## ۵.۱ انتشار خطا

وقتی خطایی در یک محاسبه رخ می‌دهد این خطا نتایج بعدی را نیز تحت تاثیر قرار خواهد داد. خوب است بدانیم که خطای ناشی از یک عمل محاسباتی چگونه در مراحل بعدی محاسبات منتشر می‌شود. بعنوان مثال فرض کنید $a$، $b$ و $c$ سه عدد ماشین بوده و بخواهیم حاصل $y = a+b+c$ را با استفاده از دو رابطه‌ی متفاوت

$$\tilde{y}_1 := (a \oplus b) \oplus c$$

و

$$\tilde{y}_2 := a \oplus (b \oplus c)$$

یافته و تفاوت احتمالی نوع انتشار خطا را با تعیین میزان خطای نسبی موجود در $\tilde{y}_1$ و $\tilde{y}_2$ بعنوان تقریب‌هایی از $y$ بیابیم. در واقع میزان خطای نسبی

$$e_{\tilde{y}_1} := |\frac{\tilde{y}_1 - y}{y}|$$

و

$$e_{\tilde{y}_2} := |\frac{\tilde{y}_2 - y}{y}|$$

را تخمین خواهیم زد. در اولین مرحله از محاسبه‌ی $\tilde{y}_1$، خطایی در تعیین مقدار $\tilde{x}_1 := a \oplus b$ رخ می‌دهد. داریم:

$$\tilde{x}_1 = (a+b)(1+\delta_1)$$

که در آن $|\delta_1| \leq u$. در مرحله‌ی دوم و پایانی داریم:

$$\tilde{y}_1 := \tilde{x}_1 \oplus c$$

و در نتیجه خطای دوم دیگری در عمل جمع ممیز شناور $\tilde{x}_1$ با $c$ رخ می‌دهد. می‌دانیم که

$$\tilde{y}_1 := (\tilde{x}_1 + c)(1+\delta_2)$$

که در آن $|\delta_2| \leq u$. پس داریم:

$$\begin{aligned}
\tilde{y}_1 &= \Big((a+b)(1+\delta_1)+c\Big)(1+\delta_2) \\
&= \Big((a+b+c)+(a+b)\delta_1\Big)(1+\delta_2) \\
&= (a+b+c)\Big(1+\frac{(a+b)}{(a+b+c)}\delta_1\Big)(1+\delta_2) \\
&= (a+b+c)\Big(1+\delta_2+\frac{(a+b)}{(a+b+c)}\delta_1(1+\delta_2)\Big)
\end{aligned} \tag{۱۰.۱}$$

در نتیجه داریم:

$$\begin{aligned}
e_{\tilde{y}_1} := |\frac{\tilde{y}_1 - y}{y}| &= |\frac{(a+b+c)\Big(1+\delta_2+\frac{(a+b)}{(a+b+c)}\delta_1(1+\delta_2)-1\Big)}{a+b+c}| \\
&= |\delta_2 + \frac{(a+b)}{(a+b+c)}\delta_1(1+\delta_2)|
\end{aligned}$$

چون $\delta_1$ و $\delta_2$ دو کمیت کوچک هستند پس $\delta_1\delta_2$ بسیار کوچک بوده و با نادیده گرفتن آن تخمین زیر را داریم:

$$e_{\tilde{y}_1} \approx |\frac{a+b}{a+b+c}|\ |\delta_1| + (1)|\delta_2|.$$

به طرز مشابه می‌توان نشان داد که

$$e_{\tilde{y}_2} \approx |\frac{b+c}{a+b+c}|\ |\delta_4| + (1)|\delta_3|,$$

که در آن $|\delta_3|, |\delta_4| \leq u$. پس اگر $|a+b|$ از $|b+c|$ کوچک‌تر باشد، خواهیم داشت

$$e_{\tilde{y}_1} < e_{\tilde{y}_2}$$

و در نتیجه انتظار این است که جواب به دست آمده با روش اول یعنی با $(a \oplus b) \oplus c$، خطای کمتری از $a \oplus (b \oplus c)$ داشته باشد.

نتیجه‌ی جالب و مهم بحث بالا این که از دیدگاه ((میزان درستی))، برای محاسبه‌ی مجموعی از اعداد در حساب ممیز شناور، بهتر است ترتیب انجام عملیاتِ جمع به صورتی باشد که اندازه‌ی مجموع‌های جزئی،

کوچک‌تر باشد! چنانچه اعدادی که می‌خواهیم مجموعشان را محاسبه کنیم، همگی هم‌علامت باشند (و نه لزوما به طور کلی)، این قاعده بدان معناست که بهتر است ابتدا اعداد را به صورت صعودی مرتب کرده و سپس اعداد کوچک‌تر را اول با هم جمع کنیم تا مجموع‌های جزئی کوچک‌تر بمانند. در تمرین ۲، اندازه‌ی کدام‌یک از مجموع‌های جزییِ $a+b$ و $b+c$ کوچک‌تر بود؟ کدام‌یک از دو ترتیب، به جواب درست‌تری منجر شد؟

## ۶.۱ پایداری عددی الگوریتم‌ها

دیدیم که می‌توان میزان حساسیت یک مساله‌ی ریاضی به خطاهای گردکردن را با عدد وضعیت آن مساله سنجید. برای حل یک مساله ممکن است الگوریتم‌های مختلفی وجود داشته باشد که برای حل مساله عملیات محاسباتی متفاوتی را در حساب ممیز شناور اجرا می‌کنند. برخی الگوریتم‌ها به خطاهای اندک در ورودی حساسیت بیشتری دارند بدین معنا که اختلالی کوچک در ورودی الگوریتم می‌تواند منجر به اختلالی بزرگ در خروجی (جواب) الگوریتم شود. مفهوم متناظر با عدد وضعیت مساله در مورد الگوریتم‌ها مفهوم پایداری عددی است.

تعریف دقیق پایداری الگوریتم بستگی به زمینه‌ی مورد بحث دارد. بعنوان مثال تعریفِ پایداری در حل عددی معادلات دیفرانسیل معمولی یا با مشتقات جزئی (جایی که نگرانی از خطای گسسته‌سازی معمولا بیش از خطای گردکردن است)، با تعریف آن در مسائل جبرخطی عددی (جایی که خطای گسسته‌سازی یا وجود ندارد و یا خطای گردکردن بر آن غالب است) تفاوت دارد. به زبان غیردقیق یک الگوریتم را به لحاظ عددی پایدار می‌نامیم اگر تمام خطاهایی که در آن رخ می‌دهد بی‌خطر باشد یعنی خروجی (جواب) الگوریتم با اختلال اندک در ورودی آن تنها دچار اختلالی اندک شود.

اگر دو الگوریتم را برای حل یک مساله درنظر بگیریم ممکن است یکی از آن‌ها به لحاظ عددی قابل اعتمادتر باشد و یا هیچیک پایدار عددی نباشند. یکی از مهمترین مسائل در آنالیز عددی پیداکردن الگوریتم‌های پایدار عددی است. یک تکنیک کلی برای بررسی پایداری عددی الگوریتم‌ها (به خصوص در مسائل جبرخطی) تحلیل خطای پسرو [1] است که توسط جیمز ویلکینسون در دهه‌ی شصت میلادی معرفی و شناسانده شد. در این راهکار هر خطای گردکردنی که در یک مرحله از الگوریتم رخ می‌دهد با یک تغییر در داده‌ی ورودی الگوریتم جایگزین شده و سپس اثر این اختلال تجزیه و تحلیل می‌شود. مطالب بیشتری در این خصوص مانند انواع پایداری عددی در دروس پیشرفته‌تر آنالیز عددی و یا در دروس مرتبط با جبرخطی عددی مطالعه می‌شوند. در ادامه تنها کلیاتی را در زمینه‌ی پایداری عددی الگوریتم‌ها روی مثال‌هایی ملموس

[1] backward error analysis

بررسی می‌کنیم.

مثال ۸. فرض کنید آرایه‌ی $a$ از صد عدد داده شده و بخواهیم مجموع اعداد عضو آرایه را روی ماشینی با دستگاه ممیزشناور $F_{10,2}^{-3,3}$ محاسبه کنیم.

قبل از اینکه الگوریتم‌های مختلف برای حل این مساله را بررسی کنیم بهتر است مشخص کنیم چه انتظاری از پاسخ الگوریتم منطقی است یعنی چه الگوریتمی را می‌توان (از الگوریتم‌های مختلفی که قرار است نهایتا روی ماشینی با دستگاه ممیزشناور $F_{10,2}^{-3,3}$ اجرا شوند)، رضایت‌بخش دانست. طبق قضیه‌ی ۲.۳.۱ که تنها برای یک‌بار اجرای عمل جمع برقرار است می‌توان فهمید که چنانچه پاسخ الگوریتمی برای جمع صد عدد ماشین دارای خطای نسبی حداکثر $\varepsilon_M$ باشد باید کاملا خرسند باشیم. در مورد دستگاه اعداد $F_{10,2}^{-3,3}$ داریم $\varepsilon_M = \beta^{-(p-1)} = 10^{-1} = 0.1$ که همان فاصله‌ی بین یک و $1.1$ که عدد ماشین بعدی عضو دستگاه است می‌باشد. پس اگر الگوریتمی بیابیم که وقتی روی این دستگاه ممیزشناور اجرا شد پاسخی با خطای نسبی حداکثر $0.1$ بدست آید راضی خواهیم بود.

ساده‌ترین الگوریتم برای تعیین جواب، اجرای یک حلقه‌ی for به صورت زیر است:

```
sum = 0;
for i=1:100
    sum = sum + a(i);
end
return sum
```

اکنون فرض کنید اولین عضو آرایه‌ی $a$ عدد $1$ بوده و نود و نه عضو دیگرش همگی مساوی $0.01$ باشند به طوری که جواب درست در حساب با دقت بینهایت برابر با $1.99$ باشد.

توجه کنید که $0.01 = 1.0 \times 10^{-2}$ یک عدد ماشین عضو دستگاه ممیزشناور $F_{10,2}^{-3,3}$ بوده و خطای گردکردنی در ذخیره‌سازی آن رخ نمی‌دهد. با این وجود در دستگاه ممیزشناور ما که تنها دو رقم دقت دارد وقتی عدد یک در متغیر sum قرار گرفت، اضافه‌کردن $0.01$ به آن هیچ تاثیری بر مقدار sum نخواهد داشت!

$$1 \oplus 0.01 = fl(1.01) = 1.0$$

چرا که دو عدد ماشین حول $1.01$ عبارتند از $1.0$ و $1.1$ و در سبک پیش‌فرض گردکردن به نزدیک‌ترین، عدد ماشینِ $1.0$ بعنوان جواب برگزیده خواهد شد. همین اتفاق نود و هشت بار دیگر نیز خواهد افتاد و خروجی

الگوریتم $1.0$ خواهد بود. خطای نسبی پاسخ این الگوریتم برابر است با:

$$\frac{|1.99-1|}{|1.99|} \approx 5 \times 10^{-1}$$

که از اپسیلون ماشین بزرگ‌تر است.

از سوی دیگر به یاد داریم که (برخلاف حساب ریاضی با دقت نامتناهی) ترتیب انجام عملیات ممیز شناور می‌تواند روی درستیِ پاسخ، تاثیرگذار باشد. با توجه به بحثی که در مورد انتشار خطا کردیم، می‌توان انتظار داشت که یک الگوریتم پایدار عددی برای حل این مساله، ترتیبی را برای انجام عمل جمع اتخاذ خواهد کرد که مجموع‌های جزئی، کمترین اندازه (قدرمطلق) را داشته باشند. پس پیشنهاد این است که آرایه‌ی $a$ را به صورت صعودی (برحسب قدرمطلق درایه‌هایش) مرتب کنیم تا اعداد کوچک‌تر به ابتدای آرایه منتقل شوند و اعداد بزرگ‌تر به انتها و سپس الگوریتم ساده‌ی قبل را روی آرایه‌ی مرتب شده اجرا کنیم. در اینجا نود و نه درایه‌ی مساوی $0.01$ به ابتدای آرایه‌ی مرتب‌شده‌ی $a$ منتقل می شوند و عدد یک که بزرگترین اندازه را دارد به انتها منتقل می‌شوند. در اولین مرحله اجرای الگوریتم قبل صفر و $0.01$ با هم جمع می‌شوند و پاسخ دقیق $0.01$ در متغیر sum قرار می‌گیرد. سپس به ازای $i=2$ داریم:

$$sum = sum + a(2) = 1.0 \times 10^{-2} \oplus 1.0 \times 10^{-2} = fl(2.0 \times 10^{-2}) = 2.0 \times 10^{-2} = 0.02.$$

شبیه همین محاسبه‌ی بدون خطا تا وقتی $i = 99$ است تکرار شده و مقدار sum قبل از مرحله‌ی پایانی الگوریتم برابر با $0.99 = 9.9 \times 10^{-1}$ که خود یک عدد ماشین است خواهد بود. نهایتا وقتی $i = 100$ شد داریم:

$$sum = sum + a(100) = 0.99 \oplus 1.0 = fl(1.99) = 2.0$$

چرا که دو عدد ماشین حول $1.99$ عبارتند از $1.9$ و $2.0$. واضح است که پاسخ این الگوریتم، تقریبِ بسیار بهتری برای جواب دقیق (در مقایسه با الگوریتم اول) است. خطای نسبی خروجی الگوریتم دوم برابر است با:

$$\frac{|1.99-2|}{|1.99|} \approx 5 \times 10^{-3}$$

که بسیار کوچک‌تر از اپسیلون ماشین است.

توجه کنید که پایداری عددی الگوریتم دوم بی‌هزینه نیست! الگوریتم دوم البته به لحاظ پایداری عددی بهتر از الگوریتم اول است اما هزینه‌ی محاسباتی بیشتری در قیاس با الگوریتم اول دارد چرا که الگوریتم دوم

نیاز به مرتب‌سازی آرایه نیز داشته و در نتیجه حجم عملیات محاسباتی بیشتری داشته و کندتر است. به یاد آورید که در مورد انواع الگوریتم‌های مرتب‌سازی (مانند مرتب‌سازی حبابی، انتخابی، درجی و ...) در اولین درس برنامه‌نویسی کامپیوتری آموخته‌ایم.

در مثال زیر یک الگوریتم ناپایدار عددی دیگر و الگوریتمی برای جایگزینی آن را بررسی می‌کنیم.

مثال ۹. مساله‌ی محاسبه‌ی دو ریشه‌ی چندجمله‌ای

$$p(x) = ax^2 + bx + c$$

را در نظر گرفته و بعنوان اولین الگوریتم برای حل این مساله فرمول دبیرستانی دلتا را درنظر بگیرید:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

اگر $b^2$ خیلی بزرگ‌تر از $|4ac|$ باشد، آنگاه $b^2 - 4ac$ تقریبا با $b^2$ برابر شده و خواهیم داشت $\sqrt{b^2 - 4ac} \approx |b|$. در این موقعیت یکی از دو فرمول $-b + \sqrt{b^2 - 4ac}$ یا $-b - \sqrt{b^2 - 4ac}$ دربرگیرنده‌ی پدیده‌ی حذف بوده و درنتیجه می‌تواند خطای گردکردن ناشی از محاسبه‌ی $fl(\sqrt{b^2 - 4ac})$ را چندین برابر کند.

بعنوان الگوریتم دومی که برای یافتن ریشه‌های چندجمله‌ای $p(x)$ پایدار عددی است، ابتدا یکی از ریشه‌ها را به کمک فرمول

$$x_1 = \frac{-(b + sign(b)\sqrt{b^2 - 4ac})}{2a}$$

که عاری از پدیده‌ی حذف است (دو حالت $b < 0$ و $b \geq 0$ را در این فرمول جداگانه بررسی کنید) می‌یابیم و سپس برای محاسبه‌ی $x_2$ از فرمول

$$x_1 x_2 = \frac{c}{a}$$

استفاده می‌کنیم.

در پایان این فصل توجه کنید که در حالت کلی اگر یک مساله‌ی خوش وضع را با الگوریتمی که به لحاظ عددی پایدار است حل کنیم می‌توانیم به جواب محاسبه‌شده اعتماد کنیم [1]. چنانچه مساله بدوضع باشد یا الگوریتم مورد استفاده ناپایدار عددی باشد نمی‌توان چندان به جواب اعتماد کرد.

[1] میزان خطای جواب محاسبه‌شده توسط الگوریتم‌های پایدار (از نوع خاصی بنام پایدار پسرو) به صورتی دقیق قابل کراندارکردن است.