数式処理 *J.JSSAC (2002)*
Vol. 9, No. 1, pp. 1 - 2

巻頭言

数式処理プログラミングの楽しみ
– 虚構の世界の実学のすすめ –

村尾 裕一

プログラミングは楽しい．正確には，プログラミングを楽しいと感ずる人間が少なからずいる．また，数学が面白いという人も世の中には沢山いる．プログラムを書くことは，コンピュータの中という実体のない世界でのこととは言え，何かを具体化していくことに他ならず，完成した時の充足感はもの作りの喜びそのものである．一方，数学は，事実を抽象的に記述し，我々の思惑では動かし得ないような事実や自然界の摂理を垣間見せてくれる．プログラミングと数学，一見似てはいるが，実は具体化と抽象化という逆方向の知的作業である．この作業，理工系の学生にとっては必須の基礎知識であり，また，かじり始めて少しはあやつれるようになると自分が少し賢くなったような気もしてくる．問題はそこから先である．筆者のようなデキの悪い学生は，抽象的な概念や訳の分からない記号の数々が日常的な言葉として使われるようになると現実が恋しくなるし，はたまた，定積分の値を求めたり整列したりするのに，色んな方法を知らなくてもいいし，そんな面倒なことをやる必要性も感じないからどうでもいいや，ということになる．でも，あの楽しみや面白さは忘れない．

数学をプログラミングの題材にすると両方の楽しみを味わうことができる．対象が数値だと，現実のものが見えて難しい．数式の計算ならば，その意義は数学が説明してくれるとしておけば虚しさを感じずに済む．計算処理の内容と意味は数学で定義・記述されている．あとは，数式をデータとしてどのように表現し，処理の方法をどのように実現するかを考えればよい．正に，プログラミングの楽しみである．一方，コンピュータは数学や式の計算の方法のことなど知る由もないのだから，プログラムにして教え込むことで自分の数学的知識をひけらかしてやればよい．既存の数式処理システム上でアルゴリズムを実装する場合でも，データとしての数式の表現法は決められているのでプログラミングの楽しみは半減するが，コンピュータに知識を与えるという知的快感が得られることに変わりはない．扱う式と計算処理の大部分は中学生の頃から馴染んできたもので，高級な数学の知識がなくとも十分に楽しめる．その計算法を紐解けば，学校で教わってきたアドホックな方法ではなく，はるかに洗練されたアルゴリズムの数々を数式処理や数理アルゴリズムの教科書で知ることとなる．こんな魅力にとりつかれた科学者が世界中に多数いる． 特に，’90 年頃からは， EU が支援する多くの研究開発プロジェクトが始まり，それまで米国中心の幾つかのグループだけで行われていた大規模なソフトウェア開発が，盛んに行われるようになった．各プロジェクトで

は，既存の処理系にとらわれずに独自のソフトウェアを開発し，固有の機能を持ったソフトウェアが数多く誕生した．その形態は，簡単なライブラリ形式のものから，現代的な機能までをも備えた大規模な処理系まで様々である．その結果として，異種間も含めたソフトウェア同志を結ぶための標準的な通信機構が開発され，また，新たな機能を実装するには，従来のように汎用の処理系への機能追加としてではなく，独自のプログラムを開発すればよいという考え方が世界中に広まっていくこととなった．基本的な機能については，必要に応じて既存のプログラムライブラリを利用することも可能である．今日，ソフトは欲しい機能のものを探してきて使うものだという風潮が広まっているが，同じ位簡単に誰もがプログラミングを楽しめる状況になっているのである．

こうした発展の背景には，計算機利用のための様々な技術が一般化したという事実がある。コンピュータサイエンスの立場から数式処理システムを見ると，プログラミングの基本的な題材の宝庫である．リストや木構造で表された数式は，演算によって大きさが変化し，メモリ管理やガーベジコレクションが当然必要になる．計算量的な解析が必要かもしれない．そもそも，数ですら構造を持つような計算処理では，どのように計算量の解析が正当なものなのか?教科書に書かれているような高速アルゴリズムに実用性はあるのか? 数式の入力は構文解析の第一歩である．独立した処理系とするには，利用者向けには使い易い入出力インタフェースも揃えたいし，何より開発者自身にとっては洗練されたデバッグの機能が欲しい．デバッグの最中に，データ構造のままの数式や長大な式の全体を見せられたのでは，デバッグどころではない．また，最近では，他の処理系やソフトウェアとの連携も考慮して設計しておくと，利用可能性も広がる． ... 等々，プログラミングの基礎から応用まですべての題材が揃っている．一方，数学を教え込むという点では，計算手法の実現は数学的にきちんと記述された算法をそのまま実装するのだから簡単かというと，なかなかそうは行かない．複雑な構造で表される式の処理は手間もかかるしメモリも浪費するので，無駄な計算は避けたい．式のデータ構造の一部だけを取り出して使うこともある．結果として複数の式を出力するアルゴリズムでも，必ずしもその全てを使うとは限らない．複雑な計算のアルゴリズムが再帰的な場合，プログラムの開発とデバッグは非常に難しい．何しろ，実装しようというアルゴリズムが，実現すべき機能は既に実現されていることを仮定しているのだから．こうした様々な難題はプログラミング技術の腕の見せ所でもある．簡潔に記述されたアルゴリズムを鵜呑みにして，行儀の良いプログラムを書いていくと，例外的な場合の処理の方が複雑で，プログラム全体を解法の本質を理解した上で設計し直さなければならないということもある．モジュラー算法はその典型で，その簡潔さに魅せられて実装を試みた EZ アルゴリズムは決して easy ではなかったという記憶がある (EZ アルゴリズムという名前の意味をイギリス人は理解してくれなかったと，開発者の一人である D. Yun 氏がこっそり教えてくれたことがある．'Z' はイギリスでは「ゼッド」と読む)．こんな具合に，実装を通してアルゴリズムというものが何なのかを学び，解法の本質を理解することにもなる．

数式処理のプログラミングや処理系の作成は，色々な勉強ができるし，様々な楽しみがある．ソフトウェアは自分で作るものという原点に立ち帰り，是非，この楽しみを自分で味わってみてもらいたい．そうして蓄えていったソフトウェア資産は，新たな研究へと発展させる礎ともなるのだから．

(編集委員 村尾裕一)