数値計算試験問題

2026/07/15 実施

pick_works_from_ans ex26_ans.ipynb -1 '9 12 15' ''

問１：線形代数，行列の４空間（25点）

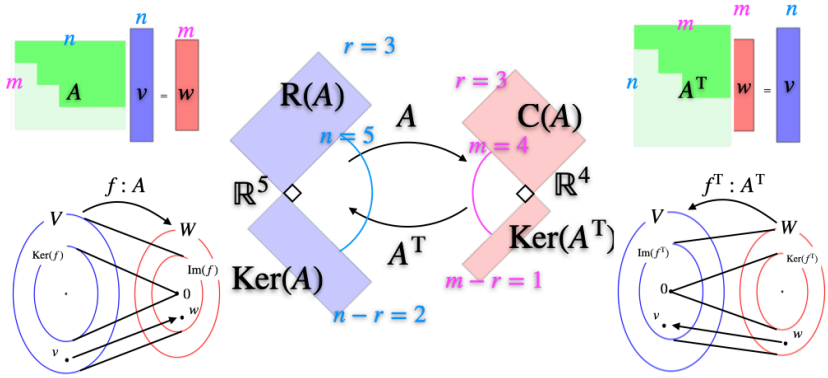
次の行列$A \in \mathbb{R}^{4\times5}$を以下の通り定義する．

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & -1 \end{pmatrix}$$

この行列 A が定義する線形写像を $f : \mathbb{R}^5 \to \mathbb{R}^4$（$f(\boldsymbol{x}) = A\boldsymbol{x}$）とする． 以下の問いに答えよ．

設問：

1. **行空間と列空間**：行列 A の行空間 $\mathrm{R}(A)$ および列空間 $\mathrm{C}(A)$ の基底をそれぞれ求めよ．
2. **像と核**: 写像 f の像 $\mathrm{Im}(f)$ および核 $\mathrm{Ker}(f)$ の次元と基底を求めよ．
3. **直交補空間の検証（行空間と核）**：$\mathbb{R}^5$ において，核 $\mathrm{Ker}(f)$ と行空間 $\mathrm{R}(A)$ が互いに直交補空間であることを，それぞれの基底を用いた内積計算により示せ．
4. **直交補空間の検証（像と左核）**：転置行列 A^{T} が定義する線形写像 $f^{\mathrm{T}} : \mathbb{R}^4 \to \mathbb{R}^5$ (随伴写像と呼ばれる)を考える．このとき，その核 $\mathrm{Ker}(f^{\mathrm{T}})$（左核と呼ばれる）が，写像 f の像 $\mathrm{Im}(f)$ の直交補空間であることを示せ．
5. **基本4空間による全体空間の分割（次元定理）**：上記の設問を参考にして，行列の基本4空間（行空間 $\mathrm{R}(A)$，列空間 $\mathrm{C}(A)$，核 $\mathrm{Ker}(A)$，左核 $\mathrm{Ker}(A^T)$）の包含関係と，それぞれの次元の間の関係を，図１を参考にまとめて説明せよ．



la_core/.../draw_four_spaces.key

図1：行列の基本４空間のイメージ図．

In [1]:
```
import sympy as sp

# 数式の見栄えを良くする設定
sp.init_printing(use_unicode=True)
a1 = sp.Matrix([1, 1, 0, 0])
a2 = sp.Matrix([0, 1, 1, 0])
a3 = sp.Matrix([0, 1, 1, 1])
a1, a2,a3
```

Out[1]:
$$\left(\begin{bmatrix}1\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\1\end{bmatrix}\right)$$

In [2]:
```
A = sp.Matrix.hstack(a1,a2,a3,a3-a2+a1,a1-a3)
A
```

Out[2]:
$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & -1 \end{bmatrix}$$

In [13]:
```
print("【ans 1：解答例】R(A)")
A.rowspace()
```

【ans 1：解答例】R(A)

Out[13]:
$$\left[\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 1 & -1 \end{bmatrix}\right]$$

In [12]:
```
print("【ans 1：解答例】C(A)")
A.columnspace()
```

【ans 1：解答例】C(A)

Out[12]:
$$\left[\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}\right]$$

In [14]:
```
print("【ans 2：解答例】Im f = C(A)")
print("【ans 2：解答例】Ker(A)")
A.nullspace()
```

【ans 2：解答例】Im f = C(A)
【ans 2：解答例】Ker(A)

Out[14]:
$$\left[\begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}\right]$$

In [15]:
```
print("【ans 3：解答例】")
row_basis = A.rowspace()
null_basis = A.nullspace()

print("--- 直交性の検証 (内積計算: r_i · n_j) ---")
for i, r in enumerate(row_basis):
    for j, n in enumerate(null_basis):
        # 内積の計算
        dot_product = r.dot(n)

        # 式の表示
        print(f"\n[行空間基底 r{i+1}] · [核の基底 n{j+1}] = {dot_product}")
```

【ans 3：解答例】
--- 直交性の検証 (内積計算: r_i · n_j) ---

[行空間基底 r1] · [核の基底 n1] = 0

[行空間基底 r1] · [核の基底 n2] = 0

[行空間基底 r2] · [核の基底 n1] = 0

[行空間基底 r2] · [核の基底 n2] = 0

[行空間基底 r3] · [核の基底 n1] = 0

[行空間基底 r3] · [核の基底 n2] = 0

In [17]:
```
print("【ans 4：解答例】")
col_basis = A.columnspace()
left_null_basis = A.transpose().nullspace()

print("--- 直交性の検証 (内積計算: c_i · ln_j) ---")
for i, c in enumerate(col_basis):
    for j, ln in enumerate(left_null_basis):
        # 内積の計算
        dot_product = c.dot(ln)

        # 式の表示
        print(f"\n[列空間基底 c{i+1}] · [左核の基底 ln{j+1}] = {dot_product}")
```

```
【ans 4：解答例】
--- 直交性の検証（内積計算: c_i · ln_j）---

[列空間基底 c1] · [左核の基底 ln1] = 0

[列空間基底 c2] · [左核の基底 ln1] = 0

[列空間基底 c3] · [左核の基底 ln1] = 0
```

【ans 5：解答例】

1. 直交補空間による全体空間の分割と次元の関係
   - 定義域 $V = \mathbb{R}^5$ において，核 $\mathrm{Ker}(f)$ と行空間 $\mathrm{R}(A)$ は互いに直交補空間の関係にある．それぞれの次元の和は $2 + 3 = 5$ となり，空間全体の次元に一致する．
   - 値域 $W = \mathbb{R}^4$ において，左核 $\mathrm{Ker}(f^{\mathrm{T}})$ と列空間 $\mathrm{C}(A)$ は互いに直交補空間の関係にある．それぞれの次元の和は $1 + 3 = 4$ となり，空間全体の次元に一致する．
   - また，$\mathrm{R}(A)$ と $\mathrm{C}(A)$ の次元はともに行列 A の階数（$\mathrm{rank} = 3$）に等しい．
2. 写像による対応関係
   - ゼロへの写像: 核 $\mathrm{Ker}(f)$ は写像 f によって W の原点へ写され，左核 $\mathrm{Ker}(f^{\mathrm{T}})$ は随伴写像 f^{T} によって V の原点へ写される．
   - 空間同士の写像: 写像 f は $\mathrm{R}(A)$ から $\mathrm{C}(A)$ への全単射として働き，逆に随伴写像 f^{T} は $\mathrm{C}(A)$ を $\mathrm{R}(A)$ へ写像する．

問２：数値積分，梁の変位（25点）

長さ $L = 6$ [m] の梁全体に，一定の荷重 $w = 3$ [kN/m]（等分布荷重）がかかっている状況を考える．

このとき，梁の各位置 x における「曲げモーメント」$M(x)$ は，力学の平衡条件より以下の二次関数として導かれる．

$$M(x) = \frac{w}{2}(Lx - x^2)$$

この梁の「端点における傾き（たわみ角）」$\theta(L)$ は，以下の積分値として定義される．

$$\theta(L) = \int_0^L \frac{M(x)}{EI} dx$$

ここで，E は材料の定数，I は断面形状の定数であり，簡単のため $\frac{1}{EI} = 1$ として計算を行う．この積分値 $\theta(L)$ は，梁の端点 $x = L$ における接線の傾き（数学的な dy/dx）に相当する物理量である．なお，この積分の理論的な真値は，解析的に計算すると $\theta(L) = \frac{w}{2EI}\left[\frac{Lx^2}{2} - \frac{x^3}{3}\right]_0^L = \frac{wL^3}{12} = 54$ となる．

設問：

1. 区間 $[0, L]$ を N 等分し，刻み幅 $h = L/N$ を用いて，中点公式および台形公式により $\theta(L)$ を近似計算するプログラムを作成せよ．
2. $N = 10, 20, 40, 80, 160$ の各ケースについて，プログラムによる計算値と真値（54）との絶対誤差を求めよ．
3. 得られた誤差を両対数グラフ（Log-Log Plot）にプロットせよ．中点公式と台形公式の誤差の大きさを比較し，なぜ中点公式の方が台形公式よりも誤差が小さくなるのかについて，積分の幾何学的な性質（曲線の凸性や誤差の打ち消し効果）の観点から考察せよ．

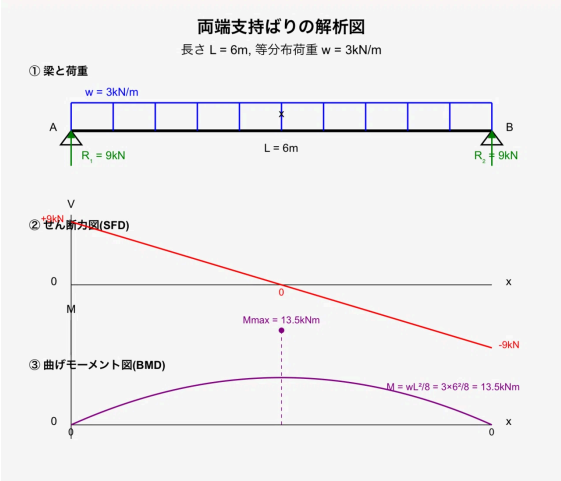


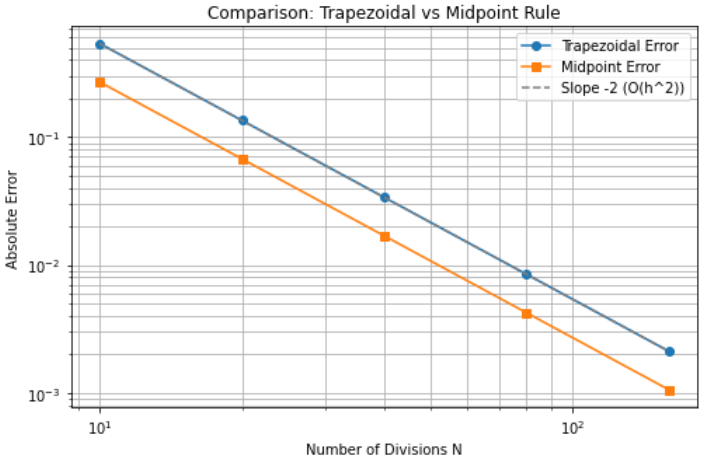
図2：梁の模式図，出典：１００日でわかる二級建築士，構造５４日目，くじらAI研究所｜建築

補足:

現実のH型鋼(H-300×150×6.5×9)では，全体の荷重は1.8t分，ヤング率 E: 205 [GPa] $= 2.05 \times 10^8$ [kN/m^2]，断面二次モーメント I: 6.5×10^{-5} [m^4]とすると 曲げ剛性 EI: $E \times I \approx 13,325$ [kN·m^2]となり，端点のたわみ角は，

$$\theta(L) = \frac{wL^3}{24EI} = \frac{3 \times 6^3}{24 \times 13,325} \approx 0.00202 \text{ [rad]} (\approx 0.116 \text{ 度})$$

In [8]:
```python
import numpy as np
import matplotlib.pyplot as plt

# 定数設定
L = 6.0
w = 3.0
true_value = 54.0

# モーメント関数
def f(x): return (w / 2.0) * (L * x - x**2)

# 1. 台形公式
def trapezoidal(N):
    x = np.linspace(0, L, N+1)
    y = f(x)
    h = L / N
    return (h / 2.0) * (y[0] + 2.0 * np.sum(y[1:N]) + y[N])

# 2. 中点公式
def midpoint(N):
    h = L / N
    x_mid = np.linspace(h/2, L - h/2, N)
    return h * np.sum(f(x_mid))

# 3. 誤差の算出
N_list = [10, 20, 40, 80, 160]
err_trap = [abs(trapezoidal(n) - true_value) for n in N_list]
err_mid = [abs(midpoint(n) - true_value) for n in N_list]

# 4. グラフの描画
plt.figure(figsize=(8, 5))
plt.loglog(N_list, err_trap, 'o-', label='Trapezoidal Error')
plt.loglog(N_list, err_mid, 's-', label='Midpoint Error')

# 理論的な傾き -2 を示す線
plt.loglog(N_list, [err_trap[0] * (n/N_list[0])**-2 for n in N_list], '--', color='gray', label='Slope -2 (O(h^

plt.xlabel('Number of Divisions N')
plt.ylabel('Absolute Error')
plt.title('Comparison: Trapezoidal vs Midpoint Rule')
plt.legend()
plt.grid(True, which="both", ls="-")
plt.show()
```



【ans 3：解答例】

中点法での結果が，台形法での結果よりも常に高い精度となる．中点を代表値にすると前後の領域で正負のキャンセルが期待できるが，この問題では被積分関数が常に凸であるため，台形では常に小さい領域になるのがこの原因と考えられる．

問３：常微分方程式，ロボットアームの振動制御（25点）

機械の制御は，常微分方程式を解くという意味で電気回路のRLCシミュレーションと等価です．

RLC直列回路における電荷 Q の時間変化は，キルヒホッフの法則より以下の微分方程式で記述されます．

$$L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{1}{C_{cap}}Q = V(t)$$

（ここで C_{cap} は静電容量を表す．テキストでは各項の順序が違うので注意）

この電気的な挙動は，機械系の振動（ロボットアームの関節など）と数学的に全く同じ形をしています．物理量は以下のように対応します．

表1：電気系と機械系の物理的対応関係

電気系 (RLC回路)	機械系 (ロボットアーム)	物理的な意味	記憶のkey
電圧 V	駆動トルク τ	システムを動かす外力	入力
インダクタンス L	慣性モーメント M	動きを維持する性質	慣性
抵抗 R	減衰係数 C	動きを妨げる性質	エネルギーの散逸
静電容量の逆数 $1/C_{cap}$	回転剛性 K	位置へ戻ろうとする性質	復元力
電荷 Q	回転角度 θ	システムの状態量	位置・変位
電流 $I = \frac{dQ}{dt}$	角速度 $\omega = \frac{d\theta}{dt}$	状態量の時間変化率	速度

この対応関係に基づき，ロボットアームの関節挙動をバネ・マス・ダンパ系（二次遅れ系）としてモデル化します． 関節の回転角度 θ は，駆動トルク τ に対して以下の二次微分方程式で記述されます．

$$M\frac{d^2\theta}{dt^2} + C\frac{d\theta}{dt} + K\theta = \tau$$

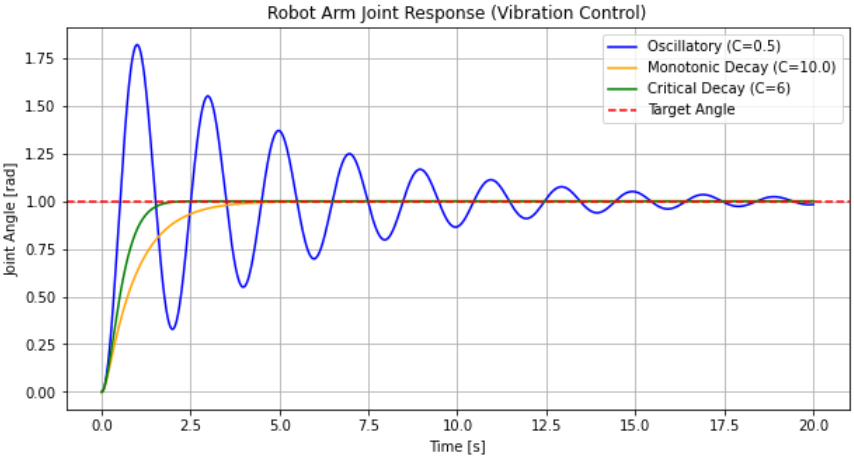
ここで，$M = 1.0$，$K = 10.0$ とし，C は関節の減衰係数を表します．

設問：

1. 授業で扱ったRLC回路の数値解法と同様の考え方（状態変数 $\omega = \frac{d\theta}{dt}$ の導入）を用い，トルク $\tau = 10$ を与えた際の角度 $\theta(t)$ を計算するプログラムを作成せよ．
2. 減衰係数 C を 0.5（不足減衰）とした場合と，10.0（過減衰）とした場合の二通りについて計算し，結果を一つのグラフにプロットせよ．グラフから「振動的な挙動」と「単調な収束挙動」の違いを観察せよ．
3. 振動が消失する境界となる「臨界減衰」の C の数値を適当に変えて，小数点以下0桁で求めよ．M と K を用いて理論的に導出しても良い． $C = 0.5, 10$ と比較して定性的な特徴を記せ．

In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt

# ロボットアーム関節モデル(バネ・マス・ダンパ系)
def euler_robot_arm(theta, omega, tau, C_val):
    M = 1.0   # 慣性
    K = 10.0  # 剛性
    # 加速度 d_omega/dt = (tau - C*omega - K*theta) / M
    d_omega = (tau - C_val * omega - K * theta) / M
    # 速度 d_theta/dt = omega
    d_theta = omega

    # オイラー法による更新
    new_theta = theta + d_theta * dt
    new_omega = omega + d_omega * dt
    return new_theta, new_omega

# シミュレーション設定
dt = 0.01
tt = np.arange(0, 20 + dt, dt)
tau = 10.0 # ステップトルク入力

# ケース1: 振動的 (C = 0.5)
theta_osc = np.zeros(len(tt))
omega_osc = np.zeros(len(tt))
C_osc = 0.5

# ケース2: 単調減衰 (C = 10.0)
theta_dec = np.zeros(len(tt))
omega_dec = np.zeros(len(tt))
C_dec = 10.0

# ケース3:  臨界減衰
theta_cri = np.zeros(len(tt))
omega_cri = np.zeros(len(tt))
C_cri = 6


# シミュレーション実行
for i in range(len(tt)-1):
    theta_osc[i+1], omega_osc[i+1] = euler_robot_arm(theta_osc[i], omega_osc[i], tau, C_osc)
    theta_dec[i+1], omega_dec[i+1] = euler_robot_arm(theta_dec[i], omega_dec[i], tau, C_dec)
```

```
    theta_cri[i+1], omega_cri[i+1] = euler_robot_arm(theta_cri[i], omega_cri[i], tau, C_cri)

# グラフ描画
plt.figure(figsize=(10, 5))
plt.plot(tt, theta_osc, label=f"Oscillatory (C={C_osc})", color="blue")
plt.plot(tt, theta_dec, label=f"Monotonic Decay (C={C_dec})", color="orange")
plt.plot(tt, theta_cri, label=f"Critical Decay (C={C_cri})", color="green")
plt.axhline(y=tau/10.0, color='r', linestyle='--', label="Target Angle") # 理論的な収束値 tau/K
plt.xlabel('Time [s]')
plt.ylabel('Joint Angle [rad]')
plt.title('Robot Arm Joint Response (Vibration Control)')
plt.legend()
plt.grid(True)
plt.show()
```



【ans 3：解答例】

C=6ぐらい，早く静止する.

解析解は，特性方程式

$$M\lambda^2 + C\lambda + K = 0$$

の判別式

$$D = C^2 - 4MK$$

より

$$C_{\text{cri}} = 2\sqrt{MK} = 2\sqrt{10} \approx 6.32$$

問４：MCMC，正方形の詰め込み問題（25点）

本問では，マルコフ連鎖モンテカルロ（MCMC）法の一種であるsimulated anneaing法(模擬焼き鈍し法)を用いて，レイアウト最適化を考えます.

問題を具体化するため，8cm×8cmの正方形領域内に，与えられた8個の正方形（面積の合計が60）

```
sizes = np.array([4.0, 4.0, 3.0, 3.0, 2.0, 2.0, 1.0, 1.0])
```

を配置するとします.

設問：

1. 配置された各正方形の重なり面積を「エネルギー（コスト関数）」として定義し，simulated anneaing法(模擬焼き鈍し法)を用いて重なりが最小になるような配置を探索するプログラムを作成せよ.
2. 実行結果として「初期配置」「エネルギー推移」「最適化後の配置」の3点を可視化せよ.

図３にランダムな初期配置とエネルギー低下の様子，そしてこのシミュレーションでの最適配置を示しています．この図に示した通り，パラメータの設定によっては，間違った最適配置となる場合がありますが，それでも答案として認めます．また，あまりややこしい試行プロセスを作成する必要はありません．SAの良さは実装のしやすさによる，開発時間の短縮にありますので.

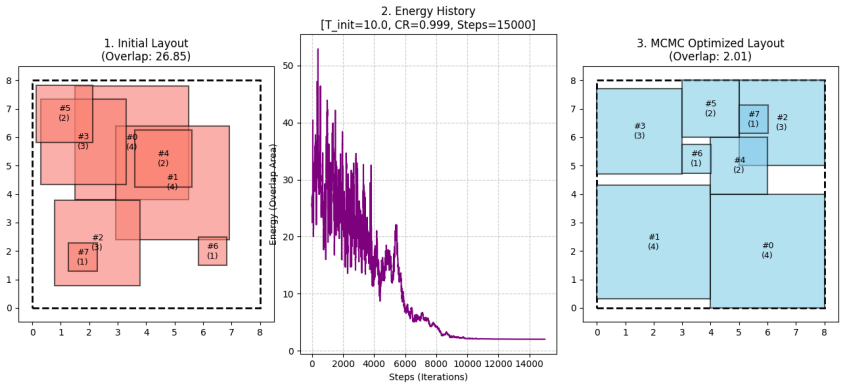


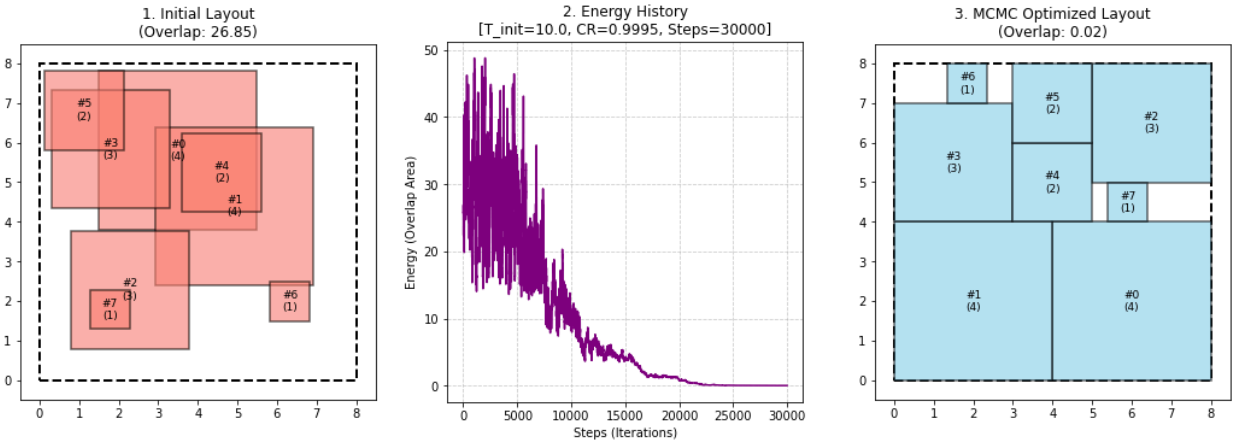
図3：ランダムな初期配置，エネルギー低下の様子と，このシミュレーションでの最適配置．

In [10]:

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

# ------------------------------------------------------------
# 1. 問題の設定と初期化
# ------------------------------------------------------------
np.random.seed(42)

# 正方形の要素(合計面積: 60)
sizes = np.array([4.0, 4.0, 3.0, 3.0, 2.0, 2.0, 1.0, 1.0])
N = len(sizes)

# 外枠の一辺
L = 8.0

# 初期配置(枠内からはみ出さないように初期化)
positions = np.array([np.random.uniform(0, L - s, 2) for s in sizes])
initial_positions = positions.copy()

# ------------------------------------------------------------
# 2. コスト関数(エネルギー関数)の定義
# ------------------------------------------------------------
def calc_overlap(x1, y1, s1, x2, y2, s2):
    """2つの正方形が重なっている面積を計算"""
    dx = min(x1 + s1, x2 + s2) - max(x1, x2)
    dy = min(y1 + s1, y2 + s2) - max(y1, y2)
    if dx > 0 and dy > 0:
        return dx * dy
    return 0.0

def compute_energy(pos, sizes):
    """全体のエネルギー（要素同士の重なり面積）を計算"""
    energy = 0.0
    for i in range(N):
        for j in range(i + 1, N):
            energy += calc_overlap(pos[i, 0], pos[i, 1], sizes[i], pos[j, 0], pos[j, 1], sizes[j])
    return energy

# ------------------------------------------------------------
# 3. MCMC / 模擬焼き鈍し法(SA)の実行とパラメータ設定
# ------------------------------------------------------------
init_T = 10.0 #10.0          # 初期温度
cooling_rate = 0.9995 #0.999   # 冷却率
steps = 30000 #15000          # ステップ数

T = init_T
current_pos = positions.copy()
current_energy = compute_energy(current_pos, sizes)

best_pos = current_pos.copy()
best_energy = current_energy

energy_history = []

for step in range(steps):
    i = np.random.randint(0, N)
    old_pos = current_pos[i].copy()

    # 現在の温度に応じたランダムな移動
    move = np.random.normal(0, 0.4 * (T + 0.05), 2)
    new_p = old_pos + move
    s = sizes[i]

    # 【エラー修正】インデックス[0], [1]でX軸・Y軸を確実に指定してはみ出し判定
    if (new_p[0] < 0) or (new_p[0] + s > L) or (new_p[1] < 0) or (new_p[1] + s > L):
        energy_history.append(current_energy)
```

```
        continue

    current_pos[i] = new_p
    new_energy = compute_energy(current_pos, sizes)
    dE = new_energy - current_energy

    if dE < 0 or np.random.uniform(0, 1) < np.exp(-dE / (T + 1e-6)):
        current_energy = new_energy
        if current_energy < best_energy:
            best_energy = current_energy
            best_pos = current_pos.copy()
    else:
        current_pos[i] = old_pos

    energy_history.append(current_energy)
    T *= cooling_rate

print(f"初期重なりエネルギー: {compute_energy(initial_positions, sizes):.4f}")
print(f"最終重なりエネルギー: {best_energy:.4f}")

# ---------------------------------------------------------
# 4. 結果の可視化(レイアウトと比率の修正)
# ---------------------------------------------------------
# 横並び3マスのグラフ(被らないように全体の横幅を 18 から 20 に拡大)
#fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 6))
fig, (ax1, ax2, ax3) = plt.subplots(
    1, 3,
    figsize=(18, 5.5),
    gridspec_kw={'width_ratios': [1, 1, 1]} # 真ん中のグラフの横幅を少し狭くして余白を作る
)



def plot_packing(ax, pos, title_text, color, energy_val):
    ax.set_xlim(-0.5, L + 0.5)
    ax.set_ylim(-0.5, L + 0.5)
    ax.add_patch(plt.Rectangle((0, 0), L, L, edgecolor='black', facecolor='none', lw=2, linestyle='--'))
    for i in range(N):
        x, y = pos[i]
        s = sizes[i]
        rect = plt.Rectangle((x, y), s, s, edgecolor='black', facecolor=color, alpha=0.6, lw=1.5)
        ax.add_patch(rect)
        ax.text(x + s/2, y + s/2, f"#{i}\n({int(s)})", ha='center', va='center', fontsize=9)
    ax.set_aspect('equal', adjustable='box')
    ax.set_title(f"{title_text}\n(Overlap: {energy_val:.2f})")

# 左側:初期配置
plot_packing(ax1, initial_positions, "1. Initial Layout", "salmon", compute_energy(initial_positions, sizes))

# 中央:エネルギー推移(アスペクト比を自動調整にして被りを防止)
ax2.plot(energy_history, color='purple', lw=1.5)
ax2.set_xlabel("Steps (Iterations)")
ax2.set_ylabel("Energy (Overlap Area)")
# タイトル部分に全てのパラメータ(T, CR, Steps)を表示
ax2.set_title(f"2. Energy History\n[T_init={init_T}, CR={cooling_rate}, Steps={steps}]")
ax2.grid(True, linestyle='--', alpha=0.6)

# 右側:MCMC実行後の最適化配置
plot_packing(ax3, best_pos, "3. MCMC Optimized Layout", "skyblue", best_energy)

# 各グラフ間の余白をきれいに調整
'''

plt.subplots_adjust(wspace=0.1) # 横の隙間を広げる(デフォルトは0.2程度)
plt.show()
'''

#plt.tight_layout()
plt.show()

print("cooling rateと試行回数を調整するとこんな単純なSAでも最適解が得られる. ")
```

```
初期重なりエネルギー: 26.8521
最終重なりエネルギー: 0.0237
```



cooling rateと試行回数を調整するとこんな単純なSAでも最適解が得られる.

In [ ]: