example 1

挑一个简单示例，看我们能不能在多轮迭代之后使得这个问题变得很困难。

```
{
        "problem": "Find the distance between the points $(2,1,-4)$ and
$(5,8,-3).$",
        "level": "Level 2",
        "type": "Precalculus",
        "solution": "The distance between $(2,1,-4)$ and $(5,8,-3)$ is\n\\
[\\sqrt{(2 - 5)^2 + (1 - 8)^2 + (-4 + 3)^2} = \\boxed{\\sqrt{59}}.\\]"
}
```

获取解决方案

- 计划过滤获取正确答案的问题，或者我们手工debug不正确的一部分再加进去做初始solution。

```
import math
def solution():
    # Coordinates of the points
    x1, y1, z1 = 2, 1, -4
    x2, y2, z2 = 5, 8, -3

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    return distance
```

evolve 1

- 计划使用GPT4 验证答案，答案正确就认为他对了。

数值增强

```
import math
def solution():
    # Coordinates of the points
    x1, y1, z1 = 12, -7, 5
    x2, y2, z2 = -3, 15, 9

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    return distance
```

What is the distance between the points $P_1 = (12, -7, 5)$ and $P_2 = (-3, 15, 9)$ in three-dimensional space?

```
import math

def solution():
    # Given points P1 and P2
    P1 = (12, -7, 5)
    P2 = (-3, 15, 9)

    # Calculate the distance between P1 and P2 using the distance formula in 3D space
    distance = math.sqrt((P2[0] - P1[0])**2 + (P2[1] - P1[1])**2 + (P2[2] - P1[2])**2)

    return distance
```

逻辑增强

```
import math
def solution():
    # Coordinates of the points
    x1, y1, z1 = 2, 1, -4
    x2, y2, z2 = 5, 8, -3

    # Compute the squared differences for each coordinate
    dx_squared = (x2 - x1) ** 2
    dy_squared = (y2 - y1) ** 2
    dz_squared = (z2 - z1) ** 2

    # New constraint: Apply a weight based on the sum of squared differences
    weight_factor = (dx_squared + dy_squared + dz_squared) ** 0.5

    # Distance formula adjusted by the weight factor
    distance = math.sqrt(dx_squared + dy_squared + dz_squared) * weight_factor
    return distance
```

Two points in three-dimensional space have coordinates $P_1(2, 1, -4)$ and $P_2(5, 8, -3)$. Compute the distance between the points, adjusting the result by a weight factor that is the square root of the sum of the squared differences of their coordinates.

```
import math

def solution():
    # Given points P1(2, 1, -4) and P2(5, 8, -3)
    P1 = (2, 1, -4)
    P2 = (5, 8, -3)

    # Compute the squared differences of the coordinates
    diff_x = P2[0] - P1[0]
    diff_y = P2[1] - P1[1]
    diff_z = P2[2] - P1[2]

    # Calculate the square of the differences
    squared_diff = diff_x**2 + diff_y**2 + diff_z**2

    # Compute the distance (Euclidean distance)
    distance = math.sqrt(squared_diff)

    # Compute the weight factor (the square root of the sum of squared differences)
    weight_factor = math.sqrt(squared_diff)

    # The final result is the distance adjusted by the weight factor
    result = distance * weight_factor

    return result
```

维度增强

```
import math
def solution():
    # Coordinates of the points in 3D space
    x1, y1, z1, t1 = 2, 1, -4, 3
    x2, y2, z2, t2 = 5, 8, -3, 7

    # 4D distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2 + (t2 - t1)^2)
    distance_4d = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2 + (t2 - t1)**2)
    return distance_4d
```

Given two points in four-dimensional space, $P_1 = (2, 1, -4, 3)$ and $P_2 = (5, 8, -3, 7)$, find the distance between these two points.

```
import math
def solution():
    # Coordinates of points P1 and P2
    P1 = (2, 1, -4, 3)
    P2 = (5, 8, -3, 7)

    # Calculate the distance between the points using the Euclidean distance formula
    distance = math.sqrt(sum((P2[i] - P1[i])**2 for i in range(4)))
    return distance
```

方法增强

```
import math
def solution():
    # Coordinates of the points
    x1, y1, z1 = 2, 1, -4
    x2, y2, z2 = 5, 8, -3

    # Using vector algebra to compute the Euclidean distance
    # Define vectors A and B as points
    A = (x1, y1, z1)
    B = (x2, y2, z2)

    # Vector difference (B - A)
    vector_diff = (B[0] - A[0], B[1] - A[1], B[2] - A[2])

    # Squared magnitude of the vector difference (dot product of the vector with itself)
    squared_magnitude = sum(v**2 for v in vector_diff)

    # The distance is the square root of the squared magnitude
    distance = math.sqrt(squared_magnitude)
    return distance
```

- 其实没有信息增益

代码续写

```
import math
def solution():
    # Coordinates of the points
    x1, y1, z1 = 2, 1, -4
    x2, y2, z2 = 5, 8, -3

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)

    # Midpoint formula: ((x1 + x2)/2, (y1 + y2)/2, (z1 + z2)/2)
    midpoint = ((x1 + x2) / 2, (y1 + y2) / 2, (z1 + z2) / 2)

    # Additional step: Calculate the angle between the vector connecting the points and the x-axis
    vector_x = x2 - x1
    vector_y = y2 - y1
    vector_z = z2 - z1
    angle_with_x_axis = math.acos(vector_x / distance)  # Angle in radians

    result = (distance, midpoint, angle_with_x_axis)
    return result
```

Question:Consider two points in three-dimensional space with coordinates (2,1,−4) and (5,8,−3). Calculate the cube of the distance between these two points.

- 返回的东西有点多，而且有无用的变量！我们可以把这些无用的变量用语法树的方式去掉，但就是在算angle的时候感觉还是不错的。
- prompt应该要求只返回一个数学内容。

破开条件

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    result = distance
    return result
```

Question:Given two points in a three-dimensional space, one point has coordinates (5, 8, -3). If the midpoint of the line segment connecting these two points is (3.5, 4.5, -3.5), find the distance between these two points.

evolve 2

- wizardLM中随机选一个，我们可以让模型评估我们的问题和solution的质量，取一个最具有教育意义的进化。
- 这里偷懒，我认为破开条件是进化的最好的。或者我们也可以看看随机的效果怎么样。

原始问题

Question:Given two points in a three-dimensional space, one point has coordinates (5, 8, -3). If the midpoint of the line segment connecting these two points is (3.5, 4.5, -3.5), find the distance between these two points.

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where
x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    result = distance
    return result
```

数值增强

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 127, -512, 61

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where
x1, y1, z1 is another point
    mx, my, mz = 215.7, -325.4, 112.2

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    result = distance
    return result
```

逻辑增强

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Introduce an additional constraint to introduce a variation in the problem
    # Consider that the distance formula includes a scaling factor proportional to the midpoint
    scaling_factor = math.sqrt(mx**2 + my**2 + mz**2)

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    # Apply the scaling factor to the final distance calculation
    distance = scaling_factor * math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)

    result = distance
    return result
```

Two points in three-dimensional space, $P_2 = (5, 8, -3)$ and $P_1 = (x_1, y_1, z_1)$, are given. The midpoint of the segment connecting these points is $M = (3.5, 4.5, -3.5)$. Find the distance between the points $P_1$ and $P_2$ after introducing a scaling factor based on the distance of the midpoint from the origin.

```
import sympy as sp

def solution():
    # Define the variables
    x1, y1, z1 = sp.symbols('x1 y1 z1')

    # Coordinates of P2 and midpoint M
    P2 = (5, 8, -3)
    M = (3.5, 4.5, -3.5)

    # Midpoint formula
    midpoint_eq = sp.Eq((x1 + P2[0])/2, M[0]), sp.Eq((y1 + P2[1])/2, M[1]), sp.Eq((z1 + P2[2])/2, M[2])

    # Solve the system of equations to find x1, y1, z1
    solution_point = sp.solve([midpoint_eq[0], midpoint_eq[1], midpoint_eq[2]], (x1, y1, z1))

    # Calculate the distance between P1 and P2 using the distance formula
    distance = sp.sqrt((P2[0] - solution_point[x1])**2 + (P2[1] - solution_point[y1])**2 + (P2[2] - solution_point[z1])**2)

    # Calculate the distance of the midpoint from the origin
    midpoint_distance = sp.sqrt(M[0]**2 + M[1]**2 + M[2]**2)

    # Apply the scaling factor to the distance
    scaled_distance = distance * midpoint_distance

    return scaled_distance
```

维度增强

```
import math

def solution():
    # Coordinates of the first point (x2, y2, z2)
    x2, y2, z2 = 5, 8, -3

    # Coordinates of the second point (x3, y3, z3)
    x3, y3, z3 = 2, -1, 4

    # Define the midpoint coordinates of the line segment between point (x2, y2, z2) and point (x3, y3, z3)
    mx, my, mz = (x2 + x3) / 2, (y2 + y3) / 2, (z2 + z3) / 2

    # Define another midpoint (mx1, my1, mz1) between (x1, y1, z1) and the new midpoint (mx, my, mz)
    mx1, my1, mz1 = (mx + x2) / 2, (my + y2) / 2, (mz + z2) / 2

    # Calculate the distance between (x2, y2, z2) and (x3, y3, z3) using the distance formula
    distance_between_points = math.sqrt((x3 - x2)**2 + (y3 - y2)**2 + (z3 - z2)**2)

    # Calculate the distance between the two midpoints (mx1, my1, mz1) and (mx, my, mz)
    distance_between_midpoints = math.sqrt((mx - mx1)**2 + (my - my1)**2 + (mz - mz1)**2)

    # Combine both distances as the new result to create a more complex solution
    result = distance_between_points + distance_between_midpoints
    return result
```

Given two points in three-dimensional space, the first with coordinates $(5, 8, -3)$ and the second with coordinates $(2, -1, 4)$, find the sum of the following two distances:

1. The distance between the two points.
2. The distance between the midpoint of the line segment connecting these two points and the midpoint of the line segment between the first point and the midpoint.

```
import math

def solution():
    # Given points
    P1 = (5, 8, -3)
    P2 = (2, -1, 4)

    # Function to calculate distance between two points in 3D
    def distance(p1, p2):
        return math.sqrt((p2[0] - p1[0])**2 + (p2[1] - p1[1])**2 + (p2[2]
- p1[2])**2)

    # 1. Distance between the two points (P1 and P2)
    dist_1 = distance(P1, P2)

    # Midpoints:
    # Midpoint of P1 and P2
    midpoint_1_2 = ((P1[0] + P2[0]) / 2, (P1[1] + P2[1]) / 2, (P1[2] +
P2[2]) / 2)

    # Midpoint of P1 and midpoint_1_2
    midpoint_1_mid1_2 = ((P1[0] + midpoint_1_2[0]) / 2, (P1[1] +
midpoint_1_2[1]) / 2, (P1[2] + midpoint_1_2[2]) / 2)

    # 2. Distance between the midpoint of the line segment (P1-P2) and the
midpoint of the line segment (P1-midpoint_1_2)
    dist_2 = distance(midpoint_1_2, midpoint_1_mid1_2)

    # Total distance
    total_distance = dist_1 + dist_2

    return total_distance
```

方法加强

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Using the inverse of the midpoint formula to calculate x1, y1, z1
    # Derived from mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Applying the generalized distance formula based on Euclidean space in 3D
    distance_squared = (x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2
    distance = math.sqrt(distance_squared)

    # Utilize a slight modification by adding a constant for more complexity
    result = distance * math.sqrt(2)  # Applying a transformation to the distance

    return result
```

Given the coordinates of a point $P_2 = (5, 8, -3)$ in 3-dimensional space, and the midpoint $M = (3.5, 4.5, -3.5)$ of the segment joining $P_2$ and another point $P_1$, find the distance between $P_1$ and $P_2$, multiplied by $\sqrt{2}$.

- 没有信息收益，但是方法加强我预估是最有收益的，初步判断是问题不适合 or Prompt比较失败

代码续写

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx - x2
    y1 = 2 * my - y2
    z1 = 2 * mz - z2

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    # Additional step: Calculate the angle between the two vectors (x1,
y1, z1) and (x2, y2, z2)
    # Dot product formula: A · B = x1*x2 + y1*y2 + z1*z2
    dot_product = x1 * x2 + y1 * y2 + z1 * z2

    # Magnitudes of the vectors: ||A|| = sqrt(x1^2 + y1^2 + z1^2), ||B|| =
sqrt(x2^2 + y2^2 + z2^2)
    magnitude_a = math.sqrt(x1**2 + y1**2 + z1**2)
    magnitude_b = math.sqrt(x2**2 + y2**2 + z2**2)

    # Cosine of the angle: cos(θ) = (A · B) / (||A|| * ||B||)
    cosine_theta = dot_product / (magnitude_a * magnitude_b)

    # Calculate the angle θ in radians
    angle = math.acos(cosine_theta)

    # Convert the angle from radians to degrees
    angle_degrees = math.degrees(angle)

    result = (distance, angle_degrees)
    return result
```

- 和原本的distance没什么关系，我认为我们应该允许把后面的程序隐藏掉一小部分，自由的解决另一个问题。 Two points $P_1$ and $P_2$ are given in 3-dimensional space. The coordinates of point $P_2$ are $(5, 8, -3)$, and the midpoint $M$ of the segment joining $P_1$ and $P_2$ is $(3.5, 4.5, -3.5)$.

Find the distance between the two points $P_1$ and $P_2$, as well as the angle between the vectors from the origin to each of the points $P_1$ and $P_2$.

```
import math

def solution():
    # Given points P2 and midpoint M
    P2 = (5, 8, -3)
    M = (3.5, 4.5, -3.5)

    # Calculate the coordinates of P1 using the midpoint formula
    P1 = (2 * M[0] - P2[0], 2 * M[1] - P2[1], 2 * M[2] - P2[2])

    # Distance between P1 and P2: Use the distance formula
    distance_P1_P2 = math.sqrt((P2[0] - P1[0])**2 + (P2[1] - P1[1])**2 +
(P2[2] - P1[2])**2)

    # Vectors from the origin to P1 and P2
    vector_P1 = P1
    vector_P2 = P2

    # Dot product of the vectors from the origin to P1 and P2
    dot_product = vector_P1[0] * vector_P2[0] + vector_P1[1] *
vector_P2[1] + vector_P1[2] * vector_P2[2]
```

```
    # Magnitudes of the vectors
    magnitude_P1 = math.sqrt(vector_P1[0]**2 + vector_P1[1]**2 + vector_P1[2]**2)
    magnitude_P2 = math.sqrt(vector_P2[0]**2 + vector_P2[1]**2 + vector_P2[2]**2)

    # Angle between the two vectors using the dot product formula: cos(theta) = dot_product / (magnitude_P1 * magnitude_P2)
    cos_theta = dot_product / (magnitude_P1 * magnitude_P2)
    angle = math.acos(cos_theta)  # Angle in radians

    return distance_P1_P2, angle
```

- 生成的回答和加强的回答，一个用的是pi，另一个是角度，其实都是对的。我们应该在回答中尽可能让他注意回答问题的答案格式，小数、分数、角度、pi之类的，或者米、厘米等单位。

条件破开

```
import math

def solution():
    # Coordinates of one point
    x2, y2, z2 = 5, 8, -3

    # Define mid-point coordinates assuming midpoint (mx, my, mz) where x1, y1, z1 is another point
    mx, my, mz = 3.5, 4.5, -3.5

    # Compute an additional parameter: the scaling factor based on the difference in x-coordinates
    scaling_factor = (x2 - mx) * 0.1 + 1.5

    # Modify midpoint values by applying scaling factor
    mx_scaled = mx * scaling_factor
    my_scaled = my * scaling_factor
    mz_scaled = mz * scaling_factor

    # Using midpoint formula to calculate x1, y1, z1
    # Midpoint formula: mx = (x1 + x2) / 2 => x1 = 2*mx - x2
    # Similarly for y and z
    x1 = 2 * mx_scaled - x2
    y1 = 2 * my_scaled - y2
    z1 = 2 * mz_scaled - z2

    # Distance formula: sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    result = distance
    return result
```

Given the coordinates of a point $P_2 = (5, 8, -3)$ and a midpoint $M = (3.5, 4.5, -3.5)$, determine the distance between the point $P_2$ and another point $P_1$, where the coordinates of $P_1$ are derived by scaling the midpoint $M$ and using the midpoint formula to find the value of $P_1$. The scaling factor is based on the difference in the $x$-coordinates of $P_2$ and the midpoint $M$.

- 因为对于midpoint的定义又加了一个缩放因子，这个缩放因子又和原本的midpoint有关，导致在逻辑表达上GPT4很难生成一个合理的问题。

evolve3

evolve2中方法加强没有信息收益，条件破开生成的问题不对。 所以只有数值增强、逻辑增强、维度增强、代码续写是好的。 从这些问题中，我觉得代码续写生成的最好，以代码续写作为原始问题。

原始问题

数值增强

逻辑增强

维度增强

方法加强

代码续写

条件破开