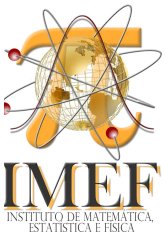
Universidade Federal do Rio Grande

Instituto de Matemática, Estatística e Física

Pós-Graduação em Modelagem Computacional



ANDRÉ LOPES BRUM

PEDRO HENRIQUE FERNANDES LOBO

VINÍCIUS HEIDTMANN AVILA

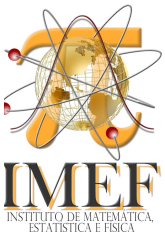
MÉTODOS NUMÉRICOS APLICADOS A EQUAÇÕES DIFERENCIAIS ORDINÁRIAS

Rio Grande-RS, 2017

Universidade Federal do Rio Grande

Instituto de Matemática, Estatística e Física

Pós-Graduação em Modelagem Computacional



ANDRÉ LOPES BRUM

PEDRO HENRIQUE FERNANDES LOBO

VINÍCIUS HEIDTMANN AVILA

MÉTODOS NUMÉRICOS APLICADOS A EQUAÇÕES DIFERENCIAIS ORDINÁRIAS

Trabalho apresentado ao programa de Pós-Graduação em Modelagem Computacional, pertencente a Universidade Federal do Rio Grande, como requisito de avaliação parcial para conclusão do curso de Métodos Numéricos Aplicados.

Rio Grande-RS, 2017

1 Objetivo

Comparar os métodos de Euler e Runge-Kutta de dois estágios ao resolver uma equação diferencial ordinária de segunda ordem.

2 Introdução

Muitos problemas em ciências e engenharias são modelados através de equações diferenciais. Porém, nem sempre elas apresentam soluções analíticas. Por isto, muitas vezes é necessário recorrer ao computador para que encontrar as soluções numéricas.

Desta maneira, para compreender a lógica por trás dos métodos Runge Kutta e de Euler, que são bastante conhecidos, iremos aplicá-los em um problema simples e recorrente na física, fazendo uma breve discusão sobre eficácia a deles.

3 Fundamentação teórica

Seja uma equação diferencial da forma:

$$\begin{aligned} \frac{d}{dt}y(t) &= f\left(t, y(t)\right) \\ y(0) &= y_0 \end{aligned} \tag{1}$$

Embora esta equação possa ter uma solução analítica, ou seja, capaz de estabelecer diferenciais contínuas em um intervalo, o computador pode imitar o processo de diferenciação se o espaço for discretizado. Para isto, vamos procurar uma solução através de uma expansão em série de Taylor:

$$y(t+\Delta t) = y(t) + \Delta t\frac{d}{dt}y(t) + \frac{\Delta t}{2!}\frac{d^2}{dt^2}y(t) + \frac{\Delta t}{3!}\frac{d^3}{dt^3}y(t) + ... + \frac{\Delta t}{n!}\frac{d^n}{dt^n}y(t)$$

Se truncarmos a série no primeiro termo e isolarmos a diferencial obtemos:

$$\frac{d}{dt}y(t) = \frac{y(t+\Delta t) - y(t)}{\Delta t} + \mathcal{O}(\Delta t) \tag{2}$$

Com isso, comparando as equações 1 e 2 e negligenciando os demais termos a partir do segundo, podemos determinar de uma forma iterativa, qual o valor da função $f\left(t, y(t)\right)$ no ponto seguinte distante de Δt. Desta forma, a expressão seguinte é conhecida como método de Euler e o cálculo da função avaliada na posição subsequente é dada por:

$$y(t+\Delta t) \approx y(t) + \Delta t f\left(t, y\left(t\right)\right) \tag{3}$$

Agora iremos reescrever a equação anterior, substituindo y_n por $y(t)$; t por t_n e Δt por h. Logo, o método de Euler é expresso na forma a seguir para obtermos uma aproximação discretizada da solução y_{n+1} avaliada em cada ponto $n+1$.

$$y_{n+1} = y_n + hf\left(t_n, y_n\right) \tag{4}$$

Obviamente para utilizarmos o método é necessário fornecer uma condição inicial para o cálculo do primeiro ponto. Além disso, há um erro associado ao truncamento realizado que vai aumentando a cada iteração. Portanto, para melhorar a aproximação, devemos adotar outro procedimento. Isto pode ser realizado através de uma quadratura numérica (ver [1, 2]) que resulta na família de métodos Runge Kutta.

Para isto, há varias maneiras de estabelecer a função que resolve numericamente a EDO e que podem ser verificados em [1, 3, 4]. Mas este assunto vai muito além da escopo deste trabalho e por isso vamos simplesmente apresentar a equação com parâmetros arbitrados que o faz ser conhecido como método *midpoint* Runge-Kutta de dois estágios.

$$\begin{aligned} k_1 &= hf(t_i, y_i) \\ k_2 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right) \\ y_{i+1} &= y_i + hk_2 \end{aligned} \tag{5}$$

4 Aplicação

Muitos problemas físicos apresentam comportamento oscilatório como sistemas massa-mola, pêndulo simples, pêndulo de torção, circuito LC, oscilador harmônico quântico, etc. Em todos estes casos, quando não há termos responsáveis pela restauração e dissipação de energia, o sistema entra no regime de movimento harmônico simples[1], ou seja, passa a oscilar eternamente. Matematicamente as equações que governam este tipo de fenômeno tem a forma apresentada a seguir:

$$\frac{d^2}{dt^2}\psi(t) = -\omega^2\psi(t) \tag{6}$$

Onde o termo ω é definido como a frequência natural do sistema, t é variável independente e $\psi(t)$ é o termo dependente. Note também que é conveniente aplicar uma redução de ordem sobre a diferencial para utilização dos métodos numéricos. Isto porque eles foram elaborados para serem utilizados em sistemas de equações diferenciais de primeira ordem. Sendo assim, reescrevendo a expressão anterior, obtemos:

$$\begin{aligned} \frac{dz}{dt} &= -\omega^2\psi \\ \frac{d\psi}{dt} &= z \end{aligned} \tag{7}$$

As equações 6 e 7 apresentam a solução dada por 8, onde A é a amplitude de oscilação e ϕ é o ângulo de fase. Ambos os termos são arbitrados como condições iniciais do problema.

$$\psi(t) = A\cos(\omega t + \phi) \tag{8}$$

5 Simulações

Em geral, métodos numéricos são normalmente utilizados quando é dificil encontrar soluções analíticas de EDOs. Entretanto, no nosso caso esta é conhecida, então é conveniente

[1] Em alguns casos, como o do pêndulo simples, a equação diferencial costuma passar por um processo de linearização para que possa ser considerado movimento harmônico simples.

fazer uma comparação entre ela e os métodos Runge-Kutta e de Euler. Para isto, vamos estabelecer as condições de contorno do problema. Desta maneira, derivando a equação 8 em relação a t, obtemos $z(t)$ apresentada a seguir:

$$\frac{d}{dt}\psi(t) = z(t) = -A\omega \sin(\omega t + \phi) \tag{9}$$

Como A e ω são positivos, ao inspecionar as equações 8 e 9 e arbitrar $\psi(0) = 1$ e $z(0) = 0$, implica em $\phi = 0$ e $A = 1$. Além disso, adotamos a constante $\omega = 1.4$. E com isso, passamos a determinar todos os termos presentes na solução dada pela equação 8.

Para resolver numericamente a EDO, construímos dois programas em *Fortran90*, capazes de resolvê-la pelo método de Euler e Runge-Kutta de dois estágios ao aplicar as relações 4 e 5 respectivamente. Cada programa utiliza o primeiro ponto obtido pelas condições de contorno e a partir dele, a cada iteração, estabelece o valor do ponto seguinte. Obviamente foi necessário reduzir a ordem da derivada (ver a relação 7) e executar o algorítimo simultaneamente nas duas diferenciais. O leitor pode ver mais detalhes através códigos fontes apresentados em anexo ou no repositório disponível em [5].

6 Resultados e discussões

Ao compilar[1] e executar os programas, o terminal[2] mostra três colunas que representam t, $\psi(t)$ e $z(t)$ respectivamente. Ao plotarmos um gráfico de $\psi(t)$ em função de t contendo a curva teórica dada pela equação 8 e os dois métodos numéricos, obtemos as figuras 1 e 2 que diferem apenas pelo tamanho do passo h.

Na figuras 1 e 2, ao compararmos as soluções numéricas com a curva teórica, verificamos que parecem haver defasagens e um aumento progressivo nas amplitudes. Mas, na verdade, isto ocorre porque a cada iteração os métodos vão acumulando erros que aumentam com o tamanho do passo h. Ao diminuirmos este, o resultado numérico passa a se aproximar mais da solução analítica.

[1] O compilador utilizado é o *gfortran* 5.4.0

[2] *Linux Lite* 3.6

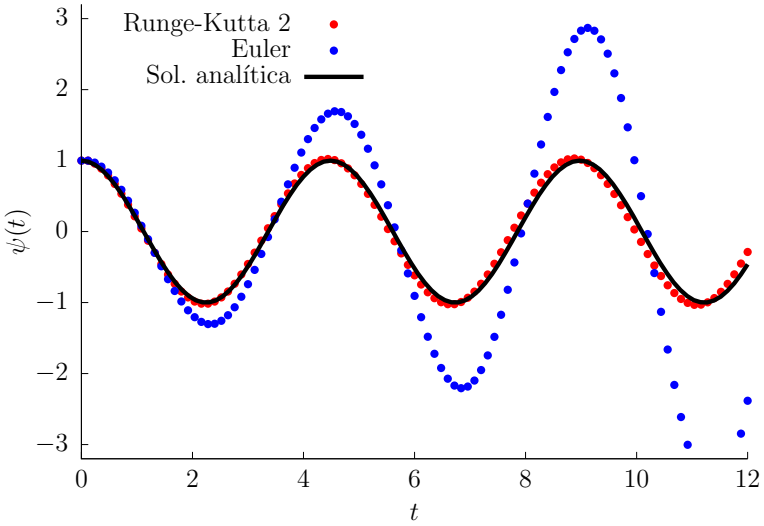


Figura 1: Solução numérica através dos métodos de Euler e Runge-Kutta de dois estágios seguido da analítica dada pela equação 8. Onde $A = 1$, $\phi = 0$ e $\omega = 1.4$. O tamanho do passo arbitrado é $h = 0.12$.

Fazendo uma comparação entre os métodos numéricos, verificamos que o de Euler se afasta da solução analítica com muito mais rapidez quando comparado com o Runge-Kutta. Isto ocorre porque a forma com que estes algorítimos são construídos levam em conta uma aproximação por quadratura numérica, fazendo com que o método de Euler se torne um caso particular (e de menor acurácia) da familia de métodos Runge-Kutta de qualquer estágio superior (ver [1, 2, 3]).

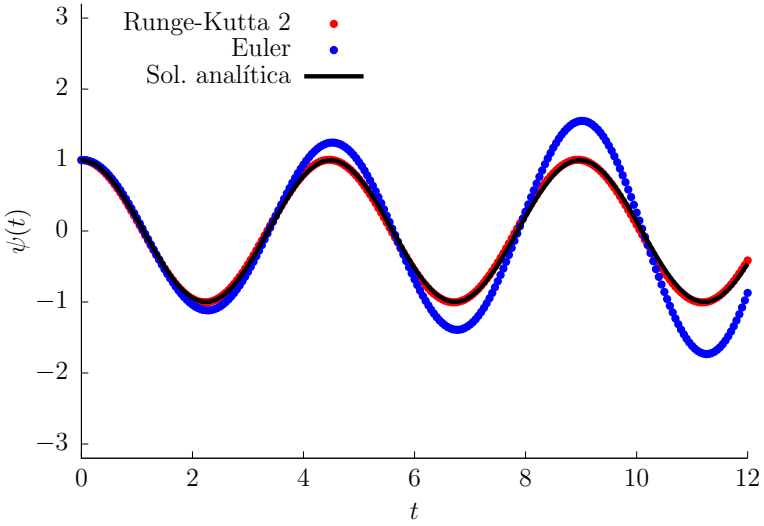


Figura 2: Solução numérica através dos métodos de Euler e Runge-Kutta de dois estágios seguido da analítica dada pela equação 8. Onde $A = 1$, $\phi = 0$ e $\omega = 1.4$. O tamanho do passo arbitrado é $h = 0.05$.

Diante do exposto, ao realizarmos as comparações, consideramos que o método

Runge-Kutta de dois estágios é mais eficaz, pois apresenta uma melhor aproximação da solução analítica. Entretanto, este possui uma equação adicional para ser resolvida a cada iteração e certamente ele exige um maior esforço computacional.

Referências

[1] ROSEN, J. S. *The Runge-Kutta equations by quadrature methods*. [S.l.], 1967. Report/Patent Number: NASA-TR-R-275, Document ID 19680000653. Disponível em: <https://ntrs.nasa.gov/search.jsp?R=19680000653>.

[2] ACKLEH, A. S. et al. Classical and modern numerical analysis: Theory, methods and practice (chapman & hall/crc numerical analysis and scientific computing series). In: . [S.l.]: Chapman and Hall/CRC, 2009. cap. 6, p. 381–459. ISBN 1420091573.

[3] ACKLEH, A. S. et al. Classical and modern numerical analysis: Theory, methods and practice (chapman & hall/crc numerical analysis and scientific computing series). In: . [S.l.]: Chapman and Hall/CRC, 2009. cap. 7, p. 381–459. ISBN 1420091573.

[4] HAIRER MICHEL ROCHE, C. L. a. E. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 1989. (Lecture Notes in Mathematics 1409). ISBN 9780387518602,0-387-51860-6.

[5] BRUM, A. L.; LOBO, P. H. F.; AVILA, V. H. *Repository on github*. Disponível em: <https://github.com/WX-78/EDOMetodosNumericos>.

Anexo A

```
module constantes
  implicit none
  real ::  omg = 1.4, psi0=1.0, z0=0.0
end module constantes

module derivadas ! Um modulo para definir as funcoes derivadas
! Esse modulo deve ser usado na subrotina que ira resolver seu
! sistema de EDOs
  implicit none
  contains
  function f(s,z) ! Aqui sao definidas as funcoes derivadas
    use constantes
    real :: s ! variavel independente
    real :: z(2) ! vetor com variaveis dependentes
    real :: f(2) ! cada f corresponde ao lado direito da EDO
                 ! dz/dx = f
    f(1) = z(2)  ! v
    f(2) = -z(1)*omg**2
  end function f
end module derivadas

module runge_kutta_2
  implicit none
  contains
  subroutine rk2(h,t,y0,y)
    use derivadas
    implicit none
    real :: y(:,:) !variaveis dependentes e derivadas
    real :: h      ! tamanho do passo
    real :: t(:)   !vetor de valores da variavel independente
    real :: y0(:)  ! condicoes iniciais
    integer :: Npassos, Neqs ! No. de passos e No. de equacoes
    integer :: i
    real, allocatable, dimension(:) :: k1,k2
    Npassos = size(t)
    Neqs = size(y0)
    allocate ( k1(Neqs), k2(Neqs))
    y(:,1)=y0 ! valores iniciais
    do i=1, Npassos-1
      k2 = f(t(i),y(:,i))
      k1 = f( t(i)+h/2.0, y(:,i) + (h/2.0)*k1 )
      y(:,i+1) = y(:,i) + h*k1
    end do
  end subroutine rk2
end module runge_kutta_2

program passing
  use constantes
  use runge_kutta_2
  implicit none
  real,parameter :: h=0.12 !alterar aqui o tamanho do passo
  integer, parameter :: Npts=2000+1
  real :: yinit(2)
  real :: y(2,Npts)
  integer :: i
```

```
56   real :: t(Npts)
57   t(1) = 0.0
58   do i = 2,Npts
59     t(i) = t(i-1)+h
60   end do
61   yinit(1) = psi0
62   yinit(2) = z0
63
64   call rk2(h,t,yinit,y)
65   write(*,*) "#    t                x              dx/dt"
66   do i=1,Npts
67     write(*,*) t(i), y(1,i), y(2,i)
68   end do
69 end program passing
```

Listagem 1: Código fonte em *Fortran90* para o método Runge-Kutta de dois estágios.

Anexo B

```
module constantes
  implicit none
  real ::  omg = 1.4, psi0=1.0, z0=0.0
end module constantes

module derivadas ! Um modulo para definir as funcoes derivadas
! Esse modulo deve ser usado na subrotina que ira resolver seu
! sistema de EDOs
  implicit none
  contains
  function f(s,z) ! Aqui sao definidas as funcoes derivadas
    use constantes
    real :: s ! variavel independente
    real :: z(2) ! vetor com variaveis dependentes
    real :: f(2) ! cada f corresponde ao lado direito da EDO
                 ! dz/dx = f
    f(1) = z(2)  ! v
    f(2) = -z(1)*omg**2
  end function f
end module derivadas

module euler
  implicit none
  contains
  subroutine metodo(h,t,y0,y)
    use derivadas
    implicit none
    real :: y(:,:)       ! variaveis dependentes e derivadas
    real :: h            ! tamanho do passo
    real :: t(:)         ! vetor de valores da variavel
                         ! independente
    real :: y0(:)        ! condicoes iniciais
    integer :: Npassos, Neqs !No. de passos e No. de equacoes
    integer :: i
    real, allocatable, dimension(:) :: k1,k2
    Npassos = size(t)
    Neqs = size(y0)
    allocate ( k1(Neqs), k2(Neqs))
    y(:,1)=y0 ! valores iniciais
    do i=1, Npassos-1
      y(:,i+1) = y(:,i) + h*f(t(i),y(:,i))
    end do
  end subroutine metodo
end module euler

program passing
  use constantes
  use euler
  implicit none
  real,parameter :: h=0.12 !alterar aqui o tamanho do passo
  integer, parameter :: Npts=2000+1
  real :: yinit(2)
  real :: y(2,Npts)
  integer :: i
  real :: t(Npts)
```

```
56    t(1) = 0.0
57    do i = 2,Npts
58      t(i) = t(i-1)+h
59    end do
60    yinit(1) = psi0
61    yinit(2) = z0
62    call metodo(h,t,yinit,y)
63    write(*,*) "#    t                x              dx/dt"
64    do i=1,Npts
65      write(*,*) t(i), y(1,i), y(2,i)
66    end do
67  end program passing
```

Listagem 2: Código fonte *Fortran90* para o método de Euler.