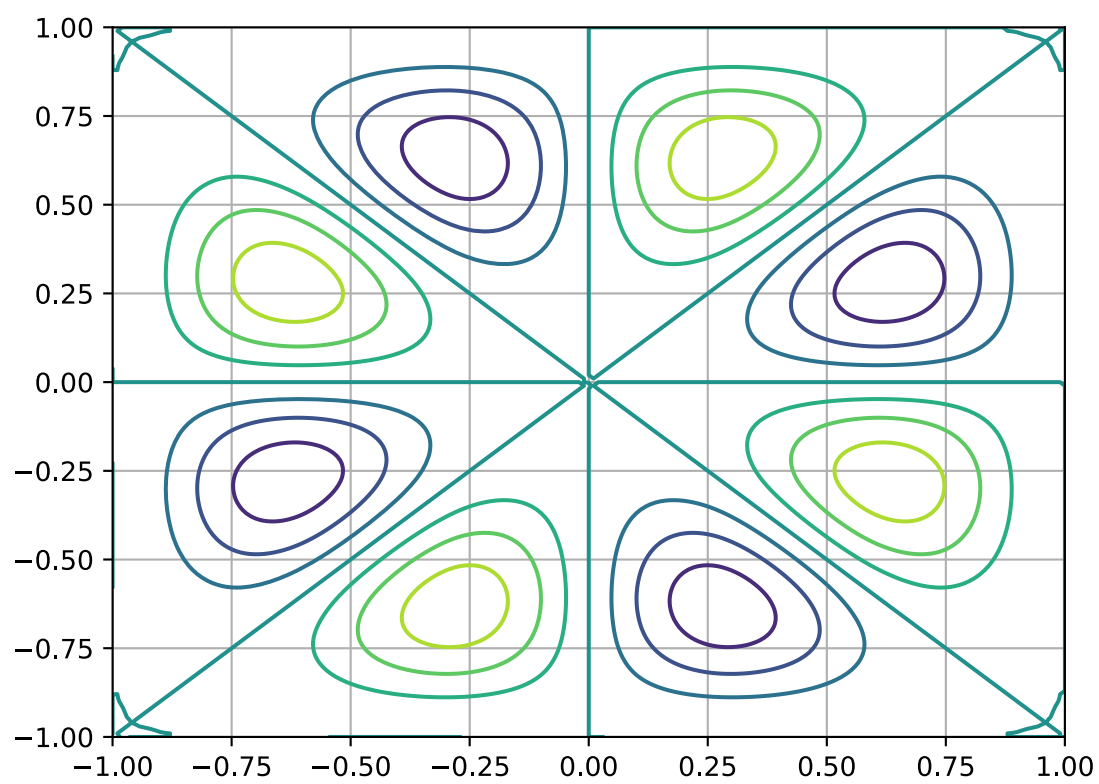
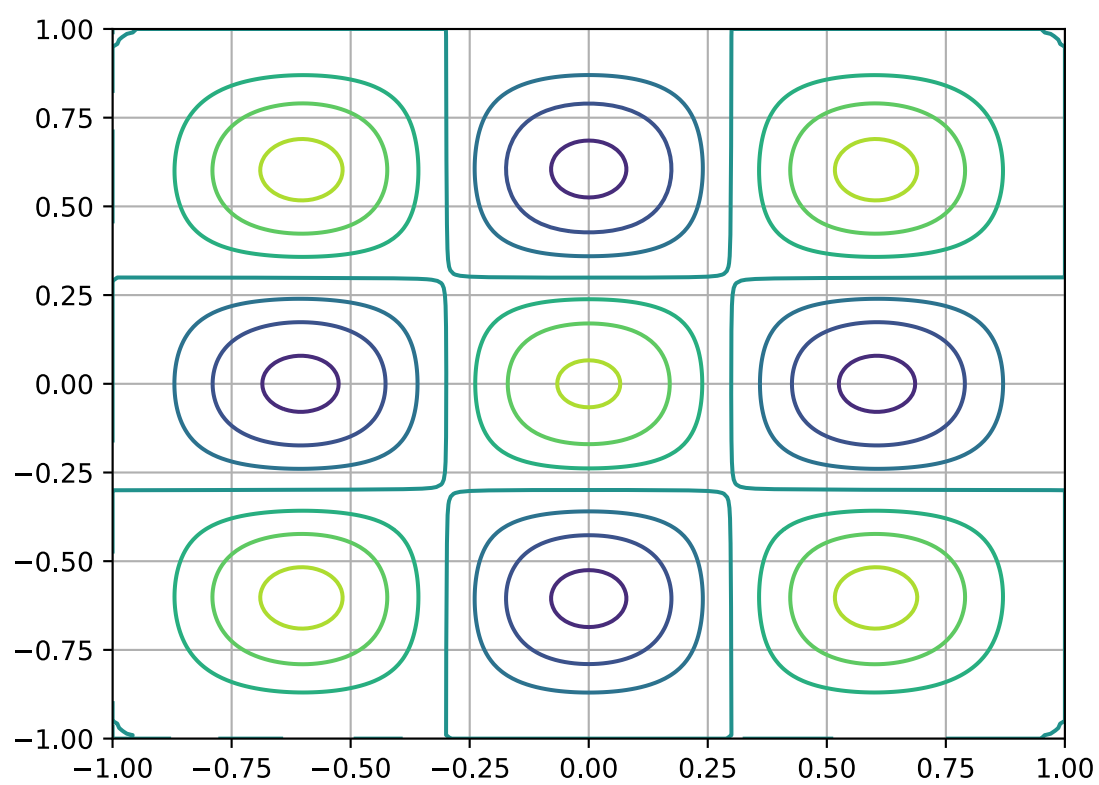
uuu = interpolator(x2,x2)
C:\Users\gary\AppData\Local\Temp\ipykernel_6208\868330203.py:44: RuntimeWarning: More than 20 figur
es have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are
retained until explicitly closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
figure(i+1)

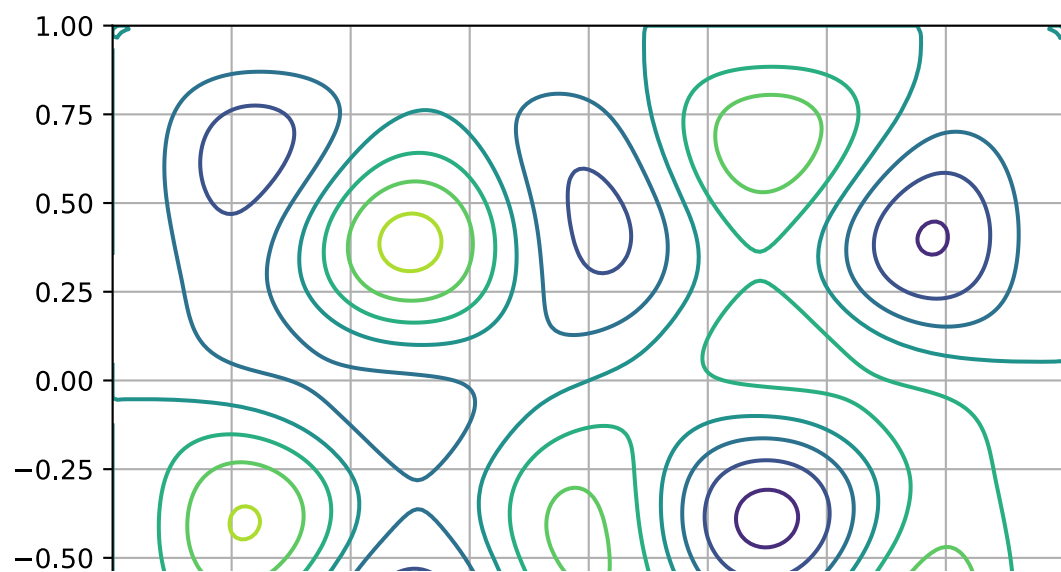
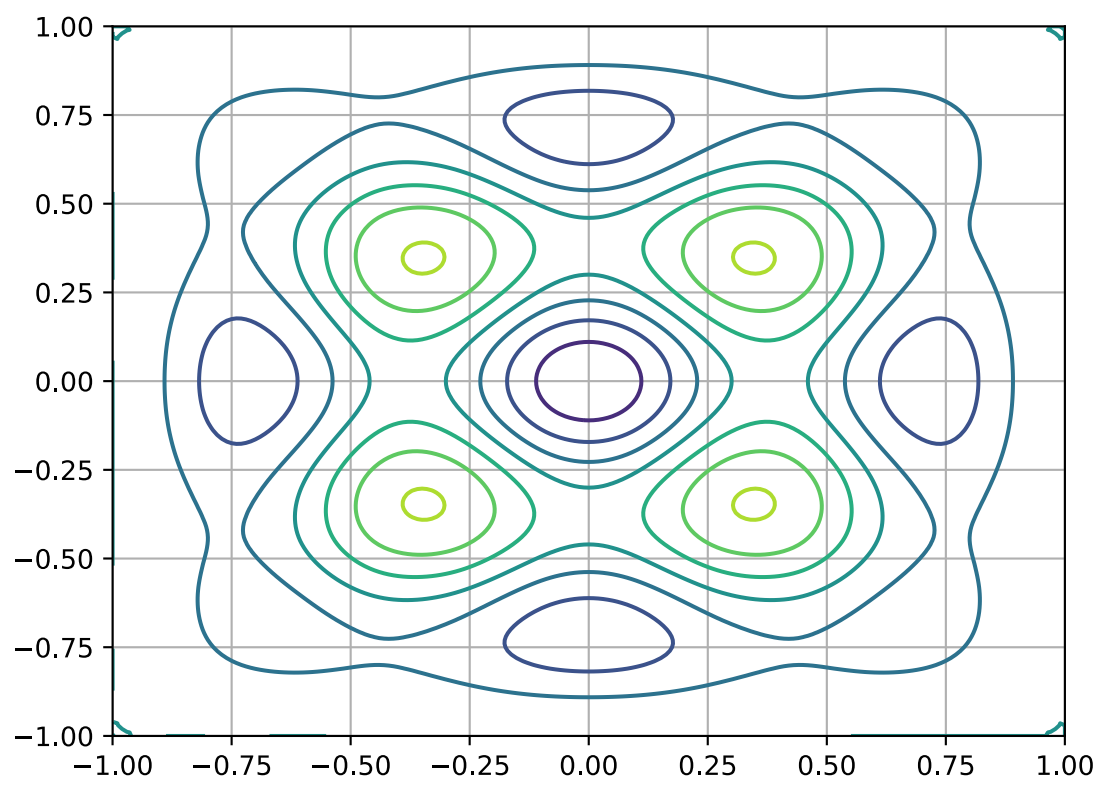
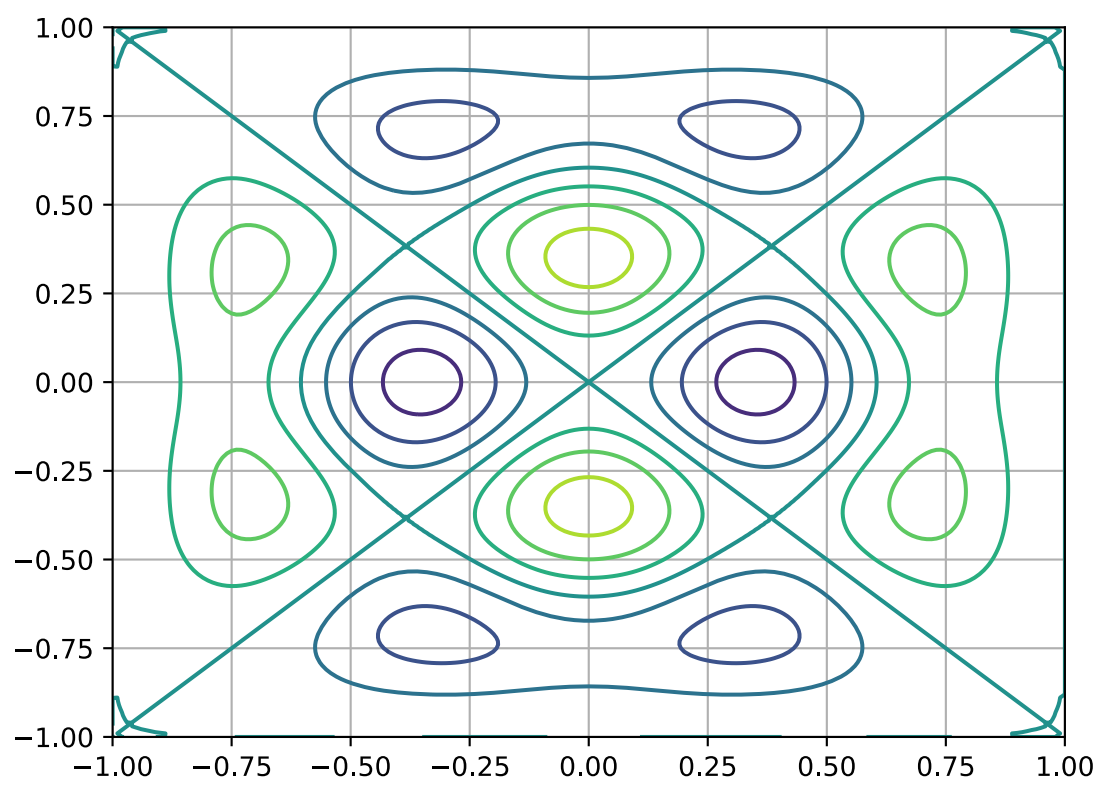
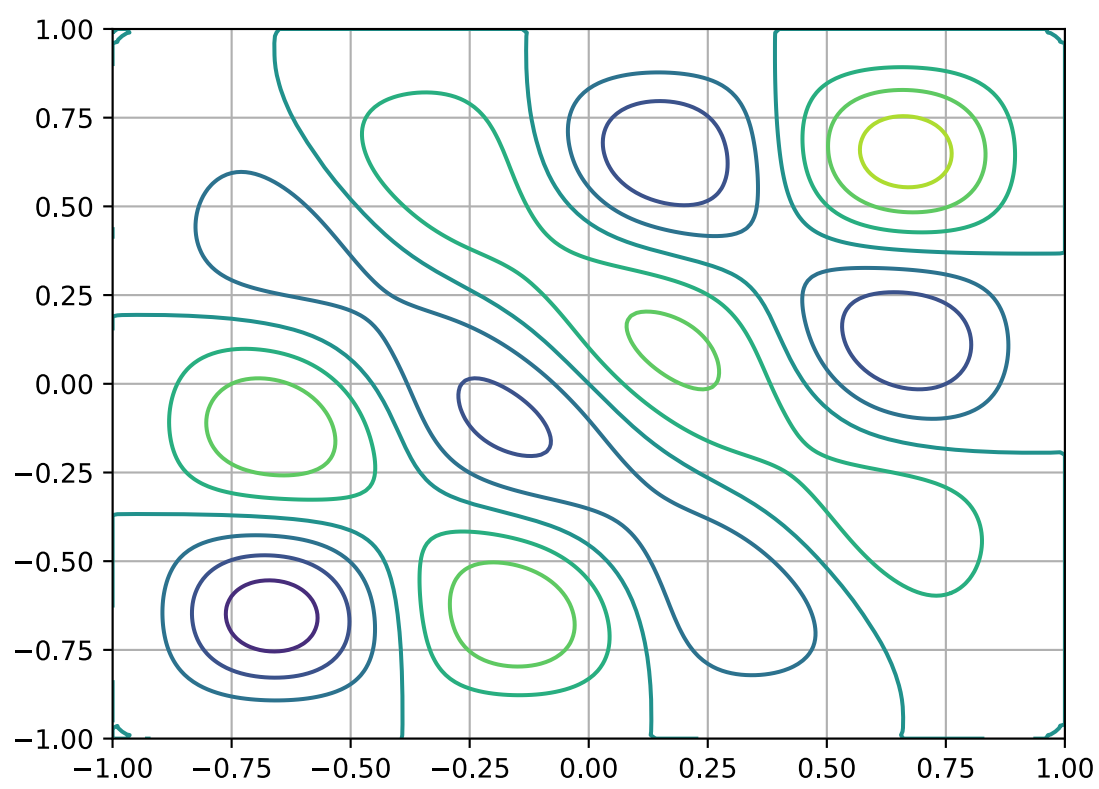
```

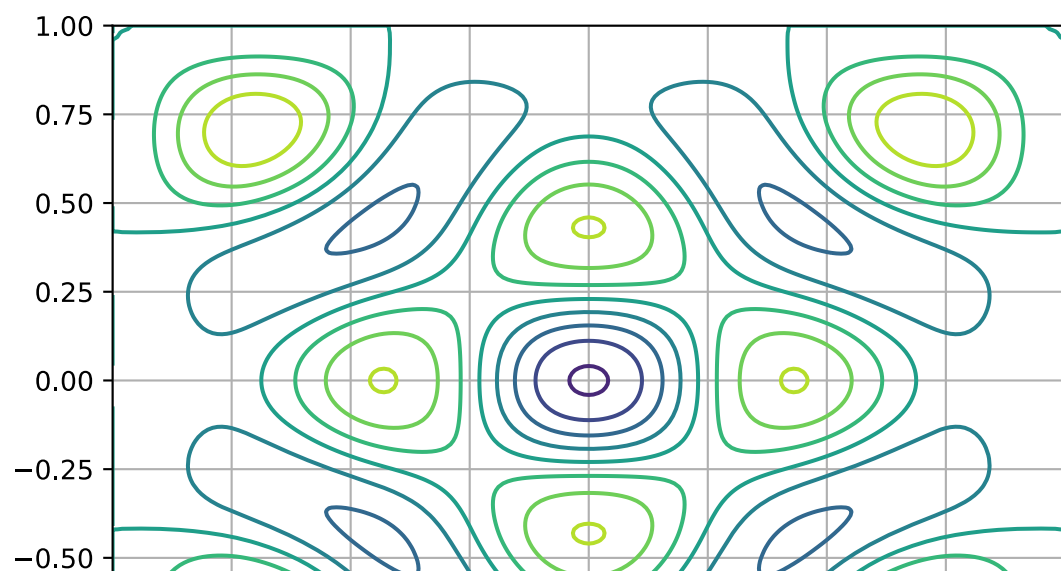
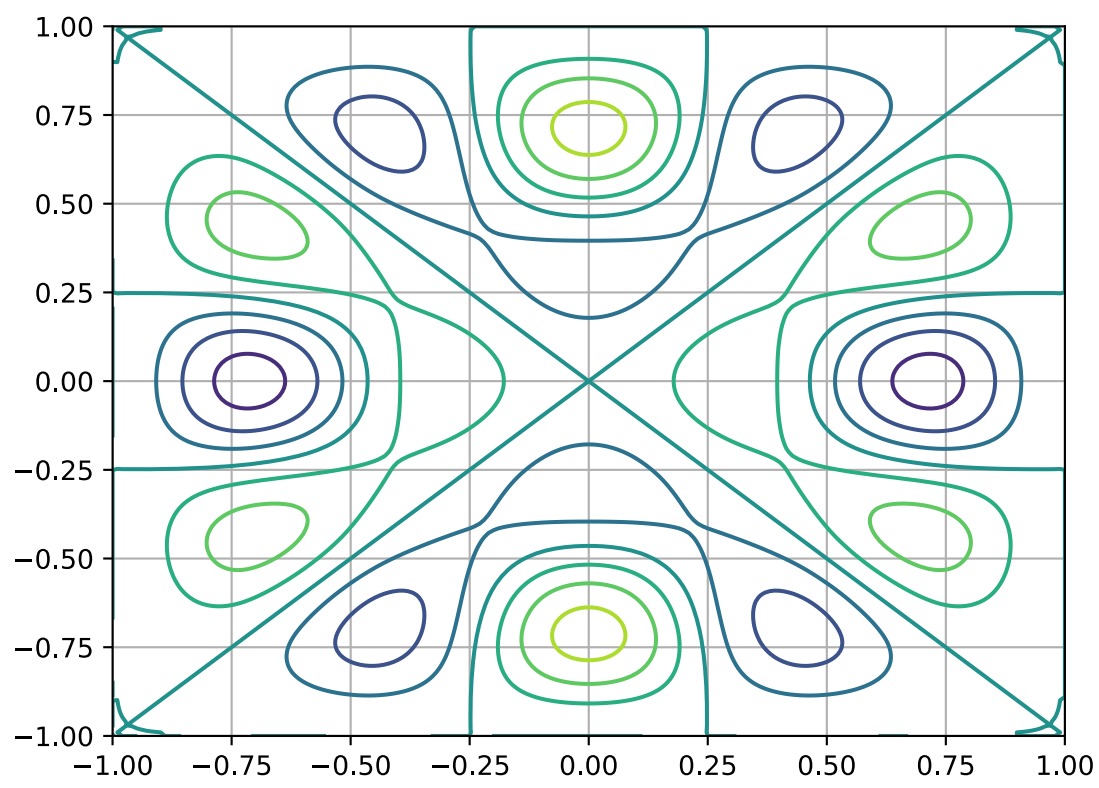
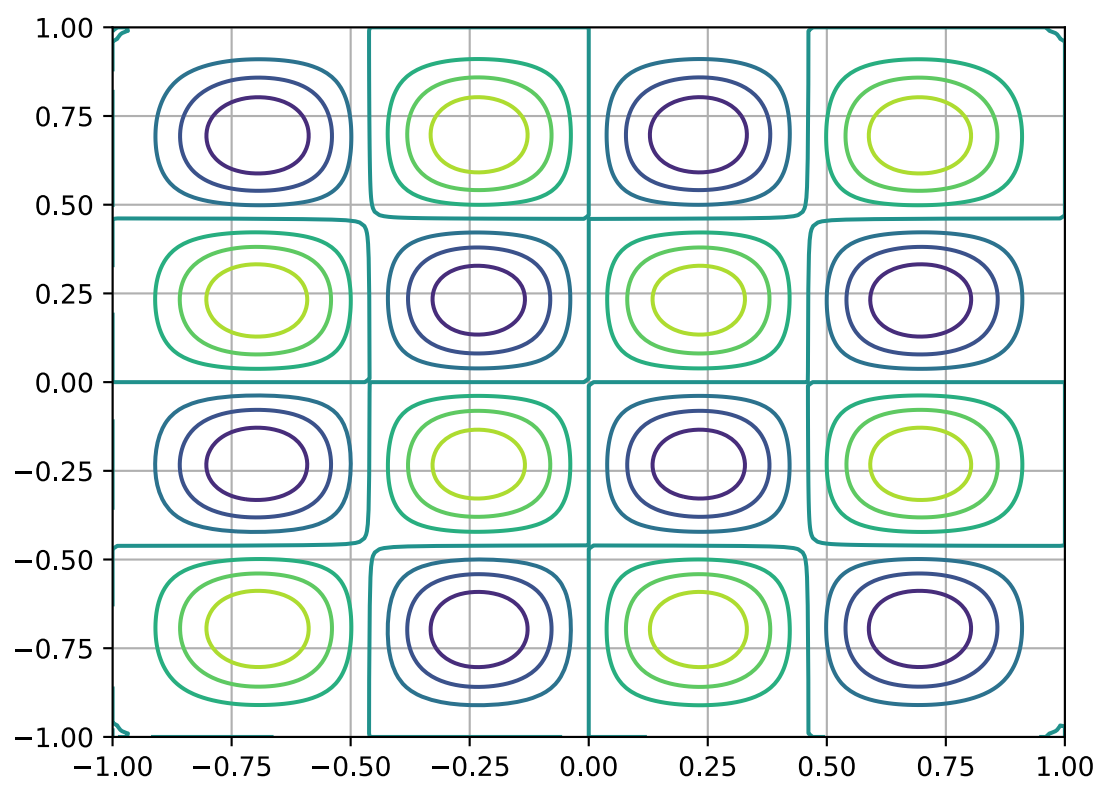
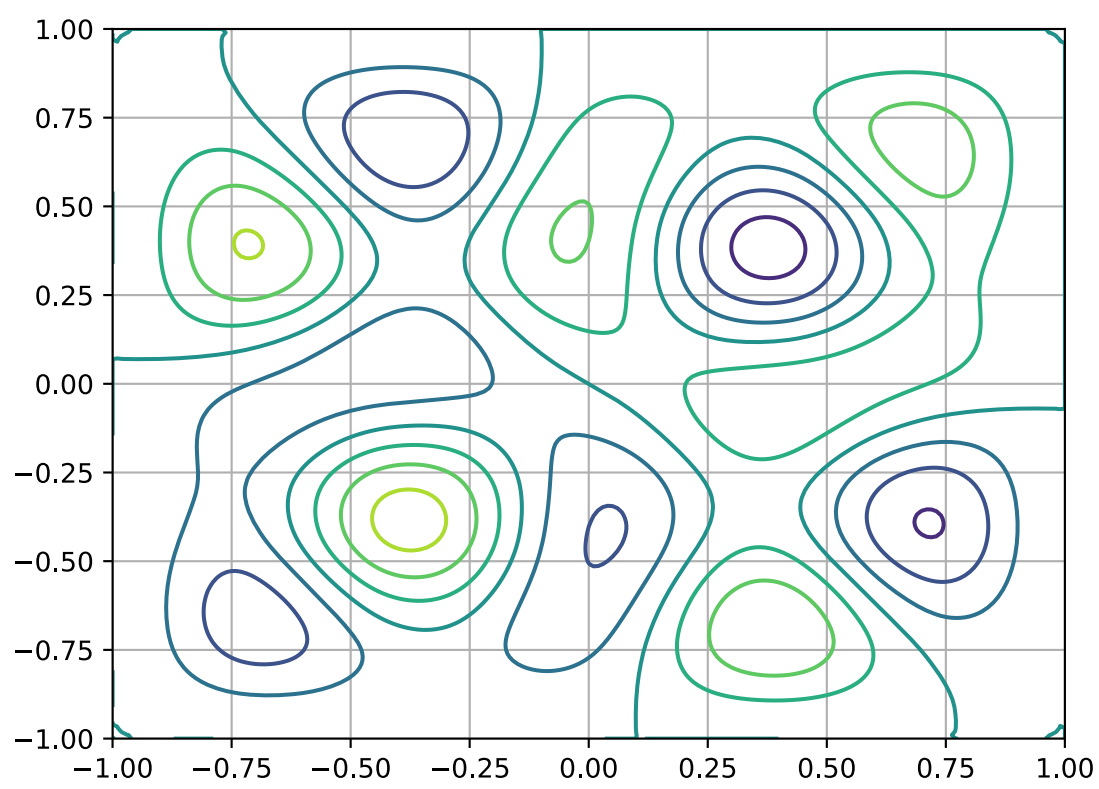


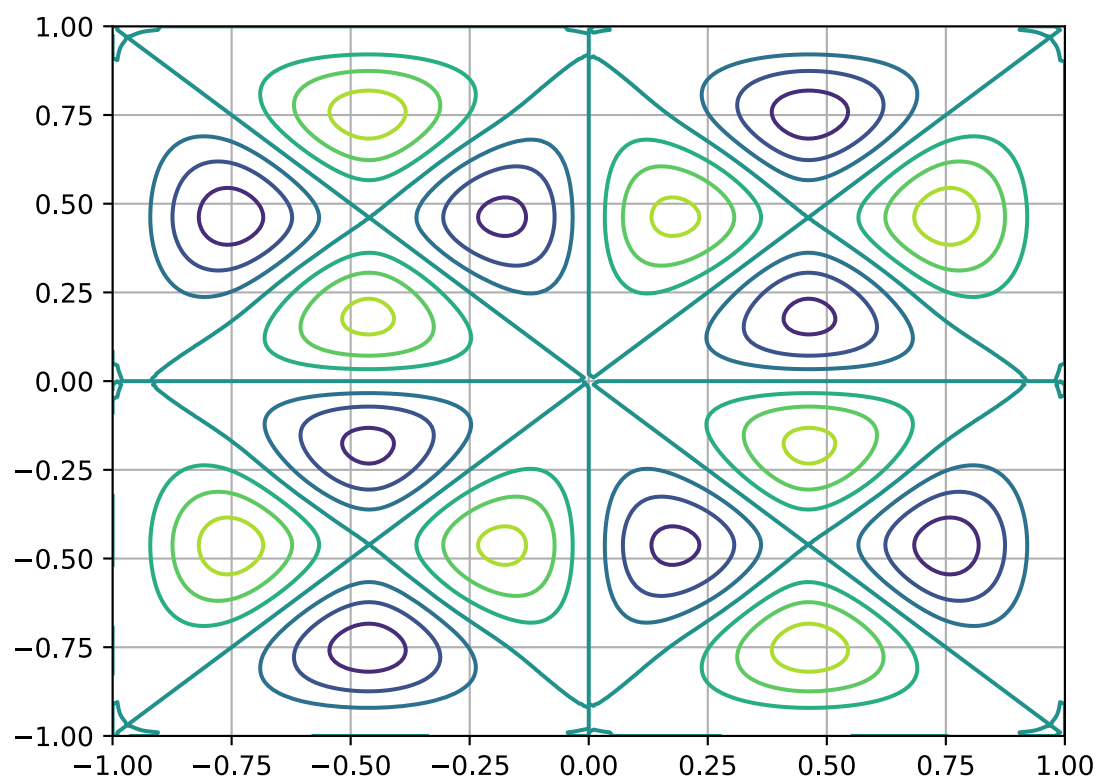
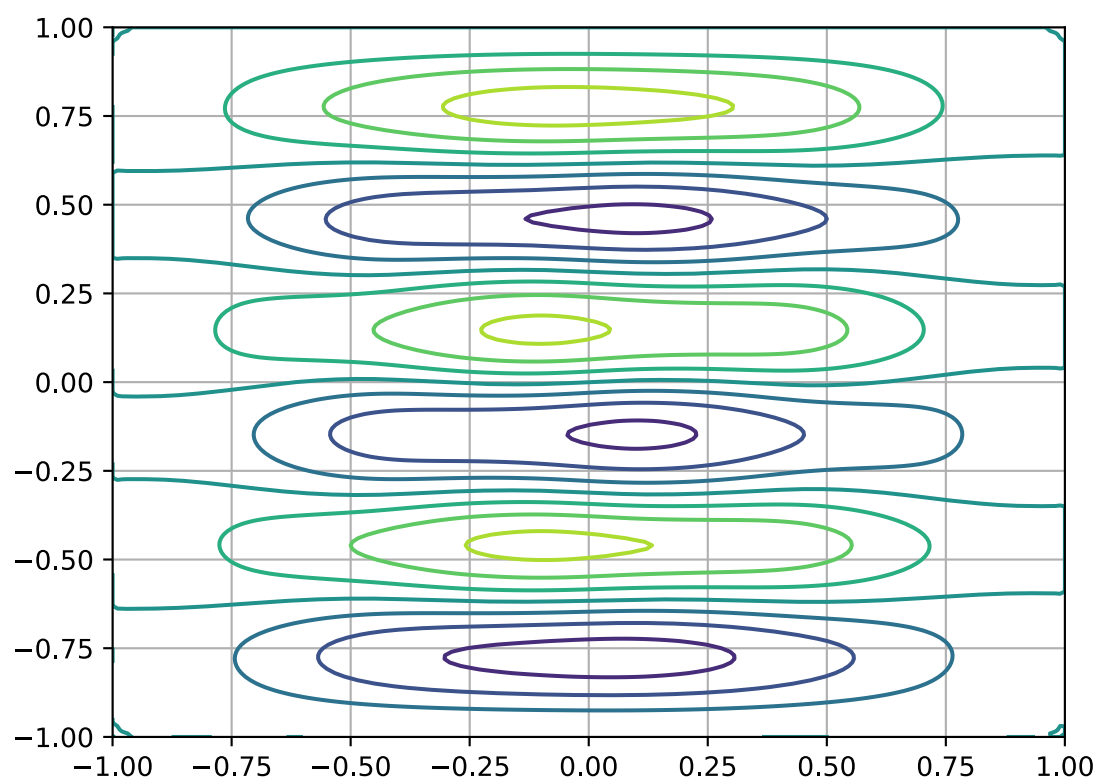
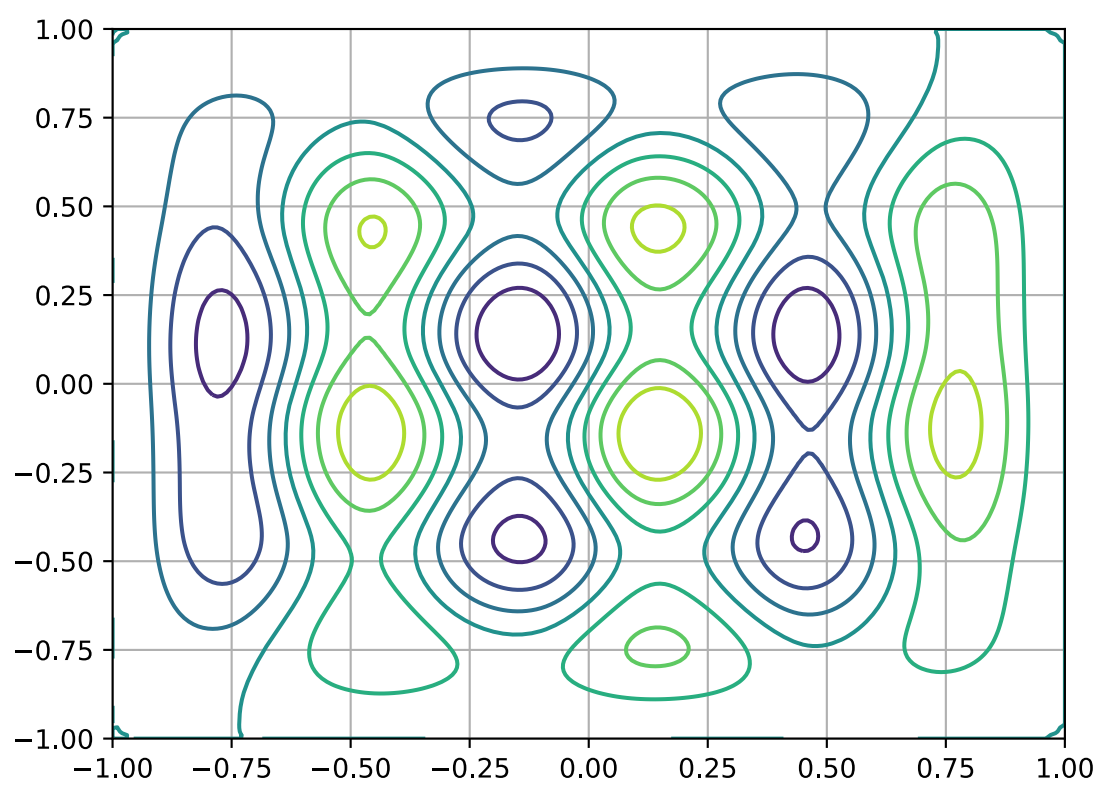












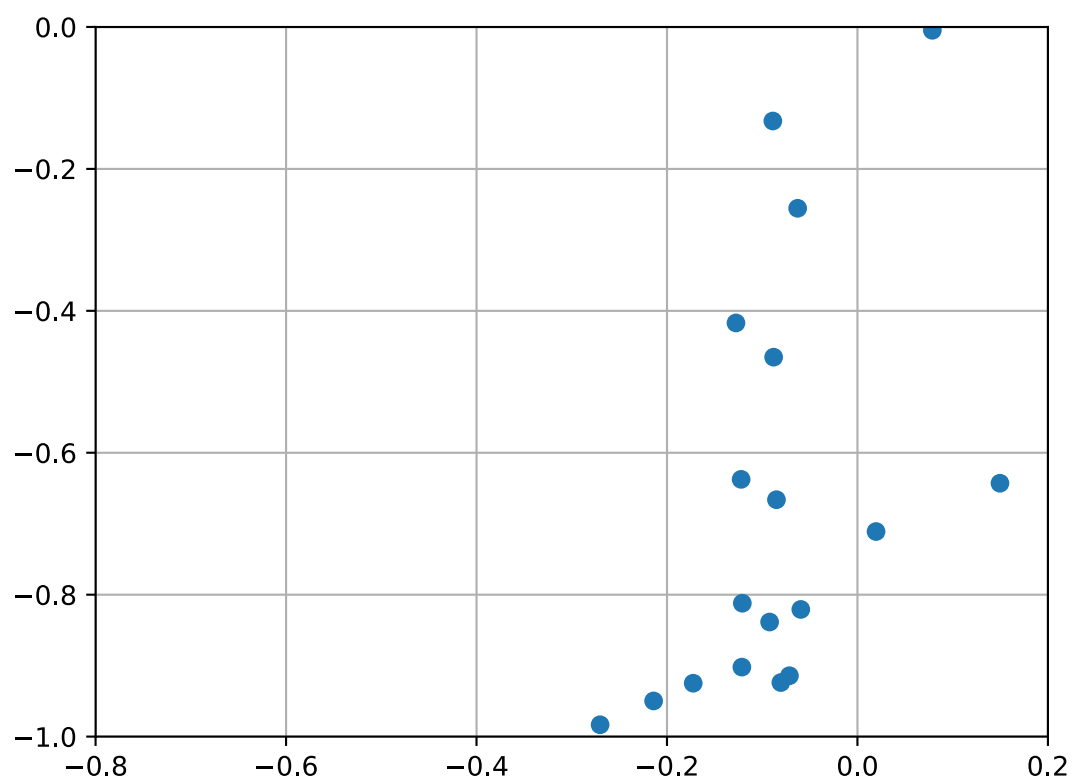
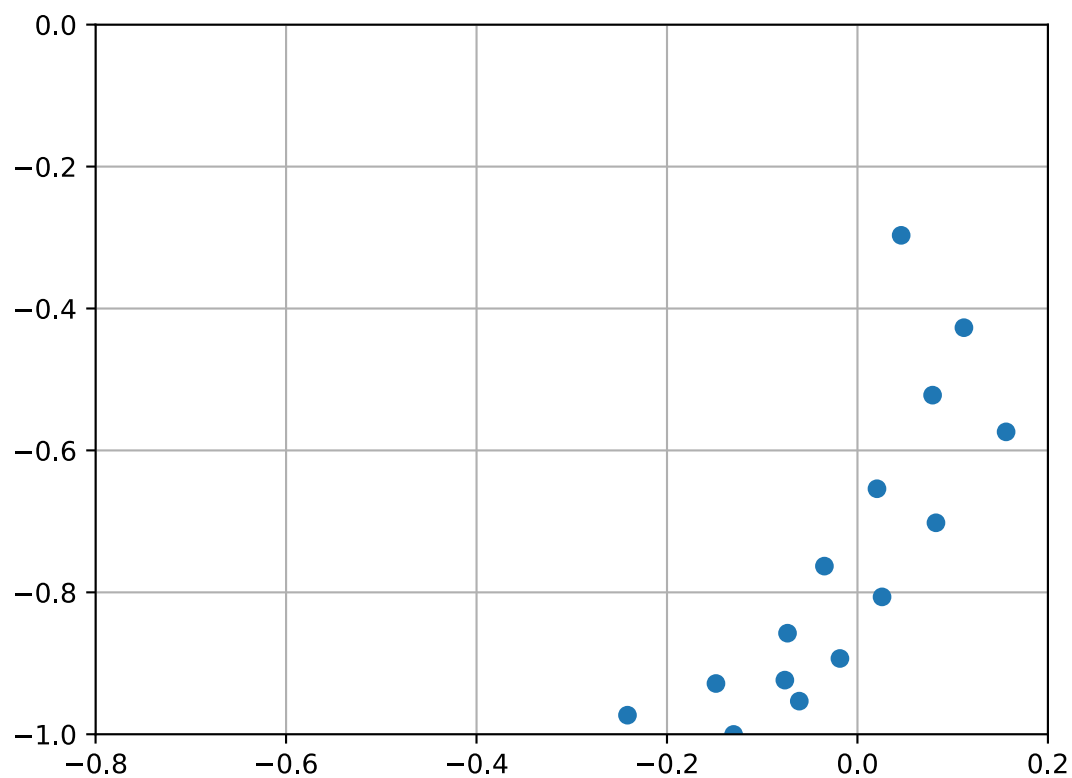
Program 40 : eigenvalues of Orr-Sommerfeld operator

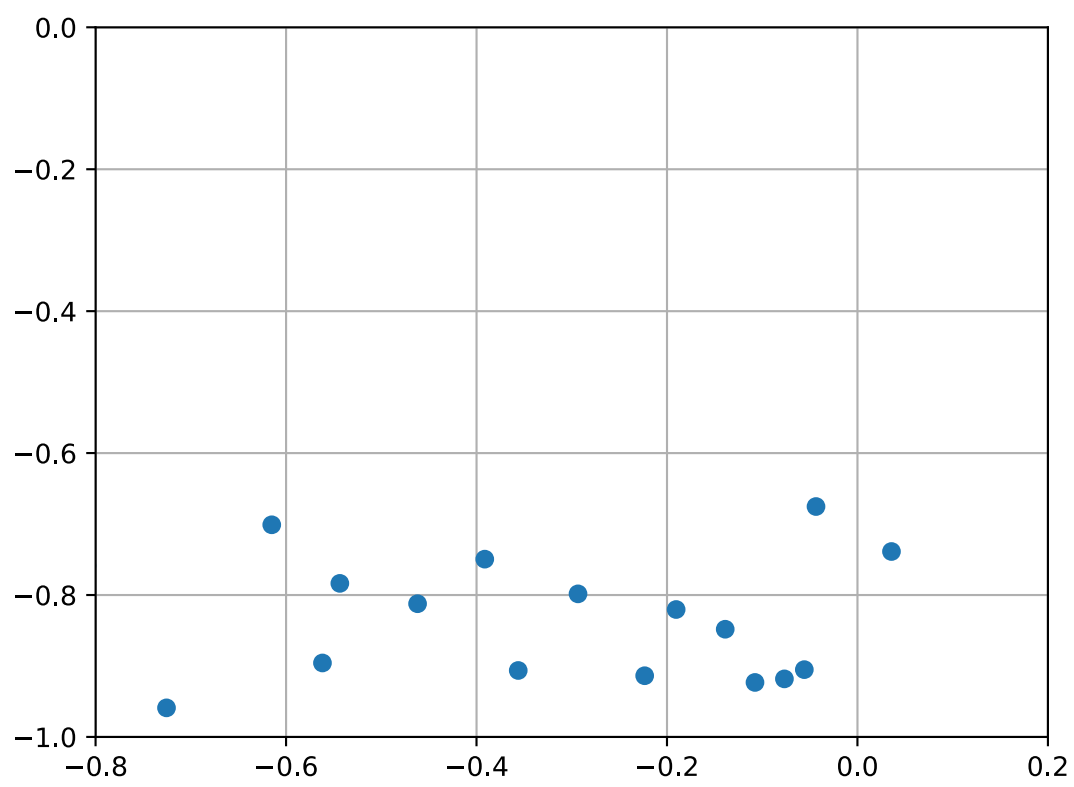
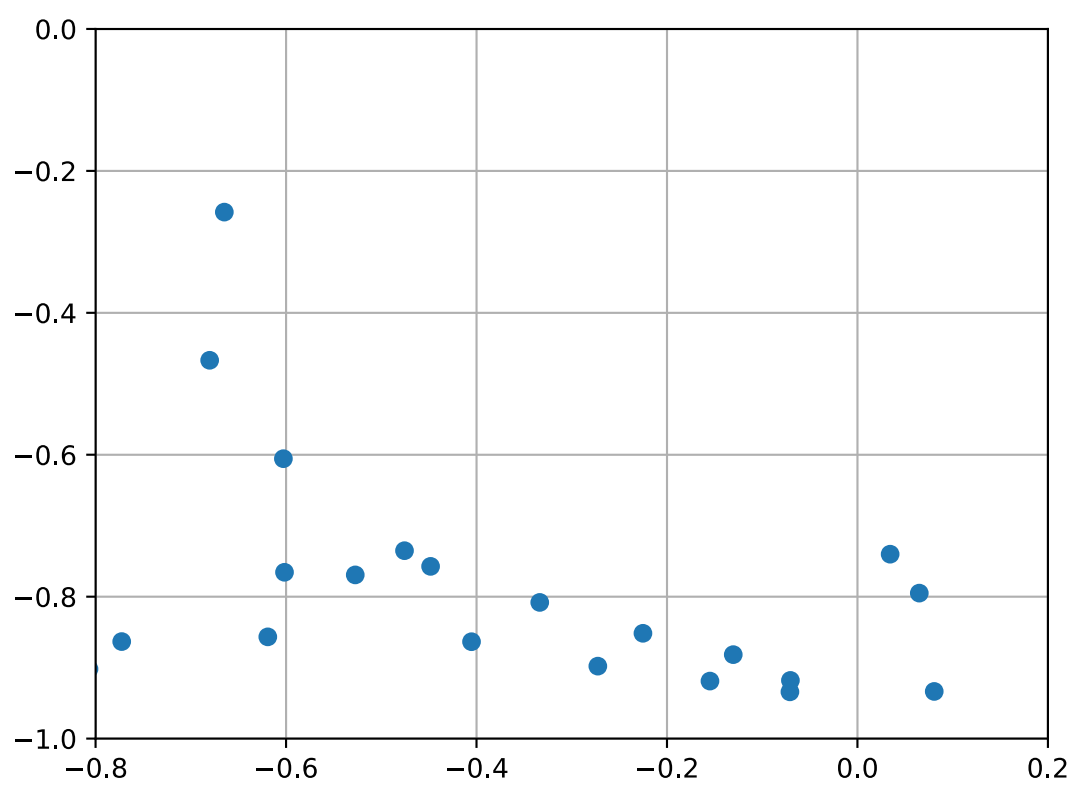
```
In [42]: 1 #from cheb import *
2 from numpy import *
3 from scipy import *
4 from scipy import linalg
5 from matplotlib.pyplot import *
6
7 # eigenvalues of Orr-Sommerfeld operator
8
9 R = 5772
10 j = 0
11 for N in arange(40,120,20): # 2nd- and 4th-order differentiation matrices:
```

```

12     D,x = cheb(N)
13     D2 = matmul(D ,D)
14     D3 = matmul(D2,D)
15     D4 = matmul(D3,D)
16     Dii = D2[1:N,1:N]
17     c = 0
18     c = append( c, 1.0/( 1.0 - x[1:N]**2) )
19     c = append( c, 0 )
20     S = diag( c )
21     M = matmul( diag(1 - x**2), D4 ) - 8*matmul( diag(x), D3 ) - 12*D2
22     D4 = matmul(M,S)
23     Div = D4[1:N,1:N]
24     # Orr-Sommerfeld operators A,B and generalized eigenvalues:
25     I = eye(N-1)
26     A = ( Div - 2*Dii + I )/R - 2*1j*I - 1j*matmul( diag( 1 - x[1:N]**2 ), ( Dii - I ) )
27     B = Dii - I
28     ee,L = linalg.eig(A,B)
29     j = j + 1
30     figure(j)
31     plot(real(ee),imag(ee),'o')
32     xlim(-.8,.2)
33     ylim(-1, 0)
34     grid(True)
35 show()
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```





In []: