A PROBLEMÁTICA DO DESENVOLVIMENTO DE SOFTWARE NUMÉRICO

Tiarajú Asmuz Diverio
Instituto de Informática PUC RS
Cx. P.1429 90620 Porto Alegre RS

RESUMO

Neste trabalho é caracterizado o software numérico (matemático), considerando-se os níveis de atividade no desenvolvimento do software e os aspectos básicos no processo de desenvolvimento do software, bem como características desejáveis em software numérico. Serão caracterizados as filosofias de software Aplicativo e instrucional, com estilo conversacional. É apresentado o SINAI-16, um software Interativo Numérico Aplicativo e Instrucional em micros de 16 bits voltado ao ensino de métodos numéricos de resolução de equação.

INTRODUÇÃO

Bibliotecas de software matemático podem ser vistas como elo de ligação entre os usuários do computador e seus problemas. Mas devido: ao recente desencadear da automação da sociedade; à falta de recursos aplicativos e instrucionais disponíveis no mercado, compatíveis com os microcomputadores existentes no Brasil; à evolução da importância e função de uma documentação consistente e simplificada e às diferentes finalidades comerciais (e não científicas) a que se destinam os softwares; os softwares matemáticos não tem acompanhado o desenvolvimento dos demais setores da Ciência da Computação e de setores que dela se utilizam.

Recentemente, com a utilização do computador no ensino e com o desenvolvimento da pesquisa em todas as áreas do conhecimento, tem-se verificado a necessidade de bibliotecas de softwares matemáticos numéricos, que contenham uma variedade de subrotinas, uma documentação consistente e simples, capaz de solucionar a grande parte das dúvidas do usuário imediatamente e uma exatidão computacional incluindo exatidão nos resultados e tempo de processamento. É preciso, ainda, que se tenha um estilo conversacional, possibilitando uma melhor ligação, interação entre o usuário e a máquina.

Para a pesquisa, existem as bibliotecas como a IMSL (International Mathematical and Statistical Libraries), a NATS (National Activity to Test Software) e a NAG (Numerical Algorithms Group) que são muito usadas. Por outro lado, deve-se iniciar a desenvolver softwares, para se adquirir experiência para um dia se poder desenvolver um produto, o qual seja comercializável. Para isto, se deve conhecer o processo de desenvolvimento, as dificuldades e as características desejáveis em softwares numéricos.

Tem-se que ter a consciência que, nesta etapa inicial, o software a ser desenvolvido dificilmente será absorvido pela indústria, por não se ter tradição em desenvolvimento e pelo fato de que os industriais preferem buscar softwares no exterior, pelas suas qualidades e confiabilidade. Por outro lado, o software poderia ser utilizado no ensino superior, em Universidades, onde, em geral, os recursos são escassos.

SOFTWARE NUMÉRICO - NATUREZA E DIFICULDADES

O termo "software numérico" ou "software matemático", inicialmente, referia-se a programas com uma performance basicamente matemática na computação de problemas referentes a métodos numéricos que resolvem problemas na Ciência e Engenharia, como é descrito em [CR077].

Software numérico é muito mais do que uma rotina programada de tarefa ou um produto da análise numérica, a qual algumas vezes, situa-se entre a matemática e a ciência da computação, sendo tão aplicada para alguns e irrelevante para outros.

Primariamente, o software numérico necessita tolerar erros. A palavra "erro" é inoportuna, porque erros numéricos não são erros no sentido usual da palavra, mas diferenças entre aproximações, que podem ser computadas num período finito de tempo.

Software numérico tem dois princípios característicos: o de tratar com aproximações de números reais e o fato de o intervalo de representação destes números varia de acordo com o computador.

O fato de que computadores diferem no tratamento de números em ponto-flutuante, significa que se precisa considerar classes de computadores. Há algumas décadas atrás, tratava-se do ambiente de um particular computador, mas hoje, considera-se sobre ambientes em classes de computadores, nas classes de linguagens. Estas classes de computadores têm diferentes intervalos numéricos, diferentes precisões e diferentes propriedades de arredondamento. Software numérico é escrito para se trabalhar nestes vários ambientes. Ele necessita ser sensível à precisão do computador, para que não se tente calcular maior exatidão do que é possível em uma máquina particular. Ele precisa ser sensível ao intervalo numérico para não permitir overflows e underflows desnecessários, e estar alerta a particularidades de erros de arredondamento.

Software numérico pode ser particularmente difícil de ser projetado, porque mesmo antes da precisão, intervalo e problemas de arredondamento, muitas tarefas numéricas são, em um sentido, insolúveis. Isto não quer dizer que o problema seja teoricamente impossível. Ele pode ser "resolvido", podendo ter como "soluçãO":

a) Um resultado correto e com exatidão requerida ou esperada pelo usuário;
b) Uma parte do programa dá que a tarefa é impossível, por não convergência ou ocorrência de algum erro;
c) Uma solução sem a exatidão desejada.

Os tipos de problemas numéricos podem ser classificados, de acordo com C. W. Gear em [GEA80], segundo três critérios que são as propriedades dos dados, propriedades dos algorítmos e propriedades do erro de arredondamento. A análise e descrição destes tipos de problemas vão além do propósito deste trabalho, embora se constate a dificuldades do projetar softwares numéricos.

Gear, ainda, menciona que grande parte das dificuldades do projeto de software numérico é devido à perda da base científica em ambiente computacional. Não existindo critérios simples que possam ser otimizados, mas um conjunto mal-definido de objetos como a confiabilidade, generalidade e utilidade que se está "tão longe" de falhar em quantificar.

Ele considera também o software numérico como parte da ciência da computação e espera que seja estudado em departamentos de ciência da computação ou informática.

NÍVEIS DE ATIVIDADES NO DESENVOLVIMENTO DE SOFTWARE

Existem alguns níveis distintos de atividades no desenvolvimento de software numérico. O trabalho feito em cada nível particular é um pouco independente do trabalho feito em outro nível, embora as diferenças entre níveis se tornem menos distintas à medida que se ganha mais experiência na preparação de software.

Cody, em seus artigos [COD82] e [COD74] identifica três níveis. O primeiro é o do desenvolvimento da matemática teórica ou análise numérica com seus métodos numéricos teóricos para resolver um particular tipo de problema. A ênfase neste nível está nas considerações teóricas, tais como análise do erro e provas de convergência.

O segundo nível da atividade é o da combinação de um ou mais métodos numéricos e sua transformação em algorítmo computacional para solucionar um determinado problema particular em um computador. O produto final deste segundo nível de atividade é muitas vezes considerado ser um programa computacional, especialmente quando o trabalho é publicado em uma linguagem algébrica de programação em um jornal profissional. Entretanto, usaremos o termo formal **algorítmo** para descrever o resultado final deste nível.

Cody define software computacional como sendo um módulo executável de programação, junto com a documentaçãoo dos algorítmos implementados em um ambiente específico.

A palavra-chave na definição de software computacional é implementação. Algorítmos na literatura não resolvem problemas em computadores, sómente implementações de algorítmos resolvem problemas. Um bom programa precisa ser baseado em um bom algorítmo, mas um bom algorítmo não garante um bom programa. Uma implementação imprópria de um ótimo algorítmo pode resultar numa péssima performance, comprometendo a reputação do algorítmo. Então, precisa-se atentar para os detalhes da implementação. Isto introduz o terceiro nível de atividade: a implementação de algorítmos para produzir um software computacional, isto é, o módulo executável de programação, junto com sua documentação.

Neste terceiro nível de atividade, a transformação de algorítmos computacionais em um software robusto freqüentemente envolve mudanças algorítmicas e inovações, as quais não podem ser feitas sem a ajuda da análise numérica.

O DESENVOLVEDOR DE SOFTWARE NUMÉRICO - O MATEMÁTICO COMPUTACIONAL

Existe uma significativa diferença entre o interesse e proximidade do analista numérico com a pessoa que desenvolve o software numérico. Esta pessoa será chamada de matemático computacional, para melhor caracterização.

O matemático computacional é um matemático que desenvolve, analisa e cria o algorítmo computacional, para obter uma solução aproximada para um problema matemático. Muitos destes problemas originam-se no campo de engenharia, da física, das ciências biológicas e sociais.

O matemático computacional tem como meta ser um suporte para outros trabalhadores e sua razão de ser é o sucesso do serviço numérico para usuários de computadores, através da criação e organização de bibliotecas de algorítmos numéricos. Eles não são, e nem pretendem ser analistas numéricos. Conquanto, o que eles são, essencialmente, numéricos fluentes, capazes de ler comandos em linguagem de alto nível, com uma apreciação por hardware e sistemas operacionais e devem ter a facilidade de escrever documentações claras e possuir habilidade de organização, somada à sensibilidade para as necessidades de usuários de computadores e ao desejo de vender seus produtos.

Os interesses do analista numérico são bem entendidos, entretanto, estão aquém das necessidades dos usuários de computadores. A força da motivação de um analista numérico é pessoal, estando muitas vezes tão envolvido em seu próprio interesse teórico, para produzir algorítmos ou programas úteis a um grupo de usuários.

Os trabalhos do analista numérico e do matemático computacional estão relacionados e não existe uma clara delimitação entre estas funções, além do que, existem muitos indivíduos, os quais atuam em ambas funções.

Duas atividades intelectuais de importância são a análise e a síntese. Analisa-se classes de problemas e classes de

métodos, mas depois, os resultados são de pouca utilidade para os analistas. Os resultados da análise precisam ser sintetizados em um programa. O matemático computacional é a pessoa interessada em ambas as atividades. E, segundo Gear, "a análise é a ciência e a síntese é a arte". O resultado será o software numérico propriamente dito.

Na aplicação da análise e síntese para produção do software numérico, o matemático computacional é, freqüentemente, requerido para desenvolver novos resultados matemáticos e para adaptar novos resultados para uso efetivo do computador digital automático. Os cálculos em algorítmos assim desenvolvidos são baseados tanto em considerações matemáticas, como em considerações computacionais.

O PROCESSO DE DESENVOLVIMENTO DO SOFTWARE

Para Johnston, em [JOH82], existem três passos básicos na criação de um software, os quais são:

a) A escolha do método básico a ser usado;
b) O projetar do algorítmo;
c) O produzir do programa como um item do software.

Ele define um **método** como uma fórmula matemática para achar a solução de um problema particular em estudo. Na escolha do método, é necessário que se tenha uma lista dos possíveis métodos a serem usados, o que implica numa pesquisa na literatura para se catalogar os existentes. Então, uma detalhada análise matemática de cada método necessita ser feita, para levantar princípios básicos, nos quais os métodos possam ser comparados, isto, juntamente com certa experiência computacional, a fim de avaliar o trabalho necessário para implementar cada método, extensões e comparações analíticas, para determinar o método ou combinação de métodos mais apropriada.

O segundo passo é o da descrição detalhada do processo computacional, que é o **algorítmo**, uma fórmula computacional projetada para resolver o problema com a exatidão quanto maior possível em um computador.

O passo final é o de converter o algorítmo em um iten do software. Este item do software é uma realização física do algorítmo, isto é, um **programa** do computador. Esta tarefa de produzir o programa, envolvendo diversas questões importantes e difíceis com respeito a estrutura aritmética usada na máquina, alocação de memória, velocidade dos dispositivos e operações do software. O objetivo deste passo é a plena utilização das capacidades computacionais disponíveis do sistema, de uma maneira mais eficiente possível.

Para Crowell e Fosdick, em [CRO77], o processo de produção de um software inicia com a análise algorítmica e prossegue através da construção do software e documentação, para vários testes e, finalmente, para a distribuição e suporte do produto do software. A exigência e o custo necessários são justificados pela utilidade de alta qualidade do software, pela eficiência do produto na sua distribuição e pelos benefícios providenciados à pesquisa em o aplicar.

Dew e James, em [DEW83], concordam com Crowell e Fosdick nas etapas do desenvolvimento do software. Eles partem, entretanto, de que, já se tenha os Métodos, não incluindo, portanto, a importante etapa da escolha do método básico a ser usado, descrita por Johnston. Eles caracterizam o desenvolvimento do software em cinco etapas, que são: o projeto e análise do algorítmo; a construção do software; uma documentação detalhada; vários testes e a distribuição e manutenção do software.

O projeto e análise de algorítmos

Na etapa do projeto e análise de algorítmos para desenvolvimento de um software, é onde são tomadas as decisões que influenciarão o software no que tange ao seu estilo, propósito e finalidade.

A produção de um software numérico não é uma simples extensão de programação básica e tem que ser bem mais do que um conjunto de programas que resolvem uma classe determinada de problemas. Programas que se pretende para distribuição pública necessitam delimitar o intervalo dos possíveis dados de entrada, de uma performance perante adversidades, compiladores e interpretadores, sistemas operacionais e características de hardware.

Nesta etapa é feita a definição de passos tomados para resolver o problema, são feitos convenções de notação, análises dos algorítmos, o relacionamento e o gerenciamento, compondo com estes elementos a filosofia e o estilo do software.

Esta etapa é uma das mais difíceis, pois, além do que já foi citado, é necessário um estudo dos métodos numéricos teóricos, para a tradução em algorítmos computacionais, o que muitas vezes requer reformulações nos métodos.

É ainda nesta etapa que se adota critérios, os quais são as características desejáveis de um software numérico, as quais são descritas mais adiante. Algumas destas são importante existirem em algorítmos a serem implementados, até para usuários sem maiores conhecimentos, tendo apenas a idéia básica do método. Estas são a **estabilidade**, o ser sensível aos erros de arredondamento, que são inevitáveis nos cálculos com números reais em computadores de precisão finita; a **exatidão**, que é a capacidade de resolver problemas com a exatidão pré-requerida na documentação, resolvendo corretamente a classe de problemas para que foi planejado, mesmo em caso de insucesso, com mensagens adequadas. Outra característica é a **eficiência**, pois quando implementado o algorítmo não deve gastar muito tempo ou espaço para armazenar ou resolver problemas.

Construção do software

A construção do software é a codificação do algorítmo, o verdadeiro código do algorítmo, incorporado no software, resultando num **programa e sua documentação**.

Nesta fase de transformação do algorítmo computacional no programa, o qual deve ter capacidade de detectar e contornar situações anormais, sem interromper a computação. O software deve conter facilidades para monitorar erros, a fim de determinar argumentos impróprios e prever problemas computacionais como underflow e overflow, antes que eles ocorram.

Esta transformação do algorítmo computacional num programa com as características acima, em geral, envolvem mudanças no algorítmo. O algorítmo alterado, então, deve ser analisado para garantir sua confiabilidade, pois uma implementação imprópria até mesmo de um magnífico algorítmo pode produzir resultados insatisfatórios, comprometendo a "reputação" do algorítmo. Ainda, um algorítmo óbvio não é, necessariamente, o melhor algorítmo e sua implementação em diferentes ambientes ou máquinas, produzem diferentes comportamentos.

Um esforço considerável é necessário para se construir um bom software, e dar a ênfase em ser capaz de transportar programas bem testados de uma instalação para outra, com mínimas modificações. Isto significa, que os matemáticos computacionais devem resistir a tentação dos programadores de explorar os recursos do sistema ou obscuras características especiais da linguagem de programação. Truques deste tipo produzem programas difíceis de serem compreendidos e podem falhar quando implementados em diferentes sistemas.

Deve-se projetar algorítmos com o propósito da adaptabilidade e codificá-los com a idéia de transportar o software de um computador para outro. Por exemplo, um teste do tipo:

" IF ABS(ERRO) < 0.5E-10 THEN ..."

não é adaptável, pois, devido a diferente precisão de cada computador, pode não ser possível obter a exatidão necessária em algumas máquinas. O programa, deste modo, irá se comportar diferentemente em diferentes sistemas, podendo falhar em alguns. Testes de erro devem ser expressos relativamente a precisão de cada computador. Uma melhor consideração para este teste seria definir no inicio do programa uma constante, por exemplo, "mi:=0.5E-10", que é a precisão do computador e usar nos testes:

" IF ABS(ERRO) < mi THEN ...".

Assim, só seria necessário modificar na passagem de um sistema para outro as constantes dependentes da máquina.

Portanto, num software numérico, alguns pontos, tais como testes de convergência, previsões de underflow e overflow, refletem o esquema de representação dos números um um computador. Estas porções do programa, que são dependentes da máquina, devem ser isoladas, tornando fácil sua alteração quando o programa é transportado para outra máquina.

Uma documentação detalhada

A documentação deve ser feita ao mesmo tempo que a construção do software, pois é parte deste produto final. A documentação influência o estilo do programa, o qual deve ser escrito para uma clara exibição da organização lógica da computação, localizando e bem definindo as partes dependentes do sistema.

A documentação precisa ser clara e bem pensada; organizada de maneira que permita um usuário ocasional obter informações necessárias facilmente.

A documentação deve ser consistente com as operações do programa, para que não haja surpresas no uso do programa.

Do ponto de vista do usuário, a documentação deve ser detalhada, contendo um enunciado clara da classe de problemas que o software, ou programa está designado a resolver, com todos os casos em que ele é particularmente aplicável; uma definição do tipo e propósito de cada parâmetro do programa; as restrições e a sua aplicação; um exemplo rodado de aplicação do programa, pois usuários irão freqüentemente procurar se basear no exemplo, adaptando-o ao caso particular que desejam resolver; deve conter uma avaliação de sua performance, isto é, informações detalhadas sobre velocidade de processamento e exatidão das respostas; enfim quaisquer características extras usadas no programa devem ser descritas e as principais seções dos programas devem ser destacadas pelo uso de comentários, dando uma estruturação clara, estando a listagem do programa disponível para quem quiser fazer modificações ou estendê-lo.

Resumindo, a documentação do software é a parte entre o que desenvolve e o usuário do produto e, por isto, deve ser completa, pertinente ao serviço a que se destina, com o nível adequado de detalhes. Quanto a que documentação produzir e como a produzir, deve-se ter em mente alguns princípios básicos:

a) Ela deve ser adequada e inteligível;
b) Deve-se testar a documentação do mesmo modo como se testa o próprio software;
c) Tem-se que escrever a documentação de modo a satisfazer seus leitores e usuários, não a você mesmo.

Vários testes

A determinação de testes numéricos nos programas matemáticos é uma tarefa complexa e muito demorada, que tem recebido muita atenção nos últimos anos.

Existem dois objetivos principais ao testar uma rotina. O primeiro é o de estabelecer experimentalmente o grau de correção do algorítmo. Para isto, seleciona-se um conjunto de problemas, chamados algumas vezes de "exercício de código", que são projetados para testar o maior número de possíveis caminhos do programa. O segundo objetivo é o de medir a performance

relativa do programa em comparação a outros. Uma vez assegurado que o programa é o algorítmo implementado, pois resolve o planejado, parte-se para a avaliação da performance. É importante entender que se está avaliando não o método numérico abstrato, mas o algorítmo implementado, o programa.

Quase toda avaliação de performance é feita pelo método de bateria, um grande conjunto de problemas para os quais o programa foi feito e preparado para resolver. A performance do programa é medida contra a de programas já existentes, registrando, por exemplo, o tempo requisitado para resolver cada problema, ou se foi convenientemente resolvido e quão exata foi sua solução.

Infelizmente, na maior parte das vezes, é difícil, mesmo após vários testes, dizer se um programa é absolutamente melhor que outro, uma vez que um programa pode resolver bem alguns problemas e outros não tão bem, enquanto que o oposto pode acontecer com outro programa. Esta é a razão de existirem bibliotecas de softwares, contendo vários programas para resolver um mesmo problema.

Distribuição e manutenção do software

Nesta etapa, pode-se justificar todo custo e esforço despendido para o desenvolvimento do software, através da distribuição do software, que pode ocorrer através de sociedades profissionais de suporte, como a "Association for Computing Machinery" (ACM), com seus grupos especiais de interesse em matemática aplicada ou numérica, como SIGMAP e SIGNUM, para os quais são enviados artigos, onde são analisados e divulgados. No Brasil, temos a Sociedade Brasileira de Matemática Aplicada e Computacional (SBMAC), que tem se preocupado em organizar uma programoteca da sociedade, e de divulgar softwares a ela enviados através de seu Boletim. Existe também a Sociedade Brasileira de Computação (SBC) com propósitos semelhantes.

A divulgação dos softwares está associada a ênfase que o software incorpora, que podem ser utilidade, explorável na pesquisa ou útil no melhoramento da produção de software.

Os projetos de software, em geral, originam-se do esforço de centros universitários, de laboratórios governamentais e de empresas privadas.

O importante, quando se desenvolve um software, é assumir a responsabilidade da manutenção do produto, incluindo correções de erros, modificações, extensões e o suprir novas necessidades do usuário.

Software numérico de boa qualidade enfrentam a expectativa de usuários, que tem sido significativamente mudada no último quarto de século, incluindo atividades de facilidades de uso. Um programa, o qual é o mais rápido e mais exato, poderá ser ignorado, se existir mesmo um número pequeno de impedimentos de uso. Os desenvolvedores de softwares comercial entenderam isto, mas muitos pesquisadores não, e ainda desprendem grande esforço para produzir um programa encorporando o melhor algorítmo, somente para ver o fruto de seu laborioso trabalho ignorado.

CARACTERÍSTICAS DESEJÁVEIS DE UM SOFTWARE NUMÉRICO

O propósito de desenvolver programas de um software numérico é providênciar ferramentas computacionais úteis para programação científica, mas, para isto, estas rotinas necessitam reunir certas características, critérios considerados essênciais, que são exatidão, eficiência, confiabilidade, validação, facilidade de uso, boa documentação, flexibilidade, modificabilidade, robustez e transportabilidade.

Muitas destas características já foram citadas, exemplificadas nas seções anteriores, mas para uma melhor formalização e caracterização, serão redefinidas, apesar da dificuldade, da dependênciaa e interrelacionamento destas características.

Exatidão e eficiência

Apesar de os critérios de um bom software tem sido mudados com a experiência e tecnologia, a exatidão e eficiência têm se mantido.

Exatidão é a capacidade do programa resolver problemas com a exatidão pré-requerida na documentação.

O critério de eficiência mudou com o desenvolvimento. No princípio, quando os computadores tinham pouca memória e esta era muito cara, um programa eficiente era um que usava a memória eficientemente, sendo o menor possível. Hoje, a memória está tão disponível e barata, fazendo com que este conceito evoluisse para velocidade de processamento; e as máquinas tornando-se cada vez mais velozes, este componente de eficiência deixou de ser tão importante.

Confiabilidade e validação

Confiabilidade é a característica que garante a consistência da operação do programa com sua documentação, resolvendo corretamente a classe de problemas para que foi planejado. Resolver corretamente refere-se a habilidade do programa de obter resultados numéricos desejados eficientemente. Nos casos em que não conseguir encontrar a solução, deve finalizar com uma mensagem de erro adequada.

Validação é um critério importante na produção de um software. O propósito da validação é estabelecer que a rotina cumprirá seu objetivo. Para isto, de inicio, necessita-se estar claro de seu objetivo, da classe de problemas que podem ser resolvidos e o que se entende pela "solução".

O processo da validação de um programa inclui uma detalhada análise matemática do algorítmo, uma prova de que o programa é uma representação fiel do algorítmo e, finalmente, um extensivo teste com representativo conjunto de problemas para testar a maior parte de possíveis caminhos do programa.

Uma apreciação imprópria do ambiente computacional durante a implementação pode degradar a confiabilidade e validação, restringindo o conjunto de problemas que podem ser resolvidos.

Facilidade de uso

O critério facilidade de uso está associado a ter boa documentação, mas também, quanta informação é requerida ao usuário para o uso de tal programa. Um programa fácil de ser usado é aquele que somente requer a entrada (dados iniciais), definições do problema a ser resolvido. Qualquer análise do problema para o propósito de setar os parâmetros do método é feito automaticamente pelo programa.

Um software fácil de ser usado é o que antecipa todas as opções que todos usuários possam querer usar, através de um controle que seleciona a opção desejada.

Boa Documentação

A documentação de um programa deve ser clara, conscisa, fácil de entender, explanando como funciona e como usar. Isto é útil não somente para decidir se, usa-se ou não o programa, como também, assistir no diagnóstico de cada dificuldade proveniente do uso. Ela deve ser organizada de maneira que permita um usuário ocasional obter informações necessárias facilmente.

Mensagens de erro são necessárias diagnosticando o acontecido, e sua gravidade. Um exemplo, numa rotina para resolver equações não lineares, pode não ser possível encontrar uma solução com a precisão requerida pelo usuário, em tal caso, a rotina deve apresentar a melhor solução que pode encontrar, dando o número de iterações e a exatidão contida em tal resultado, mais uma indicação de que não foi conseguido a exatidão requisitada.

A documentação vai além de manuais, incluindo a organização de resultado, de forma a documentar resultados, e ainda, a documentação do código, de forma a ser fácil de ser compreendido par manutenção, atualização, alterações e extensões.

Flexibilidade e modularidade

A característica de flexibilidade está associada a característica de facilidade de uso, permitindo que o usuário modifique valores de parâmetros, condições iniciais ou até mesmo de método de resolução do problema.

A modularidade é outra característica muito importante, e influência o estilo da programação, pois separa as tarefas a serem efetuadas em partes, denominadas módulos, o que gera um programa estruturado, facilitando o entendimento e manutenção do software. A modularidade é muito importante para garantir a transportabilidade de um software, uma vez que existem partes dependentes de máquina que necessitam ser modificadas, adaptadas ao novo ambiente. Estas partes são isoladas e organizadas de maneira que se possa modificar facilmente; esta parte deve ser bem identificada na documentação.

Dentro do conceito da modularidade, alguns autores introduzem a modificabilidade, que é a facilidade de se modificar e adaptar o software a um problema específico que se quer resolver. Esta questão é abordada por R. C. Bushnell, no artigo denominado: "User modifiable software" (em [RIC71], seção 5.2, p.59-66).

Robustez

A robustez é um indicador da extensão da rotina. Um método numérico envolve alguns parâmetros, os quais podem assumir valores determinados de acordo com as propriedades básicas do problema particular. A análise do processo é geralmente baseado no comportamento do problema modelo. O comportamento da maioria dos problemas não divergem, na prática, do esperado pelo modelo. Mas existem problemas em que a solução se deteriora, afastando-se do modelo. A extensão deste afastamento é uma medida de robustez, da potencialidade do programa.

Robustez se refere também, a habilidade do programa de contornar dificuldades, sem, necessariamente, interrupção do programa. Quando a interrupção se faz necessária, devido a erros, o software tem que providenciar um diagnóstico preciso do erro e sua gravidade. O problema de underflow é um exemplo. Usualmente, quando ocorre, dá-se uma mensagem de erro e considera-se o valor com zero. Se o underflow influenciar consideravelmente o resultado computado é chamado de "destrutivo", caso contrário é chamado de "não destrutivo". O problema com a mensagem de underflow é que ela não distingue entre o caso destrutivo e o não destrutivo.

Idealisticamente, deve-se reestruturar o programa de modo a evitar underflow; caso ocorra o underflow não destrutivo, o valor é substituído por zero, e caso ocorra o underflow destrutivo, deve ser dada uma mensagem de erro, diagnosticando precisamente o fato.

Transportabilidade

Transportabilidade envolve os conceitos de portabilidade e adaptabilidade. Portável é quando o programa pode ser executado corretamente em um novo ambiente sem nenhuma espécie de mudança, o que é, em geral, idealístico na prática. Se um programa pode ser transportado a outro ambiente, mantendo sua performance, com apenas poucas modificações bem documentadas, ele é adaptável ou transportável.

Estas características são relevantes a mudanças na performance do programa quando o ambiente da máquina é mudado. Isto é uma questão muito difícil, porque existem muitos fatores, os quais afetam a performance.

Idealisticamente, um software numérico deveria ser completamente portável, mas, devido as diferenças de máquinas e seus ambientes, é praticamente impossível atender este objetivo. Existem, entretanto, algumas medidas que podem ser tomadas para ajudar o grande tratamento da uniformidade da performance.

Uma das causas das diferentes performances é a variedade de arquiteturas de máquinas existentes. O remédio é fazer várias versões de rotina, uma para cada tipo de máquina. Isto não é portável, no sentido da palavra, é adaptável; mas o é para usuário.

Outra atitude para eliminar diferenças na performance é escrever o programa num dialeto "standart" da linguagem de programação, sem explorar características especiais de implementações dependentes de máquina.

Outra atitude, ainda, é definir valores que são dependentes da aritmética da máquina no inicio do programa, como constantes, e no transporte de uma máquina para outra, estes valores devem ser redefinidos de acordo com a aritmética da máquina, para evitar que se tente calcular exatidões maiores do que se pode calcular.

A INFLUÊNCIA DO TIPO DE USUÁRIO NO DESENVOLVIMENTO DE SOFTWARE

O software numérico se consistui não apenas de rotinas, mas é um programa que gerência várias rotinas numéricas, que resolvem certa classe de problemas.

A abrangência do software numérico é decorrente do tipo de usuário a que se destina, o que é manifesto pelo tipo de instituição que custeam o seu desenvolvimento, as quais, em geral, são Universidades, laboratórios governamentais e empresas privadas, individualmente ou em cooperação entre elas.

Os usuários de software influênciam o desenvolvimento de softwares, pois quando se desenvolve um software para um grupo de usuários, há limitações na diversidade de interesses e motivações que podem ser abrangidos, devido a conflitos entre conjuntos de interesses, como: entre a pesquisa e o ensino; entre a pesquisa geral e a pesquisa especializada; e entre usuários de uso geral e analistas numéricos.

Um software instrucional, como uma biblioteca de ensino, requer que métodos tradicionais estejam incluídos, uma vez que são ensinados, enquanto que uma biblioteca de pesquisa de uso geral ou um software aplicativo somente requer os métodos considerados como melhores em certa área. Num software instrucional métodos ruins são necessários para que se possa fazer comparações com os bons. Em softwares aplicativos são necessários resultados com grande exatidão, e calculados otimizadamente, enquanto que, em softwares instrucionais, a visualização de cálculos passo a passo são úteis a compreensão do método e a verificação de sua convergência.

Software aplicativo, com finalidade de pesquisa geral ou pesquisa especializada, gera outro conflito, pois grupos de pesquisa especializada, por causa da penetração em áreas numéricas particulares, requerem uma abrangência muito maior nestas áreas do que os de finalidade de pesquisa geral.

O próprio comportamento de analistas numéricos que, em geral, efetuam mudanças de conteúdos, códigos e na própria documentação, costumeiramente faz com que o software tenha características de modularidade, para que seja fácil de fazer

alterações e expansões; enquanto que para usuários de uso geral, que apenas utilizam o software de uma forma "estática", ou seja, o software e a documentação permanecem os mesmos para sempre. Isto requer um software fácil de ser usado, com boa documentação e robusto.

A documentação também deve ser adequada a finalidade do software, tanto em qualidade como em quantidade. Para software aplicativo comercial, J. D. Lomax em seu livro "Documentação de software" (LOM83), descreve a documentação em três categorias:

a) Informações preliminares;
b) Documentação do usuário;
c) Documentação de manutenção.

costituindo uma "grande coletânea" de documentação.

Para software aplicativo científico, pode ser simplificado e minimizado em documentação do usuário e documentação do programa.

Para software instrucional, a documentação é ainda compensada com um estilo conversacional nos programas, permitindo ao usuário fazer inúmeras variações nos dados e parâmetros de entrada. Desta forma, para um mesmo problema o usuário pode, facilmente, alterar um dos parâmetros e observar o efeito da mudança nos resultados, o que lhe permite um controle do comportamento da resolução.

O estilo conversacional proporciona maior interação do usuário com a máquina, dando a ele uma maior clareza dos passos que estão sendo efetuados no processamento, proporcionando, ainda, maior flexibilidade no uso do software, amenizando o "efeito místico", que era resultante de software "caixa-preta", dando assim, maiores informações.

O estilo conversacional proporciona ainda um software fácil de ser usado, desde que em cada nível de tarefa possa ser caracterizado por telas instrucionais que antecipem todas as possíveis tarefas que todos os usuários possam querer requerer. O programa neste estilo deve ser auto-documentado, para proporcionar esclarecimentos a usuários menos instruídos, e que as perguntas sejam claras e definam bem a resposta, não permitindo ambigüidades. Uma técnica usada é chave de seleção, onde na tela estão todas opções possíveis, e o usuário escolhe uma delas, através de uma letra, um número ou uma tecla. Isto também é conhecido como menu.

SINAI-16 UM EXEMPLO DE SOFTWARE INSTRUCIONAL

O SINAI-16 (Software Interativo Numérico Aplicativo e Instrucional em micros de 16 bits) foi desenvolvido no Laboratório de Programação do Instituto de Informática da PUCRS, com a finalidade de auxiliar o ensino de cálculo numérico e métodos computacionais e também servir de suporte a pessoas que em seus trabalhos necessitam resolver equações algébricas e transcendentes.

Em seus seus quase seis meses de disponibilidade, foi utulizado por diversas turmas de cálculo numérico, facilitando a

aprendizagem e aumentando o interesse dos alunos pelo assunto. Ele também foi eficiente na resolução de alguns problemas dentro da comunidade científica.

O SINAI-16 proporciona a resolução de equações polinomiais de grau menor ou igual a vinte, calculando as raízes reais e complexas, inclusive reais múltiplas. Proporciona, ainda, um módulo que possibilita traçar gráficos e imprimi-los (caso se tenha impressora gráfica). Ele contém um estudo completo deste capítulo numérico; incluindo a localização de raízes, enumeração através da visualização gráfica e em alguns casos, através de resultados obtidos, a determinação de valores iniciais para métodos iterativos; o cálculo de raízes pelos métodos tradicionais e por alguns outros de importância científica atual; e até a deflação de raízes pelo método de Briot-Ruffini.

Este pacote é interativo, proporcionando que a qualquer momento se possa modificar ou corrigir valores dos coeficientes; ou trocar a estimativa inicial, ou a exatidão desejada, e, até mesmo, trocar o método de resolução, sem que tenha que entrar novamente com os coeficientes da função.

O SINAI-16 é um pacote fácil de ser usado, devido ao estilo conversacional, e auto-documentado por tela instrucionais, contendo menus de seleção e telas de orientação e instrução, além de alguns efeitos sonoros para tornar o primeiro contato mais agradável.

A documentação é adequada ao estilo, tendo as opções de uso, antecipadas por menus, que compõem as telas instrucionais, as quais facilitam o uso do software, pois não há necessidade de conhecimento prévio para utilização do software. Além disto, os manuais são organizados de modo a conceder a informação desejada, por qualquer usuário, a qualquer momento.

Métodos implementados

Os métodos implementados na primeira seção do SINAI-16 (resolução de polinômios), visaram atender finalidades aplicativas e instrucionais, incluindo métodos que são usualmente ensinados nos cursos de cálculo numérico e métodos computacionais, bem como métodos novos ou mais eficientes e complexos que têm seu valor científico.

Capacidade Gráfica: O SINAI-16 pode traçar gráficos de polinômios de grau até vinte, com intervalos de definição variável, capaz de auxiliar a determinação da enumeração e separação de raízes.

Cálculo de Cotas: Calcular cotas, ou intervalos que contenham todas as raízes da equação, além de proporcionar boas estimativas de vlores iniciais para métodos iterativos. As cotas implementadas foram a de Cauchy, que, em geral, está próxima a maior raiz em módulo; e a Fase Inicial, que determina um círculo que contém no mínimo uma rais, estimando um valor inicial próximo a menor em módulo, além de identificar a existência de raízes complexas.

Cálculo de raízes reais de equações: Os métodos implementados foram a da Bissecção; de Newton-Rapshon; da Secante; de Newton para raízes múltiplas; o método Híbrido C e o método MIDREM.

Cálculo de raízes complexas de equações: Os métodos implementados para o cálculo de raízes complexas foram o método de Newton para complexas e o de Bairstow.

Deflacionar de raízes: As raízes calculadas reais ou complexas podem se deflacionadas, reduzindo o grau do polinômio, através do método de Briot-Ruffini e algorítmo de Horner.

Documentação

Por ser um software conversacional, o SINAI-16 é auto-documentado, contendo em seções principais telas de informações e de auxílio, descrevendo as funções implementadas, para orientar ao usuário no momento da execução e utilização do programa, bem como para instruir o usuário, suprindo assim, as finalidades aplicativas e instrucionais.

Além disto, o SINAI-16 possui dois manuais: o **Manual Instrucional do Usuário (MIU)**, que contém um conjunto completo de informações do ambiente computacional, de instalação, organização e utilização do software. Neste último item, destacam-se as fichas instrucionais que além de exemplos para ilustrar a utilização do software, contém todas as informações para pessoas que queiram utilizar métodos de resolução de equações tanto com ênfase da análise numérica; da prática de utilização do método; ou da ênfase computacional. O outro manual é denominado **Manual de Manutenção do Programa (MMP)**, que além de um conjunto básico comum ao MIU, contém informações para garantir a transportabilidade do software a outro equipamento, incluindo a listagem completa do software.

Outro módulos

Está sendo feito a seção de resolução de equações algébricas e transcendentes, estando esta em fase de depuração e ajustes finais na documentação. Esta nova seção permite traçar gráficos de equações trancendentes e o cálculo de raízes pelos métodos da Bissecção; Newton-Raphson; Secante e pelo Híbrido C.

Maiores informações podem ser obtidas na PUC - RS INFORMÁTICA (Prédio 30), caixa postal 1429, cep 90620 -Porto Alegre - RS, aos cuidados do Projeto SINAI-16.

Referência Bibliográfica

[COD74] CODY, W. J. The construction of numerical subroutine libraries. SIAM review, Philadelphia, 16(1):36-46, Jan.1974.

[COD82] CODY, W. J. Basic concepts for computational software. In: INTERNATIONAL SEMINAR ON PROBLEMS AND METHODOLOGIES IN MATHEMATICAL SOFTWARE PRODUCTION, Sorrento, Nov.3-8, 1980. Proceedings. Berlin, Springer-Verlag, 1982. p.1-23.

[CRO77] CROWELL, W.R & FOSDICK, L.D. Mathematical software production. IN: MATHEMATICAL SOFTWARE III. New York, Academic Press, 1977.

[DEW83] DEW, P.M & JAMES, K. R. Introduction to numerical computation in PASCAL. Hong Kong, Macmillan Press, 1983.

[DIV86] DIVERIO, T. A. Software Numérico Aplicativo e Instrucional. Porto Alegre, CPGCC da UFRGS, 1986. (dissertação)

[DIV86a] DIVERIO, T. A. SINAI-16 Manual instrucional do usuário. Porto Alegre, CPGCC da UFRGS, 1986.

[DIV86b] DIVERIO, T. A. SINAI-16 Manual de manutenção do programa. Porto Alegre, CPGCC da UFRGS, 1986.

[GEA80] GEAR, C. W. Numeraical Software: science or alchemy?. Advances in computer, New York, 19:229-48, 1980.

[JOH82] JONHSTON, R.L. Numerical methods a software approach. New York, John Willey & Sons, 1982.

[LOM83] LOMAX, J. D. Documentação de software. Rio de Janeiro, Campus, 1983.

[RIC71] RICE, J. R. Mathematical software. New York, Academic Press, 1971. (ACM Monograph series)