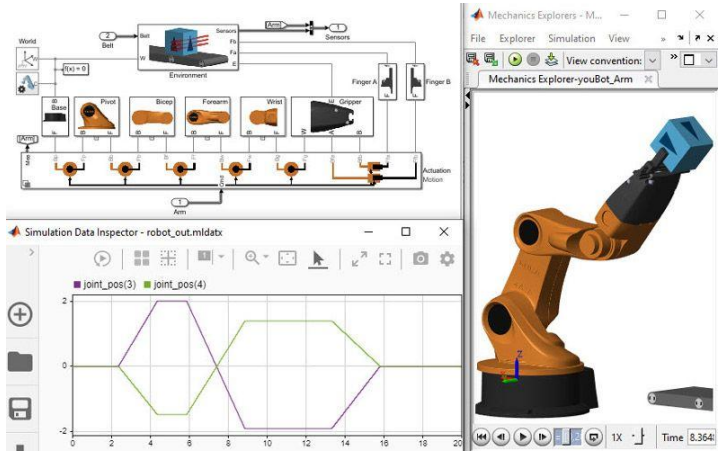
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

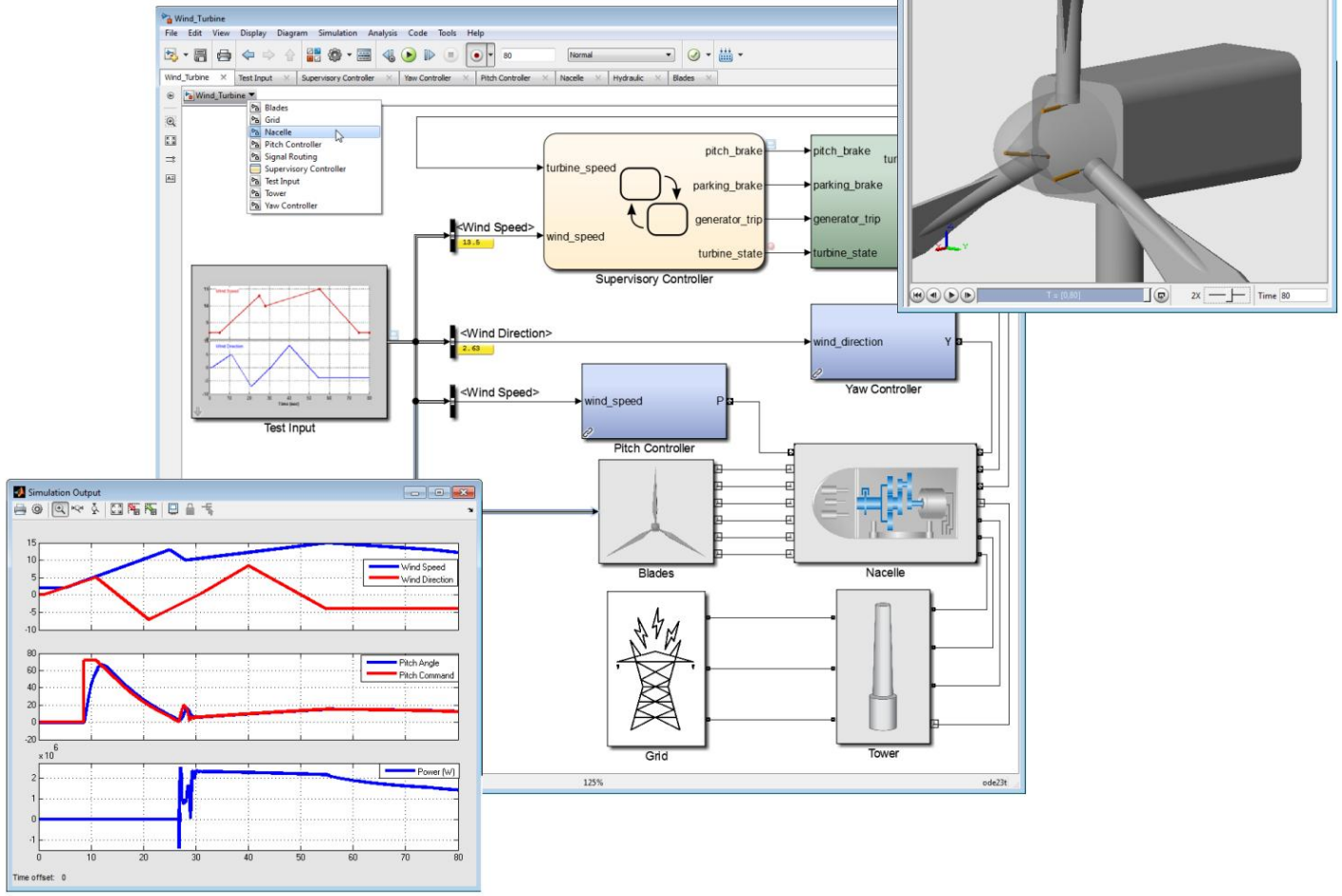
Introduction to Computer Engineering: Programming and Applications

Lecture 10
Programming Applications for Science and Engineering

Prof. Junhua Zhao
School of Science and Engineering

Programming plays an essential role in sciences and engineering

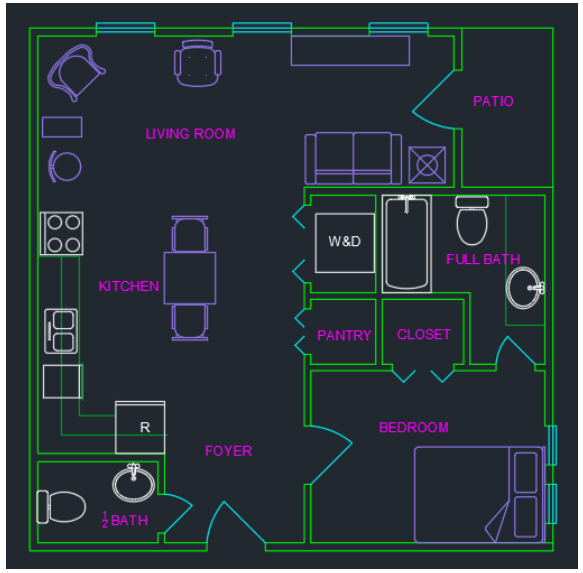


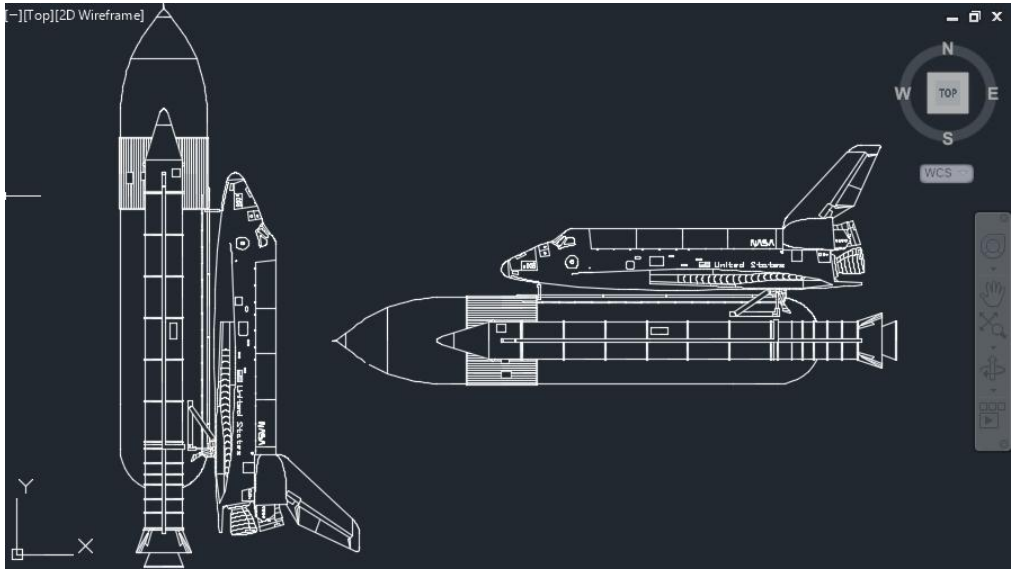


MATLAB® & SIMULINK®

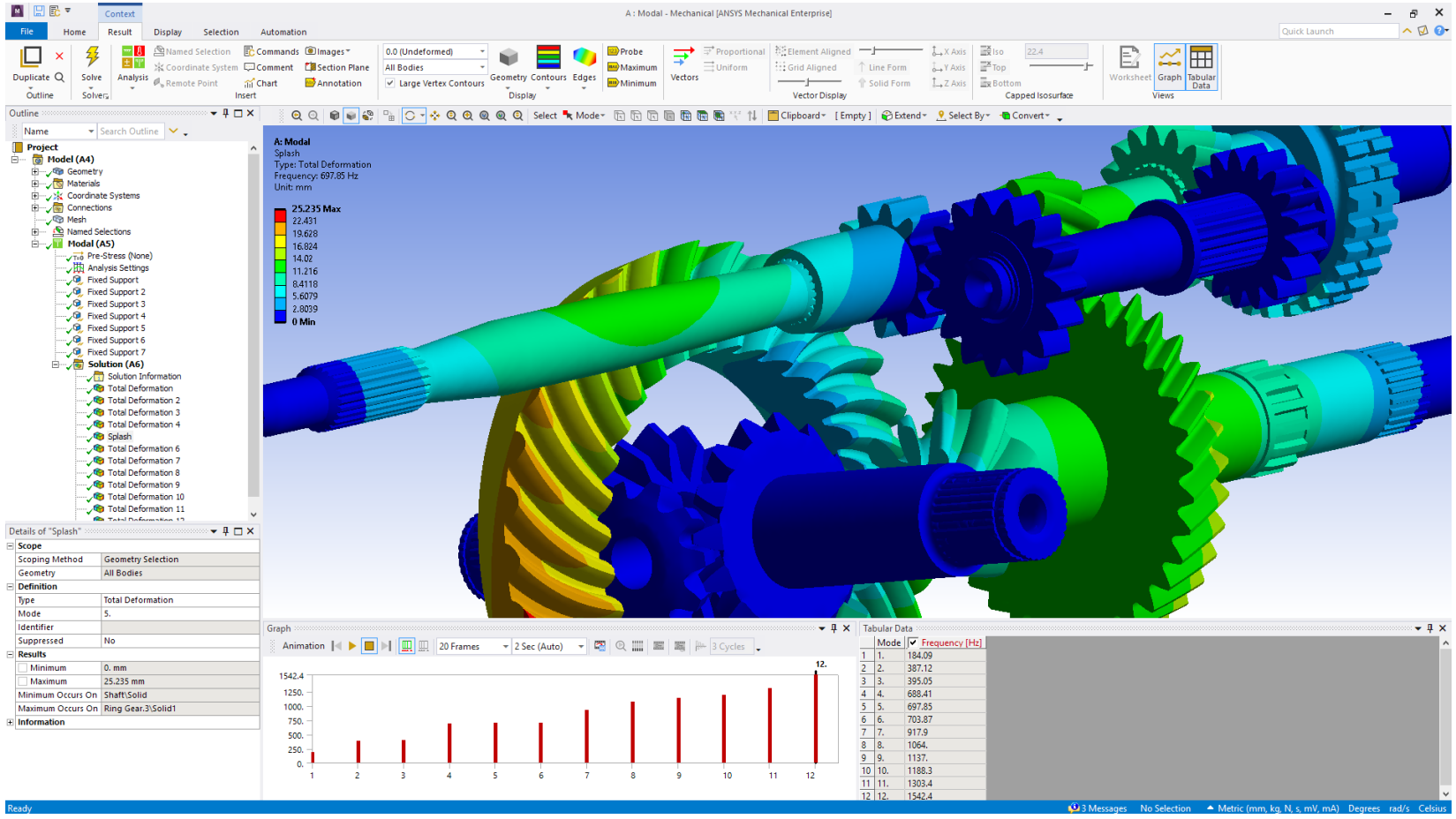
AutoCAD is a 2D and 3D computer-aided design (CAD) software application developed by Autodesk.

AUTODESK®
AUTOCAD®



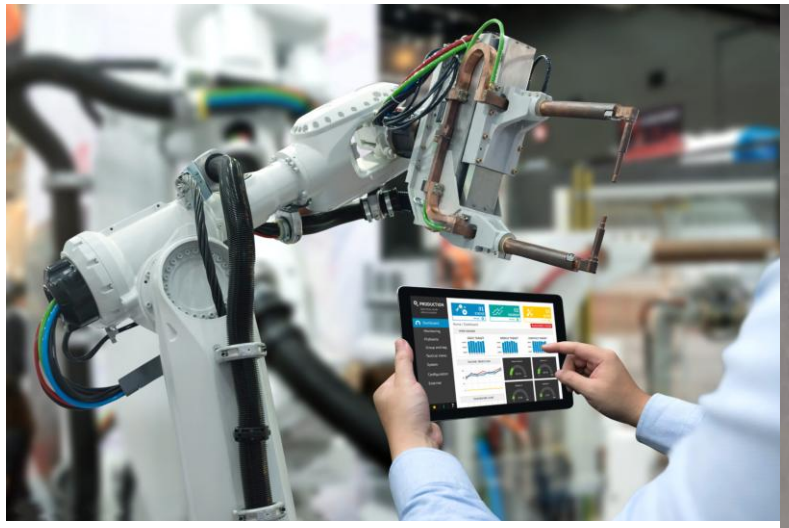


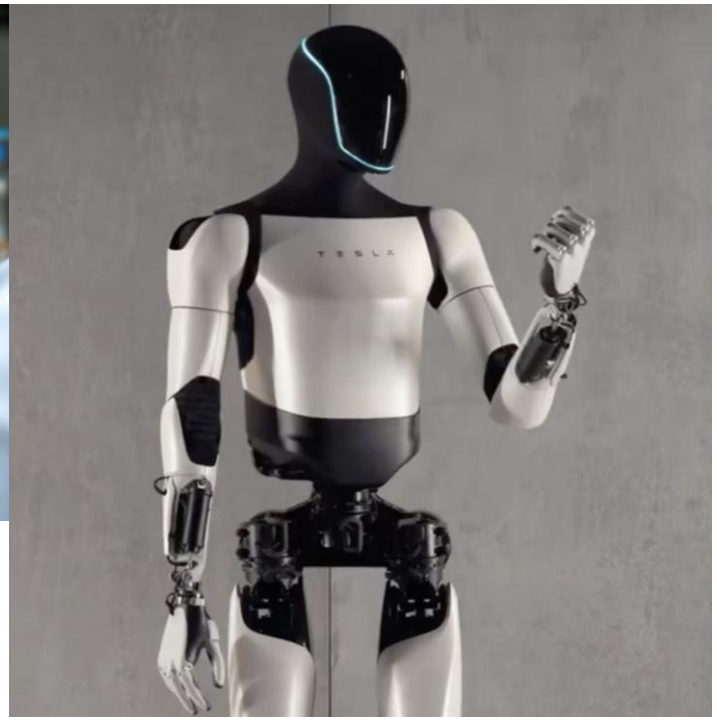
The Ansys Workbench platform integrates data across engineering simulations to create more accurate models more efficiently.



Embodied Intelligence

Embodied intelligence involves writing computer programs that enable a robot to perceive its environment, make plans and decisions, and execute tasks.





Autonomous driving

Autonomous driving, refers to vehicles equipped with systems that enable them to navigate and operate without human intervention.

Autonomous driving

Artificial Intelligence (AI) plays an important role in autonomous driving.

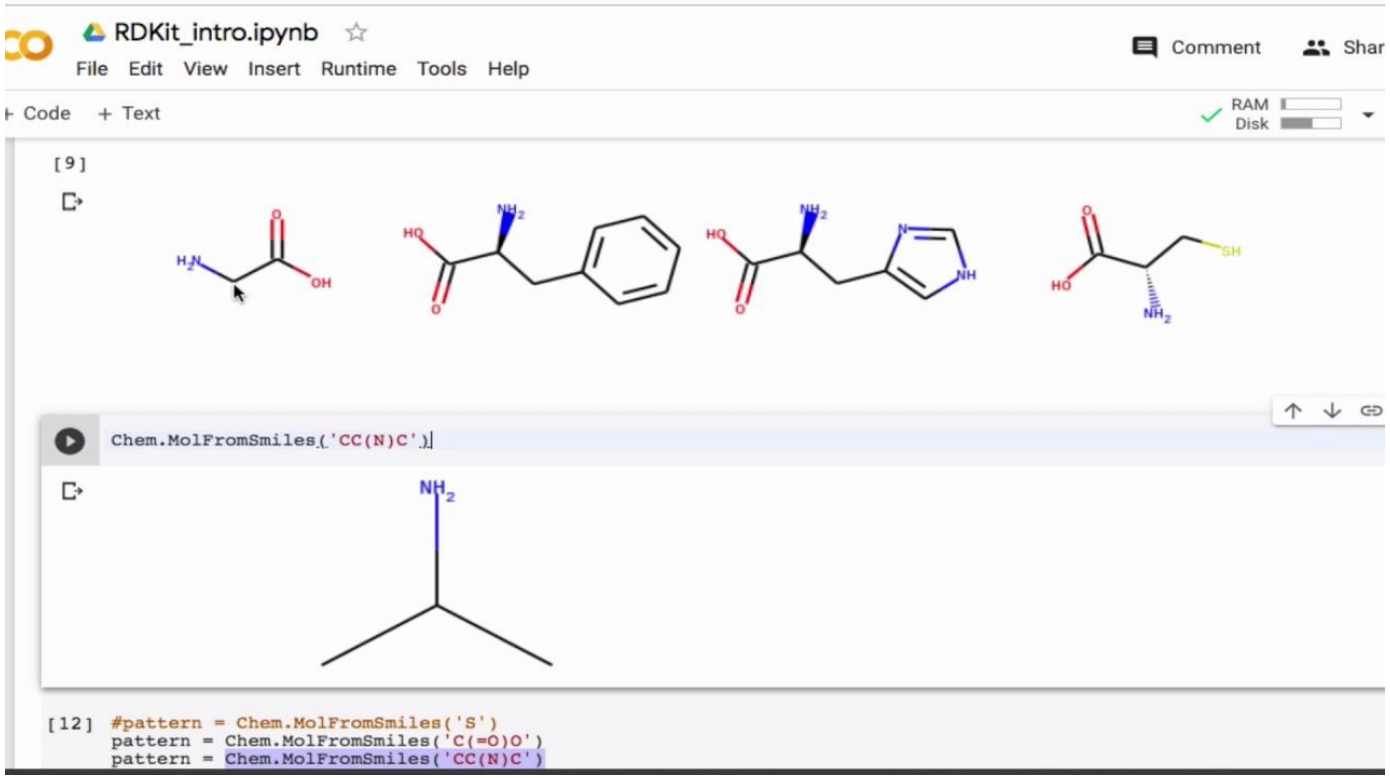
Sequence Alignments

Sequence alignment is the process of arranging two or more sequences (of DNA, RNA or protein sequences) in a specific order to identify the region of similarity between them.

Identifying the similar region enables us to infer a lot of information like what traits are conserved between species. Biopython provides extensive support for sequence alignment.

Chemical Molecular Structures Visualization

The RDKit is a collection of cheminformatics software written in C++ and Python.

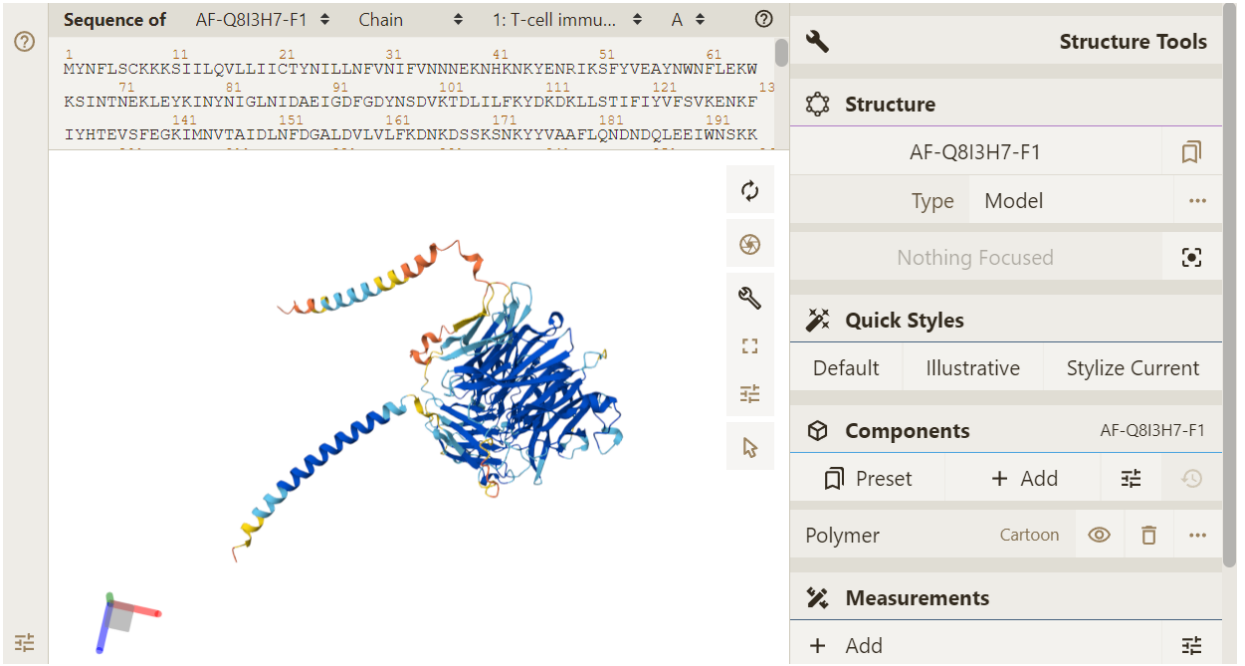


AlphaGO

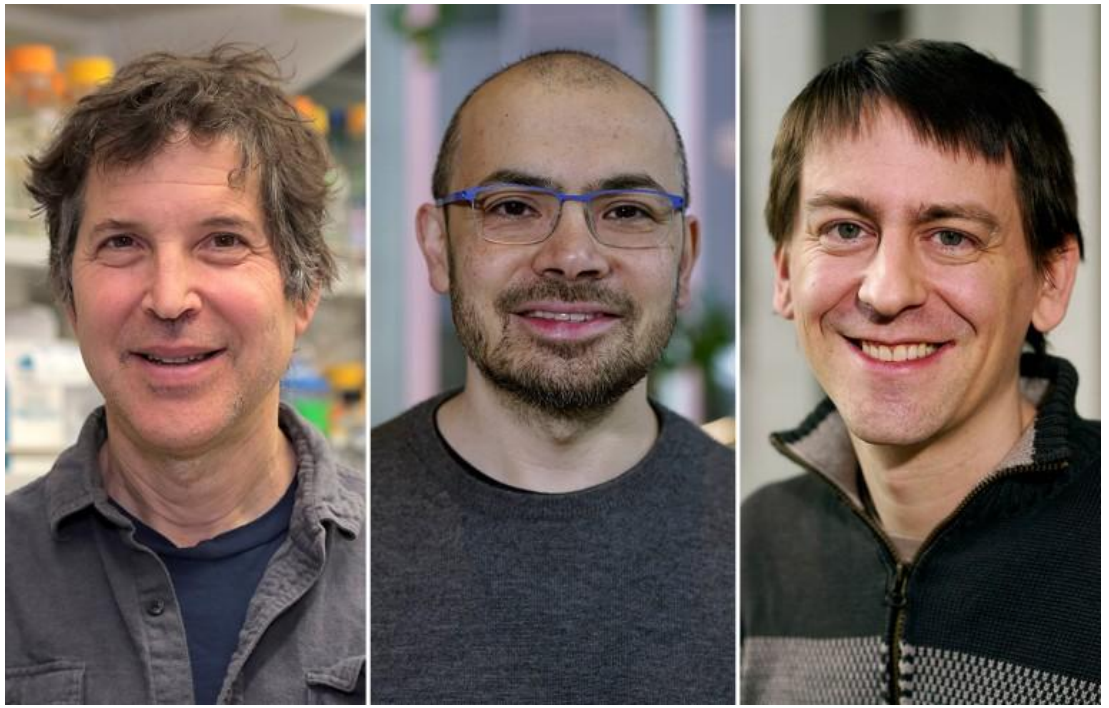
AlphaGo is a computer program that plays the board game Go. It was developed by the London-based DeepMind Technologies, an acquired subsidiary of Google.

AlphaFold

AlphaFold is an AI system developed by Google DeepMind that predicts a protein's 3D structure from its amino acid sequence. It regularly achieves accuracy competitive with experiment.



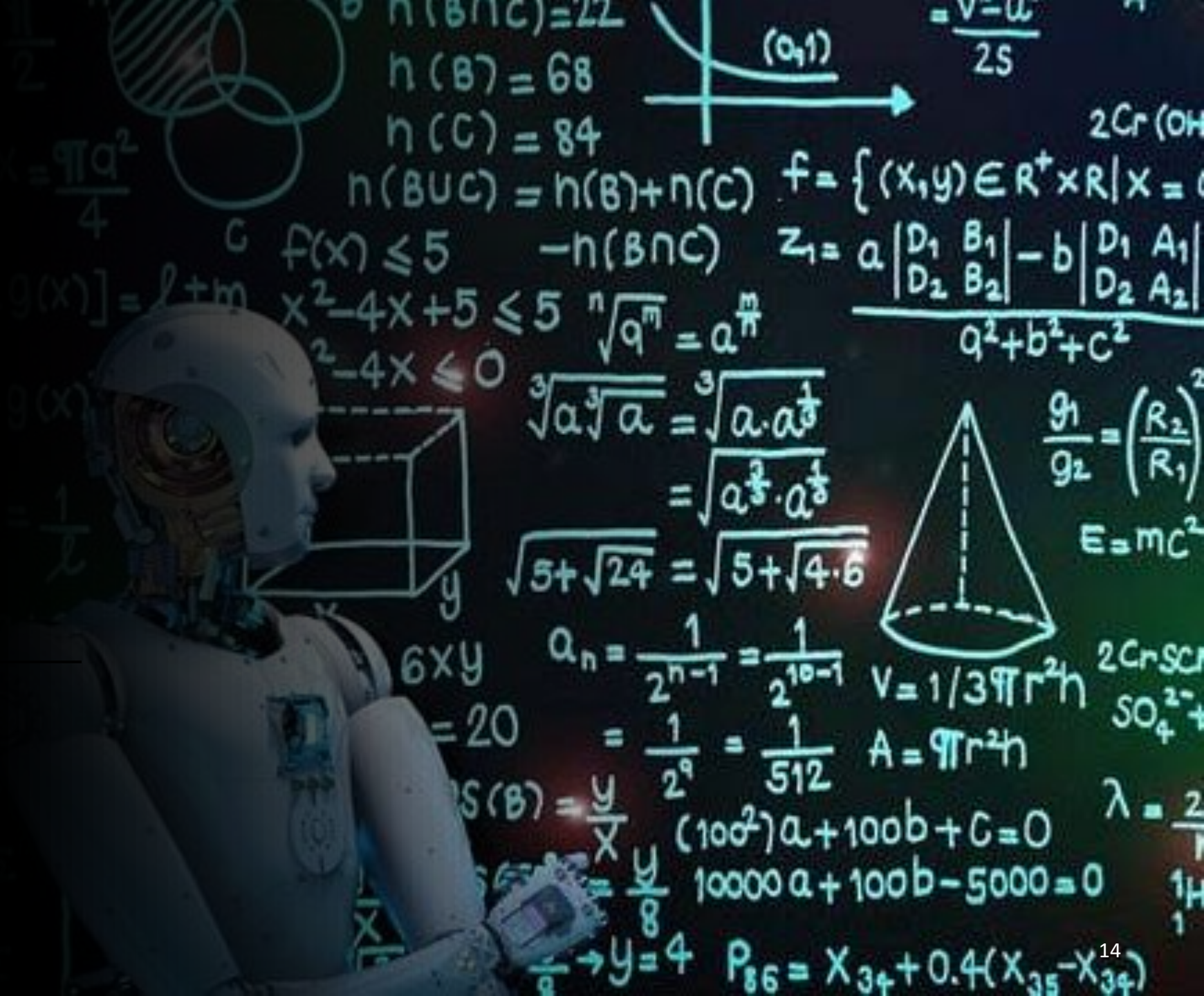
Chemistry Nobel goes to developers of AlphaFold AI that predicts protein structures.



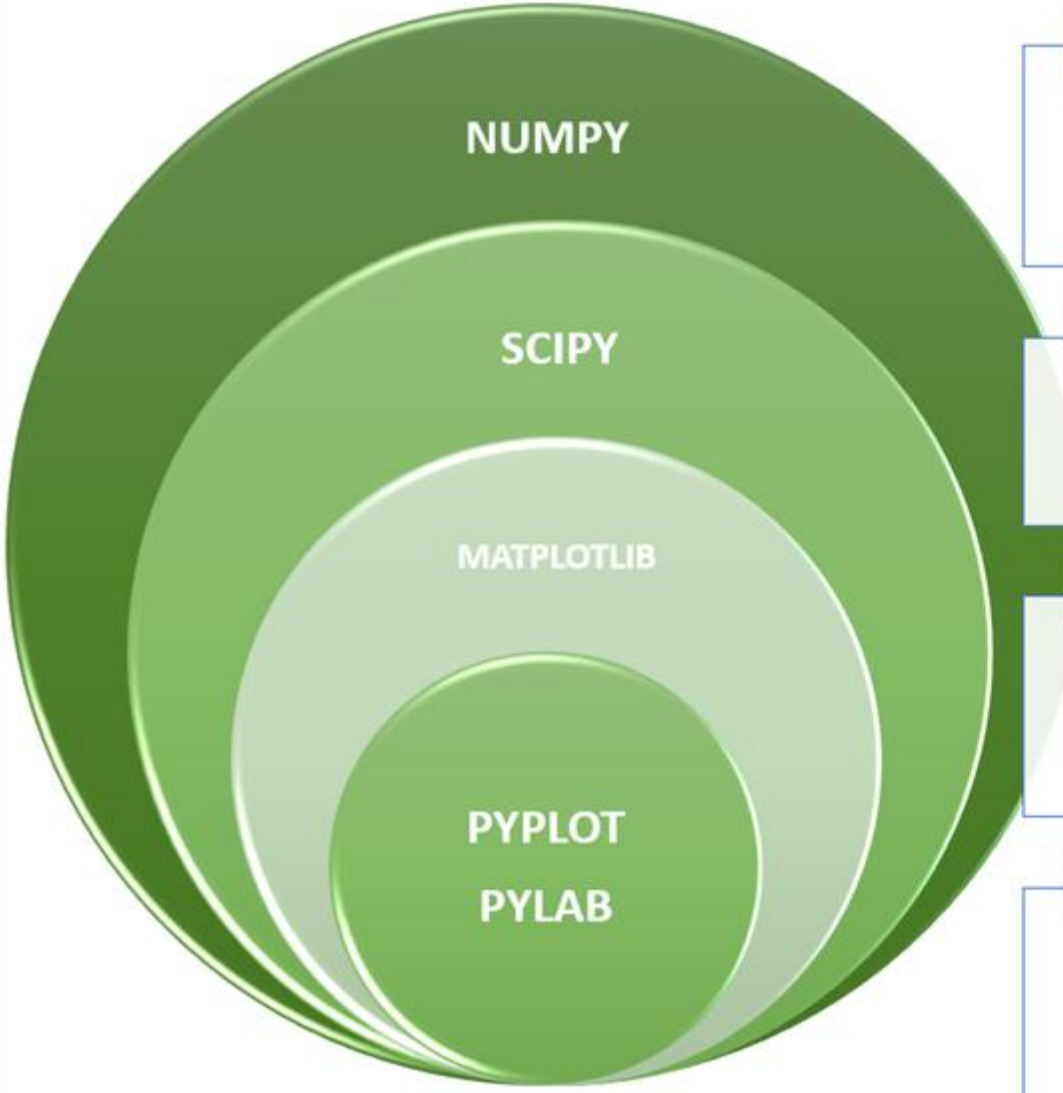
David Baker, Demis Hassabis and John Jumper (left to right) won the chemistry Nobel for developing computational tools that can predict and design protein structures. Credit: BBVA Foundation

The Nobel Prize in Physics has been awarded to two scientists, Geoffrey Hinton and John Hopfield, for their work on machine learning.

Basic Math Applications



Powerful Libraries



NUMPY

- **NumPy** is a **Python** package which is the core **library** for scientific computing, that contains a powerful N-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc.

SCIPY

- **SciPy** builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries.

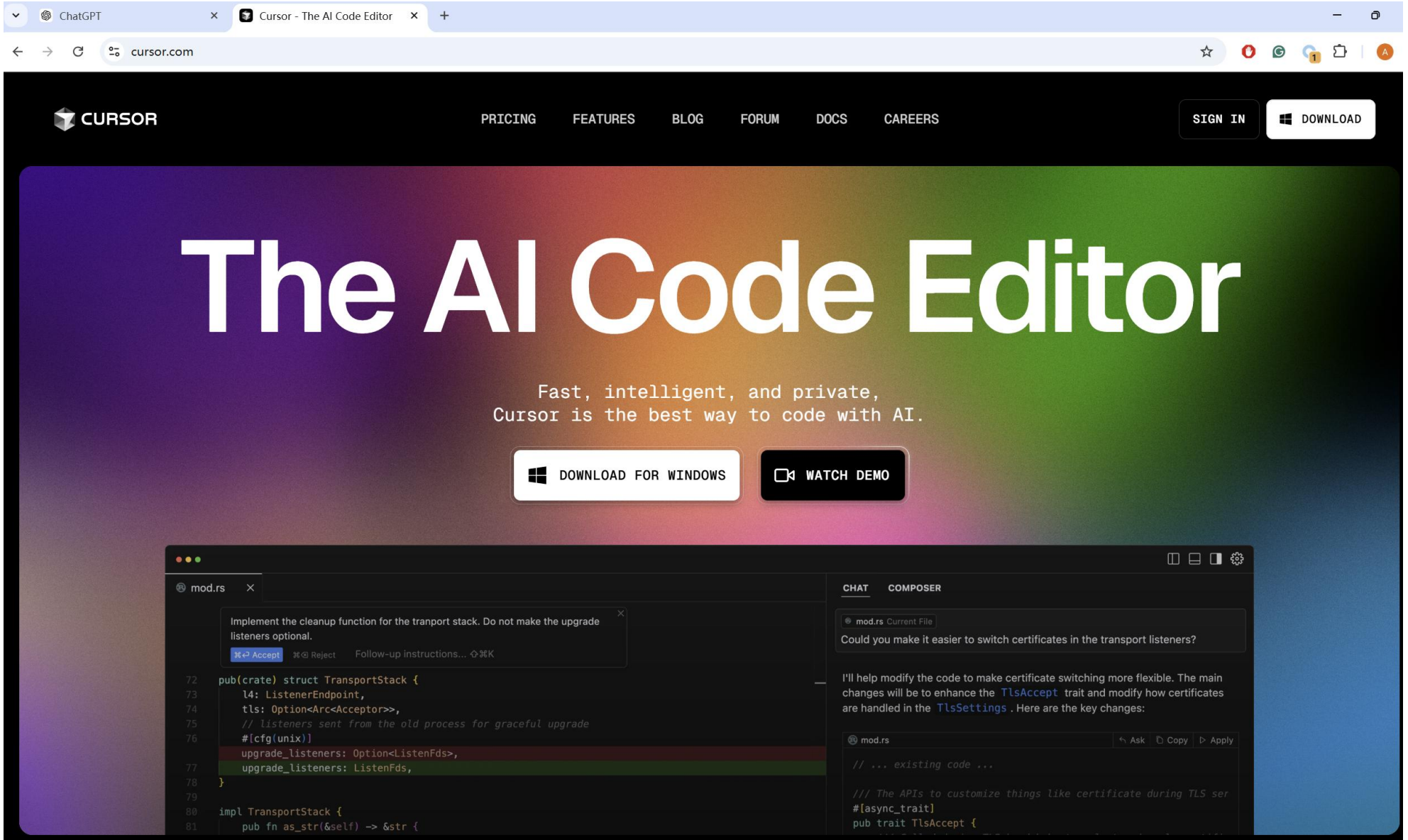
MATPLOTLIB

- **Matplotlib** is a plotting library for **Python** and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt. **SciPy** makes use of **Matplotlib**.

PYPLOT/PYLAB

- **PyLab** is a procedural interface to the Matplotlib object-oriented plotting library. **PyLab** is a convenience module that bulk imports matplotlib. **Pyplot** and NumPy is a single name space. Although many examples use **PyLab**, it is no longer recommended.

Cursor – A Powerful AI Assisted Code Editor



Powerful Libraries - Numpy

- Numerical Python
- Efficient and handy for the operations on the homogenous data, particularly stored in arrays.
- Simply, we can say Numpy is used very much for the manipulation of the **numerical data**!

```
01. import numpy as np
```

```
pip install numpy
```

Powerful Libraries - SciPy

- Scientific Python
- A collection of tools for Python, including optimization, interpolation, algebraic equations, statistics, integration, differentiation, …
- SciPy's high level syntax makes it accessible and productive for programmers from any background!

```
01. import scipy as sp
```

```
pip install scipy
```

Powerful Libraries - Matplotlib

- Data visualization
- Create static, animated, and interactive visualizations.
- It provides a number of customization options, including color, line styles, and markers, making it easy to create visualizations that meet specific needs.

```
01.  import matplotlib.pyplot as plt
```

```
pip install matplotlib
```

Optimization

Minimize $f(x) = (x-3)^2$

```
01. from scipy.optimize import minimize
02.
03. def f(x):
04.     return (x-3)**2
05.
06. res = minimize(f, x0=2)
07. print(res)
08. print(res.x)
```

```
      fun: 5.551437397369767e-17
 hess_inv: array([[0.5]])
      jac: array([-4.3254289e-13])
  message: 'Optimization terminated successfully.'
     nfev: 6
      nit: 2
     njev: 3
   status: 0
  success: True
        x: array([2.99999999])
```

```
[2.99999999]
```

Optimization

$$\text{Minimize } f(x,y) = (x-1)^2 + (y-2.5)^2$$

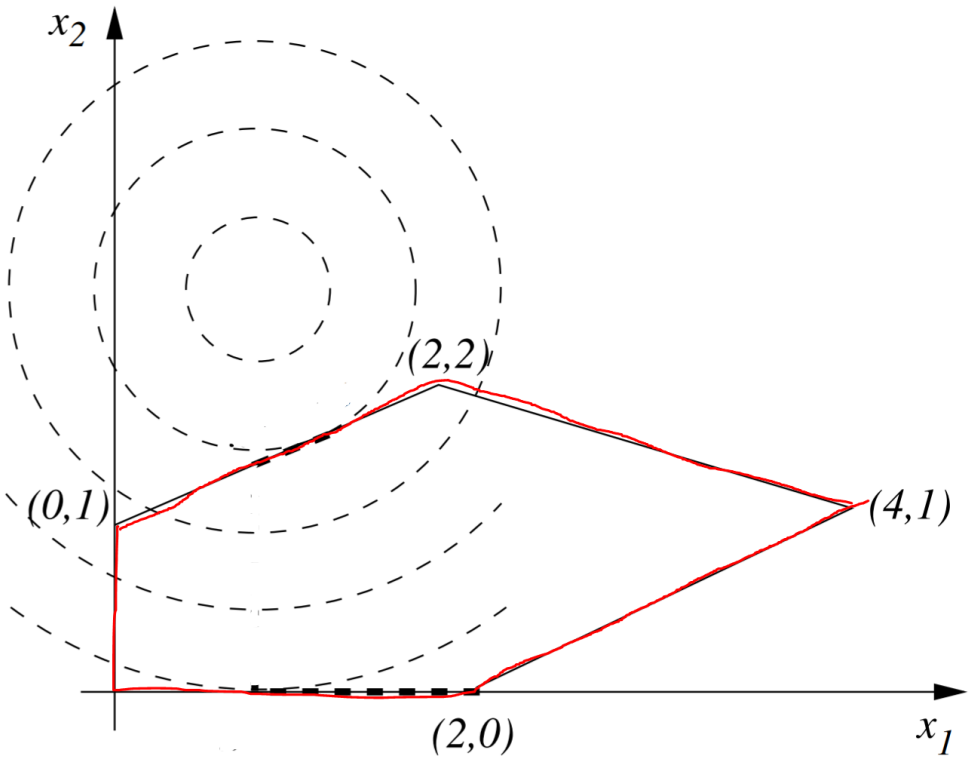
s.t.

$$x - 2y + 2 \geq 0$$
$$-x - 2y + 6 \geq 0$$
$$-x + 2y + 2 \geq 0$$
$$x \geq 0$$
$$y \geq 0$$

More Complex Problem!

- 2D function takes in vector x
- Constraints must be specified as $g_{i(x)} \geq 0$
- Bounds specified as rectangular

Optimization




```
01. f = lambda x: (x[0] - 1)**2 + (x[1] - 2.5)**2
02. cons = ({'type': 'ineq', 'fun': lambda x:  x[0] - 2 * x[1] + 2},
03.         {'type': 'ineq', 'fun': lambda x: -x[0] - 2 * x[1] + 6},
04.         {'type': 'ineq', 'fun': lambda x: -x[0] + 2 * x[1] + 2})
05. bnds = ((0, None), (0, None))
06. res = minimize(f, (2, 0), bounds=bnds, constraints=cons)
```

Optimization

```python
import numpy as np
import matplotlib.pyplot as plt

# Create a grid of points
x = np.linspace(-1, 7, 200)
y = np.linspace(-1, 7, 200)
X, Y = np.meshgrid(x, y)

# Plot constraints
plt.figure(figsize=(10, 8))

# Feasible region defined by constraints
c1 = X - 2*Y + 2 >= 0  # First constraint
c2 = -X - 2*Y + 6 >= 0  # Second constraint
c3 = -X + 2*Y + 2 >= 0  # Third constraint
c4 = X >= 0  # Bound constraint
c5 = Y >= 0  # Bound constraint

# Combined feasible region
feasible = c1 & c2 & c3 & c4 & c5

# Plot feasible region
plt.imshow(feasible, extent=[-1, 7, -1, 7], origin='lower', alpha=0.3)
```

```python
# Plot constraint lines
plt.plot(x, (x + 2)/2, 'r--', label='x - 2y + 2 = 0')
plt.plot(x, (-x + 6)/2, 'g--', label='-x - 2y + 6 = 0')
plt.plot(x, (x - 2)/2, 'b--', label='-x + 2y + 2 = 0')

# Plot optimal point
optimal_x = res.x
plt.plot(optimal_x[0], optimal_x[1], 'r*', markersize=15, label='Optimal Point')

plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Feasible Region and Optimal Solution')
plt.legend()
plt.show()
```

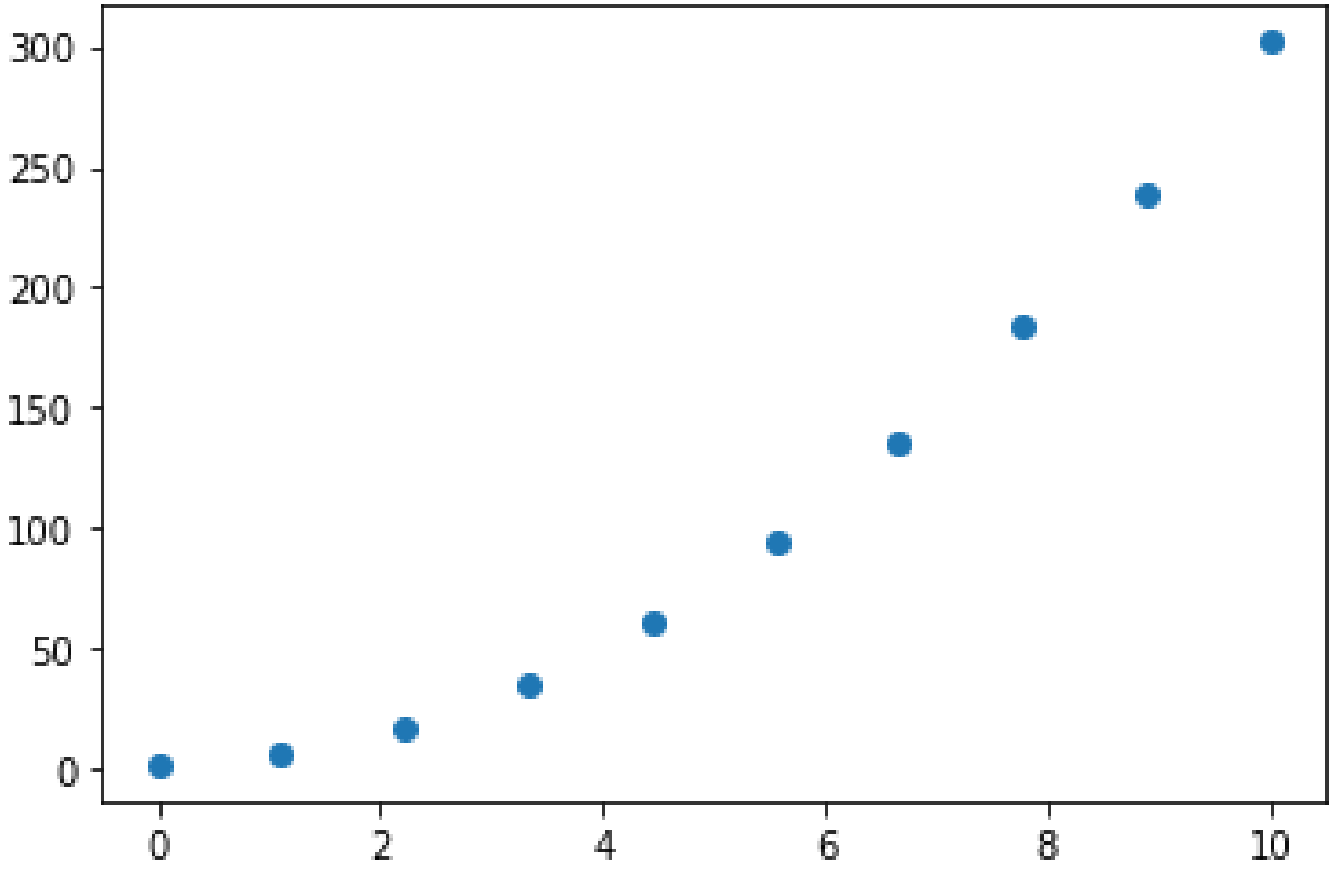
Optimization

Use different techniques for different problems.

```
Minimization of scalar function of one or more variables.

Parameters
----------
fun : callable
    The objective function to be minimized.

        ``fun(x, *args) -> float``

    where ``x`` is an 1-D array with shape (n,) and ``args``
    is a tuple of the fixed parameters needed to completely
    specify the function.
x0 : ndarray, shape (n,)
    Initial guess. Array of real elements of size (n,),
    where 'n' is the number of independent variables.
args : tuple, optional
    Extra arguments passed to the objective function and its
    derivatives (`fun`, `jac` and `hess` functions).
method : str or callable, optional
    Type of solver.  Should be one of

        - 'Nelder-Mead' :ref:`(see here) <optimize.minimize-neldermead>`
        - 'Powell'      :ref:`(see here) <optimize.minimize-powell>`
        - 'CG'          :ref:`(see here) <optimize.minimize-cg>`
        - 'BFGS'        :ref:`(see here) <optimize.minimize-bfgs>`
        - 'Newton-CG'   :ref:`(see here) <optimize.minimize-newtoncg>`
        - 'L-BFGS-B'    :ref:`(see here) <optimize.minimize-lbfgsb>`
        - 'TNC'         :ref:`(see here) <optimize.minimize-tnc>`
        - 'COBYLA'      :ref:`(see here) <optimize.minimize-cobyla>`
        - 'SLSQP'       :ref:`(see here) <optimize.minimize-slsqp>`
        - 'trust-constr':ref:`(see here) <optimize.minimize-trustconstr>`
        - 'dogleg'      :ref:`(see here) <optimize.minimize-dogleg>`
        - 'trust-ncg'   :ref:`(see here) <optimize.minimize-trustncg>`
        - 'trust-exact' :ref:`(see here) <optimize.minimize-trustexact>`
        - 'trust-krylov' :ref:`(see here) <optimize.minimize-trustkrylov>`
        - custom - a callable object (added in version 0.14.0),
          see below for description.

    If not given, chosen to be one of ``BFGS``, ``L-BFGS-B``, ``SLSQP``,
    depending if the problem has constraints or bounds.
jac : {callable,  '2-point', '3-point', 'cs', bool}, optional
    Method for computing the gradient vector. Only for CG, BFGS,
    Newton-CG, L-BFGS-B, TNC, SLSQP, dogleg, trust-ncg, trust-krylov,
    trust-exact and trust-constr.
    If it is a callable, it should be a function that returns the gradient
    vector:
```

Curve Fitting

Example 1: Want to fit the data to the curve $y = ax^2 + b$. The main goal here is determine the values of a and b.




```
01.  x_data = np.linspace(0, 10, 10)
02.  y_data = 3*x_data**2 + 2
```

Curve Fitting

Example 1: Want to fit the data to the curve $y = ax^2 + b$. The main goal here is determine the values of a and b.

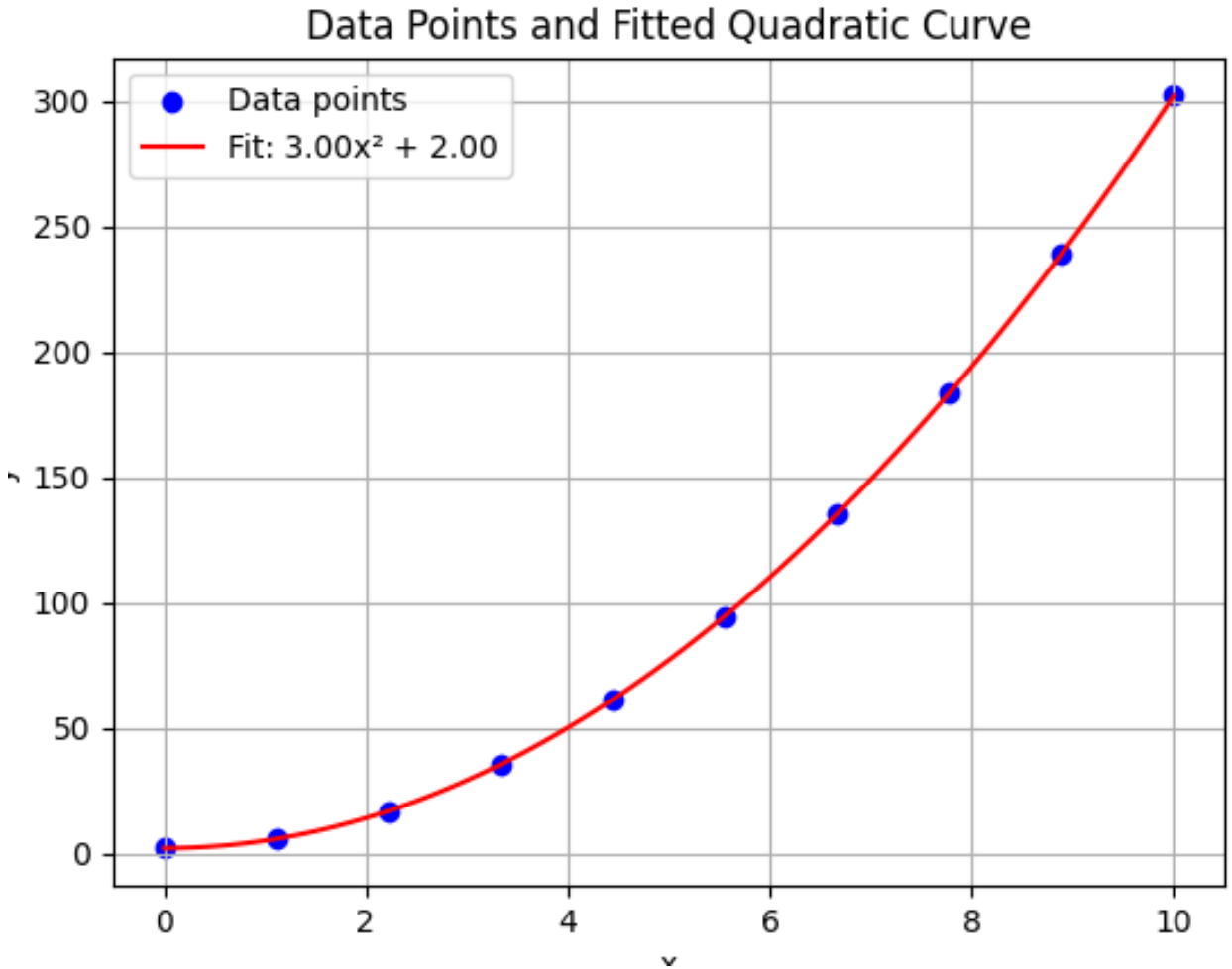
```
01. from scipy.optimize import curve_fit
02. def func(x, a, b):
03.     return a*x**2 + b
04. popt, pcov = curve_fit(func, x_data, y_data, p0=(1,1))
05. print(popt)
```

```
array([3., 2.])
```

Curve Fitting

```python
import matplotlib.pyplot as plt

# Plot the original data points
plt.scatter(x_data, y_data, color='blue', label='Data points')

# Generate smooth curve using the fitted parameters
x_fit = np.linspace(0, 10, 100)
y_fit = func(x_fit, *popt)
plt.plot(x_fit, y_fit, 'r-', label=f'Fit: {popt[0]:.2f}x² + {popt[1]:.2f}')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Data Points and Fitted Quadratic Curve')
plt.legend()
plt.grid(True)
plt.show()
```

Curve Fitting



Curve Fitting

Example 2: The equation for **spring motion** is $y(t) = Acos(\omega t + \phi)$. Want to find the natural frequency of oscillation ω for the spring.

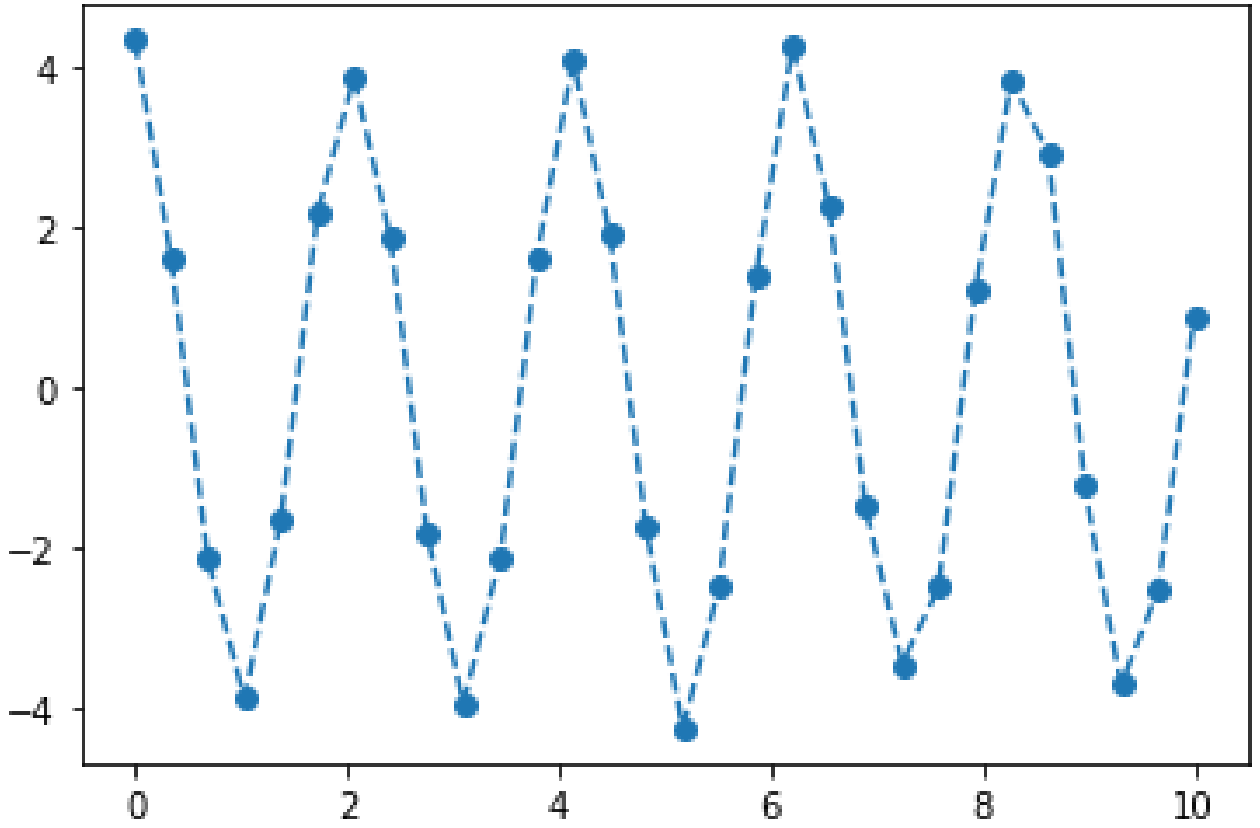
Curve Fitting

You collect the data:

```
01. t_data = np.array([ 0.   ,  0.34482759,  0.68965517,  1.03448276,  1.37931034,
02.         1.72413793,  2.06896552,  2.4137931 ,  2.75862069,  3.10344828,
03.         3.44827586,  3.79310345,  4.13793103,  4.48275862,  4.82758621,
04.         5.17241379,  5.51724138,  5.86206897,  6.20689655,  6.55172414,
05.         6.89655172,  7.24137931,  7.5862069 ,  7.93103448,  8.27586207,
06.         8.62068966,  8.96551724,  9.31034483,  9.65517241, 10.        ])
07. y_data = np.array([ 4.3303953 ,  1.61137995, -2.15418696, -3.90137249, -1.67259042,
08.         2.16884383,  3.86635998,  1.85194506, -1.8489224 , -3.96560495,
09.        -2.13385255,  1.59425817,  4.06145238,  1.89300594, -1.76870297,
10.        -4.26791226, -2.46874133,  1.37019912,  4.24945607,  2.27038039,
11.        -1.50299303, -3.46774049, -2.50845488,  1.20022052,  3.81633703,
12.         2.91511556, -1.24569189, -3.72716214, -2.54549857,  0.87262548])
```

Curve Fitting

```
01.  plt.plot(t_data,y_data,'o--')
02.  plt.show()
```

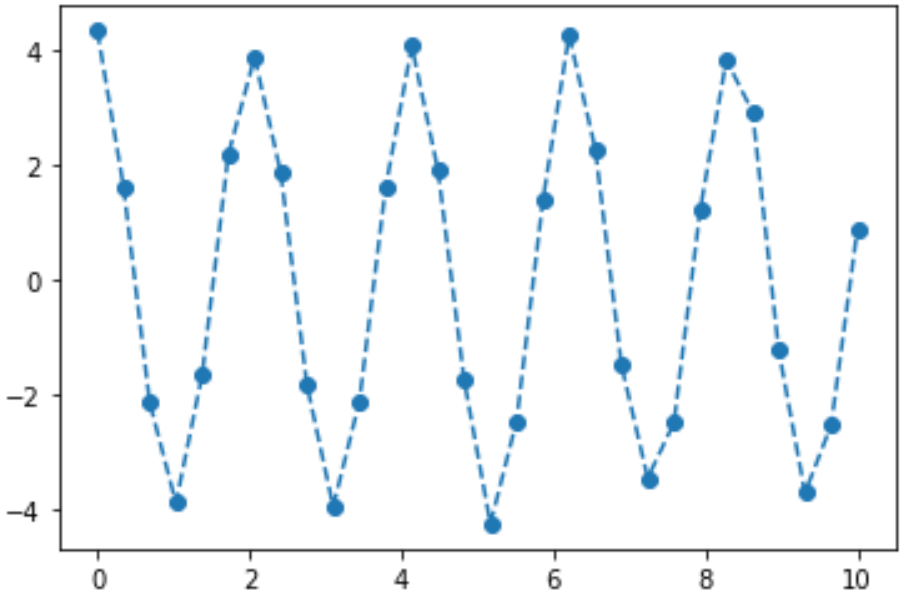


Curve Fitting

$\omega = 2\pi f \ , \ f = \frac{1}{T}, T \approx 2 \ seconds.$

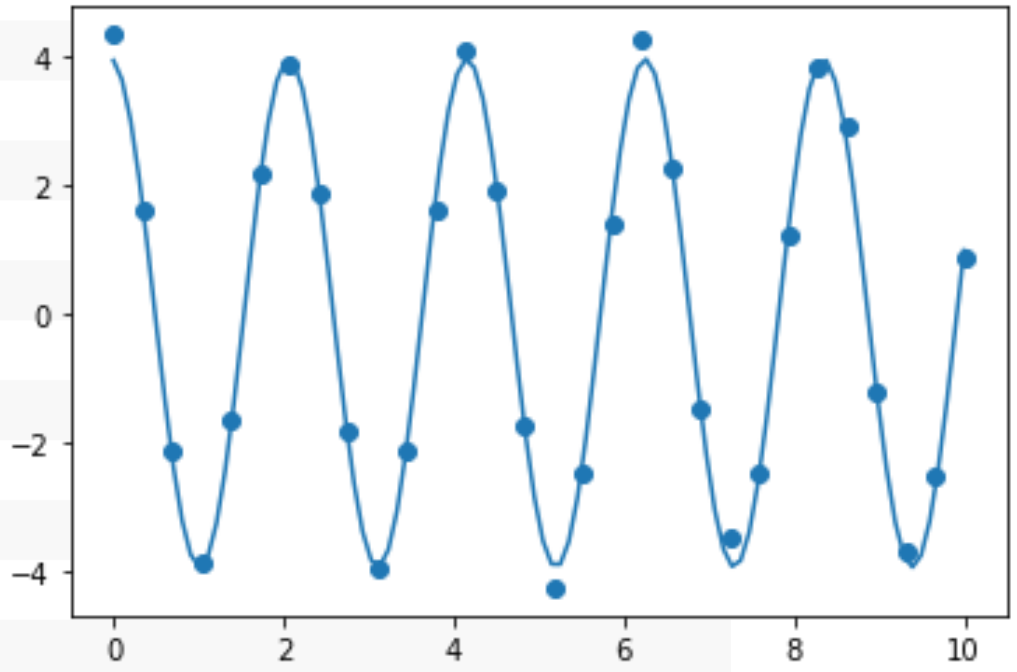
Thus, good initial guess is

- $\omega = 2\pi\left(\frac{1}{2}\right) = \pi$
- $A = 4$
- $\phi = 0$



Curve Fitting

```
01. from scipy.optimize import curve_fit
02.
03. def func(x, A, w, phi):
04.     return A*np.cos(w*x+phi)
05.
06. popt, pcov = curve_fit(func, t_data, y_data, p0=(4, np.pi, 0))
07.
08. A, w, phi = popt
09.
10. t = np.linspace(0, 10, 100)
11. y = func(t, A, w, phi)
12.
13. plt.scatter(t_data,y_data)
14. plt.plot(t,y)
15. plt.show()
16.
17. # The parameters
18. print(popt)
19. # The estimated error on the parameters
20. np.sqrt(np.diag(pcov))
```



Calculus - Differentiation

Given:

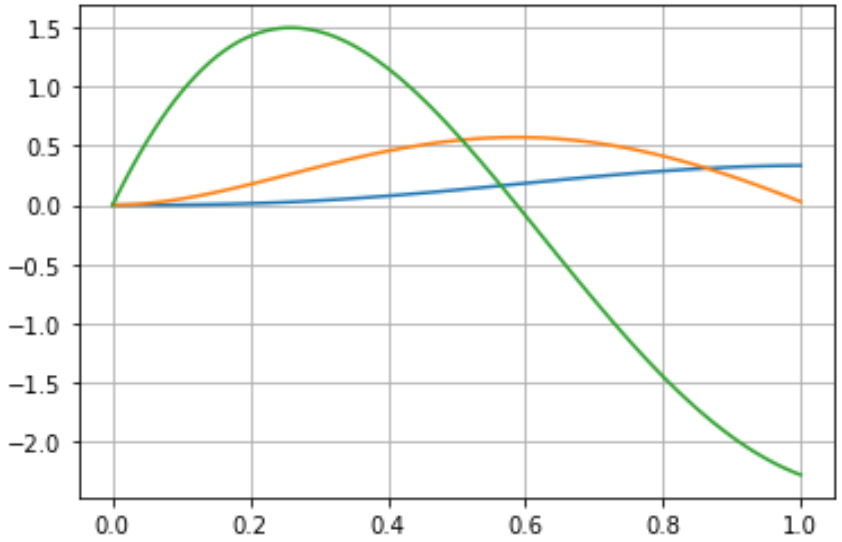
$$f(x) = x^2 \times \sin(2x) \times e^{-x}$$

How to Calculate:

$$f(x)'$$
$$f(x)''$$

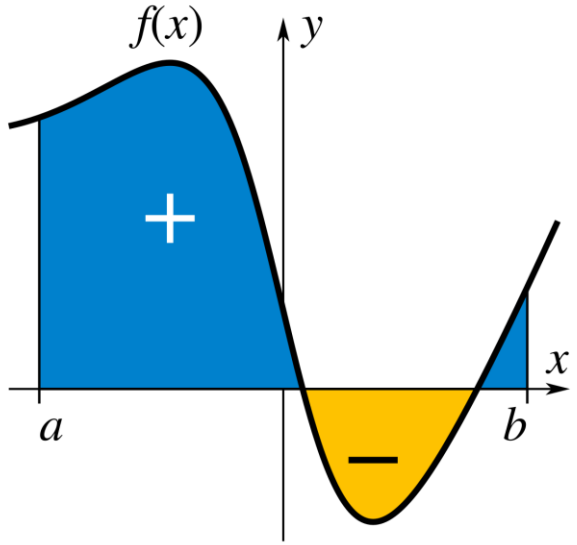
Calculus - Differentiation

```
01. import numpy as np
02. from scipy.misc import derivative
03.
04. def f(x):
05.     return x**2 * np.sin(2*x) *np.exp(-x)
06. x = np.linspace(0, 1, 100)
07.
08. plt.plot(x, f(x))
09. plt.plot(x, derivative(f, x, dx=1e-6))
10. plt.plot(x, derivative(f, x, dx=1e-6, n=2))
11. plt.grid()
12. plt.show()
```



Calculus - Integration

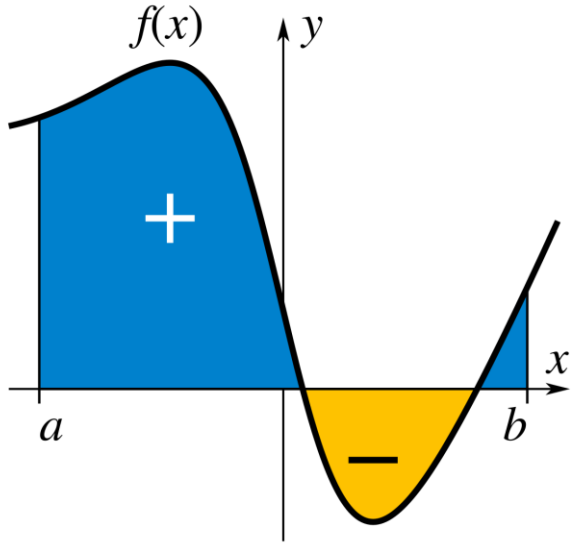
$$\int_0^1 x^2 \sin(2x)\, e^{-x}\, dx$$




```
01.  from scipy.integrate import quad
02.  integrand = lambda x: x**2 * np.sin(2*x) * np.exp(-x)
03.  integral, integral_error = quad(integrand, 0, 1)
04.  print(integral)
05.  # result: 0.14558175869954834
```

Calculus - Integration

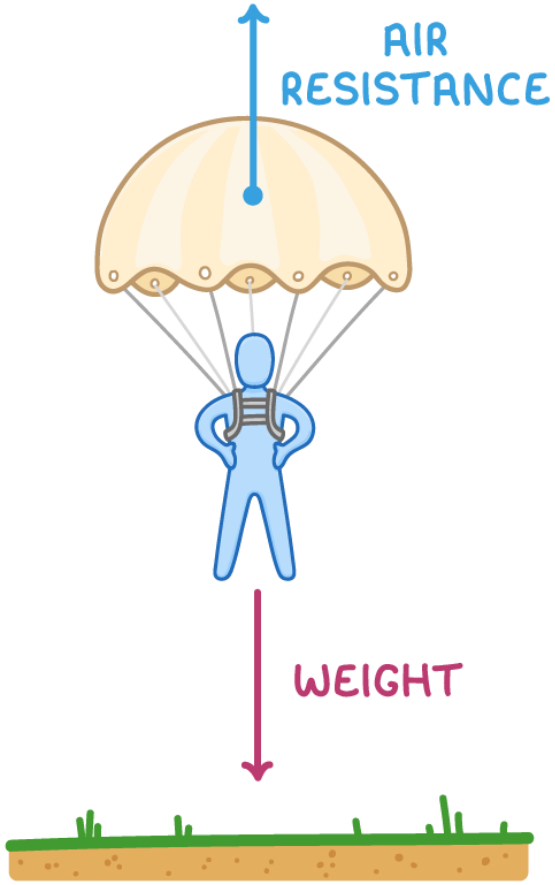
$$\int_0^1 \int_{-x}^{x^2} \sin(x + y^2)\, dydx$$




```
01.  from scipy.integrate import dblquad
02.  integrand = lambda y, x: np.sin(x+y**2)
03.  lwr_y = lambda x: -x
04.  upr_y = lambda x: x**2
05.  integral, integral_error = dblquad(integrand, 0, 1, lwr_y, upr_y)
06.  print(integral)
07.  #0.590090324408853
```

Differential Equations

Example: With ***air resistance the acceleration of a falling object*** is the acceleration of gravity minus the acceleration due to air resistance. For some objects the air resistance is proportional to the square of the velocity. For such an object we have the differential equation by Newton's Second Law.



Differential Equations

The differential equation by Newton's Second Law:

- *The rate of change of velocity is gravity minus something proportional to velocity squared:*

$$ma = mg - kv^2$$

$$\frac{dv}{dt} = g - kv^2$$

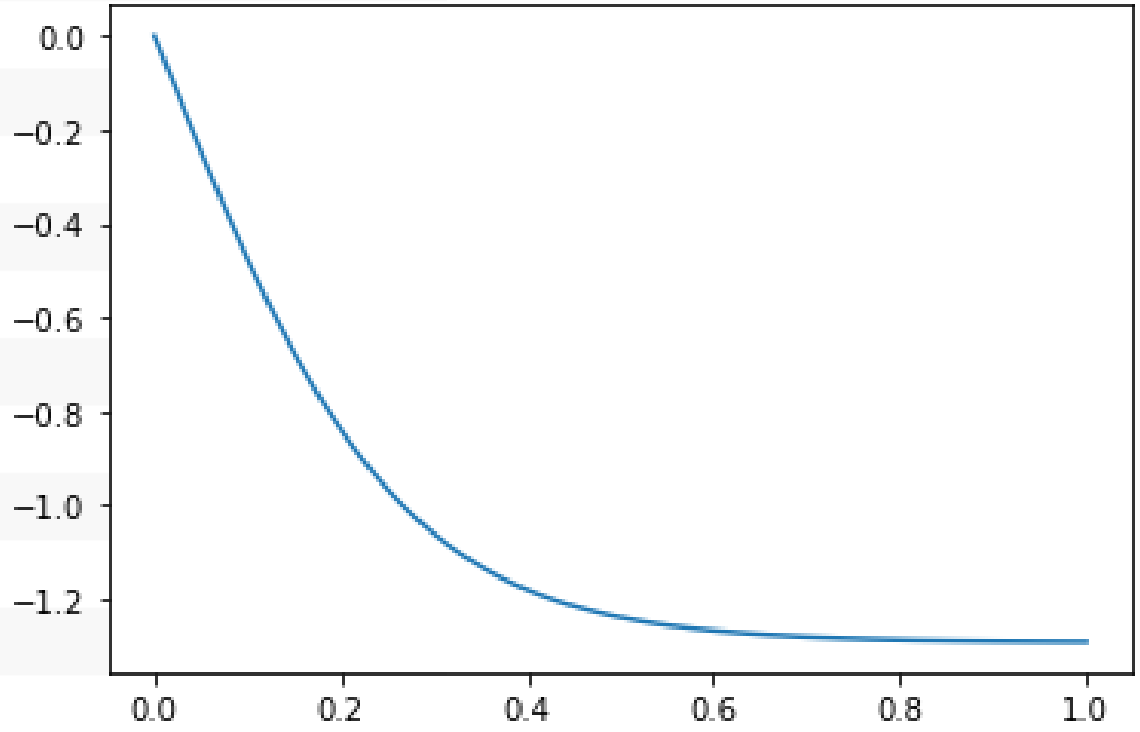
Differential Equations

Generally, the falling object with air resistance can be modeled as:

$$v' - \alpha v^2 + \beta = 0$$
$$v(0) = 0$$

Differential Equations

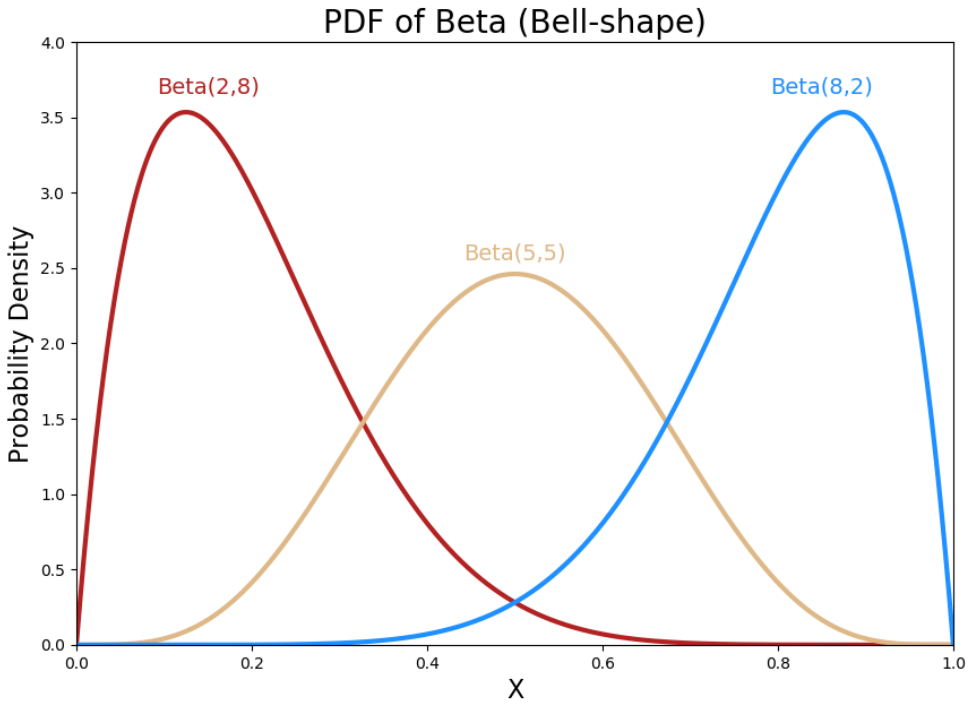
```
01. from scipy.integrate import odeint
02.
03. # All information about differential equation
04. def dvdt(v, t):
05.     return 3*v**2 - 5
06. v0 = 0
07.
08. # Solve differential equation
09. t = np.linspace(0, 1, 100)
10. sol = odeint(dvdt, v0, t)
11. v_sol = sol.T[0]
12.
13. #plot
14. plt.plot(t, v_sol)
15. plt.show()
```



Statistics

Example 1: β distribution

$$f(x;\ a, b) = \frac{\Gamma(a + b)x^{a-1}(1 - x)^{b-1}}{\Gamma(a)\Gamma(b)}, 0 \leq\ x \leq\ 1$$

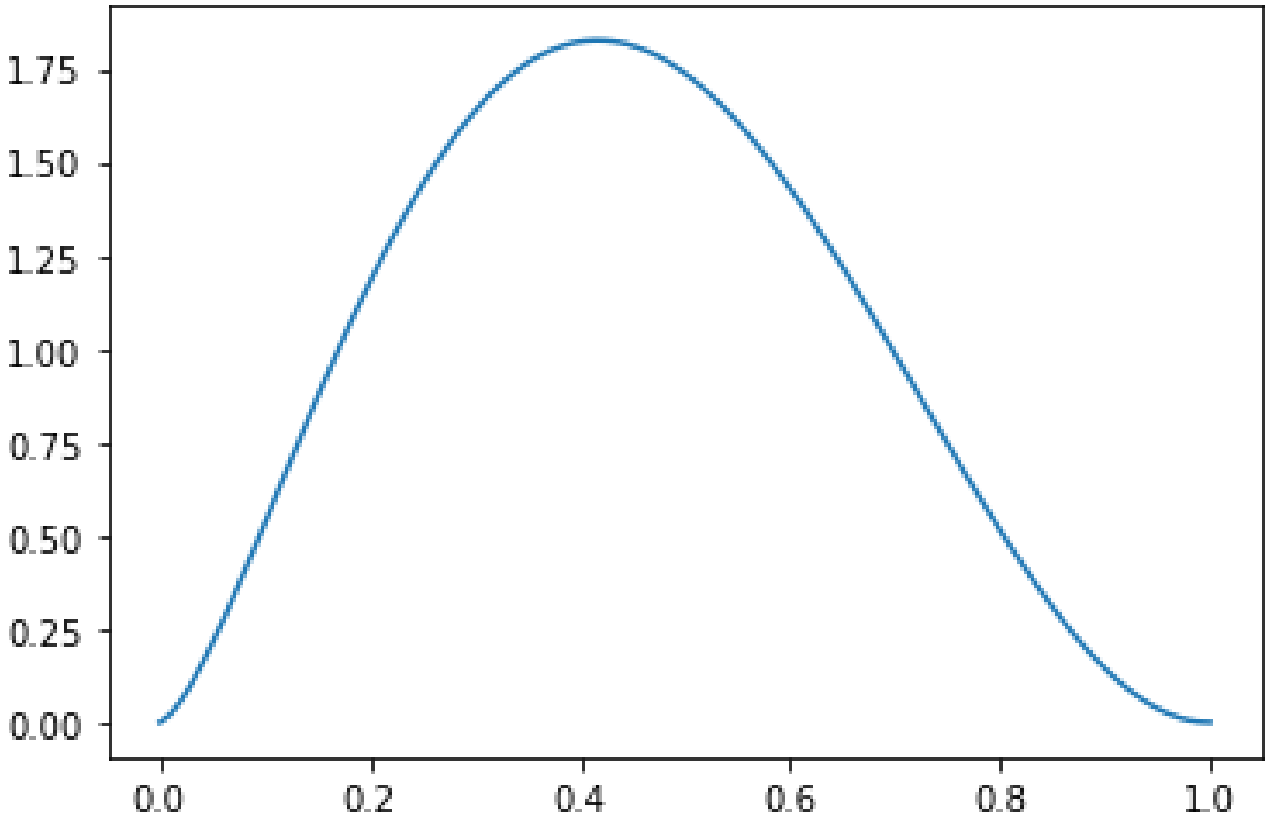


Statistics

```
01. from scipy.stats import beta
02. 
03. #Basic Statistics
04. a, b = 2.5, 3.1
05. mean, var, skew, kurt = beta.stats(a, b, moments='mvsk')
06. 
07. #Probability Distribution Plotting:
08. x = np.linspace(beta.ppf(0, a, b), beta.ppf(1, a, b), 100)
09. plt.plot(x, beta.pdf(x, a, b))
10. plt.show()
11. 
12. #Generating Random Variables:
13. r = beta.rvs(a, b, size=10)
14. print(r)
```

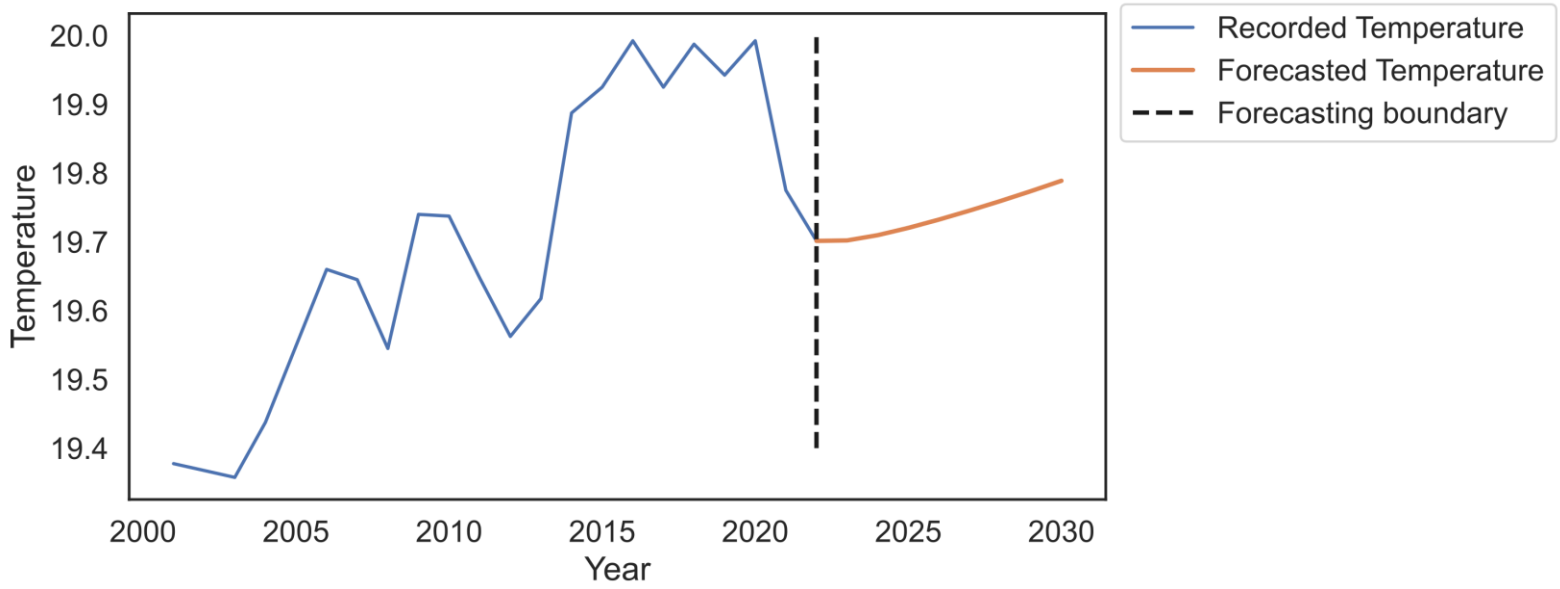
Statistics

Plotting result:



Time Series Forecasting

A time series is a sequence of observations over a certain period. The simplest example of a time series that all of us come across on a day to day basis is the change in temperature throughout the day or week or month or year.



Any data recorded with some fixed interval of time is called as time series data. This fixed interval can be hourly, daily, monthly or yearly. e.g. hourly temp reading, daily changing fuel prices, monthly electricity bill, annul company profit report etc. In time series data, time will always be independent variable and there can be one or many dependent variable.

Time series forecasting plays a pivotal role in science and engineering, as it allows professionals to predict future values based on historical data, driving better decision-making and optimization of resources.

For example: Power Systems and Energy Management, Climate Science and Environmental Engineering, …

Power Systems and Energy Management

- Load Forecasting: Utility companies rely on load forecasts to determine how much electricity needs to be generated and distributed. These forecasts help avoid blackouts or over-generation.

- Renewable Energy Forecasting: In wind or solar power systems, predicting energy output is essential for balancing supply and demand.

Developing these forecasts involves handling large datasets and implementing complex algorithms. Tools like Python, with libraries such as *pandas*, *scipy*, *scikit-learn*, and *pytorch*, are widely used to develop predictive models for energy forecasting. Programming allows for automation and testing of different models, fine-tuning forecasts to improve accuracy and reliability.

Here’s an example about doing load forecasting by Python.

Data:

date	load
2016/11/25 0:00	193.987
2016/11/25 1:00	187.12
2016/11/25 2:00	189.514
2016/11/25 3:00	190.69
2016/11/25 4:00	206.349
2016/11/25 5:00	201.094
2016/11/25 6:00	223.495
2016/11/25 7:00	230.597
2016/11/25 8:00	228.001
2016/11/25 9:00	237.563
2016/11/25 10:00	233.098
2016/11/25 11:00	248.331
2016/11/25 12:00	225.034
2016/11/25 13:00	221.596
2016/11/25 14:00	242.23
2016/11/25 15:00	235.041
2016/11/25 16:00	237.329
2016/11/25 17:00	254.505
2016/11/25 18:00	253.179
2016/11/25 19:00	236.71
2016/11/25 20:00	221.068
2016/11/25 21:00	204.747
2016/11/25 22:00	185.549
2016/11/25 23:00	172.146
2016/11/26 0:00	200.127
2016/11/26 1:00	186.238
2016/11/26 2:00	181.825
2016/11/26 3:00	182.282

Method: Autoregressive integrated moving average (ARIMA)

Reference: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

Goal: using the 80% data to train the model, and using the rest 20% to do the prediction test.

ARIMA Model

The general ARIMA model, combining autoregressive (AR), differencing (I), and moving average (MA) components, can be expressed explicitly as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q},$$

where:

- $\phi_1, \phi_2, \ldots, \phi_p$ are the autoregressive coefficients.
- $\theta_1, \theta_2, \ldots, \theta_q$ are the moving average coefficients.
- ϵ_t is the white noise (error term) at time t.

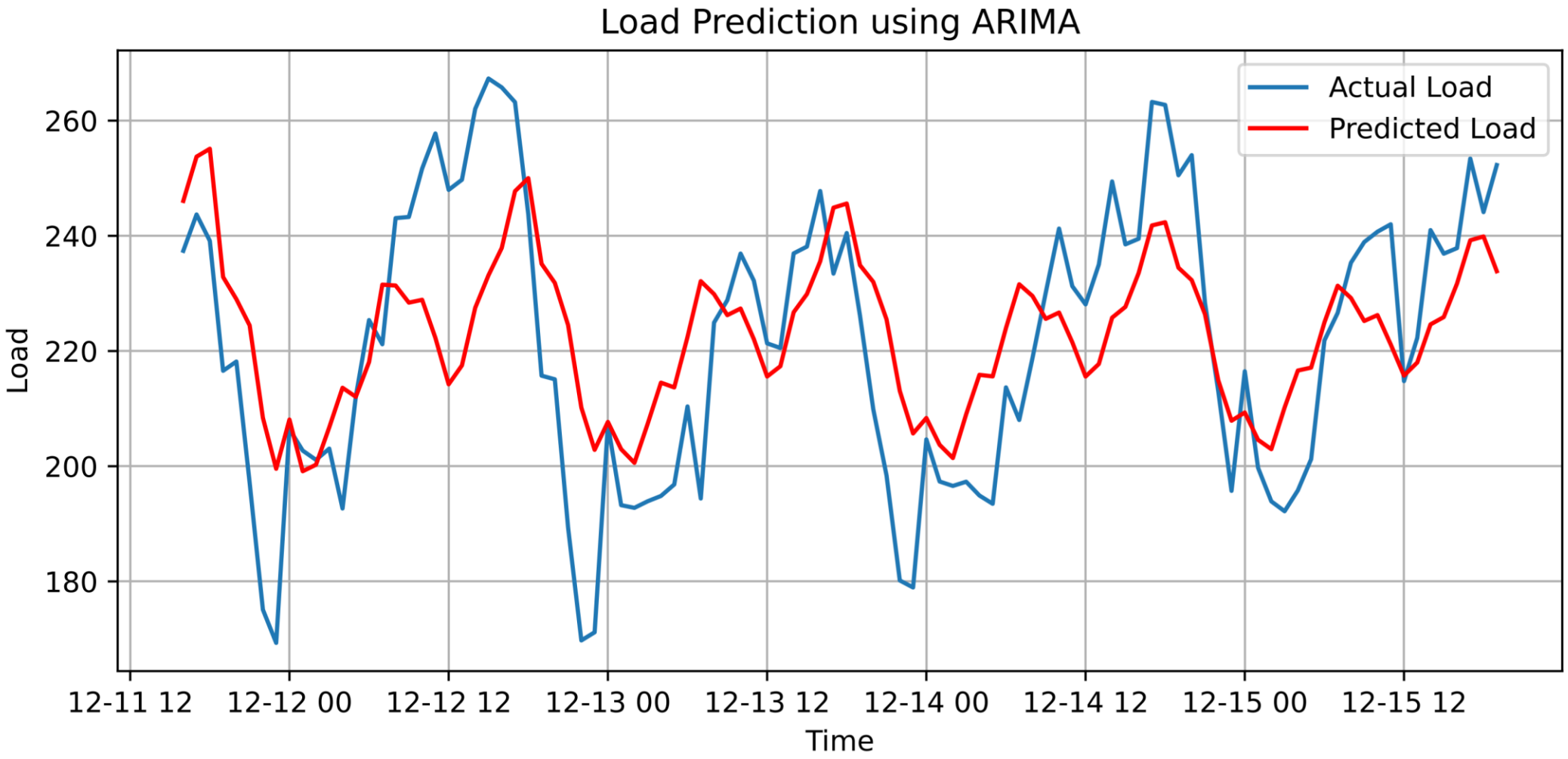
For example, if $d = 1$ (first-order differencing), the differenced value is:

$$y_t = \text{Original time series value at time } t - \text{value at } t - 1.$$

Code:

```python
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Read data
file_path = 'data.csv'
data = pd.read_csv(file_path)
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
load_data = data['load']  # get the load data

# separate train data and test data.
train_size = int(len(load_data) * 0.8)
train_data, test_data = load_data[:train_size], load_data[train_size:]

# build an ARIMA model.
model = ARIMA(train_data, order=(20, 1, 5))  # p, d, q are hyper-parameters of ARIMA.
arima_model = model.fit()

# do the prediction on the test data.
predictions = arima_model.forecast(steps=len(test_data))

# draw the prediction figure.
plt.figure(figsize=(10, 6))
plt.plot(test_data.index, test_data, label='Actual Load')
plt.plot(test_data.index, predictions, label='Predicted Load', color='red')
plt.legend(fontsize=15)
plt.xlabel('Date', fontsize=15)
plt.ylabel('Load', fontsize=15)
plt.title('Load Prediction using ARIMA', fontsize=15)
plt.grid(True)
plt.tight_layout()
plt.savefig('arima_forecast.pdf')
plt.show()
```

Results:


Load Prediction using ARIMA
Actual Load
Predicted Load
Load
260
240
220
200
180
12-11 12
12-12 00
12-12 12
12-13 00
12-13 12
12-14 00
12-14 12
12-15 00
12-15 12
Time

The graph shows that the predicted load closely aligns with the trend of the actual load. However, judging from the graph, there is still room for improvement in prediction accuracy. In fact, in the era of AI for science, more advanced AI models are often used for time series forecasting. For example, the LSTM neural network is a classic deep learning model that is widely used for sequence prediction tasks.