DETERMINISTIC OPTIMAL GROWTH MODEL

- Consider the following social planner's problem:

$$\max_{c_t, k_{t+1}} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\gamma}}{1-\gamma},$$

s.t. $c_t + k_{t+1} = k_t^\alpha + (1-\delta)k_t$, and $k_0 > 0$ is given.

BELLMAN EQUATION

- The recursive formulation is given by

$$v(k_t) = \max_{c_t, k_{t+1}} [\frac{c_t^{1-\gamma}}{1-\gamma} + \beta v(k_{t+1})],$$

s.t. $c_t + k_{t+1} = k_t^\alpha + (1-\delta)k_t$, and $k_0 > 0$ is given.

- Or

$$v(k_t) = \max_{c_t, k_{t+1}} \{ \frac{[k_t^\alpha + (1-\delta)k_t - k_{t+1}]^{1-\gamma}}{1-\gamma} + \beta v(k_{t+1}) \}$$

SOLUTION EVALUATION

- No analytical solution when $0 < \delta < 1$.
- How can one ensure the accuracy of a numerical solution when there is no analytical solution available for comparison?

WHAT RESOURCES ARE AVAILABLE?

- Euler equation

$$(\frac{c_{t+1}}{c_t})^{\gamma} = \beta[\alpha k_{t+1}^{\alpha-1} + (1-\delta)]$$

- Or

$$[\frac{k_{t+1}^{\alpha} + (1-\delta)k_{t+1} - k_{t+2}}{k_t^{\alpha} + (1-\delta)k_t - k_{t+1}}]^{\gamma} = \beta[\alpha k_{t+1}^{\alpha-1} + (1-\delta)]$$

EULER EQUATION RESIDUALS

- Suppose we have an arbitrary $\tilde{g}(k)$ that (approximately) satisfies the Euler equation for all k. Can we say that $\tilde{g}(k)$ is a good approximation to the true solution of the model?
- Yes!
    - Santos, Manuel. 2000. “Accuracy of Numerical Solutions Using the Euler Equation Residuals.” Econometrica 68(6): 1377–1402.

MAIN TAKEAWAYS

- Euler equations can be easily computed for any arbitrary policy function.
- Under standard regularity conditions, the accuracy of approximation of a **policy function** is proportional to the magnitude of its **Euler equation**.
- The constant of proportion depends on primitives like the discount factor, the curvature of the utility function, and the curvature of the value function.

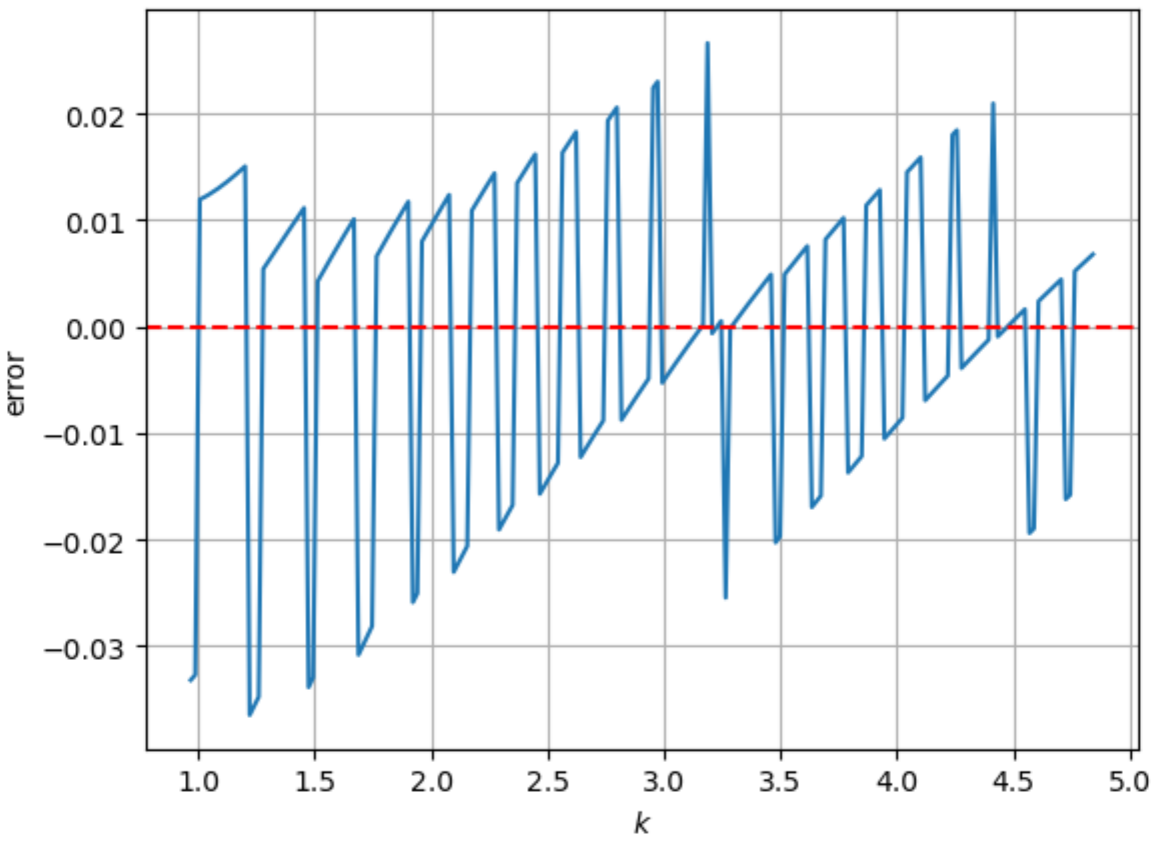
EULER EQUATION RESIDUAL

$$\epsilon = (\frac{c_{t+1}}{c_t})^{\gamma} - \beta[\alpha k_{t+1}^{\alpha-1} + (1-\delta)]$$

```
def euler_error(kgrid, g):
    n = len(kgrid)
    err = np.zeros(n)
    for i in range(n):
        k0 = kgrid[i]
        k1 = g[i]
        k1_index = np.argmin(np.abs(kgrid-k1))
        k2 = g[k1_index]
        c0 = budget(k0, k1)
        c1 = budget(k1, k2)
        err[i] = (c1/c0)**gamma - beta*(alpha*k1**(alpha-1)+1-delta)
    return err


err = euler_error(kgrid, g)
```

ILLUSTRATION




```
maximum error of 200 grids: 0.03647101167407385
```

IMPROVE THE NUMERICAL ACCURACY

- There is a significant error arising from the fact that the choice set is discrete, limited, and fixed.
- possible solution
    - increase the number of grid points
    - generate the grid points efficiently
    - interpolation

GRID NUMBER MATTERS!

grid number	maximum error
100	0.0768
200	0.0364
400	0.0192
800	0.0118
1600	0.0059

- The accuracy doubles when grid number doubles.

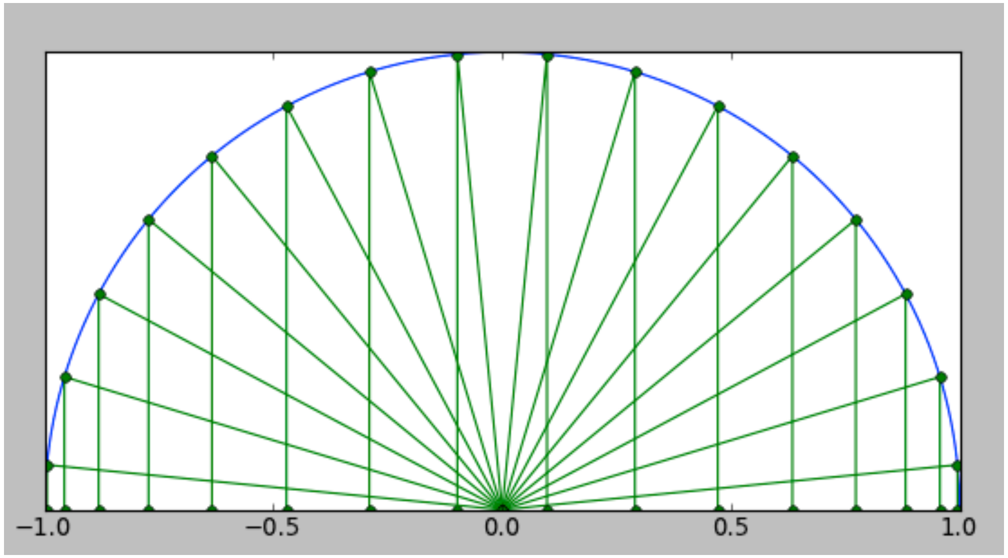
SMART GRIDS

- Use smart grids according to the curvature of the utility function, value function, and the policy function
  - power function
  - exponetial function
  - Chebyshev function

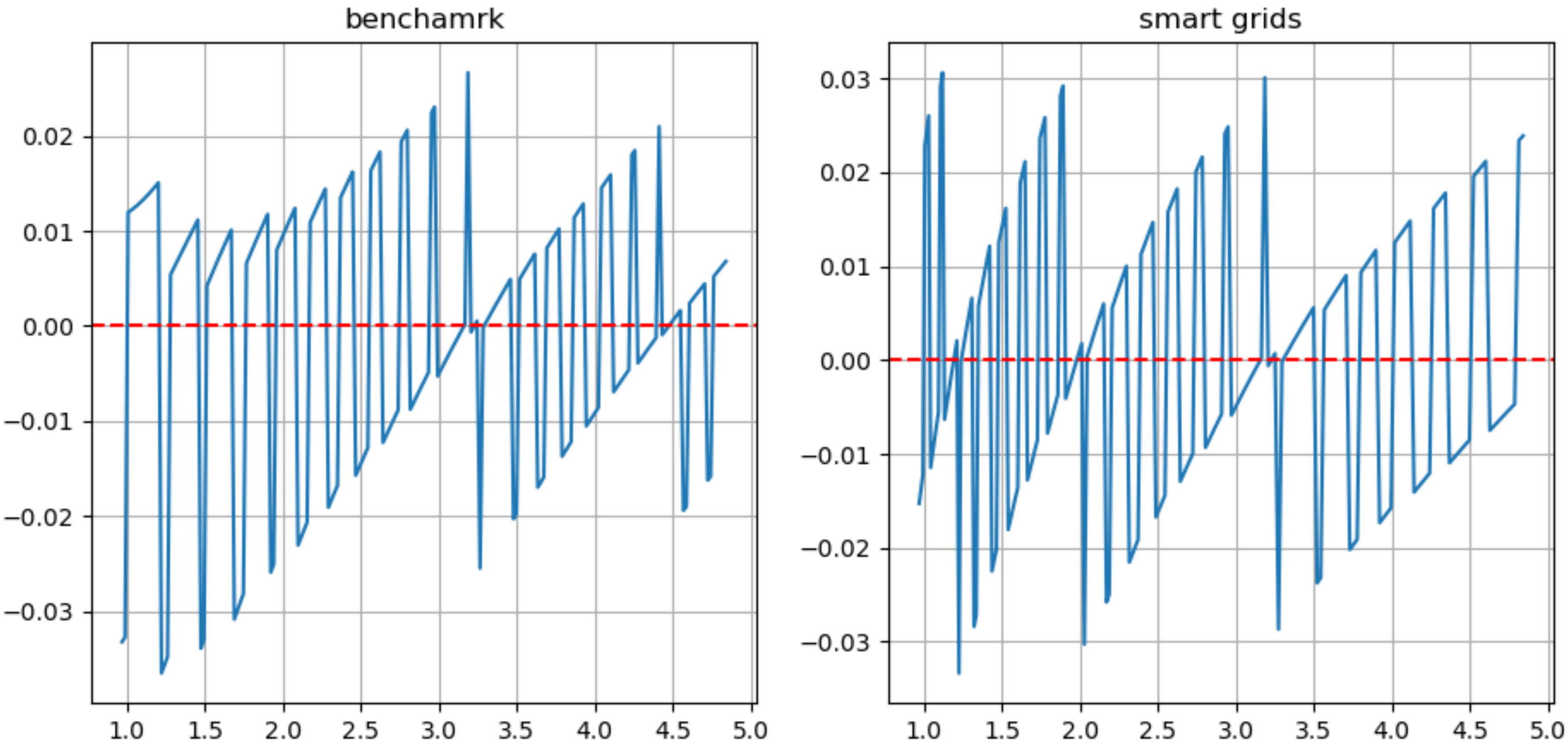
CHEBYSHEV NODES

- For a given positive integer n the Chebyshev nodes of the second kind in the closed interval $[-1, 1]$ are

$$x_k = \cos(\frac{k}{n}\pi), k = 0, 1, \ldots, n-1$$

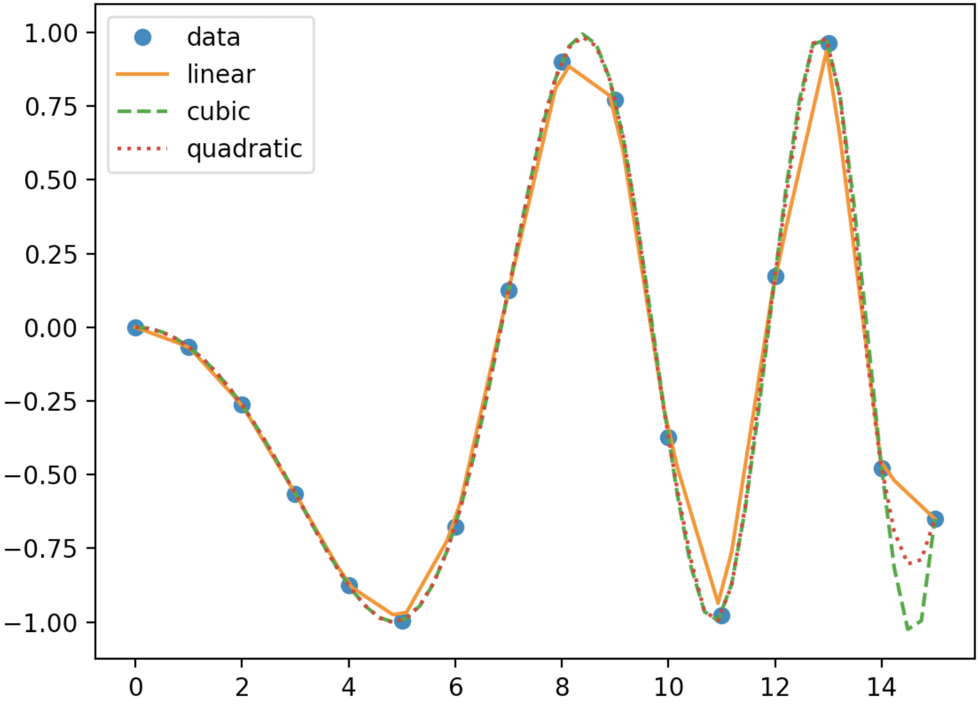


USE $\sqrt{x}$ TO GENERATE GRIDS



INTERPOLATION

- we can utilize interpolation to make each iterated function continuously.



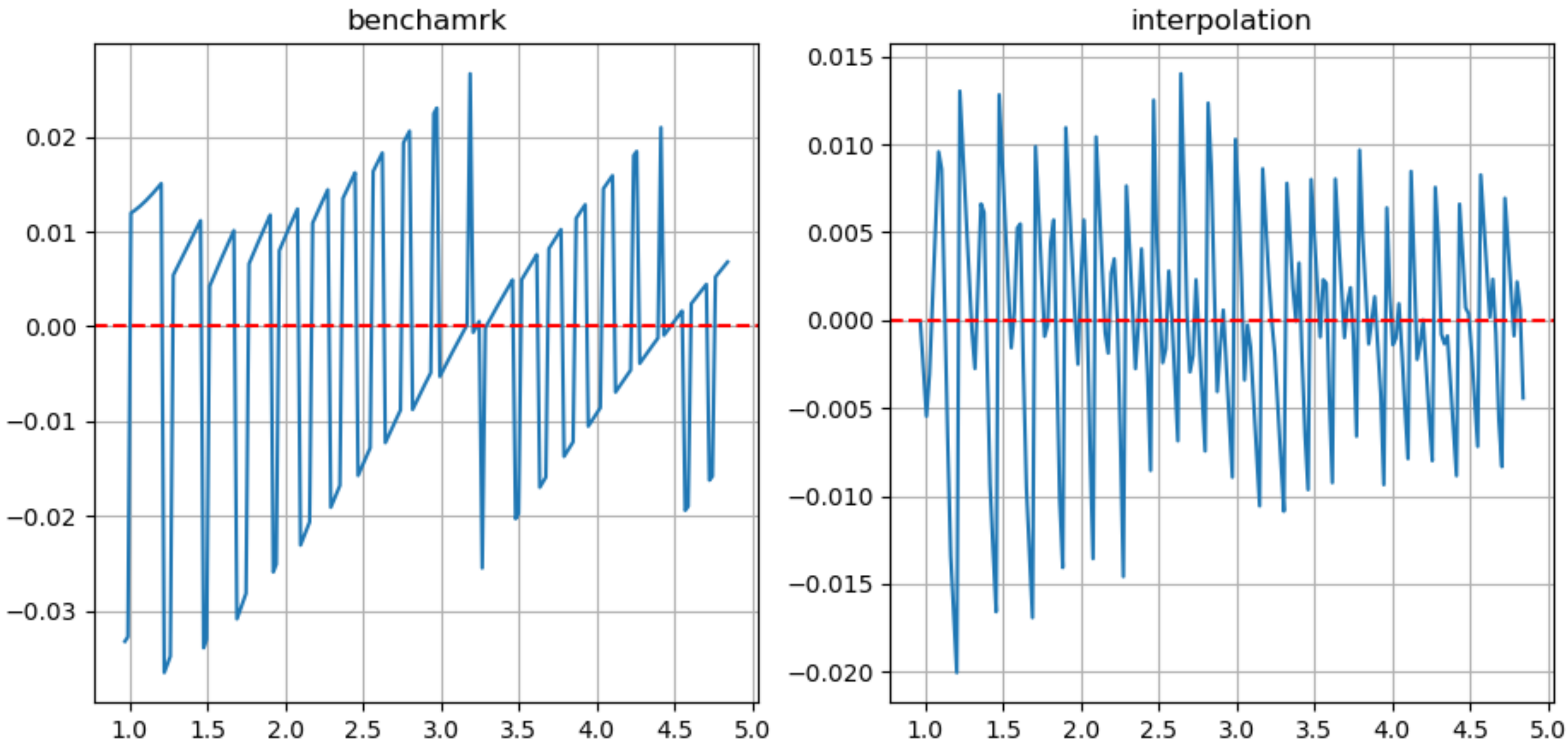
CODE EXAMPLE

```
from scipy.optimize import minimize_scalar


def V_current(k_next, k, kgrid, V):
    """objective value function to be maximized"""
    c = budget(k, k_next)
    V_interp = np.interp(k_next, kgrid, V)
    res = -(u(c) + beta * V_interp)
    return res


def V_max(k_index, kgrid, V):
    '''choose best k_next to maximize V'''
    k = kgrid[k_index]
    k_bound = budget(k, 0)   # keep non-zero consumption
    res = minimize_scalar(fun=V_current, bounds=(
        kmin, k_bound), args=(k, kgrid, V))
    V_max = -res.fun
    g_k = res.x
    return V_max, g_k
```

COMPARISON



benchamrk
0.02
0.01
0.00
−0.01
−0.02
−0.03
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
interpolation
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

COMPARISON OF MAXIMUM ERROR

grid number	benchmark	$\sqrt{x}$	interpolation
100	0.0768	0.0685	0.0481
200	0.0364	0.0333	0.0200
400	0.0192	0.0161	0.0118
800	0.0118	0.0083	0.0053
1600	0.0059	0.0043	0.0026

- The accuracy increases when considering curvature or using interpolation.

COMPARISON OF TIME

grid number	interpolation	benchmark
100	3.58	1.56
200	6.74	6.47
400	12.79	25.41
800	25.52	101.88
1600	50.56	415.94

- You can roughly estimate the required time based on the grid resolution.
    - benchmark: doubling the number of grids leads to a fourfold increase in time.
    - interpolation: doubling the number of grids leads to a twofold increase in time.

QUESTION

- Are there any methods to enhance computing speed and reduce computational time, allowing us to increase the grid resolution and enhance accuracy?