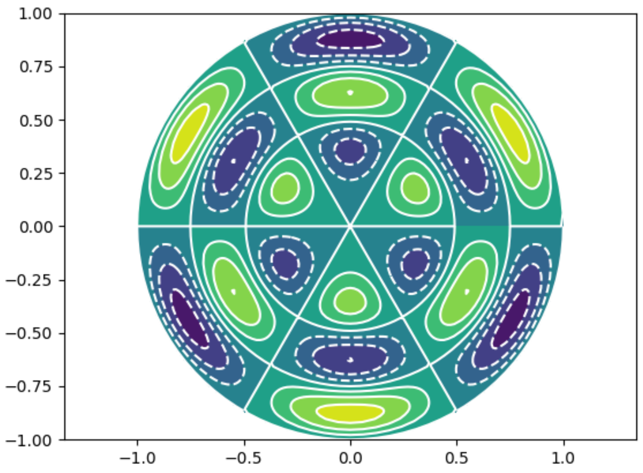
Tools for Scientific Calculations with scipy
Mathematical Programming with Python

https://people.sc.fsu.edu/∼jburkardt/classes/math1800_2023/scipy/scipy.pdf



How the surface of a drum can vibrate.

”The scipy library”

- `scipy` *offers tools for scientific computation;*
- `scipy.csgraph` *sets up and solves very large sparse linear systems;*
- `scipy.fft` *works with Fast Fourier Transforms.*
- `scipy.integrate` *estimates integrals and solves differential equations;*
- `scipy.interpolate` *uses given data to estimate a function at new arguments;*
- `scipy.io` *read and write MATLAB* `mat` *files;*
- `scipy.linalg` *carries out linear algebra operations, including solving full or rectangular linear systems, and computing eigenvalues;*
- `scipy.ndimage` *carries out image processing;*
- `scipy.optimize` *seeks the arguments that minimize a function;*
- `scipy.signal` *signal processing;*
- `scipy.spatial` *geometry functions (hull, Voronoi, Delaunay);*
- `scipy.special` *evaluates special functions*
- `scipy.stats` *evaluates statistical functions;*

1 A Python library for scientific computation

The `scipy` library adds functions that are needed for computations across fields such as biology, chemistry, engineering, physics, statistics.

In most cases, `scipy` work with arrays from `numpy` and hence are vectorized. They often rely on underlying code compiled in Fortran, C, or C++, and hence are highly optimized.

Installation instructions and documentation are available at `https://scipy.org/`.

The `scipy` library is organized into sublibraries, some of which will be considered in the following discussion.

2 scipy.special

The `scipy.special` library can evaluate many of the special functions used in probability, thermodynamics and physics. These include Airy, Bessel, Beta, Elliptic, Error, Gamma, and many statistical functions. Operations include evaluation, integration, differentiation, zeros and inverses.

To access a specific function, use an `import` statement:

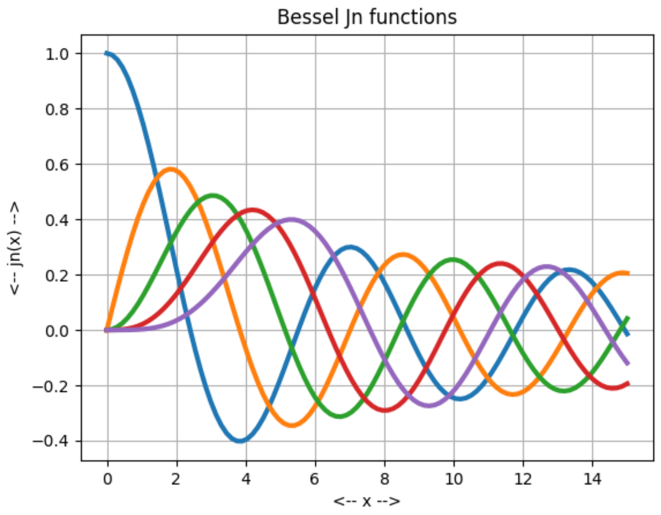
```
from scipy.special import airy
```

For help on the entire library, try:

```
help ( special )
```

As an example, `scipy.special` includes `jn(n,x)`, which evaluates the Bessel $J_n(x)$ function at the point x. Bessel functions are a generalization of the sine function, and are important in problems involving heat conduction and membrane vibration.

Here is a plot of $J_n(x)$ for $0 \leq n \leq 4$:

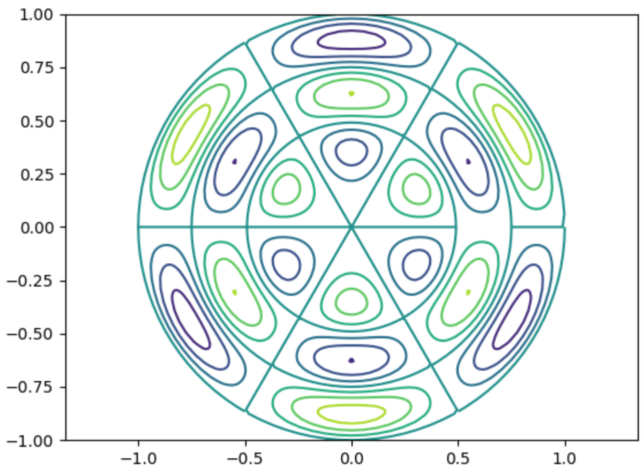


Example E8.3, from our textbook by Christian Hill, considers possible modes of vibration of a circular drum head. The parameters m and n control how many wiggles occur in the radial and angular directions. The radial variation is described by a Bessel function $J_n(r)$, while the angular variation involves $\sin(n\theta)$.

The following code computes the mode for $m = 3, n = 2$.

```
def drum_normal_modes ( ):

  from scipy.special import jn
  from scipy.special import jn_zeros
  import matplotlib.pyplot as plt
  import numpy as np
  r = np.linspace ( 0.0, 1.0, 101 )
  theta = np.linspace ( 0.0, 2.0 * np.pi, 101 )

  R, T = np.meshgrid ( r, theta )
  X = R * np.cos ( T )
  Y = R * np.sin ( T )

  n = 3
  m = 2
  bessel_zeros = jn_zeros ( n, m + 1 )
  k = bessel_zeros[m]
  Z = R * np.sin ( n * T ) * jn ( n, R * k )

  plt.clf ( )
  plt.contour ( X, Y, Z )
  plt.show ( )
```

and the resulting contour plot (green is up and blue is down) is:



3 `scipy.integrate` for integrals

The `scipy.integrate` library can estimate the integral over some interval $[a, b]$ or rectangular domain, of a function $f(x)$ given as a formula. It can also estimate integrals when the function is only available as sample data values. Special cases can handle integration over rectangular regions in 2D and 3D.

The integrator `quad()` is recommended for the most common case, in which a function $f(x)$ is to be integrated over an interval. As our integrand, we will choose the Bessel $J_v(x)$ function. Here, the suffix v indicates that the Bessel parameter is a real number, not an integer. Our integration interval is $0 \leq x \leq 4.5$ and our v parameter is 2.5.

The code to estimate the integral is as follows:

```
def bessel_j_quad ( ):
  from scipy.integrate import quad
  xmin = 0.0
  xmax = 4.5
  result, err = quad ( integrand, xmin, xmax )
  print ( '  Integral estimate is ', result )
  print ( '  Error estimate is     ', err )
  return

def integrand ( x ):
  from scipy.special import jv
  value = jv ( 2.5, x )
  return value
```

Notice that we are passing the absolute minimum information to `quad()`, and that we get an estimated error for the integral that is returned.

Other functions are available for which the user can request a specific error tolerance, require that a certain Gauss quadrature rule be used, or in other ways modify the procedure by which an integral is estimated.

4 `scipy.integrate` for differential equations

In an earlier class, we already encountered the function `solve_ivp()` for solving one or several ordinary differential equations. For completeness, we will included a short example of this calculation here, involving a version of the logistic equation:

$$\frac{dy}{dy} = 2 * y * (3 - y)$$

with initial condition $y(0) = 1$, to be solved over the interval $0 \leq t \leq 4$. Here is how we might use `solve_ivp()` and plot the resulting solution:

```
def logistic_ode ( ):
  from scipy.integrate import solve_ivp
  import matplotlib.pyplot as plt
  import numpy as np
  tmin = 0.0
  tmax = 4.0
  y0 =  np.array ( [ 1.0 ] )
  sol = solve_ivp ( logistic_dydt , [tmin,tmax], y0 )
  plt.plot ( sol.t, sol.y[0] )
  plt.show ( )
  return

def logistic_dydt ( t, y ):
  dydt = 2.0 * y * ( 3.0 - y )
  return dydt
```

Notice that the result is returned as a structure called `sol` which includes the values of time and the solution, but also a number of other pieces of information.

5 `scipy.ndimage`

The `scipy.ndimage` library provides functions for manipulating images, which might be stored as `.png` files. Typical operations include creating a new image from a portion of the original, modifying the color range, searching for edges, rotating the image, coarsening or smoothing.

Here is a simple example in which we read in an image file, display it, rotate it, and blur it using a Gaussian filter.

```
  import matplotlib.pyplot as plt
  import numpy as np
  import scipy as sp
  import scipy.ndimage as nd
  face = plt.imread ( 'face.png' )
  plt.imshow ( face )
  plt.show ( )

  face_rotated = nd.rotate ( face, 45 )
  plt.imshow ( face_rotated )
  plt.show ( )

  face_blurred = nd.gaussian_filter ( face, sigma = 3 )
  plt.imshow ( face_blurred )
  plt.show ( )
```

6 scipy.optimize

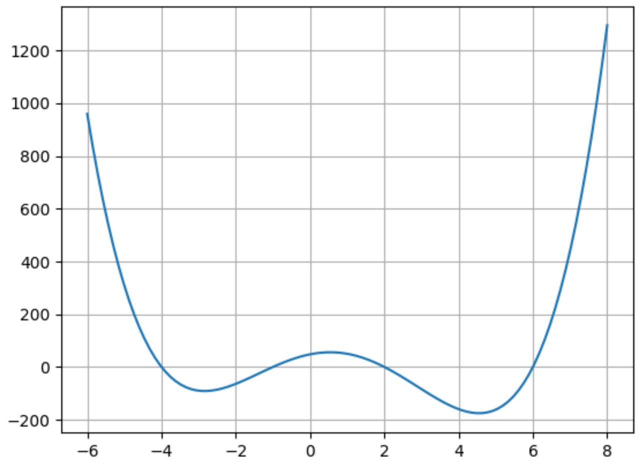
The `scipy.optimize` library is given a function $f(x)$ and seeks a value x for which $f(x)$ attains a maximum or minimum value. The argument x can be a scalar or a vector. The function $f(x)$ might itself also be a vector, in which case a least squares approach might be used, seeking to minimize $||f(x)||$. The search may be restricted to an interval or other bounded domain, or there may be other constraints imposed.

As an easily understood introduction, we consider the minimization of a polynomial function

$$p(x) = x^4 - 3x^3 - 24x^2 + 28x + 48$$

We can specify a bounding interval for the function, but if we don't, the search will begin in the interval [0,1].

Here is a plot of the function



Here is a brief code that sets up the problem and requests a minimizer:

```
def polynomial_minimize ( ):

  import numpy as np
  from scipy.optimize import minimize_scalar

  result = minimize_scalar ( p )
  print ( '  Minimizer found at ', result.x )
  print ( '  p(x) = ', p ( result.x ) )
  return

def p ( x ):
  value = x**4 - 3.0 * x**3 - 24 * x**2 + 28 * x + 48
  return p
```

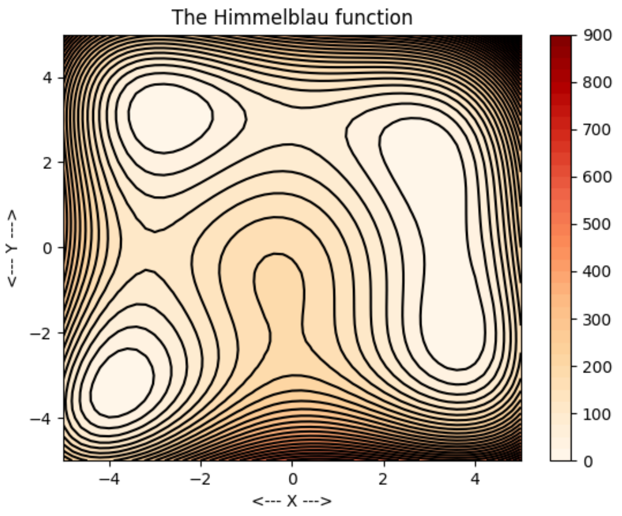
The code finds a (local) minimizer at around $x = -2.84$ where $p(x) = -91.32$. From the plot, we can see a better minimizer and we could redirect the code to search for it by adding an argument such as `bounds=[0,6]`.

As a more advanced optimization problem, we consider the minimization of the Himmelblau function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

starting from an initial guess at $(x, y) = (0, 0)$.

Here is a contour plot of the function



Here is a brief code that sets up the problem and requests a minimizer:

```
def himmelblau_minimize ( ):

  import numpy as np
  from scipy.optimize import minimize

  start = np.array ( [ 0.0, 0.0 ] )
  result = minimize ( himmelblau, start )
  print ( '  Minimizer found at ', result.x )
  print ( '  f(x,y) = ', himmelblau ( result.x ) )
  return

def himmelblau ( x ):
  f = ( x[0]**2 + x[1] - 11.0 )**2 + ( x[0] + x[1]**2 - 7.0 )**2
  return f
```

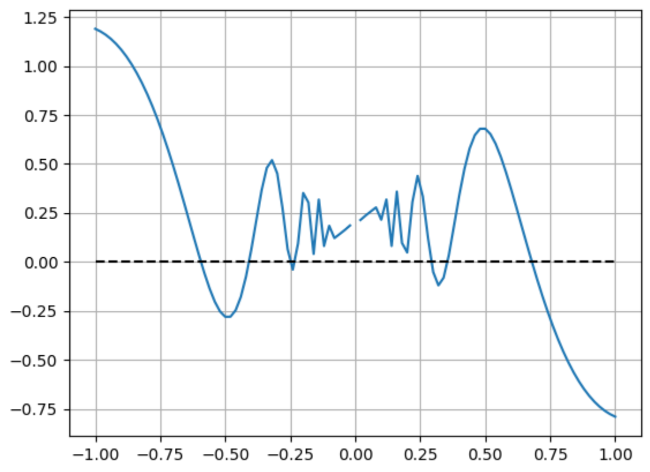
In this example, at the starting value, we have $f(0, 0) = 170$, while at the minimizer, we find $f(3, 2) = 1.3e - 13$.

The `optimize` library includes several functions which actually seek a root of a function $f(x)$, that is, a value such that $f(x) = 0$. As an example, consider the following rather nasty function:

$$f(x) = \frac{1}{5} + x\cos(\frac{3}{x})$$

for which we seek a root in the interval $-0.7 \leq x \leq -0.5$.

Here is a plot of the function



A code to seek a root could be:

```
  import numpy as np
  from scipy.optimize import brentq

  x = brentq ( lambda x: 0.5 + np.cos(3/x), -0.7, -0.5 )

  print ( '  Root found at ', x )
  print ( '  f(x) = ', 0.5 + np.cos(3/x) )
  return
```

In this example, the code estimates the root at $x = -0.59$ with $f(x) = -3.30e - 15$.

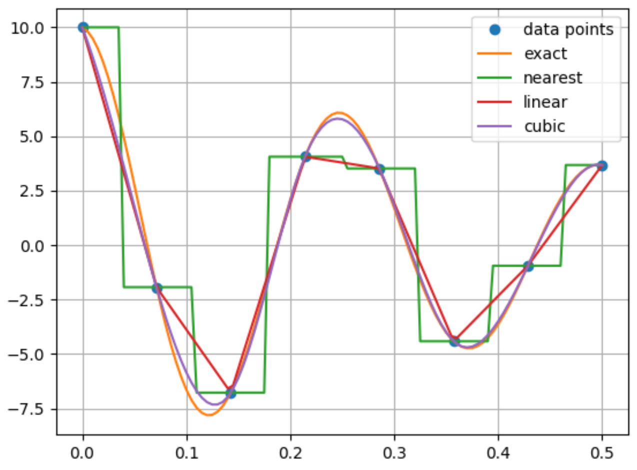
7 scipy.interpolate

The `scipy.interpolate` library is given a list of pairs of data values $\{x_i, y_i\}$, and is asked to construct an interpolating function $f(x)$ such that $f(x_i) = y_i$. The interpolating function may be a spline, a piecewise function whose parts are of degree 0, 1, 2, or 3, or a polynomial. Interpolation can also be done in higher dimensions, where, for example, a function $f(x, y) = z$ is desired which matches data triples $\{x_i, y_i, z_i\}$.

The function `interp1d()` will be used to illustrate a few of the options for spline interpolation in the 1D case. Here, we have a hidden function which we sample to get 8 data values. We then request interpolants of type 'nearest', 'linear', and 'cubic', and plot them as well as our known function.

```
def interp1d_example ( ):

  from scipy.interpolate import interp1d
  import matplotlib.pyplot as plt
  import numpy as np

  x = np.linspace ( 0.0, 0.5, 8 )
  y = f ( x )

  f_nearest = interp1d ( x, y, kind = 'nearest' )
  f_linear = interp1d ( x, y, kind = 'linear' )
  f_cubic = interp1d ( x, y, kind = 'cubic' )

  x2 = np.linspace ( 0.0, 0.5, 101 )

  plt.plot ( x, y, 'o', label = 'data points' )
  plt.plot ( x2, f(x2), label = 'exact' )
  plt.plot ( x2, f_nearest(x2), label = 'nearest' )
  plt.plot ( x2, f_linear(x2), label = 'linear' )
  plt.plot ( x2, f_cubic(x2), label = 'cubic' )
  plt.grid ( True )
  plt.legend ( )
  plt.show ( )
  return

def f ( x ):
  import numpy as np
  value = 10 * np.exp ( - 2.0 * x ) * np.cos ( 2.0 * np.pi * 4.0 * x )
  return value
```

Here is a plot showing the data points as dots, the three interpolants, and the original function which generated the data.



In this example, the piecewise linear function doesn't do badly, while the cubic spline does a very good job of producing a function similar to the original.

8 scipy.linalg

The `scipy.linalg` library includes many utilities for work in linear algebra, involving norms, multiplication of matrices and vectors, matrix factorizations, solving linear systems whether full, under-determined or over-determined, and the determination of eigenvalues.

We will look at a common problem in which we are given a matrix A, a right hand side b, and want to compute a vector x such that $A * x = b$.

The code is as follows:

```
def linalg_example ( ):
  import numpy as np

  n = 5
  A = np.random.randint ( low = 1, high = 9, size = [ n, n ] )
  A = A.reshape ( n, n )

  x = np.arange ( 1, n + 1 )

  b = np.dot ( A, x )

  x2 = np.linalg.solve ( A, b )

  print ( '  Computed solution x:' )
  print ( x2 )
  e = np.linalg.norm ( np.dot ( A, x2 ) - b )
  print ( '  ||error|| =', e )

  return
```