ALGORITMO DE RUNGE-KUTTA: DESENVOLVENDO CLASSES PARA O REÚSO NA ENGENHARIA

Lincoln C. Zamboni [1], Sergio V. D. Pamboukian [2], Edson A. R. Barros [3]

Abstract *— The numeric methods were always, in the technician-scientific area, the great link between mathematical models and practical results for engineers, researchers and students. This is due to the fact that they are of easy understanding and implementation. On the other hand, the programming languages as C++ allow the conception of classes and the modelling of the engineering problems focusing mainly the reuse of objects. In the Escola de Engenharia of the Universidade Presbiteriana Mackenzie, that it is commemorating their 110 years of existence, grows several interconected projects that use the programming as tool not only of search of the solution, but also didacticism and practice for the student. It is in this optics that the Method of Runge-Kutta of Fourth Order is used in an Object Oriented language.*

Index Terms — Numeric methods, differential equations, Runge-Kutta, programming with classes.

INTRODUÇÃO

O ensino das disciplinas que tratam, dentre outros conteúdos, sobre algoritmos, computação, programação e aplicação vivencia atualmente uma abordagem com ferramentas norteadas para o reúso de objetos.

Tais disciplinas são desenvolvidas na Escola de Engenharia da Universidade Presbiteriana Mackenzie com o notório enfoque em problemas que os estudantes necessitam resolver ao longo dos mais diversos cursos como, por exemplo, nas engenharias elétrica, civil, mecânica, de materiais e de produção e na tecnologia elétrica.

O tratamento de problemas de engenharia sob a clássica ótica de decomposição em funções tem sido paulatinamente modificado, mas não abandonado, para a decomposição em classes.

Tal tratamento, embora inicialmente mais complicado para o estudante, tem, como coluna vertebral o reúso de objetos e, neste reúso, alcança-se a simplicidade buscada pelo estudante.

É oportuno ainda citar que os tratamentos funcional e orientado a objetos exercitam uma visão sistêmica dos problemas e como resultado paralelo estimula os estudantes ao emprego de técnicas de reuso de software.

FUNÇÕES E CLASSES

Nos cursos de engenharia da universidade, procura-se fornecer aos alunos alguns conceitos fundamentais de linguagem de programação para que a mesma, em conjunto com outras disciplinas, possam auxiliá-lo na solução de problemas específicos de sua área. Estes conceitos trazidos para o âmbito da linguagem C++, e que podem ser facilmente migrados para qualquer outra linguagem orientada a objetos como Java, C#, Object Pascal, etc., são:

- tipos básicos de dados: int, double, char, bool, float, long int, long double, etc.;
- instruções de controle do fluxo: seqüência, condição (if/else, switch) e repetição (for, do/while e while);
- reúso de funções matemáticas e de tratamento de cadeias de caracteres: +, -, *, /, %, sqrt, sin, cos, atan, exp, log, etc;
- arranjos e estruturas: ponteiros, operador [], new e delete para alocação dinâmica;
- especificação e reúso de funções;
- especificação e reúso de classes;
- diagramas de classes e de atividades da UML (Unified Modeling Language);
- fôrmas de funções e fôrmas de classes da Standard Template Library (STL): string, vector, list, etc;
- reúso de objetos de interface gráfica: formulários, botões, rótulos, janelas de edição, grades, etc.

Dois conceitos fundamentais, função e classe, são abordados na especificação e reúso e divididos em duas metodologias de programação:

- Estruturada: onde se trata, em essência, o conceito de função e suas aplicações;
- Orientada a Objetos: onde se trata, em essência, o conceito de classe e suas aplicações.

Outros pontos importantes e imprescindíveis para a engenharia são:

- o uso de unidades do SI (Sistema Internacional de Unidades) nas fórmulas;
- a documentação interna do código com comentários e a geração automática de páginas de ajuda como, por exemplo, HTML (HiperText Markup Language).
- a organização do programa em unidades independentes.

1 Lincoln César Zamboni, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, lincoln.zamboni @mackenzie.com.br
2 Sergio Vicente Denser Pamboukian, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, sergiop@mackenzie.com.br
3 Edson de Almeida Rego Barros, Escola de Engenharia, Universidade Presbiteriana Mackenzie, Rua da Consolação, 930, 01302-907, Consolação, São Paulo, SP, Brazil, prof_edson@mackenzie.com.br

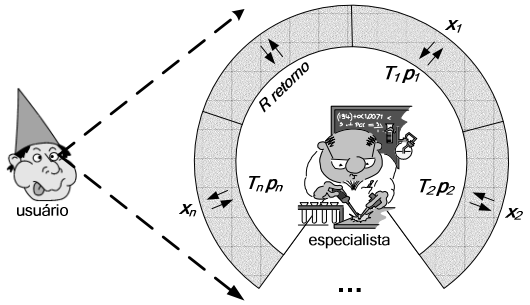

FIGURA. 1
O USUÁRIO E O ESPECIALISTA: DOIS PONTOS DE VISTA.

Neste trabalho enfocamos o conceito de classes e suas aplicações práticas na engenharia e na programação.

FÓRMULAS, ALGORITMOS E MÉTODOS

De acordo com [1], o estudante de engenharia se estereotipa como um aplicador de fórmulas. Isto se torna evidente quando, por exemplo, da solução da equação do segundo grau $a \cdot x^2 + b \cdot x + c = 0$ onde, a partir do conhecimento dos coeficientes reais *a*, *b* e *c*, determina-se, através da conhecida fórmula de Bhaskara, os dois valores da raiz *x*.

É fato que a maioria dos problemas de interesse em engenharia não possui uma fórmula algébrica fechada, como a de Bhaskara, para sua solução, mas sim um algoritmo: o algoritmo generaliza e amplia o conceito de fórmula.

A elaboração de algoritmos desenvolve no futuro engenheiro qualidades de planejamento, preparo e previsão que não devem ser omitidas no ensino universitário e, atualmente, são implementados tanto sobre a ótica da programação Estruturada, com o uso de funções, como a da Orientada a Objetos, com o uso de métodos.

Erro! Não é possível criar objetos a partir de códigos de campo de edição.
FIGURA. 2
ALGORITMOS DE EULER E DE RUNGE-KUTTA.

NOTAÇÃO COMPUTACIONAL PARA UM MÉTODO

De acordo com [1] e [2], o matemático suíço Leonhard Euler, pai da Matemática Discreta, foi o responsável pela notação de função como se conhece atualmente. Esta notação foi usada nos "Comentários de Petersburgo" em 1734. A mesma notação é utilizada pelas funções da programação estruturada e pelos métodos da programação orientada a objetos.

Em C++ utilizamos ***obj*.*f*(*x*$_1$, *x*$_2$, ..., *x*$_n$)** para chamarmos o método de nome *f* do objeto *obj*; utilizamos ***pont*->*f*(*x*$_1$, *x*$_2$, ..., *x*$_n$)** para chamarmos o método de nome *f* do objeto apontado por *pont*; utilizamos ***cla*::*f*(*x*$_1$, *x*$_2$, ..., *x*$_n$)** para chamarmos o método de nome *f* da classe *cla*.

A Figura 1 ilustra um método sob o ponto de vista computacional de um usuário e de um especialista: o usuário usa as especificações que o especialista implementa. É esta relação que existe entre o engenheiro civil e o morador; entre o engenheiro mecânico e o motorista; entre o engenheiro eletrônico e o calculista; etc.: todos estes implementam para os seus usuários.

Observe que T_1, T_2,... , T_n são, respectivamente, os tipos dos parâmetros p_1, p_2,... , p_n (domínio do método) e R o tipo de retorno (contradomínio).

Existe uma correspondência e compatibilidade entre os argumentos da chamada $x_1, x_2, ..., x_n$ (visão do usuário) e os parâmetros $p_1, p_2, ..., p_n$ (visão do especialista): ao parâmetro p_1 corresponde o argumento x_1, ao parâmetro p_2 corresponde o argumento x_2, e assim sucessivamente.

ALGORITMOS DE EULER E DE RUNGE-KUTTA

Os algoritmos de Euler (primeira ordem) e de Runge-Kutta (segunda e quarta ordens) encontrados em [1] e [3] são utilizados para resolver numericamente (1), uma equação diferencial de primeira ordem, conhecendo-se (2), sua condição inicial.

$$y' = f(x, y) \tag{1}$$

$$f(x_0, y_0) \tag{2}$$

Os três algoritmos são ilustrados pelo diagrama de atividades da Figura 2.

O que os difere é a atividade Iteragir: no de Euler utiliza-se (3), no de Runge-Kutta de segunda ordem utiliza-se (4) e no de Runge-Kutta de quarta ordem utiliza-se (5).

$$\begin{cases} k_1 = h \cdot f(x_i, y_i) \\ y_{i+1} = y_i + k_1 \end{cases} \tag{3}$$

$$\begin{cases} k_1 = h \cdot f(x_i, y_i) \\ k_2 = h \cdot f(x_i + h/2, y_i + k_1/2) \\ y_{i+1} = y_i + k_2 \end{cases} \tag{4}$$

$$\begin{cases} k_1 = h \cdot f(x_i, y_i) \\ k_2 = h \cdot f(x_i + h/2, y_i + k_1/2) \\ k_3 = h \cdot f(x_i + h/2, y_i + k_2/2) \\ k_4 = h \cdot f(x_i + h, y_i + k_3) \\ y_{i+1} = y_i + (k_1 + k_4)/6 + (k_2 + k_3)/3 \end{cases} \quad (5)$$

Os mesmos algoritmos são utilizados para resolver uma equação diferencial de ordem superior e também um sistema de equações diferenciais e, para tanto, são feitas algumas modificações algébricas na equação ou sistema que estão descritas em [3].

Tais algoritmos possuem as seguintes ordens de erro: $\boldsymbol{O}(\boldsymbol{h}^2)$ para o de Euler de primeira ordem, $\boldsymbol{O}(\boldsymbol{h}^3)$ para o de Runge-Kutta de segunda ordem, $\boldsymbol{O}(\boldsymbol{h}^5)$ para o de Runge-Kutta de quarta ordem.

Dentre os três, o de Euler é o menos utilizado e o de Runge-Kutta de quarta ordem, o mais. De uma forma geral o de Runge-Kutta de quarta ordem é o mais utilizado na resolução numérica de equações diferenciais na engenharia.

IMPLEMENTAÇÃO DAS CLASSES EM C++: PONTO DE VISTA DO ESPECIALISTA

As classes desenvolvidas para reúso podem ser vistas na Figura 3. Começamos pelo desenvolvimento da classe Vetor que é derivada da classe parametrizada vector da STL; a classe Vetor é dependente da classe string, também da STL; a classe abstrata *Euler1* fornece a base para todas as demais; a classe *Euler1Vet* é derivada da *Euler1* e também agrega três objetos da classe Vetor para os dados: *x*, *k1* e *y*.

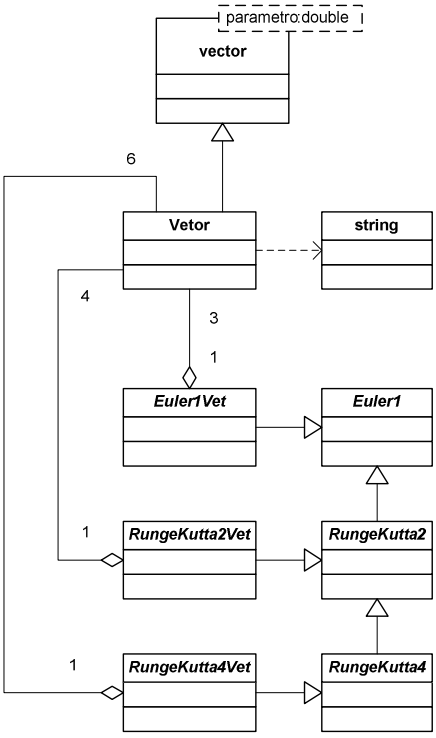

FIGURA. 3
DIAGRAMA DE CLASSES DA UML PARA O PROBLEMA.

A Figura 4 mostra um trecho do arquivo cabeçalho para a classe `Vetor` e a Figura 5 mostra um trecho do arquivo de implementação para a mesma classe. Observa-se em ambos a definição da classe dentro de um espaço de nomes identificado por `num`.

```
using namespace std;
namespace num
{
  // Tipo VetorDouble a partir da STL
  typedef vector<double> VetorDouble;
  // Classe para representar o vetor
  class Vetor: public VetorDouble
  {
    public:
      // Retorna elemento i em forma de texto
      string operator [](int i);
  };
  // Tipo Iterador a partir da STL:
  // generalização de ponteiro
  typedef Vetor::const_iterator Iterador;
}
```

FIGURA. 4
ARQUIVO CABEÇALHO UNITVETOR.H.

```
namespace num
{
  string Vetor::operator [](int i)
  {
    stringstream sai;
    sai << at(i);
    return sai.str();
  }
}
```

FIGURA. 5
ARQUIVO DE IMPLEMENTAÇÃO UNITVETOR.CPP.

```
using namespace std;
namespace num
{
  class Euler1
  {
    protected:
      int t;
      double x0;
      double y0;
      double xt;
      double h;
    public:
      Euler1(int t = 20, double x0 = 0.0,
        double y0 = 0.0, double xt = 2.0);
      ~Euler1();
      string mostraValores();
      virtual double calculaXEm(int n);
      virtual double calculaYEm(int n);
      virtual double calculaInterpYEm(double x);
      virtual double f(double x, double y) = 0;
      double k(double x, double y);
  };
}
```

FIGURA. 6
ARQUIVO CABEÇALHO UNITEULER1.H.

```
namespace num
{
  Euler1::Euler1(int t, double x0, double y0,
    double xt)
  {
    if(t <= 0) throw (string)"t inválido!";
    this->t = t;
    this->x0 = x0;
    this->y0 = y0;
    this->xt = xt;
    h = (xt - x0) / t;
  }
  Euler1::~Euler1()
  {
  }
  string Euler1::mostraValores()
  {
    stringstream sai;
    sai << "t=" << t << ", x0=" << x0
        << ", y0=" << y0 << ", xt=" << xt;
    return sai.str();
  }
  double Euler1::calculaXEm(int n)
  {
    if(n < 0 || n > t)
      throw (string)"n inválido!";
    return x0 + n * h;
  }
  double Euler1::calculaYEm(int n)
  {
    if(n < 0 || n > t)
      throw (string)"n inválido!";
    double k1n = k(x0, y0);
    double x, y = y0;
    for(int i = 1; i <= n; i++){
      x = x0 + i * h;
      y += k1n;
      k1n = k(x, y);
    }
    return y;
  }
  double Euler1::calculaInterpYEm(double x)
  {
    if(x < x0 || x > xt)
      throw (string)"x fora da faixa";
    double xb = x0, yb = y0;
    double k1n = k(xb, yb);
    int i = 1;
    double ya;
    while(x > xb){
      xb = x0 + i * h;
      ya = yb;
      yb += k1n;
      k1n = k(xb, yb);
      i++;
    }
    double xa = xb - h;
    return ya + (x - xa)*(yb - ya) / (xb - xa);
  }
double Euler1::k(double x, double y)
  {
    return h * f(x, y);
  }
}
```

FIGURA. 7
ARQUIVO DE IMPLEMENTAÇÃO UNITEULER1.CPP.

As Figuras 6 e 7 mostram a implementação da classe `Euler1` dentro do espaço de nomes `num` e as Figuras 8 e 9 mostram a implementação da classe `RungeKutta2` dentro, também, do espaço de nomes `num`. Observa-se a derivação da classe `RungeKutta2` a partir da `Euler1`.

```
namespace num
{
  class RungeKutta2: public Euler1
  {
    public:
      RungeKutta2(int t = 20, double x0 = 0.0,
        double y0 = 0.0, double xt = 2.0);
      ~RungeKutta2();
      virtual double calculaYEm(int n);
      virtual double calculaInterpYEm(double x);
  };
}
```

FIGURA. 8
ARQUIVO CABEÇALHO UNITRUNGEKUTTA2.H.

```
namespace num
{
  RungeKutta2::RungeKutta2(int t, double x0,
    double y0, double xt): Euler1(t, x0, y0, xt)
  {
  }
  RungeKutta2::~RungeKutta2()
  {
  }
  double RungeKutta2::calculaYEm(int n)
  {
    double k1n = k(x0, y0);
    double k2n = k(x0 + h/2.0, y0 + k1n/2.0);
    double x, y = y0;
    for(int i = 1; i <= n; i++){
      x = x0 + i * h;
      y += k2n;
      k1n = k(x, y);
      k2n = k(x + h/2.0, y + k1n/2.0);
    }
    return y;
  }
  double RungeKutta2::calculaInterpYEm(double x)
  {
    if(x < x0 || x > xt)
      throw (string)"x fora da faixa";
    double xb = x0, yb = y0;
    double k1n = k(xb, yb);
    double k2n = k(xb + h/2.0, yb + k1n/2.0);
    int i = 1;
    double ya;
    while(x > xb){
      xb = x0 + i * h;
      ya = yb;
      yb += k2n;
      k1n = k(xb, yb);
      k2n = k(xb + h/2.0, yb + k1n/2.0);
      i++;
    }
    double xa = xb - h;
    return ya + (x - xa)* (yb - ya) / (xb - xa);
  }
}
```

FIGURA. 9
ARQUIVO DE IMPLEMENTAÇÃO UNITRUNGEKUTTA2.CPP.

REÚSO DAS CLASSES EM C++: PONTO DE VISTA DO USUÁRIO

As Figuras 10, 11 e 12 mostram o reúso das classes `Euler1` e `RungeKutta2` em uma aplicação gerada com objetos de interface gráfica. A função *f*, vista em (1) e (2) é dada, a título de exemplo, por $f(x,y) = \cos(x)$.

```
using namespace num;
class Euler1Exemplo : public Euler1
{
  public:
    Euler1Exemplo(int t = 20, double x0 = 0.0,
      double y0 = 0.0, double xt = 2.0);
    virtual double f(double x, double y);
};
class RungeKutta2Exemplo : public RungeKutta2
{
  public:
    RungeKutta2Exemplo(int t = 20, double x0 =
      0.0, double y0 = 0.0, double xt = 2.0);
    virtual double f(double x, double y);
};
```

FIGURA. 10
ARQUIVO CABEÇALHO UNITINTERFACE.H.

```
Euler1Exemplo::Euler1Exemplo(int t, double x0,
  double y0, double xt): Euler1(t, x0, y0, xt)
{
}
double Euler1Exemplo::f(double x, double y)
{
  return cos(x);
}
RungeKutta2Exemplo::RungeKutta2Exemplo(int t,
  double x0, double y0, double xt):
  RungeKutta2(t, x0, y0, xt)
{
}
double RungeKutta2Exemplo::f(double x, double y)
{
  return cos(x);
}
```

FIGURA. 11
ARQUIVO DE IMPLEMENTAÇÃO UNITINTERFACE.CPP.

CONSIDERAÇÕES FINAIS

Neste trabalho enfocamos o conceito de classe e suas aplicações práticas na engenharia e na programação. Em [2] o leitor encontrará outras boas aplicações de classes na engenharia; em [4] um enfoque didático com objetos de interface gráfica em aplicações de engenharia; em [5] e [6] os fundamentos da programação estruturada com exemplos ligados à engenharia. Em [7] encontra-se excelentes implementações de classes e algoritmos para aplicações numéricas não só para a engenharia. [3] e [7] formam uma referência mundial na programação numérica em C/C++.

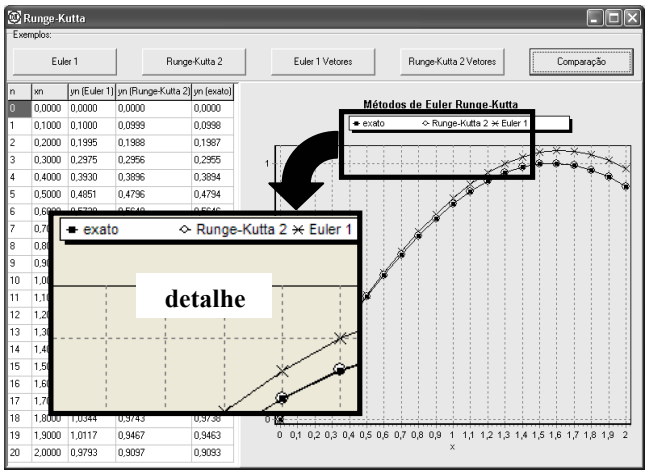

FIGURA. 12
RESULTADOS NA INTERFACE GRÁFICA.

REFERÊNCIAS

[1] ZAMBONI, L. C.; MONEZZI JÚNIOR, O., "Cálculo Numérico para Universitários". São Paulo: Páginas & Letras, 2002.

[2] ZAMBONI, L. C.; PAMBOUKIAN, S. V. D.; BARROS, E. A. R., "C++ para Universitários". São Paulo: Páginas & Letras, 2006.

[3] PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING W. T.; FLANNERY, B. P., "Numerical Recipes in C: The Art of Scientific Computing". Cambridge University Press, 1992.

[4] PAMBOUKIAN, S. V. D.; ZAMBONI, L. C.; BARROS, E. A. R., "C++ Builder para Universitários" - 2ª Edição. São Paulo: Páginas & Letras, 2003.

[5] BARROS, E. A. R.; PAMBOUKIAN, S. V. D.; ZAMBONI, L. C., "Ensinando Programação Através de Funções" – WCCSETE: World Congress on Computer Science, Engineering and Technology Education, 2006.

[6] ZAMBONI, L. C.; PAMBOUKIAN, S. V. D.; BARROS, E. A. R., "Simulação Computacional como Apoio para Outras Disciplinas" – WCCSETE: World Congress on Computer Science, Engineering and Technology Education, 2006.

[7] PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING W. T.; FLANNERY, B. P., "Numerical Recipes in C++: The Art of Scientific Computing". Cambridge University Press, 2002.