实验：解析几何综合练习（无人机定位场景）

已知无人机控制中心 $O(0,0)$，三个传感点位置为

$$A(2,3),\quad B(-1,4),\quad C(5,1).$$

定义向量

$$\overrightarrow{OA},\ \overrightarrow{OB},\ \overrightarrow{OC}.$$

---

(1) 向量与距离基础 — *范数 / 距离*

1. 写出 $\overrightarrow{OA},\ \overrightarrow{OB},\ \overrightarrow{OC}$ 的向量形式
2. 分别计算它们的范数 $\|\overrightarrow{OA}\|, \|\overrightarrow{OB}\|, \|\overrightarrow{OC}\|$
3. 计算点 A 与 B 的欧氏距离 AB，以及 A 与 C 的距离 AC

---

(2) 方向相似性 — *内积 / 夹角 / 方向判断*

1. 计算 $\langle \overrightarrow{OA}, \overrightarrow{OB} \rangle$ 与 $\langle \overrightarrow{OA}, \overrightarrow{OC} \rangle$
2. 根据余弦公式计算两组夹角（弧度与角度）
3. 判断无人机从 O 指向 A 的方向更接近 B 还是 C，并说明依据

---

(3) 正交分解与点到直线距离 — *投影 / 正交残差*

设直线 L 由 OA 的方向确定。令 H 为点 C 到 L 的垂足。

1. 写出 $\mathrm{Proj}_{OA}(\overrightarrow{OC})$ 并计算
2. 计算 CH 的长度（即点 C 到直线 L 的距离）
3. 解释为什么该距离对应的是“最短路径”，用正交/内积语言说明理由

---

(4) 标准正交基构造 — *Gram–Schmidt*

对 $\{\overrightarrow{OA}, \overrightarrow{OB}\}$ 进行 Gram–Schmidt 正交化：

1. 求 $e_1 = \dfrac{\overrightarrow{OA}}{\|\overrightarrow{OA}\|}$
2. 求 $v_\perp = \overrightarrow{OB} - \mathrm{Proj}_{OA}(\overrightarrow{OB})$
3. 求 $e_2 = \dfrac{v_\perp}{\|v_\perp\|}$
4. 验证 $\langle e_1, e_2 \rangle = 0$

---

(5) 线性组合与几何解释 — *深化范数/内积的几何意义*

考虑合成指令

$$d = 0.6\,\overrightarrow{OA} + 0.8\,\overrightarrow{OB}.$$

1. 写出 d 的坐标表达式
2. 计算 $\|d\|$ 并解释其几何含义（如位移强度或能耗近似）
3. 计算 $\langle \overrightarrow{OC}, d \rangle$ 并解释其在方向推进上的意义

---

(6) 概念应用总结 — *几何语言解释算法意义*

用自然语言简述：

- 为什么使用“正交基”能提升无人机定位计算的稳定性或效率
- 为什么“投影+内积”是距离/最优路径/最小误差的核心工具

- 将本实验中各量（范数/内积/正交基）统一在"几何结构"视角下总结

实验：解析几何综合练习（无人机定位场景）— 答题框架

已知无人机控制中心 $O(0,0)$，三个传感点位置为

$$A(2,3),\quad B(-1,4),\quad C(5,1).$$

并定义向量

$$\overrightarrow{OA},\ \overrightarrow{OB},\ \overrightarrow{OC}.$$

> 说明：本框架仅含 `numpy`，不含绘图；所有代码块可直接粘贴到 Notebook 运行。请在 `# TODO` 处完成作答。

---

(1) 向量与距离基础 — *范数 / 距离*

1. 写出 $\overrightarrow{OA},\ \overrightarrow{OB},\ \overrightarrow{OC}$ 的向量形式
2. 分别计算 $\|\overrightarrow{OA}\|,\ \|\overrightarrow{OB}\|,\ \|\overrightarrow{OC}\|$
3. 计算 AB 与 AC 的欧氏距离

```
import numpy as np

# 已知点坐标
O = np.array([0.0, 0.0])
A = np.array([2.0, 3.0])
B = np.array([-1.0, 4.0])
C = np.array([5.0, 1.0])

# 1) 向量 OA, OB, OC
OA = ...  # A - O
OB = ...  # B - O
OC = ...  # C - O

# 2) 范数
norm_OA = ...  # np.linalg.norm(OA)
norm_OB = ...
norm_OC = ...

# 3) 点间距离 AB, AC
AB = ...  # np.linalg.norm(A - B)
AC = ...

print("OA =", OA, " ||OA|| =", norm_OA)
print("OB =", OB, " ||OB|| =", norm_OB)
print("OC =", OC, " ||OC|| =", norm_OC)
print("AB =", AB, " AC =", AC)
```

In [12]:
```
import numpy as np

O = np.array([0.0, 0.0])
A = np.array([2.0, 3.0])
B = np.array([-1.0, 4.0])
C = np.array([5.0, 1.0])

OA = A - O
OB = B - O
OC = C - O

norm_OA = np.linalg.norm(OA)
norm_OB = np.linalg.norm(OB)
norm_OC = np.linalg.norm(OC)

AB = np.linalg.norm(A - B)
AC = np.linalg.norm(A - C)

print("OA =", OA, " ||OA|| =", norm_OA)
print("OB =", OB, " ||OB|| =", norm_OB)
print("OC =", OC, " ||OC|| =", norm_OC)
print("AB =", AB, " AC =", AC)
```

```
OA = [2. 3.]  ||OA|| = 3.605551275463989
OB = [-1.  4.]  ||OB|| = 4.123105625617661
OC = [5. 1.]  ||OC|| = 5.0990195135927845
AB = 3.1622776601683795  AC = 3.605551275463989
```

(2) 方向相似性 — *内积 / 夹角 / 方向判断*

1. 计算 $\langle \overrightarrow{OA}, \overrightarrow{OB} \rangle$ 与 $\langle \overrightarrow{OA}, \overrightarrow{OC} \rangle$

2. 根据余弦公式

$$\cos\theta = \frac{\langle x, y \rangle}{\|x\| \, \|y\|}$$

计算对应夹角（弧度与角度）

3. 根据计算的夹角大小，判断从 O 指向 A 的方向更接近 B 还是 C，并说明依据（即：哪个夹角更小）

In [5]:
```python
import numpy as np

# 已知点坐标（承接上一题的向量和范数）
O = np.array([0.0, 0.0])
A = np.array([2.0, 3.0])
B = np.array([-1.0, 4.0])
C = np.array([5.0, 1.0])

# 向量定义（承接上一题）
OA = A - O
OB = B - O
OC = C - O

# 范数（承接上一题）
norm_OA = np.linalg.norm(OA)
norm_OB = np.linalg.norm(OB)
norm_OC = np.linalg.norm(OC)

#  内积计算
dot_OA_OB = np.dot(OA, OB)
dot_OA_OC = np.dot(OA, OC)

#  夹角计算（余弦值、弧度、角度）
cos1 = dot_OA_OB / (norm_OA * norm_OB)
cos2 = dot_OA_OC / (norm_OA * norm_OC)
cos1 = np.clip(cos1, -1.0, 1.0)  # 修正浮点误差
cos2 = np.clip(cos2, -1.0, 1.0)

theta1_rad = np.arccos(cos1)
theta2_rad = np.arccos(cos2)
theta1_deg = np.degrees(theta1_rad)  # 转换为角度
theta2_deg = np.degrees(theta2_rad)

print("<OA,OB> =", dot_OA_OB, " angle =", theta1_deg, "deg")
print("<OA,OC> =", dot_OA_OC, " angle =", theta2_deg, "deg")

#  方向接近性判断（角度越小越接近）
closer_to = "B" if theta1_deg < theta2_deg else "C"
print("OA 更接近方向 ->", closer_to)
print("因为夹角更小")
```

```
<OA,OB> = 10.0  angle = 47.72631099390626 deg
<OA,OC> = 13.0  angle = 45.0 deg
OA 更接近方向 -> C
因为夹角更小
```

(3) 正交分解与点到直线距离 — *投影 / 正交残差*

设直线 L 由 $\overrightarrow{OA}$ 的方向确定，记 H 为点 C 到 L 的垂足。

1. 计算 $\mathrm{Proj}_{\overrightarrow{OA}}(\overrightarrow{OC})$

2. 计算 CH 的长度，即点 C 到直线 L 的最短距离
   （提示：$CH = \| \overrightarrow{OC} - \mathrm{Proj}_{OA}(\overrightarrow{OC}) \|$）

3. 说明为什么该距离是“最短路径”，要求用 **正交 / 内积为零** 的语言解释清楚

In [6]:
```python
import numpy as np

# 承接前面的向量定义
O = np.array([0.0, 0.0])
A = np.array([2.0, 3.0])
C = np.array([5.0, 1.0])
OA = A - O
OC = C - O

# 1) 计算OC在OA上的投影
```

```
OC_dot_OA = np.dot(OC, OA)  # OC与OA的内积
OA_dot_OA = np.dot(OA, OA)  # OA与自身的内积（范数平方）
proj_OC_on_OA = (OC_dot_OA / OA_dot_OA) * OA  # 投影向量

# 2) 计算残差r和CH的长度（点C到直线L的距离）
r = OC - proj_OC_on_OA  # 残差向量（OC减去投影）
CH = np.linalg.norm(r)  # 残差的范数即最短距离

print("Proj_{OA}(OC) =", proj_OC_on_OA)
print("Residual r =", r, "  CH distance =", CH)

# 正交性检查（残差r与OA应正交，内积接近0）
orth_check = np.allclose(np.dot(r, OA), 0.0)
print("<r, OA> ≈ 0 ?", orth_check)
print("因为内积接近0")
```

```
Proj_{OA}(OC) = [2. 3.]
Residual r = [ 3. -2.]   CH distance = 3.605551275463989
<r, OA> ≈ 0 ? True
因为内积接近0
```

(4) 标准正交基构造 — *Gram–Schmidt*

对 $\{\overrightarrow{OA}, \overrightarrow{OB}\}$ 执行 Gram–Schmidt 过程：

1. 计算 $e_1 = \dfrac{\overrightarrow{OA}}{\|\overrightarrow{OA}\|}$
2. 计算 $v_\perp = \overrightarrow{OB} - \mathrm{Proj}_{OA}(\overrightarrow{OB})$
   （其中 $\mathrm{Proj}_{OA}(\overrightarrow{OB}) = \dfrac{\langle \overrightarrow{OB}, \overrightarrow{OA} \rangle}{\langle \overrightarrow{OA}, \overrightarrow{OA} \rangle} \overrightarrow{OA}$）
3. 计算 $e_2 = \dfrac{v_\perp}{\|v_\perp\|}$
4. 验证 $\langle e_1, e_2 \rangle = 0$ 并用一句话解释“为何应当正交”

In [ ]:
```
import numpy as np

# e1
norm_OA = ...      # np.linalg.norm(OA)
e1 = ...           # OA / norm_OA

# Proj_{OA}(OB)
OB_dot_OA = ...
proj_OB_on_OA = ...

# v_perp 与 e2
v_perp = ...
norm_v_perp = ...
e2 = ...

print("e1 =", e1)
print("e2 =", e2)
print("<e1,e2> ≈ 0 ?", np.allclose(np.dot(e1, e2), 0.0))
```

(5) 线性组合与几何解释 — *深化范数 / 内积的几何意义*

考虑合成指令

$$d = 0.6\,\overrightarrow{OA} + 0.8\,\overrightarrow{OB}.$$

1. 写出 d 的坐标表达式（代入 OA, OB 的具体坐标）
2. 计算 $\|d\|$ 并解释其几何含义
   （例如可理解为“合成位移强度”或“能耗近似”）
3. 计算 $\langle \overrightarrow{OC}, d \rangle$
   并解释其物理/几何意义
   （如：d 在 OC 方向上的推进效果或对 C 方向贡献）

In [8]:
```
import numpy as np

# 承接前面的向量定义
```

```
O = np.array([0.0, 0.0])
A = np.array([2.0, 3.0])
B = np.array([-1.0, 4.0])
C = np.array([5.0, 1.0])
OA = A - O
OB = B - O
OC = C - O

alpha, beta = 0.6, 0.8
d = alpha * OA + beta * OB  # 合成向量d的坐标
norm_d = np.linalg.norm(d)  # d的范数
dot_OC_d = np.dot(OC, d)    # OC与d的内积

print("d =", d, " ||d|| =", norm_d)
print("<OC, d> =", dot_OC_d)
print("d对C方向有贡献")
```

```
d = [0.4 5. ]  ||d|| = 5.015974481593781
<OC, d> = 7.0
d对C方向有贡献
```

(6) 概念应用总结 — *几何语言解释算法意义*

请用简短自然段说明以下问题：

- 为什么使用"正交基"能提升定位计算的稳定性或效率
  （可从数值稳定性、简化计算、去除冗余相关性等角度解释）
- 为什么"投影 + 内积"是距离 / 最优路径 / 最小误差的核心工具
  （可从"最短距离对应正交"或"内积度量方向与贡献"角度说明）
- 如何将本实验中的范数 / 内积 / 正交基统一理解在"几何结构"视角下
  （可尝试用一句话总结为：所有操作均围绕内积诱导的几何结构展开）

  （在此作答，建议 4–8 句；可结合本题数值与计算结果。）

1）使用正交基时，由于基向量间内积为零，可消除向量间的冗余相关性。这使得向量分解的系数仅需通过简单内积计算（无需解耦合方程组），不仅简化了计算流程，还避免了因基向量高度相关导致的数值病态问题（如矩阵求逆时的误差放大），从而提升定位计算的稳定性与效率。

2）"投影 + 内积" 是核心工具的原因在于：投影的本质是寻找子空间中与目标向量距离最近的点，而此时投影残差与子空间正交（内积为零），这一性质严格保证了距离最短，对应最优路径或最小误差；内积则通过度量向量间的方向一致性，直接量化投影分量的大小，两者结合可高效求解距离、误差等核心几何问题。

3）本实验中的范数、内积、正交基可统一于内积诱导的几何结构：内积定义了向量的夹角和 "相似度"，范数（如欧氏范数）是内积诱导的向量长度度量，正交基则是该结构中相互垂直的 "坐标轴"，所有操作（如距离计算、投影分解、方向判断）均围绕内积所刻画的长度、角度和正交关系展开，共同描述了向量空间的几何性质。

实验：二维几何变换与向量空间运算综合练习

设

$$u = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad v = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad w = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

---

(1) 向量长度与夹角　*(考察：范数、内积、夹角公式)*

1. 分别计算 $\|u\|$, $\|v\|$
2. 计算 $\langle u, v \rangle$
3. 计算二者夹角并判断是锐角/直角/钝角

> Python 验证：用 `np.linalg.norm`、`np.dot`、`np.arccos` 写出对应代码验证。

---

(2) 由 Gram–Schmidt 构造正交基　*(考察：标准正交基、内积与正交化)*

对 $\{u, v\}$ 执行一次 Gram–Schmidt，得到正交基 $\{e_1, e_2\}$，
要求写出完整推导步骤，并验证 $\langle e_1, e_2 \rangle = 0$。

> Python 验证：用 `np.allclose(e1@e2,0)` 验证正交性。

---

(3) 投影与几何解释　*(考察：正交投影公式、几何含义可视化)*

1. 计算 w 在 u 方向上的正交投影 $\mathrm{Proj}_u(w)$
2. 计算投影残差 $r = w - \mathrm{Proj}_u(w)$
3. 证明残差 r 与 u 正交

> Python 验证：`np.allclose(np.dot(r,u),0)`
> 并用 Matplotlib 画出 u, w, $\mathrm{Proj}_u(w)$ 三个向量箭头图。

---

(4) 旋转变换 *(考察：二维旋转矩阵、线性变换性质)*

构造将 $\mathbb{R}^2$ 绕原点逆时针旋转 45° 的矩阵 R，并计算：

1. Ru, Rv 的显式结果
2. 判断旋转后向量是否仍保持长度与夹角性质（理论+数值验证）

> Python 验证：
> 使用 `np.array([[cosθ,-sinθ],[sinθ,cosθ]])` 并验证
> `np.allclose(np.linalg.norm(Ru), np.linalg.norm(u))`
> `np.allclose(np.dot(Ru,Rv), np.dot(u,v))`

---

(5) 几何综合解释 *(考察：整体连贯理解)*

用一段话总结本实验，回答：

> 向量内积、正交基、投影、旋转，这些操作为何都可统一理解在
> “几何结构不变或重构” 的框架下？说明它们的共同几何意义。

In [10]:
```
# 必要库
import numpy as np
import matplotlib.pyplot as plt

# 为了输出更好看的小数
np.set_printoptions(precision=4, suppress=True)

# ======= 绘图辅助函数（已给出，学生不必改动）=======
def draw_vector(ax, vec, color=None, label=None):
    ax.arrow(0, 0, vec[0], vec[1], head_width=0.12, length_includes_head=True, color=color)
    if label:
        ax.text(vec[0]*1.03, vec[1]*1.03, label)

def draw_angle_arc(ax, origin, v1, v2, n=40):
    # 在 origin 处画夹角小弧线（几何标记）
    a = np.arctan2(v1[1], v1[0])
    b = np.arctan2(v2[1], v2[0])
    # 让弧度差在 [-pi, pi]
    diff = (b - a + np.pi) % (2*np.pi) - np.pi
    ts = np.linspace(a, a + diff, n)
    r = 0.7 * min(np.linalg.norm(v1), np.linalg.norm(v2))
    xs = origin[0] + r * np.cos(ts)
    ys = origin[1] + r * np.sin(ts)
    ax.plot(xs, ys, linewidth=1)

def draw_right_angle_marker(ax, p, dir1, dir2, size=0.4):
    # 在点 p 处画直角符号（几何标记）
    d1 = dir1 / np.linalg.norm(dir1) * size
    d2 = dir2 / np.linalg.norm(dir2) * size
    q1 = p + d1
    q2 = p + d2
    ax.plot([p[0], q1[0]], [p[1], q1[1]], linewidth=1)
    ax.plot([p[0], q2[0]], [p[1], q2[1]], linewidth=1)
    ax.plot([q1[0], q1[0]+d2[0]], [q1[1], q1[1]+d2[1]], linewidth=1)

# 实验向量
u = np.array([2.0, 1.0])
v = np.array([-1.0, 2.0])
w = np.array([3.0, 4.0])
```

(1) 向量长度与夹角

考察点：范数、内积、夹角公式

- 计算 $\|u\|$, $\|v\|$
- 计算 $\langle u, v \rangle$
- 根据 $\cos\theta = \dfrac{\langle u, v \rangle}{\|u\|\|v\|}$ 求夹角 θ（弧度与角度），并判断锐角/直角/钝角

提示： 可使用 `np.linalg.norm`、`np.dot`、`np.arccos`、`np.degrees`

In [ ]:
```
norm_u = np.linalg.norm(u)
norm_v = np.linalg.norm(v)

# 计算内积
dot_uv = np.dot(u, v)

# 计算夹角（弧度与角度）
if norm_u * norm_v == 0:
    cos_theta = 0.0
else:
    cos_theta = dot_uv / (norm_u * norm_v)
cos_theta = np.clip(cos_theta, -1.0, 1.0)  # 处理数值误差
theta_rad = np.arccos(cos_theta)
theta_deg = np.degrees(theta_rad)

if np.isclose(dot_uv, 0):
    angle_type = "直角"
elif cos_theta > 0:
    angle_type = "锐角"
else:
    angle_type = "钝角"

print(f"||u|| = {norm_u:.4f}, ||v|| = {norm_v:.4f}")
print(f"<u, v> = {dot_uv:.4f}")
print(f"夹角（弧度）: {theta_rad:.4f}, 夹角（角度）: {theta_deg:.4f}°")
print(f"角度类型: {angle_type}")
```

```
||u|| = 2.2361, ||v|| = 2.2361
<u, v> = 0.0000
夹角（弧度）: 1.5708, 夹角（角度）: 90.0000°
角度类型: 直角
```

(2) 由 Gram–Schmidt 构造正交基

考察点：标准正交基、正交化、可验证性

对 $\{u, v\}$ 应用 Gram–Schmidt：

- $e_1 = \dfrac{u}{\|u\|}$
- $v_\perp = v - \mathrm{Proj}_u(v) = v - \dfrac{\langle v, u\rangle}{\langle u, u\rangle} u$
- $e_2 = \dfrac{v_\perp}{\|v_\perp\|}$

需要验证 $\langle e_1, e_2\rangle = 0$。

提示： 可使用 `np.allclose(x, 0.0)` 验证近似为零的正交性（考虑浮点误差）

In [ ]:
```
# 构造单位向量 e1（u 的单位化）
norm_u = np.linalg.norm(u)
e1 = u / norm_u

# 计算 v 在 u 上的投影向量
proj_coeff = np.dot(v, u) / np.dot(u, u)  # 投影系数
proj_v_on_u = proj_coeff * u  # 投影向量


v_perp = v - proj_v_on_u
norm_v_perp = np.linalg.norm(v_perp)
e2 = v_perp / norm_v_perp

print("e1 =", e1)
print("e2 =", e2)

dot_e1e2 = np.dot(e1, e2)
print("<e1, e2> =", dot_e1e2)
print("Orthogonal? ->", np.allclose(dot_e1e2, 0.0))
```

```
e1 = [0.89442719 0.4472136 ]
e2 = [-0.4472136   0.89442719]
<e1, e2> = 0.0
Orthogonal? -> True
```

(3) 投影与几何解释（含整合图与标注）

考察点：正交投影公式、残差正交性、可视化解释（文字+几何标记）

- 计算 $p = \mathrm{Proj}_u(w) = \dfrac{\langle w, u\rangle}{\langle u, u\rangle} u$
- 计算残差 $r = w - p$，并证明（数值验证）$\langle r, u\rangle = 0$

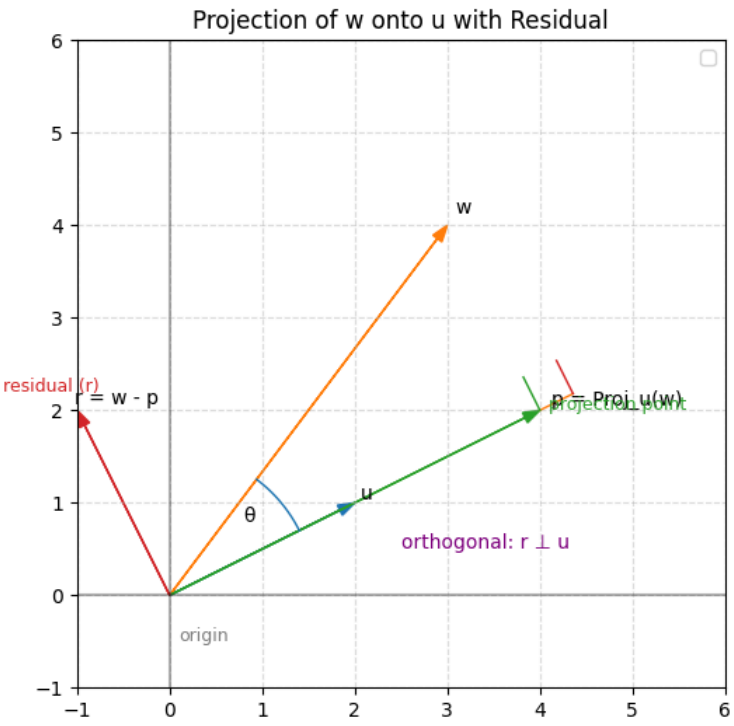
绘图要求（整合+标注）：

- 在同一坐标平面上画出 u, w, p 三个箭头
- 在图中以文字标注 "projection / residual / orthogonal"等关键字
- 画出表示 u 与 w 夹角的弧线
- 在投影点 p 处画直角符号标记 $\angle(r, u) = 90^\circ$

In [16]:
```
# ===== 计算投影与残差 =====
# 计算 w 在 u 上的正交投影 p
dot_wu = np.dot(w, u)  # <w, u>
dot_uu = np.dot(u, u)  # <u, u>
p = (dot_wu / dot_uu) * u  # 投影公式

# 计算残差 r
r = w - p

print("Proj_u(w) =", p)
print("Residual r =", r)

# 验证残差与 u 的正交性
dot_ru = np.dot(r, u)
print("<r, u> =", dot_ru, " -> orthogonal?", np.allclose(dot_ru, 0.0))

# ===== 可视化（整合一张图 + 文本说明 + 几何标记）=====
fig, ax = plt.subplots(figsize=(6,6))

# 坐标范围调整
lims = 6
ax.set_xlim(-1, lims)
ax.set_ylim(-1, lims)
ax.set_aspect('equal', 'box')
ax.grid(True, linestyle='--', alpha=0.4)
ax.axhline(0, color='k', alpha=0.3)
ax.axvline(0, color='k', alpha=0.3)

# 绘制向量箭头
draw_vector(ax, u, color='C0', label='u')
draw_vector(ax, w, color='C1', label='w')
draw_vector(ax, p, color='C2', label='p = Proj_u(w)')
draw_vector(ax, r, color='C3', label='r = w - p')  # 残差向量

# 文本标注关键概念
ax.text(0.1, -0.5, "origin", fontsize=9, color='gray')
ax.text(p[0]+0.1, p[1], "projection point", fontsize=9, color='C2')
ax.text(r[0]-0.8, r[1]+0.2, "residual (r)", fontsize=9, color='C3')
ax.text(2.5, 0.5, "orthogonal: r ⊥ u", fontsize=10, color='purple')

# 绘制 u 与 w 的夹角弧线
draw_angle_arc(ax, origin=np.array([0,0]), v1=u, v2=w)
ax.text(0.8, 0.8, "θ", fontsize=10)  # 标记夹角

# 在投影点 p 处绘制直角符号（标记 r 与 u 垂直）
draw_right_angle_marker(ax, p, dir1=u, dir2=r, size=0.4)

ax.set_title("Projection of w onto u with Residual")
ax.legend(loc='upper right')
plt.show()
```

```
Proj_u(w) = [4. 2.]
Residual r = [-1.  2.]
<r, u> = 0.0  -> orthogonal? True
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_25304\3064303140.py:49: UserWarning: No artists with labels found to put in lege
nd.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
  ax.legend(loc='upper right')
```



(4) 旋转变换（45° 逆时针）

考察点：二维旋转矩阵、长度与内积不变

构造旋转矩阵

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \qquad \theta = 45^\circ.$$

计算 Ru 与 Rv，并验证旋转后仍保持几何结构：

- 长度保持：$\|Ru\| = \|u\|$，$\|Rv\| = \|v\|$
- 内积保持：$\langle Ru, Rv\rangle = \langle u, v\rangle$

提示： `np.cos/np.sin` 需要弧度；用 `np.deg2rad(45)` 进行度→弧度转换。

In [17]:
```
# 构造旋转矩阵 R（ϑ=45°，逆时针旋转）
theta = np.deg2rad(45)  # 转换为弧度
cos_theta = np.cos(theta)
sin_theta = np.sin(theta)
R = np.array([
    [cos_theta, -sin_theta],
    [sin_theta, cos_theta]
])

# 计算旋转后的向量
Ru = R @ u  # 等价于 np.dot(R, u)
Rv = R @ v  # 等价于 np.dot(R, v)

print("R =\n", R)
print("Ru =", Ru)
print("Rv =", Rv)

# 验证长度保持不变
ok_len_u = np.allclose(np.linalg.norm(Ru), np.linalg.norm(u))
ok_len_v = np.allclose(np.linalg.norm(Rv), np.linalg.norm(v))

# 验证内积保持不变
ok_dot = np.allclose(np.dot(Ru, Rv), np.dot(u, v))

print("||Ru|| == ||u|| ?", ok_len_u)
print("||Rv|| == ||v|| ?", ok_len_v)
print("<Ru,Rv> == <u,v> ?", ok_dot)
```

```
R =
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
Ru = [0.70710678 2.12132034]
Rv = [-2.12132034  0.70710678]
||Ru|| == ||u|| ? True
||Rv|| == ||v|| ? True
<Ru,Rv> == <u,v> ? True
```

(5) 几何综合解释（简述）

考察点：整体连贯理解、抽象与统一视角

用一小段话回答下列问题：

> 内积用于度量向量之间的长度与夹角；正交基通过重构坐标系使几何关系更清晰； 投影将向量分解为"沿子空间的部分"与"正交部分"，体现正交分解结构； 旋转属于保持内积结构的等距线性变换，保证长度和夹角不变。
>
> 请说明上述操作如何可以统一理解在"几何结构的保持或重构"这一抽象框架下。

（在此用 4–8 句话作答；建议从"度量结构（内积）→ 正交化（基变换）→ 投影（最短距离/正交分解）→ 旋转（等距变换）"的逻辑串起来。）

这些操作围绕 "几何结构" 的核心 —— 由内积定义的长度与夹角展开：内积是度量基础，为所有几何关系提供量化标准；正交基通过重构坐标系，在不改变内积度量的前提下简化几何关联；投影将向量分解为正交分量，保持残差与子空间的垂直结构；旋转作为等距变换，严格保留长度和夹角。它们或直接保持原有几何结构，或在不破坏核心度量的前提下优化表达，最终统一于 "几何结构的保持或重构" 框架，服务于空间关系的分析。

提交与自检清单

- [√] (1) 数值结果合理，夹角判断有解释
- [√] (2) e_1, e_2 正交且单位化，并用 `np.allclose` 验证通过
- [√] (3) 图中含文字与几何标记（夹角弧线、直角符号），且 `<r,u> \approx 0`
- [√] (4) 验证旋转变换后长度与内积保持不变
- [×] (5) 几何综合解释部分具有抽象性与连贯性（概念之间存在"因果/蕴含/应用"联系）

In [ ]: