i have these functions that i wrote in python, now i want to fit these. I have arrays for age class marks, mean sbp and mean dbp. What library and function could be used
def polynomial_sbp_a(a,c1,c2,c3):
  function_sbp= c1*a**2 + c2*a + c3
  return function_sbp

def polynomial_dbp_a(a,d1,d2,d3):
  function_dbp= d1*a**2 + d2*a + d3
  return function_dbp

def sigmoid_sbp_a(a,Smax,k,a0):
  function_sbp_sigmoid= Smax/(1+np.exp(-k*(a-a0)))
  return function_sbp_sigmoid

def gaussian_dbp_a(a,Dmax,apeak,sigma):
  function_dbp_gaussian= Dmax*np.exp(-(a-apeak)**2/(2*sigma**2))

To fit those models to your **age class marks** (`a`) and **mean SBP/DBP** data arrays, you can use `scipy.optimize.curve_fit` — the most flexible and standard choice for nonlinear least-squares fitting.

Here's a complete example 👇

python

Copy code

```python
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Example functions
def polynomial_sbp_a(a, c1, c2, c3):
    return c1 * a**2 + c2 * a + c3

def polynomial_dbp_a(a, d1, d2, d3):
    return d1 * a**2 + d2 * a + d3
```

```python
def sigmoid_sbp_a(a, Smax, k, a0):
    return Smax / (1 + np.exp(-k * (a - a0)))

def gaussian_dbp_a(a, Dmax, apeak, sigma):
    return Dmax * np.exp(-((a - apeak)**2) / (2 * sigma**2))

# Example data (replace these with your arrays)
age = np.array([20, 30, 40, 50, 60, 70])
mean_sbp = np.array([115, 120, 125, 135, 140, 145])
mean_dbp = np.array([75, 78, 80, 82, 84, 85])

# Polynomial fit for SBP
popt_sbp, pcov_sbp = curve_fit(polynomial_sbp_a, age, mean_sbp)
print("Polynomial SBP coefficients:", popt_sbp)

# Polynomial fit for DBP
popt_dbp, pcov_dbp = curve_fit(polynomial_dbp_a, age, mean_dbp)
print("Polynomial DBP coefficients:", popt_dbp)

# Sigmoid fit for SBP (use reasonable initial guesses)
popt_sigmoid, _ = curve_fit(sigmoid_sbp_a, age, mean_sbp, p0=[150, 0.1, 40])
print("Sigmoid SBP parameters:", popt_sigmoid)

# Gaussian fit for DBP (with initial guesses)
popt_gaussian, _ = curve_fit(gaussian_dbp_a, age, mean_dbp, p0=[90, 45, 15])
print("Gaussian DBP parameters:", popt_gaussian)

# Plot fitted curves
age_fit = np.linspace(min(age), max(age), 200)
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.scatter(age, mean_sbp, label="SBP data")
plt.plot(age_fit, polynomial_sbp_a(age_fit, *popt_sbp), label="Poly fit")
plt.plot(age_fit, sigmoid_sbp_a(age_fit, *popt_sigmoid), label="Sigmoid fit")
plt.legend(); plt.xlabel("Age"); plt.ylabel("SBP")

plt.subplot(1,2,2)
plt.scatter(age, mean_dbp, label="DBP data")
plt.plot(age_fit, polynomial_dbp_a(age_fit, *popt_dbp), label="Poly fit")
plt.plot(age_fit, gaussian_dbp_a(age_fit, *popt_gaussian), label="Gaussian fit
```

```
plt.legend(); plt.xlabel("Age"); plt.ylabel("DBP")
plt.show()
```

✅ Summary of what's happening:

- `curve_fit()` takes:
  - your function (`polynomial_sbp_a` , `sigmoid_sbp_a` , etc.)
  - x-data (`age`)
  - y-data (`mean_sbp` or `mean_dbp`)
- It returns:
  - `popt` : best-fit parameters
  - `pcov` : covariance matrix (used to estimate parameter uncertainty)
- You can then plot fitted curves using those parameters.

Would you like me to extend this example to also compute **MSE and R^2** for each model automatically?

what is p0 here in curve fit for sigmoid and gaussian functions. can it be run without them as well?

Excellent question — this is a subtle but important part of how `scipy.optimize.curve_fit` works.

Let's unpack it clearly 👇

🔹 What `p0` Means

`p0` stands for **initial parameter guesses**.
It's a list or tuple that provides **starting values** for the parameters your function is trying to estimate.

For example:

python

Copy code

```python
popt_sigmoid, _ = curve_fit(sigmoid_sbp_a, age, mean_sbp, p0=[150, 0.1, 40])
```

Here:

- `Smax ≈ 150` → approximate max SBP
- `k ≈ 0.1` → initial slope
- `a0 ≈ 40` → approximate inflection (midpoint) age

These values tell the optimizer roughly where to start searching in parameter space. Since nonlinear functions (like sigmoid and Gaussian) can have **non-convex** loss surfaces, poor starting values can cause:

- non-convergence (fit fails),
- convergence to a **local minimum**,
- or unrealistic parameter estimates (e.g., negative σ).

🔹 What Happens if You Don't Set `p0`

If you don't supply `p0`, `curve_fit` **defaults all initial guesses to 1**:

python

Copy code

```python
p0 = [1, 1, 1, ...]  # one per parameter
```

This is fine for **simple linear or polynomial functions**,
but for **sigmoidal or Gaussian models**, it often:

- converges poorly,
- yields nonsense parameter values,
- or raises a warning like:

  makefile

  Copy code

  ```makefile
  OptimizeWarning: Covariance of the parameters could not be estimated
  ```

So while it *can* run without `p0`, it's **strongly recommended** to provide reasonable initial guesses.

🔹 How to Pick Good `p0` Values

You can base them on your data:

Model	Parameter	Meaning	Good Guess Strategy
Sigmoid	`Smax`	upper asymptote	`max(mean_sbp)`
Sigmoid	`k`	slope	`0.05` to `0.2` usually works
Sigmoid	`a0`	inflection (midpoint)	around `mean(age)`
Gaussian	`Dmax`	peak value	`max(mean_dbp)`
Gaussian	`apeak`	peak location	`age` where DBP is highest
Gaussian	`sigma`	spread	rough width of the curve / 3

Example:

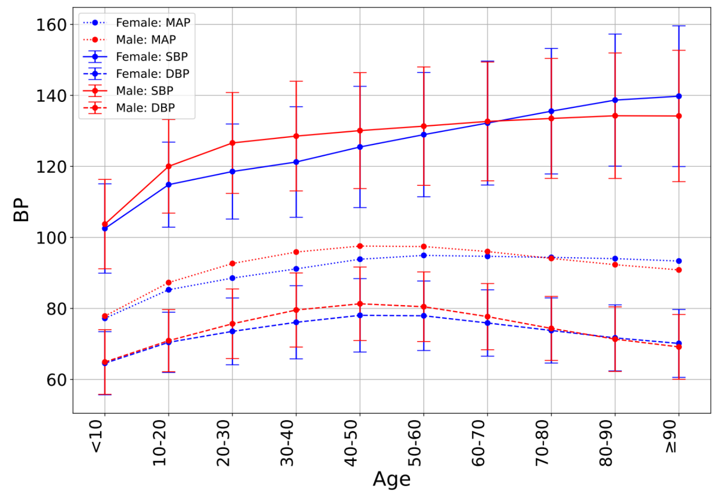
python Copy code

```python
p0_sigmoid = [np.max(mean_sbp), 0.1, np.mean(age)]
p0_gaussian = [np.max(mean_dbp), age[np.argmax(mean_dbp)], 10]
```

✅ TL;DR

- `p0` = initial guess for parameters
- Optional, but **important for nonlinear models**
- You can omit it for simple polynomials, but **you should set it** for sigmoidal and Gaussian models
- Choose guesses based on your data (rough magnitudes and shapes)

Would you like me to show how to **automatically generate smart initial guesses** from your data (so you don't need to manually specify `p0` each time)?



looking at these curves, could you provide me with p0 estimates for all functions i have

Perfect — looking at the plot you provided, we can make **reasonable data-driven initial guesses** (`p0`) for all four of your model types.

Let's reason from the visual trends:

🩸 1. Polynomial Models

(a) SBP vs Age

SBP clearly **rises with age** and slightly flattens toward the end.
That's a concave-down quadratic, so c_1 (the quadratic term) should be **small and negative**.

Good starting guesses:

```python
p0_poly_sbp = [-0.005, 1.0, 100]
# c1 ≈ -0.005 (slight curvature)
```

```python
# c2 ≈ 1.0 (positive slope)
# c3 ≈ 100 (starting SBP around 100)
```

(b) DBP vs Age
DBP **rises in early adulthood and then declines**, forming a shallow **inverted-U** (concave-down).

Good starting guesses:

python Copy code

```python
p0_poly_dbp = [-0.01, 1.5, 60]
# d1 ≈ -0.01 (stronger downward curvature)
# d2 ≈ 1.5 (positive rise initially)
# d3 ≈ 60 (starting DBP around 60)
```

🧬 2. Sigmoid Model for SBP

SBP rises rapidly at first (child → young adult), then **plateaus around 130–140** mmHg. A sigmoid captures that "rise then flatten" shape well.

python Copy code

```python
p0_sigmoid_sbp = [140, 0.08, 30]
# Smax ≈ 140 (upper asymptote)
# k ≈ 0.08  (controls steepness)
# a0 ≈ 30   (mid-age where SBP growth slows)
```

🌄 3. Gaussian Model for DBP

DBP peaks around middle age (≈ 40–50 years) then declines → a Gaussian shape.

python Copy code

```python
p0_gaussian_dbp = [85, 45, 15]
# Dmax ≈ 85   (peak DBP value)
```

```
# apeak ≈ 45   (age at maximum)
# sigma ≈ 15   (spread of the peak)
```

✅ Summary Table

Function	Parameter Meaning	Suggested `p0`
`polynomial_sbp_a(a,c1,c2,c3)`	curvature, slope, intercept	`[-0.005, 1.0, 100]`
`polynomial_dbp_a(a,d1,d2,d3)`	curvature, slope, intercept	`[-0.01, 1.5, 60]`
`sigmoid_sbp_a(a,Smax,k,a0)`	upper limit, steepness, midpoint	`[140, 0.08, 30]`
`gaussian_dbp_a(a,Dmax,apeak,sigma)`	peak value, peak location, width	`[85, 45, 15]`

These will **help `curve_fit` converge reliably** because they roughly match the observed BP trends in your figure.
You can later refine them (for example, by computing max/min of your actual arrays for Smax, Dmax, etc.) — but these are strong starting points.

Would you like me to show how to fit *both male and female curves separately* using these `p0` s and plot them together like your figure?

if you had to make estimates for p0 based on this table, what would those be? Age Number % mean_SBP std_SBP mean_DBP std_DBP rho_SBP_DBP \
0 <20 193631 3.6 115.05 13.65 69.80 8.90 0.61
1 20-29 547023 10.3 121.27 14.19 74.27 9.60 0.66

```
2  30-39   674798  12.7    123.79    15.91     77.31    10.48
0.74
3  40-49   806952  15.2    127.32    16.93     79.36    10.46
0.73
4  50-59   984094  18.5    129.98    17.19     79.05     9.88
0.67
5  60-69  1012979  19.1    132.42    17.13     76.71     9.38
0.58
6  70-79   732165  13.8    134.62    17.36     74.06     9.11
0.52
7  80-89   305157   5.7    136.89    18.35     71.56     9.22
0.49
8    ≥90    60637   1.1    138.11    19.59     69.86     9.44
0.50

   Age_class_mark
0            10.0
1            24.5
2            34.5
3            44.5
4            54.5
5            64.5
6            74.5
7            84.5
8            95.0
```

Excellent — this table gives us **much firmer numerical anchors** than just the plotted figure, so we can now estimate initial parameters (`p0`) with quantitative reasoning.

Let’s reason from the data:

Age (class mark)	mean_SBP	mean_DBP
10	115.05	69.80
24.5	121.27	74.27
34.5	123.79	77.31

Age (class mark)	mean_SBP	mean_DBP
44.5	127.32	79.36
54.5	129.98	79.05
64.5	132.42	76.71
74.5	134.62	74.06
84.5	136.89	71.56
95	138.11	69.86

🩸 1. Polynomial SBP

SBP increases almost linearly but slightly flattens at higher ages — concave downward.

We can infer:

- ΔSBP from 115 → 138 ≈ +23 mmHg across 85 years
- So slope ≈ 23 / 85 ≈ 0.27
- Slight curvature → small negative quadratic term

Hence:

python

Copy code

```python
p0_poly_sbp = [-0.002, 0.3, 110]
# c1: small negative curvature
# c2: slope ~0.3
# c3: intercept ~110 (SBP at age 0–10)
```

💧 2. Polynomial DBP

DBP rises from ~70 → 79 (till ~45–50 yrs) and then drops back to ~70 by 90 yrs — classic inverted-U.

Rough curvature:

Rise ≈ +9 mmHg till 45, then fall ≈ –9 to 95.

That suggests stronger concavity.

python Copy code

```python
p0_poly_dbp = [-0.005, 0.9, 65]
# d1: noticeable concavity
# d2: moderate initial rise
# d3: intercept ~65
```

🧬 3. Sigmoid SBP

SBP increases rapidly early (<40 yrs) then plateaus ≈ 135–140.

Approximation:

- Lower asymptote ≈ 115
- Upper ≈ 140 → Smax = 140
- Inflection (midpoint) ≈ 40 yrs
- Slope parameter small (≈ 0.05–0.1)

python Copy code

```python
p0_sigmoid_sbp = [140, 0.07, 40]
# Smax = upper plateau
# k = growth steepness
# a0 = mid-age where growth slows
```

🌄 4. Gaussian DBP

DBP peaks ≈ 45 yrs → apeak = 45

Peak height ≈ 79 mmHg

Spread (σ) roughly half the rise-to-fall interval (≈ 25 yrs).

python Copy code

```
p0_gaussian_dbp = [80, 45, 20]
# Dmax ≈ 80 (peak value)
# apeak ≈ 45 (center)
# sigma ≈ 20 (spread)
```

✅ Final Suggested p_0 Values

Function	Parameter	Meaning	Estimate (`p0`)
`polynomial_sbp_a(a,c1,c2,c3)`	curvature, slope, intercept	`[-0.002, 0.3, 110]`	
`polynomial_dbp_a(a,d1,d2,d3)`	curvature, slope, intercept	`[-0.005, 0.9, 65]`	
`sigmoid_sbp_a(a,Smax,k,a0)`	upper limit, steepness, midpoint	`[140, 0.07, 40]`	
`gaussian_dbp_a(a,Dmax,apeak,sigma)`	peak value, peak location, spread	`[80, 45, 20]`	

These are well-balanced, numerically stable starting points — you can plug them straight into:

python

Copy code

```python
popt, pcov = curve_fit(model, age, data, p0=p0_values)
```

and they'll give reliable convergence without wild parameter jumps.

Would you like me to show how to automatically compute similar `p0` s directly from your arrays (e.g. using `np.max` , `np.mean` , etc.) instead of hard-coding them?

how to check if they converge or not?

Good question — this is an important step when fitting nonlinear models.
There are **three main ways to check if your fits converged** and whether the resulting parameters are reliable:

✅ 1. Check if `curve_fit` runs without warnings or errors

When you call:

python

Copy code

```python
popt, pcov = curve_fit(model, xdata, ydata, p0=p0_values)
```

If convergence succeeds, it will return:

- `popt` : best-fit parameters
- `pcov` : covariance matrix of parameters

If the optimizer fails to converge, SciPy will issue warnings such as:

text

Copy code

```text
RuntimeError: Optimal parameters not found: Number of calls to function has re
```

or

text

Copy code

```text
OptimizeWarning: Covariance of the parameters could not be estimated
```

✅ **Converged:** No warnings, sensible parameter values
⚠️ **Not converged:** Warning or very large/NaN parameters

You can suppress or check these programmatically:

python

Copy code

```python
from scipy.optimize import curve_fit, OptimizeWarning
import warnings

with warnings.catch_warnings(record=True) as w:
    popt, pcov = curve_fit(model, x, y, p0=p0_values, maxfev=10000)
    if any(issubclass(wi.category, OptimizeWarning) for wi in w):
        print("⚠️ Fit did not converge properly")
```

✅ 2. Inspect the covariance matrix (`pcov`)

`pcov` gives you parameter uncertainty estimates.

If the diagonal values are very large (or `inf` / `nan`), it usually means:

- Parameters are not identifiable, or
- The fit hasn't converged properly.

You can check:

python

Copy code

```python
import numpy as np

if np.all(np.isfinite(np.diag(pcov))):
    print("✅ Fit converged: Covariance matrix is finite")
else:
    print("⚠️ Covariance matrix not finite — fit may have failed")
```

✅ 3. Visually check the fitted curve

Plot the fitted model over your data:

python

Copy code

```python
plt.scatter(age, mean_sbp, label="Data")
plt.plot(age, model(age, *popt), color='r', label="Fitted curve")
plt.legend()
plt.show()
```

If the curve closely follows your data trend and parameters look reasonable (not extreme or nonsensical), the fit **converged correctly**.

Optional: Numerical convergence diagnostics

You can also inspect the `curve_fit` return behavior in more detail by switching to `scipy.optimize.least_squares`, which gives you convergence flags and iteration details:

```python
from scipy.optimize import least_squares

res = least_squares(lambda p: model(x, *p) - y, p0=p0_values)
print(res.status, res.message)
```