カハンの加算アルゴリズムの実装と考察

数値計算法 第一回レポート

hinshiba　瀕死

出題日 2025 年 10 月 02 日

提出日 2025 年 12 月 09 日

1. 概要

本レポートでは計算機上での加算の際に発生しうる情報落ちによる誤差を軽減する手法の 1 つであるカハンの加算アルゴリズムについて実装し，原理を考察，他のアルゴリズムとの比較を行う．

2. 実装

今回作成したカハンの加算アルゴリズムの Python による実装をソースコード 1 に示す．なお，実行可能なプログラムは GoogleForm にて別途提出してある．

```python
def kahan_add(vals: NDArray[np.float32]) -> np.float32:
    ans = np.float32(0)
    comp: np.float32 = np.float32(0)
    for x in vals:
        x -= comp
        temp: np.float32 = ans + x
        comp = (temp - ans) - x
        ans = temp
    return ans
```

ソースコード 1: Python による実装

6 行目の，変数 temp にいままでの総和 ans と入力 x を加算する操作を繰り返し行うことで加算が達成される． このとき，加算後の temp はには情報落ちの誤差等が含まれている可能性がある．これを 7 行目によって求め，変数 comp に保存する． そして，次回の加算前に，comp を考慮した値に入力 x を調整するという実装である．

3. 結果

ここでは非常に簡単な実行結果のみとし，詳細な考察と比較は考察で行う．

ソースコード 2 をソースコード 1 の定義の下で実行した結果をテキスト 1 に示す．

この結果から，カハンの加算アルゴリズムでは 2 + eps 時の誤差を補正して，最終的にはより近い解を得られていることがわかる．

```python
# マシンイプシロンの近似値
eps = np.finfo(np.float32).eps

print(f"{np.float32(2) + eps + eps =}")
print(f"{kahan_add(np.array([2, eps, eps], dtype=np.float32))=}")
```

ソースコード 2: 実行例

4. 考察

本章ではカハンの加算アルゴリズムを含めた 3 つのアルゴリズムを比較し，誤差の評価と，どのような点が優位であるのかについて考察する． 考察するアルゴリズムは入力を順番に足すもっとも単純なもの，応用解析の講義において説明された絶対値が小さい値から足していくもの，カハンの加算アルゴリズムの 3 つである．

4.1. 理論的説明

設定として，$x_1, x_2 ... x_{n+1} (x_i \geq 0)$の総和を求めるものとし，加算前には誤差を持っていなかったものとする．また，減算における誤差は小さい[1] ので誤差は常に無視できるものとする．

以降において，i回目の加算で生じる相対誤差をε_i，変数 comp の値をc_i，誤差を含むx_{i+1}までの総和をS_iとする．

まず，1 回目は0との加算で誤差が発生しないものとし，

$$S_1 = x_1 + 0, \quad \varepsilon_i = 0, \quad c_1 = 0$$

$i (i > 1)$回目では，

$$\begin{aligned} S_i &= \{(x_i - c_{i-1}) + S_{i-1}\}(1 + \varepsilon_i) \\ &= x_i - c_{i-1} + S_{i-1} + \varepsilon_i(x_i - c_{i-1} + S_{i-1}) \\ c_i &= S_i - S_{i-1} - (x_i - c_{i-1}) \\ &= \varepsilon_i(x_i - c_{i-1} + S_{i-1}) \end{aligned}$$

また，

$$\begin{aligned} S_i &= x_i - c_{i-1} + S_{i-1} + \varepsilon_i(x_i - c_{i-1} + S_{i-1}) \\ &= x_i - c_{i-1} + x_{i-1} - c_{i-2} + S_{i-2} + \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2}) \\ &+\varepsilon_i\{x_i - c_{i-1} + x_{i-1} - c_{i-2} + S_{i-2} + \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2})\} \\ c_{i-1} &= \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2}) \end{aligned}$$

より，

$$\begin{aligned} S_i &= x_i - \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2}) + x_{i-1} - c_{i-2} + S_{i-2} + \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2}) \\ &+\varepsilon_i\{x_i - \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2}) + x_{i-1} - c_{i-2} + S_{i-2} + \varepsilon_{i-1}(x_{i-1} - c_{i-2} + S_{i-2})\} \\ &= x_i + x_{i-1} - c_{i-2} + S_{i-2} + \varepsilon_i\{x_i + x_{i-1} - c_{i-2} + S_{i-2}\} \end{aligned}$$

よって，この操作を繰り返すと，

$$S_i = x_i + x_{i-1} + ... + x_1 - c_0 + \varepsilon_i\{x_i + x_{i-1} ... x_1 - c_0\}$$

となる． したがって，このアルゴリズムにおいては，加算の回数に誤差が依存しないことがわかる．

```
np.float32(2) + eps + eps =np.float32(2.0)
kahan_add(np.array([2, eps, eps], dtype=np.float32))=np.float32(2.0000002)
```

テキスト 1: 実行結果

4.2. 他のアルゴリズムとの比較

比較として絶対値が小さい値から足す場合や，単純に足す場合について考える．

1 回目は0との加算で誤差が発生しないものとし，

$$S_1 = x_1 + 0, \quad \varepsilon_i = 0$$

$i(i > 1)$回目では，

$$\begin{aligned} S_i &= (x_i + S_{i-1})(1 + \varepsilon_i) \\ &= x_i + S_{i-1} + \varepsilon_i(x_i + S_{i-1}) \\ &= \{x_i + (x_{i-1} + S_{i-2})(1 + \varepsilon_{i-1})\} + \varepsilon_i\{x_i + (x_{i-1} + S_{i-2})(1 + \varepsilon_{i-1})\} \\ &= x_i + x_{i-1} + S_{i-2} + \varepsilon_{i-1}\{x_i + x_{i-1} + S_{i-2}\} + \\ &\varepsilon_{i\{x_i + x_{i-1} + S_{i-2}\}} + \varepsilon_i\varepsilon_{i-1}\{x_i + x_{i-1} + S_{i-2}\} \\ &\cdots \end{aligned}$$

ここで，$\varepsilon_i \ll 1$とすると，$\varepsilon_i\varepsilon_j$は無視できるほど小さい． このとき，上式より 1 回加算するごとにε_iの積となる項が 1 つづつ増えるので，これら誤差が互いに打ち消しあわないとすると，概ねnの増え方に比例して誤差も増えていくこととなる．

絶対値が小さい順に足すとすると，それぞれのε_iは単純に足すより小さくなると考えられるが，こちらもまた概ねnの増え方に比例して誤差も増えていくこととなる．また，比較を用いたソートの過程で，最低でも$O(n \log n)$の時間計算量を持つことになる[2] ため計算量が悪化する．

したがって，このアルゴリズムにおいては，加算の回数に誤差が依存しないことがわかる．

ここまでの比較を表 1 にまとめる．

アルゴリズム	**誤差の増え方**	**時間計算量**
単純な加算	nに比例	$O(n)$
絶対値の低い順に加算	nに比例	$O(n \log n)$
カハンの加算アルゴリズム	一定	$O(n)$

表 1: アルゴリズムの比較

このことから，カハンの加算アルゴリズムは時間計算量を単純な加算からそのままに，誤差を大きく減らすことのできるアルゴリズムであるということがわかった．

参考文献

[1] Kahan, W., Pracniques: further remarks on reducing truncation errors, Commun. ACM (1965), Vol. 8, pp. 40–41

[2] 浅野 哲夫, 和田 幸一, 増澤 利光, アルゴリズム論, オーム社 (2003), p. 100