Universidade Federal de Santa Maria
Centro de Ciências Naturais e Exatas
Departamento de Física
Física Licenciatura Plena Noturno

Soluções Numéricas de Equações Diferenciais Ordinárias: Problema de Valor Inicial

Métodos Numéricos e Computacionais
Professor Tiago Martinuzzi Buriol
Acadêmico Dherik França Menezes
Matrícula 201413262

Santa Maria, 27 de Abril de 2018.

Prefácio

1. Motivação

Os Métodos estudados anteriormente mostraram como resolver equações diferenciais usando técnicas analíticas com Integração ou Expansão em Séries. A ênfase nos métodos anteriores era encontrar uma expressão exata para a solução.

Mas como se sabe, existem muitos problemas importantes em Física, Matemática e Engenharia, por exemplo, especialmente problemas não lineares, nos quais os métodos anteriores ou não se aplicam ou são muito complexos para serem utilizados.

Neste trabalho será apresentada uma abordagem alternativa, a utilização de métodos numéricos aproximados para se obter uma aproximação precisa da solução de um *Problema de Valor Inicial*, comumente chamado de **P.V.I.**.

Esses métodos serão apresentados de forma simples, ou seja, uma única equação escalar de primeira ordem. Os programas apresentados podem ser executados utilizando a linguagem de programação *Python*, em máquinas que executam distribuições *Linux* ou *Microsoft Windows*.

2. Introdução

Equações Diferenciais são utilizadas para modelar problemas que surgem em várias áreas do conhecimento que envolvam a alteração de algumas variáveis em relação a outra. A maioria destes problemas requer a solução de um P.V.I., isto é, a solução de uma equação diferencial que satisfaça determinada condição inicial.

Na maioria das situações encontradas na vida real, a equação diferencial que modela o problema é complexa demais para ser resolvida exatamente (*resolvida a mão*) e uma entre as duas abordagens é aplicada para aproximar a solução. A primeira abordagem é simplificar a equação diferencial para uma que possa ser resolvida exatamente e depois utilizar a solução da equação simplificada para aproximar a solução da equação original. A outra abordagem usa métodos de aproximação da solução do problema original. Essa é a abordagem empregada comumente, uma vez que os métodos de aproximação fornecem resultados mais precisos e informações mais realísticas sobre os erros.

Os métodos que serão apresentados neste trabalho não produzem uma aproximação no contínuo para a solução do problema de valor inicial. Em vez disso, as aproximações encontradas em determinados pontos especificados, e, frequentemente, igualmente espaçados.

3. Desenvolvimento

São necessárias algumas definições e resultados da *Teoria das Equações Diferenciais Ordinárias* antes de considerar métodos de aproximação de soluções de problemas de valores iniciais.

Os Problemas de Valores Iniciais obtidos por meio da observação de fenômenos físicos geralmente aproximam a situação verdadeira, de modo que é necessário saber se pequenas variações no enunciado do problema introduzem variações correspondentemente pequenas na solução da observação do fenômeno. Isso é importante devido a introdução de erro de arredondamento quando métodos numéricos são utilizados.

3.1 Problema de Valor Inicial

Para uma equação diferencial de *n*-ésima ordem, o problema em (1)

$$\textit{Resolva:} \quad a_n(x)\frac{d^n y}{dx^n} + a_{n-1}(x)\frac{d^{n-1}y}{dx^{n-1}} + \cdots + a_1(x)\frac{dy}{dx} + a_0(x)y = g(x) \tag{1}$$

$$\textit{Sujeita a:} \quad y(x_0) = y_0,\ \ y'(x_0) = y_0', \cdots, y^{(n-1)}(x_0) = y_0^{(n-1)} \tag{2}$$

em que $y_0, y_0', \cdots, y_0^{(n-1)}$, em (2), são constantes arbitrárias, é chamado de um ***Problema de Valor Inicial***. Os valores específicos $y(x_0) = y_0,\ \ y'(x_0) = y_0', \cdots, y^{(n-1)}(x_0) = y_0^{(n-1)}$ são chamados de *Condições Iniciais* – C.I., do problema de valor inicial. Procura-se uma solução em algum intervalo *I* contento x_0.

No caso de uma Equação Linear de Segunda Ordem – Equação (3), uma solução (4) – condição inicial, para o problema de valor inicial

$$a_2(x)\frac{d^2y}{dx^2} + a_1(x)\frac{dy}{dx} + a_0(x)y = g(x) \tag{3}$$

$$\begin{cases} y(x_0) = y_0 \\ y'(x_0) = y_0' \end{cases} \tag{4}$$

é uma função que satisfaça a equação diferencial no intervalo *I* cujo gráfico passa pelo ponto (x_0, y_0) com inclinação m igual a y_0'. Veja Figura 1 que ilustra soluções da equação diferencial (ED) ponto (x_0, y_0) com inclinação $m = y_0'$.

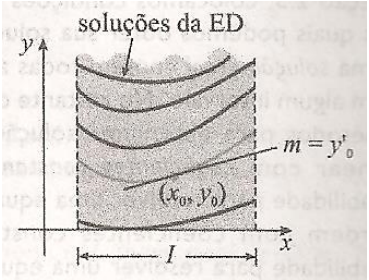

Figura 1: Soluções da ED e reta de inclinação contendo (x_0, y_0).

O próximo Teorema fornece condições suficientes para a existência de uma única solução para a Equação (2).

TEOREMA 1: Existência de uma Única Solução

Sejam $a_n(x), a_{n-1}(x), \cdots, a_1(x), a_0(x)$ e $g(x)$ contínuas em um intervalo *I* com $a_n(x) \neq 0$ para todo x neste intervalo. Se $x = x_0$ é algum ponto de deste intervalo, então existe uma única solução $y(x)$ para o *problema de valor inicial* (2) neste intervalo.

3.2 Análise do Problema de Valor Inicial

Em geral, uma análise de um P.V.I. simples (de grau um), é da forma descrita em (5):

$$\begin{cases} y' = f(x, y(x)) \\ y(x_0) = y(0) \end{cases} \tag{5}$$

Onde $y(x_0) = y(0)$ é a condição inicial para análise do P.V.I.

Para iniciar o estudo de Soluções Numéricas para Equações Diferenciais Ordinárias (EDO's) considera-se a resolução do seguinte P.V.I. com condição inicial $y(0) = 0$ mostrado abaixo:

$$\begin{cases} y' = y - 2 \\ y(0) = 0 \end{cases} \tag{6}$$

A análise do P.V.I. mostrado em (6) será feita de duas formas: (a) *Análise Quantitativa* e (b) *Análise Numérica*.

(a) ***Análise Quantitativa***

Para encontrar a Solução Geral do P.V.I., deve-se resolver o P.V.I. da Equação (6):

$$y' = y - 2 \tag{7}$$

$$\text{Sendo} \quad y' = \frac{dy}{dx} \tag{8}$$

Substituindo (8) em (7)

$$\Leftrightarrow \frac{dy}{dx} = y - 2 \tag{9}$$

Rearranjando os termos em (9)

$$\Leftrightarrow \frac{dy}{y - 2} = dx \tag{10}$$

continuação: Resolução do P.V.I. da Equação (6):

Integrando ambos os lados da igualdade em (10):

$$\Leftrightarrow \int \frac{dy}{y-2} = \int dx$$

$$\Leftrightarrow ln|y-2| + c_1 = x + c_2$$

$$\Leftrightarrow ln|y-2| = x + c_2 - c_1 \quad (11)$$

Sendo $C = c_2 - c_1$ (12)

Substituindo (12) em (11)

$$\Leftrightarrow ln|y-2| = x + C \quad (13)$$

Aplicando e em ambos os lados da igualdade em (13)

$$\Leftrightarrow e^{ln|y-2|} = e^{(x+C)}$$

$$\Leftrightarrow e^{ln|y-2|} = e^{(x+C)}$$

$$\Leftrightarrow y - 2 = e^x \cdot e^C \quad (14)$$

Definindo $e^C = A$ em (14) e substituindo em (14)

$$\Leftrightarrow y - 2 = e^x \cdot A \quad (15)$$

Reescrevendo (15)

$$\Leftrightarrow y(x) = Ae^x + 2 \quad (16)$$

Reescrevendo (16) para encontrar o valor da constante A

$$\Leftrightarrow y(x_0) = Ae^x + 2 \quad (17)$$

O resultado encontrado em (16) é a *solução geral do P.V.I.*.

Da condição inicial $y(0) = 0$, em que $x_0 = 0$ e $y_0 = 0$, apresentada em (6), calcula-se a constante A para encontrar a solução do P.V.I.

Então, substituindo a condição inicial em (17):

$$y(0) = 0$$

$$\Leftrightarrow 0 = Ae^0 + 2$$

Onde $e^0 = 1$

$$\Leftrightarrow 0 = A \cdot (1) + 2$$

$$\Leftrightarrow 0 = A + 2$$

$$\Leftrightarrow -2 = A$$

$$\Leftrightarrow A = -2 \quad (18)$$

Retomando a Equação (16), $y(x) = Ae^x + 2$, e substituindo o resultado encontrado em (18):

$$y(x) = Ae^x + 2$$

$$\Leftrightarrow y(x) = -2e^x + 2$$

Portanto, a *solução do P.V.I.* é:

$$\boxed{y_E(x) = -2e^x + 2} \quad (19)$$

Coloca-se um subíndice E na Equação (19), pois o resultado, quando define-se valores para o x, obtém-se o *valor real/exato*, o qual é proveniente da solução exata do P.V.I..

(b) ***Análise Numérica***

Sabe-se que, a partir da Equação (20):

$$y'(x) = \lim_{h \to 0} \frac{y(x+h) - y(x)}{h} \tag{20}$$

$$\Leftrightarrow y'(x) \cong \frac{y(x+h) - y(x)}{h}$$

$$\Leftrightarrow Y(x+h) \cong y(x) + h\,Y'(x) \tag{21}$$

Mas pelo P.V.I., reescreve-se o resultado em (21):

$$\Leftrightarrow y(x+h) \cong y(x) + h\,f(x, y(x)) \tag{22}$$

Para o nosso P.V.I.:

$$\Leftrightarrow y(x+h) \cong y(x) + h\,(y(x) - 2) \tag{23}$$

Mas, como $y_0 = 0$ e, sem perda de generalidade, considere o intervalo h, sendo $h - 0{,}1$, reescreve-se a Equação (21):

$$Y(x_n + h) = y(x_n) + h\,(y(x_n) - A) \tag{24}$$

ou

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{25}$$

Sendo $f(x_n, y_n)$ em (25) igual $y' = y - 2$, então

$$\Leftrightarrow y_{n+1} = y_n + h\,y'$$

$$\Leftrightarrow y_{n+1} = y_n + h(y_n - 2) \tag{26}$$

Reescrevendo (24)

$$y(x_n + h) = y(x_n) + h\,(y(x_n) - A) \tag{27}$$

Substituindo $x_n = 0$; $h = 0{,}1$; $y(x_n) = 0$ e $A = -2$ em (27)

$$\Leftrightarrow y(0 + 0{,}1) = y(0) + 0{,}1\,(y(0) - 2)$$

Com $y(0) = 0$

$$\Leftrightarrow y(0{,}1) = 0 + 0{,}1\,(0 - 2)$$

$$\Leftrightarrow y(0{,}1) = 0{,}1\,(-2)$$

$$\Leftrightarrow y(0{,}1) = -0{,}2 \tag{28}$$

Conforme o P.V.I., $y(0) = 0 \Leftrightarrow y(x_0) = y_0$, portanto, $x_0 = 0$ e $y_0 = 0$, exatamente como mostra (28).

Para, $x_0 = 0$, $h = 0$ (pois está na origem 0) e $A = 2$ aplicando na Equação (27) encontra-se o valor de $y(0)$:

$$y(x_n + h) = y(x_n) + h\,(y(x_n) - A)$$

$$\Leftrightarrow y(x_0 + h) = y(x_0) + h\,(y(x_0) - A)$$

$$\Leftrightarrow y(0 + 0) = y(0) + 0\,(y(0) - 2)$$

$$\Leftrightarrow y(0) = 0 + 0\,(0 - 2)$$

$$\Leftrightarrow y(0) = 0 + 0\,(-2)$$

$$\Leftrightarrow y(0) = 0 + 0$$

$$\Leftrightarrow y(0) = 0 \tag{29}$$

Para $h = 0{,}1$ e $A = 2$ aplicando na Equação (27), obtém-se o resultado, $y(0{,}1)$, mostrado em (28).

Para calcular a próxima estimativa, utiliza-se $h = 0{,}1$, $A = 2$, $x_1 = 0{,}1$ e $y(x_1) = -0{,}2$ – resultado (28), e aplica-se na Equação (27),

$$\begin{gathered} y(x_n + h) = y(x_n) + h\,(y(x_n) - A) \\ \Leftrightarrow y(x_1 + h) = y(x_1) + h\,(y(x_1) - A) \\ \Leftrightarrow y(0{,}1 + 0{,}1) = y(0{,}1) + 0{,}1\,(y(0{,}1) - 2) \\ \Leftrightarrow y(0{,}2) = -0{,}2 + 0{,}1\,(-0{,}2 - 2) \\ \Leftrightarrow y(0{,}2) = -0{,}2 + 0{,}1\,(-2{,}2) \\ \Leftrightarrow y(0{,}2) = -0{,}2 - 0{,}22 \\ \Leftrightarrow y(0{,}2) = -0{,}42 \end{gathered} \tag{30}$$

O resultado obtido em (30) significa $y(x_2 = 0{,}2) = -0{,}42$. Este resultado será utilizado para encontrar o próximo y.

A estimativa seguinte, com $h = 0{,}1$, $A = 2$ e $y(x_2 = 0{,}2) = -0{,}42$, é encontrada utilizando novamente a Equação (27):

$$\begin{gathered} y(x_n + h) = y(x_n) + h\,(y(x_n) - A) \\ \Leftrightarrow y(x_2 + h) = y(x_2) + h\,(y(x_2) - A) \\ \Leftrightarrow y(0{,}2 + 0{,}1) = y(0{,}2) + 0{,}1\,(y(0{,}2) - 2) \\ \Leftrightarrow y(0{,}3) = -0{,}42 + 0{,}1\,(-0{,}42 - 2) \\ \Leftrightarrow y(0{,}3) = -0{,}42 + 0{,}1\,(-2{,}42) \\ \Leftrightarrow y(0{,}3) = -0{,}42 - 0{,}242 \\ \Leftrightarrow y(0{,}3) = -0{,}662 \end{gathered} \tag{31}$$

Reunindo os resultados obtidos em (28), (29), (30) e (31), apresentamos a Tabela 1:

Tabela 1: Comparação entre os valores da aproximação e solução exata do P.V.I..

x_n	$y_n(x)$ (aproximação)	$y_E(x)$ Valor real/exato (vem da solução exata do P.V.I.) $y_E(x) = -2e^x + 2$	**Erro Absoluto – EA** $EA = \lvert y_E(x) - y_n(x)\rvert$	**Erro Relativo – ER** $ER = \dfrac{\lvert EA \cdot 100\%\rvert}{\lvert y_E(x)\rvert}$
$x_0 = 0{,}0$	$y_0 = y(0{,}0) = 0{,}0$	$y_E(0{,}0) = 0{,}0$	0,0000	0,00%
$x_1 = 0{,}1$	$y_1 = y(0{,}1) = -0{,}2$	$y_E(0{,}1) = -0{,}2103$	0,0103	~ 4,90%
$x_2 = 0{,}2$	$y_2 = y(0{,}2) = -0{,}42$	$y_E(0{,}2) = -0{,}4428$	0,0228	~ 5,15%
$x_3 = 0{,}3$	$y_3 = y(0{,}3) = -0{,}662$	$y_E(0{,}0) = -0{,}6997$	0,0377	~ 5,39%

A Tabela 1 reúne os valores da aproximação de $y_n(x)$, o valor $y_E(x)$ – que é o valor real/exato proveniente da solução exata do P.V.I., cujos valores são calculados pela Equação (19), o *Erro*

Absoluto (vindo do módulo da diferença entre $|y_E(x) - y_n(x)|$) e o *Erro Relativo* mostra o quanto, em porcentagem, o valor do *Erro Absoluto* é em relação ao valor calculado pela solução exata do P.V.I..

A Figura 2 compara as soluções por aproximação e pela solução do P.V.I..

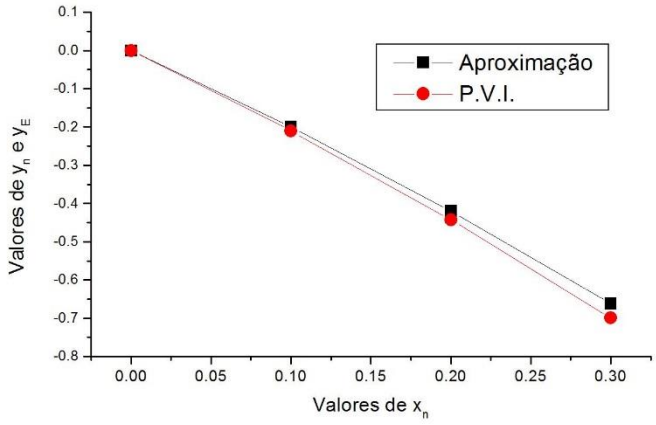

Figura 2: Gráfico comparando as soluções gerado no *Microcal Origin*.

Conforme Figura 2, vê-se que os resultados da Aproximação obtidos em (28), (29), (30) e (31), apresentados ao longo da curva com o símbolo ■, ficam muito próximos aos valores da solução exata do P.V.I., representado pelo símbolo ● ao longo da curva.

A partir da Análise Numérica feita anteriormente para a obtenção dos resultados (28), (29), (30) e (31), apresentaremos o *Método de Euler* como sendo o método utilizado para determinar as aproximações. Este método, *de Euler*, e outros que serão apresentados, constituem uma série de métodos para obter as Soluções Numéricas de Equações Diferenciais Ordinárias para o Problema de Valor Inicial.

3.3 Polinômios de Taylor[1]

Percebe-se que

$$y(x_0 + h) \cong y(x_0) + h\, y'(x_0) \tag{32}$$
Onde $x = x_0 + h$ e $f(x_0, y_0) = y'(x_0)$

Essa aproximação, descrita na Equação (32), é o *polinômio de Taylor de Grau 1* proveniente da *Expansão de Taylor* da função $y(x_0 + h)$.

De fato,

$$y(x_0 + h) = y(x_0) + (x_0 + h - x_0) \cdot y'(x_0) + \cdots \tag{33}$$
$$\Leftrightarrow y(x_0 + h) = y(x_0) + h\, y'(x_0) + Oh^2 \tag{34}$$

[1]**Brook Taylor** (1685-1731) foi responsável por adicionar na Matemática um novo ramo agora chamado o "Cálculo das Diferenças Finitas". Inventou a integração por partes e descobriu a célebre fórmula conhecida como a expansão de Taylor, de qual a importância permaneceu não reconhecida até 1772, quando Lagrange proclamou isto como o princípio básico do Cálculo Diferencial.

Onde Oh^2, na Equação (34), é *Erro de Truncamento Local – ETL* – para o polinômio de Taylor de Grau 1. Esse erro ocorre devido a desprezar-se os valores da expansão que aparecem no final da Equação (33), referidos por ⋯, ao considerar apenas os termos $y(x_0 + h) = y(x_0) + h\,y'(x_0) +$.

A partir da Equação (34), encontra-se os Polinômios de Grau Maior que 1, sendo eles:

$$\boxed{y_{n+1} = y_n + h\,f(x_n, y_n) + \frac{h^2}{2!}y''_n(x)} \tag{35}$$

Onde $y''_n(x)$ representa o erro de truncamento local denotado por Oh^3

$$\boxed{y_{n+1} = y_n + h\,f(x_n, y_n) + \frac{h^2}{2!}y''_n(x) + \frac{h^3}{3!}y'''_n(x)} \tag{36}$$

Onde $y''_n(x)$ representa o erro, denotado por Oh^3
e $y'''_n(x)$ representa o erro denotado por Oh^4

As Equações (35) e (36) são denotadas como *Expressões 'primeiras' do Método de Taylor*. Pois derivam-se da Equação (34).

3.4 Métodos de Euler[2]

Com os Polinômios de Taylor, vistos anteriormente, constrói-se os demais Métodos de Euler (ou de Taylor), os quais serão vistos a seguir.

3.4.1 Método de Euler ou Método de Taylor de Ordem 1

O Método de Euler também é conhecido como *Método de Taylor* ou *Método da Reta Tangente (ou das Tangentes)* ou *Método de Diferenças Elementares*.

Este método constitui uma das técnicas mais simples para aproximar soluções de equações diferencias. Supor que se queira aproximar a solução do problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0 \tag{37}$$

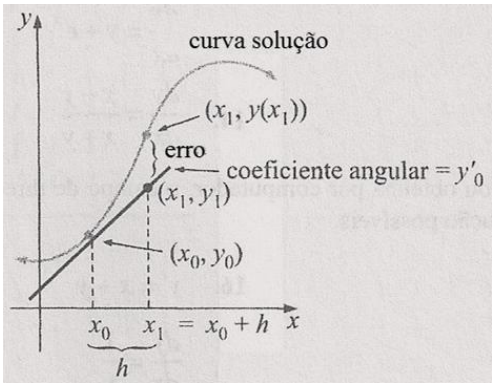

Figura 3: Ilustração do Método de Euler com a curva solução, erro e coeficiente angular.

[2]**Leonard Euler** (1707-1783), o mais prolífico dos matemáticos, foi o primeiro a apresentar ao público da área o uso de métodos de diferenças elementares para encontrar aproximações de soluções de equações diferenciais.

Sendo h um incremento (passo) positivo no eixo-*x*, então, conforme ilustra a Figura 4, encontra-se um ponto $(x_1, y_1) = (x_0 + h, y_1)$ na tangente à curva solução desconhecida em (x_0, y_0).

A partir da Figura 4, o coeficiente angular da reta será:

$$\frac{y_1 - y_0}{(x_0 + h) - x_0} = y'_0 \tag{38}$$

ou

$$y_1 = y_0 + h y'_0 \tag{39}$$

onde $y'_0 = f(x_0, y_0)$. Rotula-se $x_0 + h$ como x_1 e o ponto (x_1, y_1) na tangente é uma aproximação do ponto $\big(x_1, y(x_1)\big)$ da curva solução; isto é, $y_1 \approx y(x_1)$. Naturalmente, a precisão da aproximação depende do tamanho do incremento (passo) h. De modo geral, se deve escolher esse tamanho do passo *razoavelmente pequeno* para se obter uma melhor aproximação do valor exato da solução do P.V.I..

Ao supor um valor uniforme (constante) de h, pode-se obter uma sucessão de pontos, veja Figura 4, $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ que devem estar próximos s pontos $(x_1, y(x_1)), (x_2, y(x_2)), \cdots, (x_n, y(x_n))$.

Utilizando (x_1, y_1), obtém-se o valor de y_2, que é a ordenada de um ponto em uma *nova tangente*.

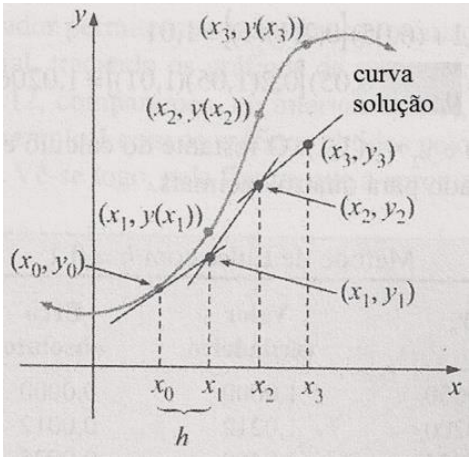

Figura 4: Sucessão de pontos com passo constante.

$$\frac{y_2 - y_1}{h} = y'_1 \tag{40}$$

ou

$$y_2 = y_1 + h\, y'_1 = y_1 + h\, f(x_1, y_1) \tag{41}$$

De modo geral, decorre que

$$y_{n+1} = y_n + hy'_n \quad (42)$$

$$\boxed{\Leftrightarrow y_{n+1} = y_n + hf(x_n, y_n)} \quad (43)$$

Onde $x_n = x_0 + nh$
e $f_n = f(x_n, y_n)$
$n = 0, 1, 2, \cdots$

Portanto, a Equação (43) é o ***Método de Taylor de Ordem 1*** ou ***Método de Euler*** ou ***Método das Tangentes*** ou ***Método da Reta Tangente*** ou ***Método de Diferenças Elementares***:

$$y(x_n + h) = y(x_n) + h \cdot f(x_n, y_n)$$

$$\boxed{\Leftrightarrow y_{n+1} = y_n + hf_n} \qquad \begin{cases} f_n = f(x_n, y_n) \\ n = 0, 1, 2, \cdots \end{cases} \quad (44)$$

3.4.1.1 Exemplo de Aplicação 1 – Método de Taylor de Ordem 1

A seguir, segue um código de programação para a solução de um P.V.I., utilizando o programa computacional *Python*, que ilustra o Método de Taylor de Ordem 1. Esse código gera um gráfico que compara a aproximação da solução da equação diferencial com a solução exata do P.V.I..

Use o Método de Euler para integrar numericamente a equação: $\frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8$ de $x = 0$ a $x = 4$ com um tamanho de passo de $h = 0{,}5$. A condição inicial em $x = 0$ é $y = 1$. A solução exata é dada por $y = -5.x^4 + 4x^3 - 10x^2 + 8.5x + 1$.

Solução:

- Com um passo $h = 0{,}5$, tem-se $x_0 = 0{,}0; x_1 = 0{,}5; x_2 = 1{,}0; x_3 = 1{,}5; \ldots; x_8 = 4{,}0$. Assim, a partir da equação

$$y_{i+1} = y_i + f(x_i, x_{i+1})h$$

com

$$f(x, y) = \frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8{,}5$$

e sabendo que em $x_0 = 0$ temos $y_0 = 1$, obtém-se os valores de $y_1, y_2, y_3, \ldots, y_8$.

- A seguir, o código computacional, em Python, que resolve o P.V.I. indicado acima.

continuação: Exemplo de Aplicação 1 – Método de Taylor de Ordem 1

```
% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

h = 0.5 #passo
x0 = 0.0
xn = 4.0

# criamos a lista com os pontos xi
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# lista com y0, a ser preenchida pelos yi
Y = [1.0]

# entramos com f(x,y)
f = lambda x: -2*x**3 + 12*x**2 - 20*x + 8.5

# laço para ir calculando os yi e preenchendo a lista
for x in X[0:-1]:
    Y.append(Y[-1] + f(x)*h ) #Y[-1] é o último yi calculado

# imprime
print ("pontos yi:", Y)

# plota
plt.plot(X, Y, "r-")
plt.grid()
plt.show()
```

```
pontos xi: [ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4. ]
pontos yi: [1.0, 5.25, 5.875, 5.125, 4.5, 4.75, 5.875, 7.125, 7.0]
```

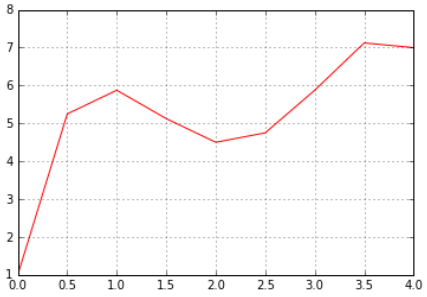

Figura 5: Gráfico com pontos da aproximação.

```
# plotando a solução exata para comparar
Xex = np.arange(0, 4.1, 0.1)
Yex = []
sol_ex = lambda x: -0.5*x**4 + 4*x**3 - 10*x**2 + 8.5*x + 1.0

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

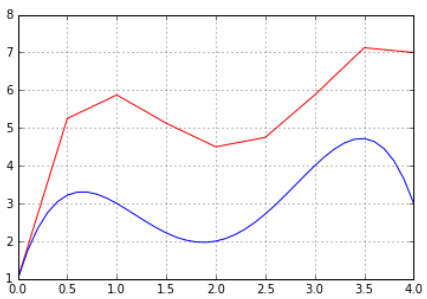

Figura6: Gráfico comparando os pontos da aproximação com a solução exata do P.V.I..

Nesse exemplo usamos um polinômio simples para a equação diferencial para facilitar a análise de erro a seguir. Logo,

$$\frac{dy}{dx} = f(x)$$

Um caso mais geral envolveria EDOs que dependam de x e y,

$$\frac{dy}{dx} = f(x, y)$$

Observações:

1. O exemplo e o código computacional foram copiados do *Repositório de material para a disciplina de Métodos Numéricos da UFSM* editado e mantido pelo Professor Tiago Buriol.

Endereço da página: https://github.com/tiagoburiol

Endereço do conteúdo: https://github.com/tiagoburiol/NUMETHODS

2. Para os demais Métodos de Taylor, *Ordem 2*, *3* e *P*, não serão apresentados Exemplos de Aplicação e códigos computacionais, mas apenas os resumos destes métodos, pois o intuito deste trabalho são os Métodos de Runge-Kutta.

3.4.2 Método de Taylor de Ordem 2

$$\boxed{y_{n+1} = y_n + hf_n + \frac{h^2}{2!} f'_n} \tag{45}$$

$$f_n = f(x_n, y_n)$$
$$f'_n = f_x(x_n, y_n) + f_y(x_n, y_n) \cdot f(x_n, y_n)$$
$$n = 0, 1, 2, \cdots$$

3.4.3 Método de Taylor de Ordem 3

$$\boxed{y_{n+1} = y_n + hf_n + \frac{h^2}{2!}f'_n + + \frac{h^3}{3!}f''_n} \tag{46}$$

$$f_n = f(x_n, y_n)$$
$$f'_n = f_x(x_n, y_n) + f_y(x_n, y_n) \cdot f(x_n, y_n)$$
$$f''_n = f_{xx}(x_n, y_n) + 2f_{xy}(x_n, y_n) \cdot f(x_n, y_n) +$$
$$+f_{yy}(x_n, y_n) \cdot f(x_n, y_n) + \left(f_y(x_n, y_n)\right)^2 \cdot f(x_n, y_n) +$$
$$+f_x(x_n, y_n) \cdot f_y(x_n, y_n)$$
$$n = 0, 1, 2, \cdots$$

3.4.4 Método de Taylor de Ordem P

$$\boxed{\begin{aligned} y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + + \frac{h^3}{3!}y'''(x_n) + \\ + \cdots + \frac{h^P}{P!}y^{(P)}(x_n) + \frac{h^{(P+1)}}{(P+1)!}y^{(P+1)}\xi_n \end{aligned}} \tag{47}$$

$$x_n < \xi_n < x_n + h$$

3.5 Métodos de Runge[3]-Kutta[4]

Provavelmente um dos mais populares, e mais precisos também, procedimentos numéricos utilizados para obtenção de soluções aproximadas para um problema de valor inicial de primeira ordem $y' = f(x, y)$, $y(x_0) = y_0$ são os ***Métodos de Runge-Kutta***.

Os Métodos de Taylor que foram apresentados anteriormente têm a propriedade desejável de erro de truncamento local – ETL – de alta ordem, mas apresentam a desvantagem de exigir a determinação e o cálculo em diversos pontos das derivadas de $f(x, y)$. Esse é um procedimento complicado e que consome muito tempo na maioria dos problemas, de modo que os Métodos de Taylor, na prática, nem sempre são utilizados.

Ao optar pela utilização dos Métodos de Runge-Kutta, estes têm o erro de truncamento local – ETL – de alta ordem dos Métodos de Taylor ao mesmo tempo em que elimina, a necessidade de calcular as derivadas de $f(x, y)$.

A seguir, serão apresentados os demais Métodos de Runge-Kutta de diferentes ordens.

---

[3]**Carl David Tolmé Runge** (1856-1927), matemático e físico alemão, trabalhou muitos anos em espectroscopia. A análise de dados o levou a considerar problemas em computação numérica, e o Método de Runge-Kutta teve origem em seu artigo sobre soluções numéricas de equações diferenciais de 1895. O método foi estendido para sistemas de equações em 1901 por [4]**Martin Wilhelm Kutta** (1867-1944). Kutta era um matemático alemão que trabalhava com aerodinâmica e é, também, muito conhecido por suas contribuições importantes à Teoria Clássica do Aerofólio.

3.5.1 Método de Runge-Kutta de Ordem R

A formulação geral para o Método de Runge-Kutta de Ordem R é:

$$\boxed{y_{n+1} = y_n + h \cdot f(x_n, y_n, h)} \tag{48}$$

$$com \quad f(x, y, h) = \sum_{r=1}^{R} c_r k_r$$

$$e \quad k_1 = f(x, y)$$

$$k_r = f\left(x + a_r h, y + h \sum\nolimits_{s=1}^{r-1} b_{rs}\, k_s\right); \quad r = 2, 3, \cdots, R$$

$$a_r = \sum\nolimits_{s=1}^{r-1} b_{rs}\,; \quad r = 2, 3, \cdots, R$$

Sendo que a partir da Equação (48) obtém-se as demais ordens do Método de Runge-Kutta.

3.5.2 Método de Runge-Kutta de Ordem 1

Fundamentalmente, todos os Métodos de Runge-Kutta são generalizações da fórmula básica de Euler, Equação (49) – ver Equação (44), página 13 – na qual a função coeficiente angular f é substituída por uma média ponderada dos coeficientes angulares sobre o intervalo $x_n \leq x \leq x_{n+1}$. Istó é,

$$y_{n+1} = y_n + h \cdot (w_1 k_1 + w_2 k_2 + \cdots + w_m k_m) \tag{49}$$

Sendo $w_1 k_1 + w_2 k_2 + \cdots + w_m k_m$ a média ponderada

Aqui w_i, $i = 1, 2, \cdots, m$ são constates que geralmente satisfazem $w_1 + w_2 + \cdots + w_m = 1$, e cada k_i, $i = 1, 2, \cdots, m$ é a função f calculada em um ponto selecionado (x, y) para o qual $x_n \leq x \leq x_{n+1}$. Será visto que os k_i são definidos recursivamente. O número m é chamado de ordem do método. É importante observar que adotando $m = 1$, $w_1 = 1$ e $k_1 = f(x, y)$, obtém-se a fórmula de Euler $y_{n+1} = y_n + hf(x_n, y_n)$.

Consequentemente, o Método de Euler é dito ser chamado de *Método de Runge-Kutta de Ordem 1*.

A média ponderada, Equação (49), é formada por parâmetros escolhidos de modo que (49) esteja de acordo dom um Polinômio de Taylor de grau m.

Então, se uma função $y(x)$ possui $k+1$ derivadas que são contínuas em um intervalo aberto contendo a e x, então escreve-se:

$$y(x) = y(a) + y'(a)\frac{x-a}{1!} + y''(a)\frac{(x-a)^2}{2!} + \cdots + y^{(k+1)}(c)\frac{(x-a)^{k+1}}{(k+1)!} \qquad (50)$$

onde c é algum número entre a e x. Substituindo a por x_n e x por $x_{n+1} = x_n + h$, então reescreve-se a Equação (50):

$$y(x_{n+1}) = y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \cdots + \frac{h^{k+1}}{(k+1)!}y^{(k+1)}(c) \qquad (51)$$

onde c é agora algum número entre x_n e x_{n+1}. Quando $y(x)$ for uma solução de $y' = f(x,y)$, no caso $k = 1$, e o resto $\frac{1}{2}h^2y''(c)$ for pequeno, vê-se que o Polinômio de Taylor $y(x_{n+1}) = y(x_n) + hy'(x_n)$ de **grau um** concorda com a equação de aproximação do Método de Euler:

$$\begin{aligned} & y_{n+1} = y_n + hy'_n \\ \Leftrightarrow\ & \boxed{y_{n+1} = y_n + hf(x_n, y_n)} \end{aligned} \qquad (52)$$

$$x_n = x_0 + nh$$
$$n = 0, 1, 2, \cdots$$

3.5.3 Métodos de Runge-Kutta de Ordem 2

Para continuar a ilustrar (49), considere agora o Método de Runge-Kutta de 2ª Ordem. Este Método consiste em obter constantes, ou parâmetros, c_1 e c_2.

Partindo da Fórmula Geral:

$$\boxed{y_{n+1} = y_n + h[c_1K_1 + c_2K_2]} \qquad (53)$$

$$com \quad \begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + a_2h, y_n + ha_2K_1) \end{cases}$$

As constantes c e a envolvidas são soluções do sistema:

$$\begin{cases} c_1 + c_2 = 1 \\ c_1 + c_2 = 1/2 \end{cases} \left[\begin{matrix} 2\ equações \\ 3\ incógnitas \end{matrix}\right\} infinitas\ possibilidades$$

As duas soluções mais usuais são:

$$(i)\quad c_1 = 0; c_2 = 1 \quad e \quad a_2 = 1/2$$
$$(ii)\quad c_1 = c_2 = 1/2 \quad e \quad a_2 = 1$$

3.5.3.1 Método de Euler Modificado – Runge-Kutta de Ordem 2

Para a escolha (i), obtém-se:

$$\boxed{y_{n+1} = y_n + hK_2} \tag{54}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^1/_2\, h, y_n + {}^1/_2\, hK_1\right)$$
$$n = 0, 1, 2, \cdots$$

3.5.3.2 Método de Euler Melhorado – Runge-Kutta de Ordem 2

Para a escolha (ii), obtém-se:

$$\boxed{y_{n+1} = y_n + {}^h/_2\,[K_1 + K_2]} \tag{55}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f(x_n + h, y_n + hK_1)$$
$$n = 0, 1, 2, \cdots$$

3.5.3.3 Exemplo de Aplicação 2 – Método de Runge-Kutta de Ordem 2

A seguir, um exemplo que compara as aproximações obtidas pelo Método de Euler Melhorado e pelo Método de Euler Modificado.

Dado o P.V.I.:

$$\begin{cases} y' = x - y + 2 \\ y(0) = 2 \\ \in [0; 0{,}3]\ e\ h = 0{,}1 \end{cases}$$

Calcule a solução numérica pelo Método de Euler Modificado e pelo Método de Euler Melhorado.

Solução:

- Omitiu-se a resolução da Equação Diferencial de Ordem 1 e apresenta-se a solução exata do P.V.I.: $y(x) = x + 1 + e^x$

- Tabela Demonstrativa:

Tabela 1: Comparação entre os valores da aproximação e solução exata do P.V.I..

$\boldsymbol{n}$	$\boldsymbol{x_n}$	$\boldsymbol{y_n(x)}$ (aproximação) (i) Método de Euler Modificado	$\boldsymbol{y_n(x)}$ (aproximação) (ii) Método de Euler Melhorado	$\boldsymbol{y_E(x)}$ Valor real/exato (vem da solução exata do P.V.I.) $y_E(x) = x + 1 + e^x$
0	0,0	2,000	2,000	2,00000
1	0,1	2,005	2,005	2,00484
2	0,2	2,0190	2,014975	2,01873
3	0,3	2,0412	2,03757	2,04082

- Método (I) – Método de Euler Modificado:

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + 0{,}1 \cdot K_2 \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + 0{,}05; y_n + 0{,}05 \cdot K_1) \end{cases} \qquad n = 0, 1, 2, \cdots$$

$\boldsymbol{n = 0}$	$y_1 = y_0 + 0{,}1\ K_2$
	$K_1 = f(x_0, y_0)$
	$K_2 = f(x_0 + 0{,}05; y_0 + 0{,}05\ K_1)$
$K_1 = f(x_0, y_0) = f(0{,}2) = 0 - 2 + 2 = 0$	
$K_2 = f(x_0 + 0{,}05; y_0 + 0{,}05 \cdot K_1) = f(0{,}05; 2 + 0{,}05 \cdot 0) = f(0{,}05; 2) = 0{,}05 - 2 + 2 = 0{,}05$	
$y_1 = y_0 + 0{,}1 \cdot K_1 = 2 + (0{,}1 \cdot 0{,}05) = 2{,}005$	

$\boldsymbol{n = 1}$	$y_2 = y_1 + 0{,}1\ K_2$
	$K_1 = f(x_1, y_1)$
	$K_2 = f(x_1 + 0{,}05; y_1 + 0{,}05\ K_1)$
$K_1 = f(0{,}1; 2{,}005) = 0{,}1 - 2{,}005 + 2 = 0{,}095$	
$K_2 = f(0{,}1 + 0{,}05;\ 2{,}005 + 0{,}05(0{,}095)) = f(0{,}15; 2{,}00975) = 0{,}15 - 2{,}00975 + 2 = 0{,}14025$	
$y_2 = 2{,}005 + 0{,}1(0{,}14025) = 2{,}019025$	

$\boldsymbol{n = 2}$	$y_3 = y_2 + 0{,}1\ K_2$
	$K_1 = f(x_2, y_2)$
	$K_2 = f(x_2 + 0{,}05; y_2 + 0{,}05\ K_1)$
$K_1 = f(0{,}2; 2{,}0190) = 0{,}2 - 2{,}0190 + 2 = 0{,}181$	
$K_2 = f(0{,}2 + 0{,}05; 2{,}0190 + 0{,}05(0{,}181)) = f(0{,}25; 2{,}02805) = 0{,}25 - 2{,}02805 + 2 \cong 0{,}222$	
$y_3 = 2{,}0190 + 0{,}1(0{,}222) = 2{,}0412$	

- Método (II) – Método de Euler Melhorado:

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + h/2 \cdot [K_1 + K_2] = y_n + 0{,}1/2 \cdot [K_1 + K_2] = y_n + 0{,}05 \cdot [K_1 + K_2] \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + 0{,}1; y_n + 0{,}1 \cdot K_1) \end{cases}$$
$n = 0, 1, 2, \cdots$

continuação: Método (II) – Método de Euler Melhorado

$\boldsymbol{n = 0}$	$y_1 = y_0 + 0{,}05 \cdot [K_1 + K_2]$
	$K_1 = f(x_0, y_0)$
	$K_2 = f(x_0 + 0{,}1; y_0 + 0{,}1\ K_1)$
$K_1 = f(0{,}2) = x_0 - y_0 + 2 = 0 - 2 + 2 = 0$	
$K_2 = f(x_0 + 0{,}1; y_0 + 0{,}1 \cdot K_1) = f(0 + 0{,}1; 2 + 0{,}1) = f(0{,}1; 2) = 0{,}1 - 2 + 2 = 0{,}1$	
$y_1 = y_0 + 0{,}05 \cdot [0 + 0{,}1] = 2 + (0{,}05 \cdot 0{,}1) = 2{,}005$	

$\boldsymbol{n = 1}$	$y_2 = y_1 + 0{,}05 \cdot [K_1 + K_2]$
	$K_1 = f(x_1, y_1)$
	$K_2 = f(x_1 + 0{,}1; y_1 + 0{,}1\ K_1)$
$K_1 = f(x_1, y_1) = f(0{,}1; 2{,}005) = 0{,}1 - 2{,}005 + 2 = 0{,}1 - 0{,}005 = 0{,}095$	
$K_2 = f(x_1 + 0{,}1; y_1 + 0{,}1 \cdot K_1) = f(0{,}1 + 0{,}1; 2{,}005 + (0{,}1 \cdot 0{,}095)) = f(0{,}2; 2{,}0146)$ $= 0{,}2 - 2{,}0146 + 2 = 0{,}1854$	
$y_2 = y_1 + 0{,}05 \cdot [K_1 + K_2] = 2{,}005 + 0{,}05\ [0{,}005 + 0{,}1854] = 2{,}005 + (0{,}005 \cdot 0{,}1904) = 2{,}01452$	

$\boldsymbol{n = 2}$	$y_3 = y_2 + 0{,}05 \cdot [K_1 + K_2]$
	$K_1 = f(x_2, y_2)$
	$K_2 = f(x_2 + 0{,}1; y_2 + 0{,}1\ K_1)$
$K_1 = f(x_2, y_2) = f(0{,}2; 2{,}015) = 0{,}2 - 2{,}015 + 2 = 0{,}2 - 0{,}015 = 0{,}185$	
$K_2 = f(x_2 + 0{,}1; y_2 + 0{,}1 \cdot K_2) = f(0{,}2 + 0{,}1; 2{,}015 + (0{,}1 \cdot 0{,}185)) = f(0{,}3; 2{,}0335)$ $= 0{,}3 - 2{,}0335 + 2 = 0{,}2665$	
$y_2 = y_2 + 0{,}05 \cdot [K_1 + K_2] = 2{,}015 + 0{,}05\ [0{,}185 + 0{,}2665] = 2{,}03757$	

3.5.4 Métodos de Runge-Kutta de Ordem 3

Assim como no Método de Runge-Kutta de 2ª Ordem, este Método também consiste em obter constantes, ou parâmetros, c_1, c_2 e c_3.

Partindo da Fórmula Geral:

$$\boxed{y_{n+1} = y_n + h[c_1K_1 + c_2K_2 + c_3K_3]} \tag{56}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f(x_n + ha_2, y_n + hb_{21}K_1),\ b_{21} = a_2$$
$$K_3 = f(x_n + ha_3, y_n + hb_{31}K_1 + hb_{32}K_2)$$
$$a_2 = b_{21} \quad e \quad a_3 = b_{31} + b_{32}$$

Cujas constantes, estão relacionadas no sistema abaixo:

$$\begin{cases} c_1 + c_2 + c_3 = 1 \\ c_2a_2 + c_3a_3 = 1/2 \\ c_3b_{32}a_2 = 1/6 \\ c_2a_2^2 + c_3a_3^2 = 1/3 \end{cases}$$

As soluções usuais são:

(i) $c_1 = 1/4; c_2 = 0$ e $c_3 = 3/4$ → Método de Heun

(ii) $c_2 = c_3 = 3/8$ e $c_1 = 1/4$ → Método de Nystrom

3.5.4.1 Método de Heun – Runge-Kutta de Ordem 3 (I)

Para a escolha (i), obtém-se:

$$\boxed{y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3K_3]} \tag{57}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

3.5.4.2 Método de Nystrom – Runge-Kutta de Ordem 3 (II)

Para a escolha (ii), obtém-se:

$$\boxed{y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3/2(K_2 + K_3)]} \tag{58}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

3.5.4.3 Método de Runge-Kutta de Ordem 3 (III)

Uma solução para o sistema não linear que define este método de Runge-Kutta de Ordem 3 é dada por:

$$(iii) \quad c_1 = 2/9; c_2 = 3/9\ e\ \ c_3 = 4/9$$
$$a_2 = 1/2; b_{21} = 1/2$$
$$a_3 = 3/4; b_{31} = a_3 - b_{32} = 0; b_{32} = 3/4$$

$$\boxed{y_{n+1} = y_n + {}^{h}/_{9}\,(2K_1 + 3K_2 + 4K_3)} \tag{59}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{3}/_{4}\,h, y_n + {}^{3}/_{4}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

3.5.4.3 Exemplo de Aplicação 3 – Método de Runge-Kutta de Ordem 3

A seguir, um exemplo que compara as aproximações obtidas pelo Método de Heun e pelo Método de Nystrom.

Dado o P.V.I.:

$$\begin{cases} y' = x - y + 2 \\ y(0) = 2 \\ \in [0;0{,}3]\ e\ h = 0{,}1 \end{cases}$$

Calcule a solução numérica pelo Método de Heun e pelo Método de Nystrom.

Solução:

- Omitiu-se a resolução da Equação Diferencial de Ordem 1 e apresenta-se a solução exata do P.V.I.: $y(x) = x + 1 + e^x$

- Tabela Demonstrativa:

Tabela 1: Comparação entre os valores da aproximação e solução exata do P.V.I..

$\boldsymbol{n}$	$\boldsymbol{x_n}$	$\boldsymbol{y_n(x)}$ (aproximação) (I) Método de Heun	$\boldsymbol{y_n(x)}$ (aproximação) (II) Método de Nystrom	$\boldsymbol{y_E(x)}$ Valor real/exato (vem da solução exata do P.V.I.) $y_E(x) = x + 1 + e^x$
0	0,0	2,0000	2,0000	2,0000
1	0,1	2,0048 (285)	2,0048 (375)	2,00484
2	0,2	2,0187 (216)	2,0187 (295)	2,01873
3	0,3	2,0408 (018)	2,0265 (317)	2,04082

- Método (I) – Método de Heun:

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3K_3] \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right) \\ K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right) \end{cases} \qquad n = 0, 1, 2, \cdots$$

$\boldsymbol{n = 0}$	$K_1 = f(x_0, y_0) = f(0,2) = 0 - 2 + 2 = 0$
	$K_2 = f(0 + 0{,}0333; 2 + 1/3(0{,}1) \cdot 0) = f(0{,}0333; 2) = 0{,}0333 - 2 + 2 = 0{,}0333$
	$K_3 = f(0 + 0{,}0666; 2 + 0{,}00222) = f(0{,}0666; 2{,}00222) = 0{,}0666 - 2{,}00222 + 2 = 0{,}06438$
$y_1 = y_0 + 0{,}025(0{,}19314) = 2 + 0{,}0048285 = 2{,}0048285$	

$\boldsymbol{n = 1}$	$K_1 = f(x_1, y_1) = f(0{,}1; 2{,}0048285) = 0{,}1 - 2{,}0048285 + 2 = 0{,}0951715$
	$K_2 = f(0{,}1 + 0{,}0333; 2{,}0048285 + 0{,}0031724) = f(0{,}1333; 2{,}0080009) =$ $= 0{,}1333 - 2{,}0080009 + 2 = 0{,}1252991$
	$K_3 = f(0{,}1 + 0{,}0667; 2{,}0048285 + 0{,}0083533) = f(0{,}1667; 2{,}0131818) =$ $= 0{,}1667 - 2{,}0131818 + 2 = 0{,}1535182$
$y_2 = 2{,}0048285 + 0{,}025(0{,}0951715 + 3(0{,}1535182)) = 2{,}0048285 + 0{,}138931 = 2{,}0187216$	

continuação: Método (I) – Método de Heun:

$\boldsymbol{n=2}$	$K_1 = f(x_2, y_2) = f(0{,}2; 2{,}0187216) = 0{,}2 - 2{,}0187216 + 2 = 0{,}1812784$
	$K_2 = f(0{,}2 + 0{,}0333; 2{,}0187216 + 0{,}0060426) = f(0{,}2333; 2{,}0247642) =$ $= 0{,}2333 - 2{,}0247642 + 2 = 0{,}2085358$
	$K_3 = f(0{,}2 + 0{,}0666; 2{,}0187216 + 0{,}0139024) = f(0{,}2666; 2{,}032624) =$ $= 0{,}2666 - 2{,}032624 + 2 = 0{,}233976$
$y_3 = 2{,}0187216 + 0{,}025(0{,}1812784 + 3(0{,}233976)) = 2{,}0187216 + 0{,}0220802 = 2{,}0408018$	

- Método (II) – Método de Nystrom:

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3/2(K_2 + K_3)] \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_1\right) \\ K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right) \end{cases} \qquad n = 0, 1, 2, \cdots$$

$\boldsymbol{n=0}$	$K_1 = f(x_0, y_0) = f(0, 2) = 0 - 2 + 2 = 0$
	$K_2 = f(0 + 0{,}0667; 2 + 0{,}0044) = f(0{,}0667; 2{,}0044) = 0{,}0667 - 2{,}0044 + 2 = 0{,}0623$
	$K_2 = f(0 + 0{,}0667; 2 + 0) = f(0{,}0667; 2) = 0{,}0667 - 2 + 2 = 0{,}0667$
$y_1 = 2 + 0{,}025 \cdot [0 + 3/2(0{,}0667 + 0{,}0623)] = 2 + 0{,}025(0{,}1935) = 2{,}0048375$	

$\boldsymbol{n=1}$	$K_1 = f(x_1, y_1) = f(0{,}1; 2{,}0048375) = 0{,}1 - 2{,}0048375 + 2 = 0{,}0951625$
	$K_2 = f(0{,}1 + 0{,}0667; 2{,}0048375 + 0{,}0063442) = f(0{,}1667; 2{,}0111817) =$ $= 0{,}1667 - 2{,}0111817 + 2 = 0{,}1555183$
	$K_3 = f(0{,}1 + 0{,}0667; 2{,}0048375 + 0{,}0103679) = f(0{,}1667; 2{,}0152054) =$ $= 0{,}1667 - 2{,}0152054 + 2 = 0{,}1514946$
$y_2 = 2{,}0048375 + 0{,}025 \cdot [0{,}0951625 + 3/2(0{,}1555183 + 0{,}154946)] = 2{,}0048375 + 0{,}0138920 =$ $= 2{,}0187295$	

$\boldsymbol{n=2}$	$K_1 = f(x_2, y_2) = f(0{,}2; 2{,}0187295) = 0{,}2 - 2{,}0187295 + 2 = 0{,}1812705$
	$K_2 = f(0{,}2 + 0{,}0667; 2{,}0187295 + 0{,}0120847) = f(0{,}2667; 2{,}0308142) =$ $= 0{,}2667 - 2{,}0308142 + 2 = 0{,}2358858$
	$K_3 = f(0{,}2 + 0{,}0667; 2{,}0187295 + 0{,}0157257) = f(0{,}2667; 2{,}0344552) =$ $= 0{,}2667 - 2{,}0344552 + 2 = 0{,}2322448$
$y_3 = 2{,}0187295 + 0{,}025 \cdot [0{,}1812705 + 3/2(0{,}2358858 + 0{,}2322448)] = 2{,}0187295 + 0{,}0078022 =$ $= 2{,}0265317$	

3.5.5 Métodos de Runge-Kutta de Ordem 4

Um dos processos numéricos mais populares e, ao mesmo tempo, mais preciso para obter soluções aproximadas de equações diferencias é o Método de Runge-Kutta de 4ª Ordem.

O Método de Runge-Kutta de Ordem 4 consiste em determinar constantes apropriadas tais que uma fórmula

$$y_{n+1} = y_n + aK_1 + bK_2 + cK_3 + dK_4 \tag{60}$$

concorde com um polinômio de Taylor até h^4, isto é, até o quinto termo. Tal como apresentado na Equação (53), os K_i são múltiplos constantes de $f(x, y)$ calculados em pontos selecionados. A dedução dos termos necessita de muitos cálculos, que não são apresentados neste trabalho,

por isso enunciam-se os resultados. A partir da Equação (60) será apresentada duas variações de quarta ordem.

Enquanto outras fórmulas de quarta ordem são facilmente obtidas, o algoritmo resumido em (61) é amplamente utilizado e reconhecido como uma ferramenta computacional valiosa que é as vezes é referido também como o *Método de Runge-Kutta Clássico*.

Aconselha-se observar atentamente as fórmulas em (61), pois K_2 depende de K_1, K_3 depende de K_2 e K_4 depende de K_3. Além disso, K_2 e K_3 envolvem aproximações para o coeficiente angular no ponto médio $x_n + \frac{1}{2}h$ do intervalo $x_n \leq x \geq x_{n+1}$.

3.5.5.1 Método de Runge-Kutta de Ordem 4 (I)

$$\boxed{y_{n+1} = y_n + {}^{h}/_{6}\,[K_1 + 2(K_2 + K_3) + K_4]} \tag{61}$$

$$\begin{aligned}
&K_1 = f(x_n, y_n)\\
&K_2 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_1\right)\\
&K_3 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_2\right)\\
&K_4 = f(x_n + h, y_n + hK_3)\\
&n = 0, 1, 2, \cdots
\end{aligned}$$

3.5.5.2 Método de Runge-Kutta de Ordem 4 (II)

$$\boxed{y_{n+1} = y_n + {}^{h}/_{8}\,[K_1 + 3(K_2 + K_3) + K_4]} \tag{62}$$

$$\begin{aligned}
&K_1 = f(x_n, y_n)\\
&K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right)\\
&K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n - {}^{1}/_{3}\,hK_1 + hK_2\right)\\
&K_4 = f(x_n + h, y_n + hK_1 - hK_2 + hK_3)\\
&n = 0, 1, 2, \cdots
\end{aligned}$$

3.5.5.3 Exemplo de Aplicação 4 – Método de Runge-Kutta de Ordem 4

A seguir, um exemplo que compara as aproximações obtidas pelos Métodos (I) e (II) de Ordem 4.

Dado o P.V.I.:

$$\begin{cases} y' = x - y + 2 \\ y(0) = 2 \\ \in [0; 0{,}3]\ e\ h = 0{,}1 \end{cases}$$

Calcule a solução numérica pelos Métodos (I) e (II) de Ordem 4.

Solução:

- Omitiu-se a resolução da Equação Diferencial de Ordem 1 e apresenta-se a solução exata do P.V.I.: $y(x) = x + 1 + e^x$

- Tabela Demonstrativa:

Tabela 1: Comparação entre os valores da aproximação e solução exata do P.V.I..

n	x_n	$y_n(x)$ (aproximação) Método (I)	$y_n(x)$ (aproximação) Método (II)	$y_E(x)$ Valor real/exato (vem da solução exata do P.V.I.) $y_E(x) = x + 1 + e^x$
0	0,0	2,00000	2,00000	2,00000
1	0,1	2,00642	2,00475	2,00484
2	0,2	2,02016	2,01865	2,01873
3	0,3	2,04211	2,04075	2,04082

- Método (I):

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + {}^{h}/_{6}\,[K_1 + 2(K_2 + K_3) + K_4] \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_1\right) \\ K_3 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_2\right) \\ K_4 = f(x_n + h, y_n + hK_3) \end{cases} \qquad n = 0, 1, 2, \cdots$$

$n = 0$	$K_1 = f(x_0, y_0) = f(0,2) = 0 - 2 + 2 = 0$
	$K_2 = f(0 + 1/2(0{,}1); 2 + 1/2(0{,}1)(0)) = f(0{,}05; 2) = 0{,}05 - 2 + 2 = 0{,}05$
	$K_3 = f(0 + 1/2(0{,}1); 2 + 1/2(0{,}1)(0{,}05)) = f(0{,}1; 2{,}0025) = 0{,}1 - 2{,}0025 + 2 = 0{,}0975$
	$K_4 = f(0 + 0{,}1; 2 + 0{,}1(0{,}0975)) = f(0{,}1; 2{,}00975) = 0{,}1 - 2{,}00975 + 2 = 0{,}09025$
$y_1 = 2 + 0{,}1/6[0 + 2(0{,}05 + 0{,}0975) + 0{,}09025] = 2{,}00642$	

$n = 1$	$K_1 = f(x_1, y_1) = f(0{,}1;\ 2{,}00642) = 0{,}1 - 2{,}00642 + 2 = 0{,}09358$
	$K_2 = f(0{,}1 + 1/2(0{,}1); 2{,}00642 + 1/2(0{,}1)(0{,}09358)) = f(0{,}15; 2{,}01110) = 0{,}15 - 2{,}01110 + 2 = 0{,}1389$
	$K_3 = f(0{,}1 + 1/2(0{,}1); 2{,}00642 + 1/2(0{,}1)(0{,}1389)) = f(0{,}15; 2{,}01336) = 0{,}15 - 2{,}01336 + 2 = 0{,}13664$
	$K_4 = f(0{,}1 + 0{,}1; 2{,}00642 + 0{,}1(0{,}13664)) = f(0{,}2; 2{,}02008) = 0{,}2 - 2{,}02008 + 2 = 0{,}17992$
$y_2 = 2{,}00642 + 0{,}1/6[0{,}09358 + 2(0{,}1389 + 0{,}13664) + 0{,}17992] = 2{,}02016$	

$n = 2$	$K_1 = f(x_2, y_2) = f(0{,}2; 2{,}02016) = 0{,}2 - 2{,}02016 + 2 = 0{,}17984$
	$K_2 = f(0{,}2 + 1/2(0{,}1); 2{,}02016 + 1/2(0{,}1)(0{,}17984)) = f(0{,}25; 2{,}02915) = 0{,}25 - 2{,}02915 + 2 = 0{,}22085$
	$K_3 = f(0{,}2 + 1/2(0{,}1); 2{,}02016 + 1/2(0{,}1)(0{,}22085)) = f(0{,}25; 2{,}03120) = 0{,}25 - 2{,}03120 + 2 = 0{,}21880$
	$K_4 = f(0{,}2 + 0{,}1; 2{,}02016 + 0{,}1(0{,}21880)) = f(0{,}3; 2{,}04204) = 0{,}3 - 2{,}04204 + 2 = 0{,}25796$
$y_3 = 2{,}02016 + 0{,}1/6[0{,}17984 + 2(0{,}22085 + 0{,}21880) + 0{,}25796] = 2{,}04211$	

- Método (II)

$h = 0{,}1$; $y_0 = 2$; $x_0 = 0$ e $f(x_n, y_n) = x_n - y_n + 2$

$$\begin{cases} y_{n+1} = y_n + {}^{h}/_{8}\,[K_1 + 3(K_2 + K_3) + K_4] \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right) \\ K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n - {}^{1}/_{3}\,hK_1 + hK_2\right) \\ K_4 = f(x_n + h, y_n + hK_1 - hK_2 + hK_3) \end{cases} \qquad n = 0, 1, 2, \cdots$$

$\boldsymbol{n = 0}$	$K_1 = f(x_0, y_0) = f(0, 2) = 0 - 2 + 2 = 0$
	$K_2 = f(0 + 1/3(0{,}1); 2 + 1/3(0{,}1)(0)) = f(0{,}03333; 2) = 0{,}03333 - 2 + 2 = 0{,}03333$
	$K_3 = f(0 + 2/3(0{,}1); 2 - 1/3(0{,}1)(0) + 0{,}1(0{,}03333)) = f(0{,}06667; 2{,}00333) = 0{,}06667 - 2{,}00333 + 2 = = 0{,}06334$
	$K_4 = f\left(0 + 0{,}1; 2 + 0{,}1(0) - 0{,}1(0{,}03333) + 0{,}1(0{,}06334)\right) = f(0{,}1; 2{,}00967) = 0{,}1 - 2{,}00967 + 2 = 0{,}09033$
$y_1 = 2 + 0{,}1/8[0 + 3(0{,}03333 + 0{,}06334) + 0{,}09033] = 2{,}00475$	

$\boldsymbol{n = 1}$	$K_1 = f(x_1, y_1) = f(0{,}1;\ 2{,}00475) = 0{,}1 - 2{,}00475 + 2 = 0{,}09525$
	$K_2 = f(0{,}1 + 1/3(0{,}1); 2{,}00475 + 1/3(0{,}1)(0{,}09525)) = f(0{,}13333; 2{,}0793) = 0{,}13333 - 2{,}00793 + 2 = = 0{,}12540$
	$K_3 = f(0{,}1 + 2/3(0{,}1); 2{,}00475 - 1/3(0{,}1)(0{,}09525) + 0{,}1(0{,}12540)) = f(0{,}16667; 2{,}01412) = = 0{,}16667 - 2{,}01412 + 2 = 0{,}15255$
	$K_4 = f\left(0{,}1 + 0{,}1; 2{,}00475 + 0{,}1(0{,}09525) - 0{,}1(0{,}12540) + 0{,}1(0{,}15255)\right) = f(0{,}2; 2{,}01699) = = 0{,}2 - 2{,}01699 + 2 = 0{,}18301$
$y_2 = 2{,}00475 + 0{,}1/8[0{,}09525 + 3(0{,}12540 + 0{,}15255) + 0{,}18301] = 2{,}01865$	

$\boldsymbol{n = 2}$	$K_1 = f(x_2, y_2) = f(0{,}2;\ 2{,}01865) = 0{,}2 - 2{,}01865 + 2 = 0{,}18135$
	$K_2 = f(0{,}2 + 1/3(0{,}1); 2{,}01865 + 1/3(0{,}1)(0{,}18135)) = f(0{,}23333; 2{,}02470) = 0{,}23333 - 2{,}02470 + 2 = = 0{,}20863$
	$K_3 = f(0{,}2 + 2/3(0{,}1); 2{,}01865 - 1/3(0{,}1)(0{,}18135) + 0{,}1(0{,}20863)) = f(0{,}26667; 2{,}0347) = = 0{,}26667 - 2{,}03347 + 2 = 0{,}23320$
	$K_4 = f\left(0{,}2 + 0{,}1; 2{,}01865 + 0{,}1(0{,}18135) - 0{,}1(0{,}20863) + 0{,}1(0{,}23320)\right) = f(0{,}3; 2{,}03924) = = 0{,}3 - 2{,}03924 + 2 = 0{,}26076$
$y_3 = 2{,}01865 + 0{,}1/8[0{,}18135 + 3(0{,}20863 + 0{,}23320) + 0{,}26076] = 2{,}04075$	

3.5.6 Método de Runge-Kutta de Ordem 5 de Butcher[5]

Quando resultados mais precisos são necessários, o *Método de Runge-Kutta de Quinta Ordem de Butcher* é recomendado:

$$\boxed{y_{n+1} = y_n + {}^{h}/_{90}\,[7K_1 + 32K_3 + 12K_4 + 32K_5 + 7K_6]} \qquad (63)$$

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f\left(x_n + {}^{1}/_{4}\,h, y_n + {}^{1}/_{4}\,hK_1\right) \\ K_3 &= f\left(x_n + {}^{1}/_{4}\,h, y_n + {}^{1}/_{8}\,hK_1 + {}^{1}/_{8}\,hK_2\right) \\ K_4 &= f\left(x_n + {}^{1}/_{2}\,h, y_n - {}^{1}/_{2}\,hK_2 + hK_3\right) \\ K_5 &= f\left(x_n + {}^{3}/_{4}\,h, y_n + {}^{3}/_{16}\,hK_1 + {}^{9}/_{16}\,hK_4\right) \\ K_6 &= f\left(x_n + h, y_n - {}^{3}/_{7}\,hK_1 + {}^{2}/_{7}\,hK_2 + {}^{12}/_{7}\,hK_3 - {}^{12}/_{7}\,hK_4 + {}^{8}/_{7}\,hK_5\right) \\ n &= 0, 1, 2, \cdots \end{aligned}$$

[5]**John Charles Butcher** (1933-), é um matemático neozelandês especializado em métodos numéricos para a solução de Equações Diferenciais Ordinárias. Butcher trabalha em métodos multiestágio para problemas de valor inicial, como Runge-Kutta e métodos lineares gerais. Mais recentemente, ele está investigando um novo tipo de método com estabilidade idêntica à de um método Runge-Kutta.

Fórmulas Runge-Kutta de ordem superior como o Método de Butcher estão disponíveis, mas, em geral, além, dos Métodos de Ordem 4, o ganho em acurácia é contrabalançado pelo esforço e pela complexidade computacionais adicionais.

3.5.7 Métodos de Runge-Kutta-Cash-Karp[6]

Este método é uma das melhores e mais modernas versões do algoritmo de Runge-Kutta com possibilidade de controle de precisão e, portanto, com possibilidade de variação automática do passo de integração na solução de um sistema de Equações com Derivadas Ordinárias (EDO).

Dessa forma, a EDO pode ser resolvida com a Equação (65), por exemplo, e a estimativa de erro obtida, como a diferença das estimativas de Quinta e Quarta Ordens do Método de Cash-Karp. Os coeficientes particulares utilizados para o método foram desenvolvidos por Cash e Karp em 1990.

3.5.7.1 Método de Runge-Kutta-Cash-Karp de Ordem 4

A seguir, a estimativa de Ordem 4 é apresentada.

$$\boxed{y_{n+1} = y_n + h\left(\frac{37}{378}K_1 + \frac{250}{621}K_3 + \frac{125}{594}K_4 + \frac{512}{1.771}K_6\right)} \tag{64}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^1/_5\, h, y_n + {}^1/_5\, hK_1\right)$$
$$K_3 = f\left(x_n + {}^3/_{10}\, h, y_n + {}^3/_{40}\, hK_1 + {}^9/_{40}\, hK_2\right)$$
$$K_4 = f\left(x_n + {}^3/_5\, h, y_n + {}^3/_{10}\, hK_1 - {}^9/_{10}\, hK_2 + {}^6/_5\, hK_3\right)$$
$$K_5 = f\left(x_n + h, y_n - {}^{11}/_{54}\, hK_1 + {}^5/_2\, hK_2 - {}^{70}/_{27}\, hK_3 + {}^{35}/_{27}\, hK_4\right)$$
$$K_6 = f\left(x_n + \frac{7}{8}h, y_n + \frac{1.631}{55.296}hK_1 + \frac{175}{512}hK_2 + \frac{575}{13.824}hK_3 + \frac{44.275}{110.592}hK_4 + \frac{253}{4.096}hK_5\right)$$
$$n = 0, 1, 2, \cdots$$

3.5.7.2 Método de Runge-Kutta-Cash-Karp de Ordem 5

E para a estimativa de Ordem 5, tem-se que:

$$\boxed{y_{n+1} = y_n + h\left(\frac{2.825}{27.648}K_1 + \frac{18.575}{48.384}K_3 + \frac{13.525}{55.296}K_4 + \frac{277}{14.336}K_5 + \frac{1}{4}K_6\right)} \tag{65}$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^1/_5\, h, y_n + {}^1/_5\, hK_1\right)$$
$$K_3 = f\left(x_n + {}^3/_{10}\, h, y_n + {}^3/_{40}\, hK_1 + {}^9/_{40}\, hK_2\right)$$
$$K_4 = f\left(x_n + {}^3/_5\, h, y_n + {}^3/_{10}\, hK_1 - {}^9/_{10}\, hK_2 + {}^6/_5\, hK_3\right)$$
$$K_5 = f\left(x_n + h, y_n - {}^{11}/_{54}\, hK_1 + {}^5/_2\, hK_2 - {}^{70}/_{27}\, hK_3 + {}^{35}/_{27}\, hK_4\right)$$
$$K_6 = f\left(x_n + \frac{7}{8}h, y_n + \frac{1.631}{55.296}hK_1 + \frac{175}{512}hK_2 + \frac{575}{13.824}hK_3 + \frac{44.275}{110.592}hK_4 + \frac{253}{4.096}hK_5\right)$$
$$n = 0, 1, 2, \cdots$$

[6] **J. R. Cash, A. H. Karp** (1990). Em Análise Numérica, Cash-Karp é um método para a resolução de Equações Diferenciais Ordinárias (EDOs). Ele é um dos métodos tipo Runge-Kutta para a resolução numérica de equações diferenciais. De fato, ele faz seis avaliações de função para calcular soluções acuradas de Quarta e Quinta Ordem. A diferença entre elas é então tomada como erro para a aproximação de quarta ordem. Essa estimativa do erro é bastante conveniente para algoritmos de integração com tamanho de passo adaptativo.

3.5.8 Método de Runge-Kutta-Fehlberg[7]

Os métodos anteriores mostram que a precisão de um método numérico para aproximar soluções de equações diferenciais pode ser melhorado reduzindo-se o tamanho do passo h. Essa melhora da precisão é usualmente obtida com algum custo, ou seja, o aumento do tempo de cálculo e maior possibilidade de cometer erros de arredondamento.

Em geral, no intervalo de aproximação podem existir subintervalos nos quais um tamanho de passo relativamente grande é suficiente e outros subintervalos nos quais um tamanho de passo menor é necessário para que o erro de truncamento seja mantido dentro de um limite desejado.

Métodos numéricos que utilizam um tamanho de passo variável são chamados de **Métodos Adaptativos**. Uma das rotinas adaptativas mais utilizadas é o **Método de Runge-Kutta-Fehlberg**.

Como Fehlberg empregou dois métodos de Runge-Kutta de ordens diferentes, os *Métodos de Quarta* e *Quinta Ordem*, este algoritmo é frequentemente designado como **Método RKF45**.

Será visto que o Método de Runge-Kutta de Quarta Ordem utilizado em RKF45 não é igual ao indicado em (61).

Para obter a estimativa do erro em RKF45 é necessário calcular duas previsões, usando métodos de Runge-Kutta de ordens diferentes. Os resultados podem então ser subtraídos para obtenção do Erro Local – EL – como também uma estimativa para o passo h ideal estabelecendo um esquema adaptativo, ou seja, um esquema em que o valor de h varia automaticamente a partir de um valor da tolerância imposto pelo usuário.

Como em cada passo deve-se utilizar um método de Runge-Kutta duas vezes (duas ordens diferentes), essa abordagem tem como desvantagem o alto consumo de recursos computacionais.

O Método de Fehlberg contorna esse problema, uma vez que os coeficientes calculados na interpolação da função de ajuste desse método podem ser aproveitados tanto na versão de Quinta Ordem quanto na versão de Quarta Ordem.

Assim, a abordagem de Fehlberg fornece uma estimativa do Erro de Truncamento – ET – (ou Erro Local) com apenas seis cálculos de função com a diferença das estimativas de quarta e quinta ordens calculadas previamente.

O Erro Local EL pode ser determinado por:

$$\boxed{EL = -\frac{K_1}{360} + \frac{128}{4.275}K_3 + \frac{2.187}{75.240}K_4 - \frac{K_5}{50} - \frac{2}{55}K_6} \qquad (66)$$

[7]**Erwin Fehlberg** (1911-1990) nasceu na Alemanha, recebeu seu doutorado da Technical University of Berlin em 1942, emigrou para os Estados Unidos depois da Segunda Guerra Mundial e trabalhou na NASA por muitos anos. O Método de Runge-Kutta-Fehlberg foi publicado pela primeira vez em um Relatório Técnico da NASA em 1969. Fehlberg desenvolveu esta e outras técnicas de correção de erros enquanto trabalha nas instalações da NASA, em Huntsville, Alabama. Em 1969, ele recebeu uma Medalha de Feitos Científicos Excepcionais da NASA por este trabalho.

O valor S mais apreciado para o passo h é obtido multiplicando o valor atual de h pelo valor fornecido em (67):

$$S = \left(\frac{\epsilon h}{2\left|y_{n+1}^{5ª\ Ordem} - y_{n+1}^{4ª\ Ordem}\right|}\right)^{1/4}$$

$$\Leftrightarrow S = \left(1/2\right)^{1/4}\left(\frac{\epsilon h}{\left|y_{n+1}^{5ª\ Ordem} - y_{n+1}^{4ª\ Ordem}\right|}\right)^{1/4}$$

$$\Leftrightarrow \boxed{S = 0{,}840886\left(\frac{\epsilon h}{\left|y_{n+1}^{5ª\ Ordem} - y_{n+1}^{4ª\ Ordem}\right|}\right)^{1/4}} \quad (67)$$

onde ϵh é a tolerância definida pelo usuário.

3.5.8.1 Método de Runge-Kutta-Fehlberg de Ordem 4

A aproximação de valores para a solução de um Problema de Valor Inicial para este método é conforme a Equação (66):

$$\boxed{y_{n+1} = y_n + \frac{25}{216}K_1 + \frac{1.408}{2.565}K_3 + \frac{2.197}{4.104}K_4 - \frac{1}{5}K_5} \quad (68)$$

E as Equações dos coeficientes são:

$$K_1 = hf(x_n, y_n)$$

$$K_2 = hf\left(x_n + \frac{1}{4}h, y_n + \frac{1}{4}K_1\right)$$

$$K_3 = hf\left(x_n + \frac{3}{8}h, y_n + \frac{3}{32}K_1 + \frac{9}{32}K_2\right)$$

$$K_4 = hf\left(x_n + \frac{12}{13}h, y_n + \frac{1.932}{2.197}K_1 - \frac{7.200}{2.197}K_2 + \frac{7.296}{2.197}K_3\right)$$

$$K_5 = hf\left(x_n + h, y_n + \frac{439}{216}K_1 - 8K_2 + \frac{3.680}{513}K_3 - \frac{845}{4.104}K_4\right)$$

$$K_6 = hf\left(x_n + \frac{1}{2}h, y_n - \frac{8}{27}K_1 + 2K_2 - \frac{3.544}{2.565}K_3 + \frac{1.859}{4.104}K_4 - \frac{11}{40}K_5\right)$$

$$n = 0, 1, 2, \cdots$$

3.5.8.2 Método de Runge-Kutta-Fehlberg de Ordem 5

A melhor aproximação de valores para a solução de um P.V.I. é dada por (69):

$$\boxed{y_{n+1} = y_n + \frac{16}{135}K_1 + \frac{6.656}{12.825}K_3 + \frac{28.561}{56.430}K_4 - \frac{9}{50}K_5 + \frac{2}{55}K_6} \quad (69)$$

E as Equações dos coeficientes são:

$$K_1 = hf(x_n, y_n)$$

$$K_2 = hf\left(x_n + \frac{1}{4}h, y_n + \frac{1}{4}K_1\right)$$

$$K_3 = hf\left(x_n + \frac{3}{8}h, y_n + \frac{3}{32}K_1 + \frac{9}{32}K_2\right)$$

$$K_4 = hf\left(x_n + \frac{12}{13}h, y_n + \frac{1.932}{2.197}K_1 - \frac{7.200}{2.197}K_2 + \frac{7.296}{2.197}K_3\right)$$

$$K_5 = hf\left(x_n + h, y_n + \frac{439}{216}K_1 - 8K_2 + \frac{3.680}{513}K_3 - \frac{845}{4.104}K_4\right)$$

$$K_6 = hf\left(x_n + \frac{1}{2}h, y_n - \frac{8}{27}K_1 + 2K_2 - \frac{3.544}{2.565}K_3 + \frac{1.859}{4.104}K_4 - \frac{11}{40}K_5\right)$$

$$n = 0, 1, 2, \cdots$$

4. Discussão e Conclusão

Uma Equação Diferencial não necessariamente possui solução. Porém, mesmo quando uma solução existe, podemos não ser capazes de exibi-las em sua forma explícita ou implícita – podemos ter que nos contentar com uma aproximação da solução.

A ideia é utilizar a Equação Diferencial para construir algoritmos que aproximam de fato as coordenadas dos pontos em uma curva solução.

A escolha de um método para resolver numericamente uma equação diferencial exige muitas considerações e escolhas: precisão e tempo computacional que se quer atingir. Em geral, escolhem-se os métodos de passo único, em especial o Método de Runge-Kutta, devido a precisão e o fato de ser fácil de programar.

Entretanto, uma séria desvantagem de tais métodos é que o membro direito da equação diferencial deve ser calculado muitas vezes em cada passo.

Viu-se, a partir da fórmula de Euler, as maneiras de resolver o *Problema de Valor Inicial* de uma Equação Diferencial Ordinária.

Os Métodos de Euler com as Expansões em Polinômios de Taylor são conhecidos hoje como *Métodos de Runge-Kutta*.

Dentre os Métodos de Runge-Kutta, o *Método Clássico* sofre os mesmos defeitos que outros métodos com tamanho de passo fixo para problemas onde o erro de truncamento local varia muito no intervalo de interesse. Esses defeitos estimularam o desenvolvimento de Métodos de Runge-Kutta Adaptativos, que providenciam a modificação do tamanho do passo automaticamente à medida que os cálculos avançam, de modo a manter o erro de truncamento local próximo ou abaixo de um nível de tolerância especificado.

Referências

Análise Numérica – Richard L. Burden e J. Douglas Faires. 3ª Edição. São Paulo: Cengage Learning, 2013;

Cálculo Numérico: Aprendizagem com apoio de software. Selma Arenales e Arthur Darezzo. São Paulo: Cengage Learning, 2013;

Cálculo Numérico: Aprendizagem com apoio de software. Selma Arenales e Arthur Darezzo. 2ª Edição. São Paulo: Cengage Learning, 2015;

Equações Diferenciais, Volume 2 – Dennis G. Zill e Michael R. Cullen. 3ª Edição. São Paulo: Pearson Makron Books, 2001;

Equações Diferenciais Elementares e Problemas de Valores de Contorno – William E. Boyce e Richard C. DiPrima. 9ª Edição. Rio de Janeiro: LTC, 2013;

Matemática Avançada para Engenharia, Volume 1 – Dennis G. Zill e Michael R. Cullen. 3ª Edição. Porto Alegre: Bookman, 2009;

Erwin Fehlberg (1969). **Low-order classical Runge-Kutta formulas with step size control and their application to some heat transfer problems**. NASA Technical Report 315

metodo de runge kutta Fehlberg, acesso em 07 de Abril de 2018.

Disponível em:

< www.feq.unicamp.br/~nunhez/eq502/modulo09.pdf >

< https://web.archive.org/web/20061209105041/http://math.fullerton.edu/mathews/n2003/RungeKuttaFehlbergMod.html >

< https://repositorio-aberto.up.pt/bitstream/10216/12677/2/Texto%20integral.pdf >

< http://www2.ic.uff.br/~otton/graduacao/metodosnumericos/Runge_Kutta_Fehlberg.pdf >

< http://www.professorglobal.cbpf.br/mediawiki/index.php/M%C3%A9todos_de_Runge-Kutta >

< https://www.linux.ime.usp.br/~cef/mac499-00/monografias/julian/relatorio/node9.html >

< https://www.linux.ime.usp.br/~cef/mac499-00/monografias/julian/relatorio/node10.html >

< https://es.wikipedia.org/wiki/M%C3%A9todo_de_Runge-Kutta-Fehlberg >

Método de Runge-Kutta de Ordem 5 de Butcher, acesso em 22 de Abril de 2018.

Disponível em:

< http://www.sawp.com.br/blog/?p=1930 >

< https://en.wikipedia.org/wiki/John_C._Butcher >

< https://www.math.auckland.ac.nz/~butcher/ >

Runge-Kutta de Cash-Karp, acesso em 22 de Abril de 2018.

Disponível em:

< https://pt.wikipedia.org/wiki/Método_de_Cash-Karp >

< https://dl.acm.org/citation.cfm?doid=79505.79507 >*****

* J. R. Cash, A. H. Karp. "A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides", ACM Transactions on Mathematical Software 16: 201-222, 1990. doi: 10.1145/79505.79507

< http://www.elegio.it/mc2/rk/doc/p201-cash-karp.pdf >

< http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK5/RKcoeff5c_4.pdf >

< http://www.elegio.it/mc2/rk/doc/cki-js.htm >

-Leituras sobre o Método de Runge-Kutta-Fehlberg – RKF45, acesso em 23 de Abril de 2018.

< https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19690021375.pdf >

< https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta%E2%80%93Fehlberg_method >

< http://mathfaculty.fullerton.edu/mathews/n2003/rungekuttafehlbergmod.html >

-Artigos de Erwin Fehlberg disponíveis no NASA Technical Reports Server (NTRS):

< https://ntrs.nasa.gov/search.jsp?Nm=4294033827|Author|Fehlberg,%20E.&N=0 >

APÊNDICES

APÊNDICE I

Métodos para as Soluções Numéricas de Equações Diferenciais Ordinárias

❖ MÉTODOS DE TAYLOR

➢ MÉTODO DE TAYLOR DE ORDEM 1 – MTO-1
– MÉTODO DE EULER ou MÉTODO DAS TANGENTES –

$$y(x_n + h) = y(x_n) + h \cdot f(x_n, y_n)$$

$$\boxed{\Leftrightarrow y_{n+1} = y_n + hf_n} \qquad \begin{cases} f_n = f(x_n, y_n) \\ n = 0, 1, 2, \cdots \end{cases} \qquad (1)$$

➢ MÉTODO DE TAYLOR DE ORDEM 2 – MTO-2

$$\boxed{y_{n+1} = y_n + hf_n + \frac{h^2}{2!} f'_n} \qquad (2)$$

$$f_n = f(x_n, y_n)$$
$$f'_n = f_x(x_n, y_n) + f_y(x_n, y_n) \cdot f(x_n, y_n)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE TAYLOR DE ORDEM 3 – MTO-3

$$\boxed{y_{n+1} = y_n + hf_n + \frac{h^2}{2!} f'_n + + \frac{h^3}{3!} f''_n} \qquad (3)$$

$$f_n = f(x_n, y_n)$$
$$f'_n = f_x(x_n, y_n) + f_y(x_n, y_n) \cdot f(x_n, y_n)$$
$$f''_n = f_{xx}(x_n, y_n) + 2f_{xy}(x_n, y_n) \cdot f(x_n, y_n) +$$
$$+ f_{yy}(x_n, y_n) \cdot f(x_n, y_n) + \left(f_y(x_n, y_n)\right)^2 \cdot f(x_n, y_n) +$$
$$+ f_x(x_n, y_n) \cdot f_y(x_n, y_n)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE TAYLOR DE ORDEM P

$$y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + + \frac{h^3}{3!}y'''(x_n) + \\ + \cdots + \frac{h^P}{P!}y^{(P)}(x_n) + \frac{h^{(P+1)}}{(P+1)!}y^{(P+1)}\xi_n \quad (4)$$

$$x_n < \xi_n < x_n + h$$

❖ MÉTODOS DE RUNGE-KUTTA

➢ MÉTODO DE RUNGE-KUTTA DE ORDEM R – RK-R

$$y_{n+1} = y_n + h \cdot f(x_n, y_n, h) \quad (5)$$

$$com \quad f(x, y, h) = \sum_{r=1}^{R} c_r k_r$$

$$e \quad k_1 = f(x, y)$$

$$k_r = f\left(x + a_r h, y + h\sum_{s=1}^{r-1} b_{rs}\, k_s\right); \quad r = 2, 3, \cdots, R$$

$$a_r = \sum_{s=1}^{r-1} b_{rs}; \quad r = 2, 3, \cdots, R$$

➢ MÉTODO DE RUNGE-KUTTA DE 1ª ORDEM – RK-1

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (6)$$

$$x_n = x_0 + nh$$

$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE RUNGE-KUTTA DE 2ª ORDEM – RK-2

– Fórmula Geral –

$$\boxed{y_{n+1} = y_n + h[c_1K_1 + c_2K_2]} \quad (7)$$

$$com \ \begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + a_2h, y_n + ha_2K_1) \end{cases}$$

As constantes c e a envolvidas são soluções do sistema:

$$\begin{cases} c_1 + c_2 = 1 \\ c_1 + c_2 = 1/2 \end{cases} \left[\begin{matrix} 2 \ equações \\ 3 \ incógnitas \end{matrix} \right\} infinitas \ possibilidades$$

As duas soluções mais usuais são:

$$(i) \ \ c_1 = 0; c_2 = 1 \ \ e \ \ a_2 = 1/2$$

$$(ii) \ \ c_1 = c_2 = 1/2 \ \ e \ \ a_2 = 1$$

- Para a escolha (i), tem-se:

MÉTODO DE EULER MODIFICADO – RK-2 (I)

$$\boxed{y_{n+1} = y_n + hK_2} \quad (8)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^1/_2\, h, y_n + {}^1/_2\, hK_1\right)$$
$$n = 0, 1, 2, \cdots$$

- Para a escolha (ii), tem-se:

MÉTODO DE EULER MELHORADO – RK-2 (II)

$$\boxed{y_{n+1} = y_n + {}^h/_2\, [K_1 + K_2]} \quad (9)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f(x_n + h, y_n + hK_1)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE RUNGE-KUTTA DE 3ª ORDEM – RK-3

– Fórmula Geral –

$$\boxed{y_{n+1} = y_n + h[c_1K_1 + c_2K_2 + c_3K_3]} \qquad (10)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f(x_n + ha_2, y_n + hb_{21}K_1),\ b_{21} = a_2$$
$$K_3 = f(x_n + ha_3, y_n + hb_{31}K_1 + hb_{32}K_2)$$
$$a_2 = b_{21} \quad e \quad a_3 = b_{31} + b_{32}$$

Cujas constantes estão relacionadas no sistema:

$$\begin{cases} c_1 + c_2 + c_3 = 1 \\ c_2a_2 + c_3a_3 = 1/2 \\ c_3b_{32}a_2 = 1/6 \\ c_2a_2^2 + c_3a_3^2 += 1/3 \end{cases}$$

As soluções usuais são:

$$(i)\ \ c_1 = 1/4;\ c_2 = 0; c_3 = 3/4 \longmapsto \text{Método de Heun}$$
$$(ii)\ \ c_1 = 1/4;\ c_2 = c_3 = 3/8 \quad \longmapsto \text{Método de Nystrom}$$

- Para a escolha (i), tem-se:

MÉTODO DE HEUN – RK-3 (I)

$$\boxed{y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3K_3]} \qquad (11)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

- Para a escolha (ii), tem-se:

MÉTODO DE NYSTROM – RK-3 (II)

$$\boxed{y_{n+1} = y_n + {}^{h}/_{4}\,[K_1 + 3/2(K_2 + K_3)]} \qquad (12)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n + {}^{2}/_{3}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

- Para (iii), tem-se:

MÉTODO DE RUNGE-KUTTA III – RK-3 (III)

$$(iii)\quad c_1 = 2/9; c_2 = 3/9\ e\ \ c_3 = 4/9$$
$$a_2 = 1/2; b_{21} = 1/2$$
$$a_3 = 3/4; b_{31} = a_3 - b_{32} = 0; b_{32} = 3/4$$

$$\boxed{y_{n+1} = y_n + {}^{h}/_{9}\,(2K_1 + 3K_2 + 4K_3)} \qquad (13)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{3}/_{4}\,h, y_n + {}^{3}/_{4}\,hK_2\right)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE RUNGE-KUTTA DE 4ª ORDEM – RK-4

RK-4 (I)*

$$\boxed{y_{n+1} = y_n + {}^{h}/_{6}\,[K_1 + 2(K_2 + K_3) + K_4]} \qquad (14)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{1}/_{2}\,h, y_n + {}^{1}/_{2}\,hK_2\right)$$
$$K_4 = f(x_n + h, y_n + hK_3)$$
$$n = 0, 1, 2, \cdots$$

RK-4 (II)

$$\boxed{Hy_{n+1} = y_n + {}^{h}/_{8}\,[K_1 + 3(K_2 + K_3) + K_4]} \qquad (15)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{3}\,h, y_n + {}^{1}/_{3}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{2}/_{3}\,h, y_n - {}^{1}/_{3}\,hK_1 + hK_2\right)$$
$$K_4 = f(x_n + h, y_n + hK_1 - hK_2 + hK_3)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODO DE RUNGE-KUTTA DE 5ª ORDEM DE BUTCHER – RKB-5

RKB-5

$$\boxed{y_{n+1} = y_n + {}^{h}/_{90}\,[7K_1 + 32K_3 + 12K_4 + 32K_5 + 7K_6]} \qquad (16)$$

$$K_1 = f(x_n, y_n)$$
$$K_2 = f\left(x_n + {}^{1}/_{4}\,h, y_n + {}^{1}/_{4}\,hK_1\right)$$
$$K_3 = f\left(x_n + {}^{1}/_{4}\,h, y_n + {}^{1}/_{8}\,hK_1 + {}^{1}/_{8}\,hK_2\right)$$
$$K_4 = f\left(x_n + {}^{1}/_{2}\,h, y_n - {}^{1}/_{2}\,hK_2 + hK_3\right)$$
$$K_5 = f\left(x_n + {}^{3}/_{4}\,h, y_n + {}^{3}/_{16}\,hK_1 + {}^{9}/_{16}\,hK_4\right)$$
$$K_6 = f\left(x_n + h, y_n - {}^{3}/_{7}\,hK_1 + {}^{2}/_{7}\,hK_2 + {}^{12}/_{7}\,hK_3 - {}^{12}/_{7}\,hK_4 + {}^{8}/_{7}\,hK_5\right)$$
$$n = 0, 1, 2, \cdots$$

➢ MÉTODOS DE RUNGE-KUTTA-CASH-KARP

MÉTODO DE RUNGE-KUTTA-CASH-KARP DE 4ª ORDEM – RKCK-4

$$y_{n+1} = y_n + h\left(\frac{37}{378}K_1 + \frac{250}{621}K_3 + \frac{125}{594}K_4 + \frac{512}{1.771}K_6\right) \quad (17)$$

$K_1 = f(x_n, y_n)$

$K_2 = f\left(x_n + {}^1/_5\, h, y_n + {}^1/_5\, hK_1\right)$

$K_3 = f\left(x_n + {}^3/_{10}\, h, y_n + {}^3/_{40}\, hK_1 + {}^9/_{40}\, hK_2\right)$

$K_4 = f\left(x_n + {}^3/_5\, h, y_n + {}^3/_{10}\, hK_1 - {}^9/_{10}\, hK_2 + {}^6/_5\, hK_3\right)$

$K_5 = f\left(x_n + h, y_n - {}^{11}/_{54}\, hK_1 + {}^5/_2\, hK_2 - {}^{70}/_{27}\, hK_3 + {}^{35}/_{27}\, hK_4\right)$

$K_6 = f\left(x_n + \frac{7}{8}h, y_n + \frac{1.631}{55.296}hK_1 + \frac{175}{512}hK_2 + \frac{575}{13.824}hK_3 + \frac{44.275}{110.592}hK_4 + \frac{253}{4.096}hK_5\right)$

$n = 0, 1, 2, \cdots$

MÉTODO DE RUNGE-KUTTA-CASH-KARP DE 5ª ORDEM – RKCK-5

$$y_{n+1} = y_n + h\left(\frac{2.825}{27.648}K_1 + \frac{18.575}{48.384}K_3 + \frac{13.525}{55.296}K_4 + \frac{277}{14.336}K_5 + \frac{1}{4}K_6\right) \quad (18)$$

$K_1 = f(x_n, y_n)$

$K_2 = f\left(x_n + {}^1/_5\, h, y_n + {}^1/_5\, hK_1\right)$

$K_3 = f\left(x_n + {}^3/_{10}\, h, y_n + {}^3/_{40}\, hK_1 + {}^9/_{40}\, hK_2\right)$

$K_4 = f\left(x_n + {}^3/_5\, h, y_n + {}^3/_{10}\, hK_1 - {}^9/_{10}\, hK_2 + {}^6/_5\, hK_3\right)$

$K_5 = f\left(x_n + h, y_n - {}^{11}/_{54}\, hK_1 + {}^5/_2\, hK_2 - {}^{70}/_{27}\, hK_3 + {}^{35}/_{27}\, hK_4\right)$

$K_6 = f\left(x_n + \frac{7}{8}h, y_n + \frac{1.631}{55.296}hK_1 + \frac{175}{512}hK_2 + \frac{575}{13.824}hK_3 + \frac{44.275}{110.592}hK_4 + \frac{253}{4.096}hK_5\right)$

$n = 0, 1, 2, \cdots$

➢ MÉTODOS DE RUNGE-KUTTA-FEHLBERG

MÉTODO DE RUNGE-KUTTA-FEHLBERG DE 4ª ORDEM – RKF-4

Aproximação de valores para a solução de um P.V.I.:

$$y_{n+1} = y_n + \frac{25}{216}K_1 + \frac{1.408}{2.565}K_3 + \frac{2.197}{4.104}K_4 - \frac{1}{5}K_5 \qquad (19)$$

MÉTODO DE RUNGE-KUTTA-FEHLBERG DE 5ª ORDEM – RKF-5

Melhor aproximação de valores para a solução de um P.V.I.:

$$y_{n+1} = y_n + \frac{16}{135}K_1 + \frac{6.656}{12.825}K_3 + \frac{28.561}{56.430}K_4 - \frac{9}{50}K_5 + \frac{2}{55}K_6 \qquad (20)$$

Erro de Truncamento (ou Erro Local), utilizado como função de ajuste do passo:

$$EL = -\frac{K_1}{360} + \frac{128}{4.275}K_3 + \frac{2.197}{75.240}K_4 - \frac{K_5}{50} - \frac{2}{55}K_6 \qquad (21)$$

Equações dos Coeficientes:

$$K_1 = hf(x_n, y_n)$$

$$K_2 = hf\left(x_n + \frac{1}{4}h, y_n + \frac{1}{4}K_1\right)$$

$$K_3 = hf\left(x_n + \frac{3}{8}h, y_n + \frac{3}{32}K_1 + \frac{9}{32}K_2\right)$$

$$K_4 = hf\left(x_n + \frac{12}{13}h, y_n + \frac{1.932}{2.197}K_1 - \frac{7.200}{2.197}K_2 + \frac{7.296}{2.197}K_3\right)$$

$$K_5 = hf\left(x_n + h, y_n + \frac{439}{216}K_1 - 8K_2 + \frac{3.680}{513}K_3 - \frac{845}{4.104}K_4\right)$$

$$K_6 = hf\left(x_n + \frac{1}{2}h, y_n - \frac{8}{27}K_1 + 2K_2 - \frac{3.544}{2.565}K_3 + \frac{1.859}{4.104}K_4 - \frac{11}{40}K_5\right)$$

$$n = 0, 1, 2, \cdots$$

APÊNDICE II

Códigos em Python:
Métodos de Runge-Kutta

- Runge-Kutta de Ordem 2: Método de Euler Modificado – Método do Ponto Médio

Código em Python: ***RK2i***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 21:12:54 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 2
#  Método de Euler Modificado ou Método do Ponto Médio
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK2i-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 2 - Método de Euler Modificado - RK2i
def RK2i (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + 0.5*h , y + 0.5*h*k1)
    y1 = y + h*k2

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK2i (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Runge-Kutta de Ordem 2: Método de Euler Melhorado – Método de Heun

Código em Python: ***RK2ii***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:12:34 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 2
#  Método de Euler Melhorado ou Método de Heun
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK2ii-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 2 - Método de Euler Melhorado - RK2ii
def RK2ii (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + h , y + h*k1)
    y1 = y + ((h/2)*(k1 + k2))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK2ii (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Runge-Kutta de Ordem 3 (I) – Método de Heun

Código em Python: ***RK3i***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:24:54 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 3 (I)
#  Método de Heun
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK3i-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 3 - Método de Heun - RK3i
def RK3i (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/3)*h) , y + ((1/3)*h*k1))
    k3 = f(x + ((2/3)*h) , y + ((1/3)*h*k2))
    y1 = y + (((1/4)*h) * (k1 + 3*k3))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK3i (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Runge-Kutta de Ordem 3 (II) – Método de Nystrom

Código em Python: ***RK3ii***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:32:53 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 3 (II)
#  Método de Nystrom
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK3ii-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 3 - Método de Nystrom - RK3ii
def RK3ii (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/3)*h) , y + ((1/3)*h*k1))
    k3 = f(x + ((2/3)*h) , y + ((1/3)*h*k2))
    y1 = y + (((1/4)*h) * (k1 + 3*k3))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK3ii (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Método de Runge-Kutta de Ordem 3 (III)

Código em Python: ***RK3iii***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:38:02 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 3 (III)
#
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK3iii-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 3 - RK3iii
def RK3iii (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/2)*h) , y + ((1/2)*h*k1))
    k3 = f(x + ((3/4)*h) , y + ((3/4)*h*k2))
    y1 = y + (((1/9)*h) * ((2*k1) + (3*k2) + (4*k3)))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK3iii (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Método de Runge-Kutta de Ordem 4 (I)

Código em Python: ***RK4i***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:43:23 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 4 (I)
#
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK4i-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 4 - RK4i
def RK4i (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/2)*h) , y + ((1/2)*h*k1))
    k3 = f(x + ((1/2)*h) , y + ((1/2)*h*k2))
    k4 = f(x + h , y + (h*k3))
    y1 = y + (((1/6)*h) * (k1 + (2*(k2 + k3)) + k4))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK4i (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Método de Runge-Kutta de Ordem 4 (II)

Código em Python: ***RK4ii***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:49:49 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem 4 (II)
#
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK4ii-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 4 - RK4ii
def RK4ii (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/3)*h) , y + ((1/3)*h*k1))
    k3 = f(x + ((2/3)*h) , y - ((1/3)*h*k1) + (h*k2))
    k4 = f(x + h , y + (h*k1) - (h*k2) + (h*k3))
    y1 = y + (((1/8)*h) * (k1 + (3*(k2 + k3)) + k4))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK4ii (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

- Método de Runge-Kutta de Ordem 5 de Butcher

Código em Python: ***RK5B***

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 27 23:53:53 2018

@author: Dherik França HP
"""

#############################################################
#
#  Métodos Numéricos e Computacionais
#  Runge-Kutta de Ordem  de Butcher
#
#  -Baseado no Programa 'euler.py'
#  Proprietário: Professor Tiago Martinuzzi Buriol
#  -RK5B-
#  Proprietário: Dherik França Menezes
#  27/04/2018
#############################################################
#

# Importar as Bibliotecas:
#% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import math

# Inserir o passo:
h = 0.1

# Intervalo x:
x0 = 0.0
xn = 0.3

# Definir k2=0
k2=0

# Lista com os pontos xi:
# Imprimir os pontos xi:
X = np.arange(x0, xn+h, h)
print ("pontos xi:", X)

# Lista com y0, a ser preenchida pelos yi:
# Problema de Valor Inicial: y(0.0) = 2.0
Y = [2.0]

# Função a ser integrada numericamente:
f = lambda x,y: x - y + 2
```

```
# Método utilizado:
# Método de Runge-Kutta de Ordem 5 de Butcher - RK5B
def RK5B (f, x, y, h):
    k1 = f(x , y)
    k2 = f(x + ((1/4)*h) , y + ((1/4)*h*k1))
    k3 = f(x + ((1/4)*h) , y + ((1/8)*h*k1) + ((1/8)*h*k2))
    k4 = f(x + ((1/2)*h) , y - ((1/2)*h*k2) + (h*k3))
    k5 = f(x + ((3/4)*h) , y + ((3/16)*h*k1) + ((9/16)*h*k4))
    k6 = f(x + h , y - ((3/7)*h*k1) + ((2/7)*h*k2) + ((12/7)*h*k3) - ((12/7)*h*k4)+
((8/7)*h*k5))

    y1 = y + (((1/90)*h) * ((7*k1) + (32*k3) + (12*k4) + (32*k5) + (7*k6))

    return (y1)

# Laço para calcular os yi e preencher a lista:
for x in X[0:-1]:
    Y.append(RK5B (f,x,Y[-1],h) ) #Y[-1] é o último yi calculado

# Imprimir os pontos yi:
print ("pontos yi:", Y)

# Plotar o Gráfico da Aproximação:
plt.plot(X, Y, "r-")
plt.grid()
plt.show()

# Plotar a Solução Exata do P.V.I. para comparar com a Aproximação:
Xex = np.arange(0, .31, 0.01)
Yex = []
sol_ex = lambda x: x + 1 + (math.exp(-x))

for x in Xex:
    Yex.append(sol_ex(x))

plt.plot(X, Y, "r-", Xex, Yex, "b-")
plt.grid()
plt.show()
```

APÊNDICE III

A Variable Order Runge-Kutta Method for Value Problems with Rapidly Varying Right-hand Sides

Jeff R. Cash, Alan H. Karp

ACM Transactions on Mathematical Software, Vol. 16, No. 3, September 1990

A Variable Order Runge—Kutta Method for Value Problems with Rapidly Varying Right-hand Sides

Article *in* ACM Transactions on Mathematical Software · September 1990

DOI: 10.1145/79505.79507 · Source: DBLP

A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides

J. R. CASH
Imperial College
and
ALAN H. KARP
IBM Scientific Center

Explicit Runge-Kutta methods (RKMs) are among the most popular classes of formulas for the approximate numerical integration of nonstiff, initial value problems. However, high-order Runge-Kutta methods require more function evaluations per integration step than, for example, Adams methods used in PECE mode, and so, with RKMs, it is especially important to avoid rejected steps. Steps are often rejected when certain derivatives of the solution are very large for part of the region of integration. This corresponds, for example, to regions where the solution has a sharp front or, in the limit, some derivative of the solution is discontinuous. In these circumstances the assumption that the local truncation error is changing slowly is invalid, and so any step-choosing algorithm is likely to produce an unacceptable step. In this paper we derive a family of explicit Runge-Kutta formulas. Each formula is very efficient for problems with smooth solutions as well as problems having rapidly varying solutions. Each member of this family consists of a fifth-order formula that contains imbedded formulas of all orders 1 through 4. By computing solutions at several different orders, it is possible to detect sharp fronts or discontinuities before all the function evaluations defining the full Runge-Kutta step have been computed. We can then either accept a lower order solution or abort the step, depending on which course of action seems appropriate. The efficiency of the new algorithm is demonstrated on the DETEST test set as well as on some difficult test problems with sharp fronts or discontinuities.

Categories and Subject Descriptors: G.1.7 [**Numerical Analysis**]; Ordinary Differential Equations

Additional Key Words and Phrases: Differential equations, Runge-Kutta methods, singularities

1. INTRODUCTION

Explicit Runge-Kutta methods (RKMs) are among the most popular classes of formulas for the numerical integration of the nonstiff initial value problem

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0. \tag{1}$$

Authors' addresses: J. R. Cash, Department of Mathematics, Imperial College, South Kensington, London SW7 2AZ, England; A. H. Karp, IBM Scientific Center, 1530 Page Mill Road, Palo Alto, CA 94304.

Because of their one-step nature, Runge-Kutta methods are self-starting, and can change the step size of integration as often and by as much as required. Efficient Adams formulas, such as the widely used code STEP [14], can also change step size at every step, but in order to reduce overhead and to enhance its theoretical support, it is strongly biased against doing so. RKMs also have the advantage that the theory supporting the interpolation in a variable step Adams code is less well developed than for Runge-Kutta methods.

One frequently quoted disadvantage of RKMs is that they require more function evaluations per step than do linear multistep methods used in PEC or PECE mode, and so may be uncompetitive with Adams methods if function evaluations are very expensive. However, one of the present authors (AHK) has been heavily involved in a spectral method for the solution of oil-reservoir problems. These investigations have shown that, despite the fact that function evaluations are extremely expensive, the great flexibility of step size allowed by RKMs often solves these problems somewhat more efficiently, and with considerably less storage space, than when using Adams methods.

This is due primarily to the erratic behavior of the solution, which typically has smooth regions coupled to very sharp fronts. An effective method for such problems needs to make frequent changes of step size. In solving these oil-reservoir problems, it was very noticeable that in regions where the solution trajectory was particularly rough, there were many rejected steps, which resulted in an expensive integration. In part, the present investigation arose from a desire to predict that a step will be rejected and to quit, or accept a lower order solution, before all the function evaluations required to complete a full Runge-Kutta step had been computed.

It is clear that, even though linear multistep methods use fewer function evaluations per step than Runge-Kutta methods, this advantage can be lost when the solution requires frequent changes in step size. Thus the claim that Runge-Kutta methods are generally less efficient than linear multistep methods when function evaluations are expensive—because they do more function evaluations per step—is not always valid. In practice it is often found that Runge-Kutta methods take larger steps, on average; and it is the distance divided by the cost that is important. However, when the step size is restricted by, for example, output requirements or nonsmooth behavior such as singularities, so that Runge-Kutta methods are unable to take a relatively large step, it is the absolute cost that is relevant.

For the oil-reservoir problems that one of us has solved, it is the flexibility of the step-size selection in the Runge-Kutta methods that is important. Typically the step size is cut by a large factor every 20 or so steps and by smaller amounts every 5 to 10 steps. While a Runge-Kutta method can decrease the step size a small amount, or increase the step size by a large amount at every step (typical codes allow a step increase by a factor of 5), linear multistep methods (as typified by [14]) can only halve the step size or double it every few steps. The ability of Runge-Kutta methods to change the step size by small amounts, as needed by the problem, in addition to increasing the step size rapidly when entering a smooth region, gives the Runge-Kutta methods a larger average step size than multistep methods on our reservoir problems.

Once we became interested in problems with solutions exhibiting very sharp fronts, it was natural for us to consider discontinuous initial value problems which are, in a sense, the limiting case. We use the term *discontinuous initial value problem* rather loosely to refer to a problem with a solution having a discontinuous derivative of some order. Such problems have recently become the focus of renewed attention [4, 7, 8], and there is considerable interest in solving them efficiently, reliably, and automatically. Although discontinuous IVPs are not our main topic of interest, we show in Section 4 that the algorithms developed in this paper do perform very efficiently on a large class of such problems.

A very widely used fixed-order Runge-Kutta code is RKF45 of Shampine and Watts [16]. The RKM on which this code is based uses a total of six function evaluations per step to give a main formula of order 5 and an imbedded formula of order 4. The difference between these two solutions gives a local error estimate in the fourth-order solution. If this error is less than a prescribed tolerance, it is normally the fifth-order solution that is accepted (i.e., local extrapolation is performed). This procedure is, of course, simply the well-known Fehlberg imbedding technique applied to a 5(4) formula.

An important extension to this idea was given by Bettis [1], and further developed by Verner [17]. Verner constructed special Runge-Kutta formulas, which he called CSIRK methods, that contain a complete set of imbedded RKMs. He gave families of order p that contain imbedded methods of all orders $1, 2, \ldots, p-1$ for $p = 5, 6, 7$, and 8. One possible application of these formulas is in the construction of a family of variable-order, imbedded Runge-Kutta methods with order $\leq p$. Shampine et al. [15] used a similar idea to choose between 4(3) and 8(7) explicit Runge-Kutta methods. They showed that there is often a big advantage in varying the order of Runge-Kutta codes. An important conclusion of Shampine et al. is that Runge-Kutta methods are very efficient if the order is properly matched to the accuracy required. Furthermore, they claim that if one could select at each step the best fixed-order Runge-Kutta code for that step, the resulting algorithm would compete with the best Adams codes in terms of derivative evaluations, and would possess several important advantages. We have found this to be especially true for problems having solutions with sharp fronts or discontinuities where a low-order solution often has very good accuracy while a higher order one has very poor accuracy.

Although there are some similarities between our approach and that of Shampine et al. [15], the two differ in several important respects. First, we implement a complete set of Runge-Kutta formulas, whereas they consider only 4(3) and 8(7) formulas. Second, we only accept a lower order solution that passes the error test if higher order solutions fail this test. This strategy is justified later in this paper. Finally, we are mainly interested in applying our formulas to problems having regimes where the solution is "rough" or discontinuous. In particular, our approach has proved to be very effective for integrating through points where certain low-order derivatives of the solution are either discontinuous or extremely large. For such problems, a low-order formula is normally more effective than a higher order one, and also has the possibility of giving a reasonable error estimate. Shampine et al. explicitly exclude this class of problem.

The variable order RKM which we describe reduces substantially the number of function evalutions for many of the problems we have solved and requires only a slight increase in overhead, which can normally be neglected if function evaluations are reasonably expensive. The basic formula we use is (6, 5) CSIRK, derived using the theory developed by Verner [17]. A six-stage CSIRK formula allows the construction of RKMs of all orders from 1 to 5, but as we explain in Section 3, we only allow the possibility of accepting orders 2, 3, or 5. At each step we use Fehlberg imbedding to compute an estimate of the error in lower order solutions. If the local error estimate in a low-order solution indicates that the step is likely to be rejected at order 5, we immediately reject the step and investigate the possibility of accepting a lower order solution.

This strategy has a number of advantages. If the solution has a sharp front or discontinuity, we frequently get the largest increase in step size in fewer than six function evaluations. More importantly, since we can often detect when the step needs to be cut, without computing all six function evaluations, the penalty for rejecting a step is greatly reduced. We have modified the code RKF45 of Shampine and Watts to incorporate these ideas. A listing of the code is available from one of the present authors (JRC) and from NETLIB.

2. A MODIFIED CSIRK METHOD

In this section we derive a special Runge-Kutta formula of the form

$$
\begin{array}{c|cccccc}
0 & 0 & & & & & \\
c_2 & a_{21} & 0 & & & & \\
c_3 & a_{31} & a_{32} & 0 & & & \\
c_4 & a_{41} & a_{42} & a_{43} & 0 & & \\
c_5 & a_{51} & a_{52} & a_{53} & a_{54} & 0 & \\
c_6 & a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & 0 \\
\hline
 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6
\end{array}
\tag{2}
$$

which has order 5, and also contains a complete family of imbedded Runge-Kutta formulas of orders 1, 2, 3, and 4. Runge-Kutta formulas with a complete family of imbedded methods, called CSIRK methods, have been studied in detail by Verner [17], who derived CSIRK formulas of order 5, 6, 7, and 8. Because of the special application we have in mind, we derive a different CSIRK formula from the one given by Verner. In what follows, we describe the design criteria for our formulas.

Our first concern is to derive formulas and error estimates that are of "good quality" for general initial value problems. With this in mind, we chose the coefficients of our CSIRK formulas so that their local truncation errors have the form suggested by Dormand and Prince [5] and Shampine [13]. It is useful to give a brief summary of these ideas here.

For discussion, we wish to compute a numerical solution of the initial value problem (1) using a Runge-Kutta formula of order p. We further assume that the numerical solution y_n at the step point x_n has already been computed and that our pth-order Runge-Kutta formula gives a solution y_{n+1} at the step point

$x_{n+1} = x_n + h$. The local solution $u(x)$ is defined to be the solution of

$$\frac{du}{dx} = f(x, u), \quad u(x_n) = y_n. \tag{3}$$

It is well known that

$$\begin{aligned} &u(x_n + h) - y_{n+1} \\ &\quad = h^{p+1} \sum_{j=1}^{\lambda_{p+1}} T_{p+1,j} D_{p+1,j} + h^{p+2} \sum_{j=1}^{\lambda_{p+2}} T_{p+2,j} D_{p+2,j} + O(h^{p+3}), \end{aligned} \tag{4}$$

where λ_{p+1} and λ_{p+2} are integers depending on p, the $D_{p+i,j}$ are elementary differentials depending on the problem, and the $T_{p+i,j}$ are error coefficients depending on the Runge-Kutta formula. It has been suggested by Dormand and Prince [5] that every elementary differential should contribute to both the local error and its estimate. We have constructed our formulas in line with this suggestion. From Eq. (4), we see that this requirement is $T_{p+1,j} \neq 0$ for all j.

The next design criterion of interest concerns the relative size of the local error in the imbedded formula compared with that of the higher order formula. The larger the error in the imbedded formula, the more accurate the local error estimate will tend to be. However, the step control procedure will be more conservative, resulting in smaller steps and a generally more expensive integration. This question has been investigated by Shampine [13], and we have derived our formulas with his suggestions in mind. For further comments on this subject and for additional references, see Cash [3].

There is a third important design criterion we need to take into account: the fact that we are particularly interested in nonsmooth solutions. We wish to choose the c_i so that they span the range [0, 1] with reasonable uniformity. To explain the need for this criterion, we describe our approach for detecting nonsmooth behavior, which can best be done by considering imbedded formulas of order 2 and 3. The second-order formula involves function evaluations at points x_n and $x_n + c_2 h$, while the third-order formula involves function evaluations at $x_n + c_i h$, $1 \leq i \leq 4$. Embedded in the second-order formula we have a formula of order 1, and an estimate of the error in the first- and second-order solutions can be obtained by imbedding in the usual way. If the order 1 and 2 solutions are sufficiently accurate (in a precisely defined sense, to be explained in Section 3), then we go on to compute the fourth-order solution with the expectation that the step length is sufficiently small to allow a highly accurate, high-order solution to be computed. If, however, the first-order solution has good accuracy while the second-order solution has very poor accuracy, then we would expect a "trouble spot" in the range $[x_n + c_2 h, x_n + c_4 h]$. If this were the case, we would accept the order 1 solution and reduce the step (as described in Section 3). Similarly, we have a well-defined course of action if the first-order solution has poor accuracy.

Given this strategy, it is clear that we would like the c_i to span the range [0, 1] and to be reasonably equally spaced, with one of the $c_i = 1$. This choice gives us a very good chance of detecting bad behavior in the right-hand side of Eq. (1), if it occurs. After having derived all the order relations for our CSIRK formulas, we attempted to choose the coefficients to satisfy all of these (somewhat impre-

cise) design criteria. The formula that we finally derived is:

$$
\begin{array}{c|cccccc}
0 & 0 & & & & & \\
\frac{1}{5} & \frac{1}{5} & 0 & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & 0 & & & \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & 0 & & \\
1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & 0 & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & 0 \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \quad \text{Order 5} \\
 & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4} \quad \text{Order 4} \\
 & \frac{19}{54} & 0 & -\frac{10}{27} & \frac{55}{54} & 0 & 0 \quad \text{Order 3} \\
 & -\frac{3}{2} & \frac{5}{2} & 0 & 0 & 0 & 0 \quad \text{Order 2} \\
 & 1 & 0 & 0 & 0 & 0 & 0 \quad \text{Order 1}
\end{array}
\tag{5}
$$

When used as a fixed-order method, we refer to Eq. (5) as RKFNC; when used as a variable order method, as VRKF.

It is interesting to examine the first few terms in the local error expansion of this formula and to compare them with the errors in certain other widely used 5(4) formulas. A reasonable way of measuring the local error in a pth-order formula, which has been used by several other authors, is to compute

$$\| T_{p+1} \|_2 \equiv \sqrt{\sum_{j=1}^{\lambda_{p+1}} T_{p+1,j}^2} \quad \text{and} \quad \| T_{p+2} \|_2 \equiv \sqrt{\sum_{j=1}^{\lambda_{p+2}} T_{p+2,j}^2}.$$

(See Eq. (4).) These relations give the 2-norm of the error coefficients of order $p + 1$ and $p + 2$ which are the first two nonvanishing terms in Eq. (4). For the formula described in this paper and two reference formulas, the details are given in Table I.

We note that, using these measures to quantify the local error, RKFNC has considerably smaller terms in its local error than either RKF45 or Verner's fifth-order CSIRK method. The hope is that on problems with reasonably smooth solutions, using RKFNC will result in increased efficiency. As we will see in Section 4, this hope is realized for the DETEST test set. Before giving any

Table I. Error Terms

Formula		Order	$\|T_5\|_2$	$\|T_6\|_2$	$\|T_7\|_2$
RKF45	Main	5	0	0.0034	0.0068
	Embedded	4	0.0018	0.0058	—
Verner 5(4)	Main	5	0	0.0037	0.0070
	Embedded	4	0.0012	0.0054	—
RKFNC	Main	5	0	0.0009	0.0013
	Embedded	4	0.0005	0.0011	—

numerical results, however, we first explain in detail our computational procedure for dealing with any regions where the solution of our problem exhibits particularly nonsmooth behavior.

3. A STRATEGY FOR DEALING WITH NONSMOOTH BEHAVIOR

The Runge-Kutta formula derived in the previous section has the special property that it contains imbedded solutions of all orders less than five. In addition, the formula has been designed so that the first five c_i values span the range [0, 1] with reasonable uniformity, so that we have a very good chance of spotting bad behavior in f if it occurs. Our aim is to derive an automatic strategy that allows us to quit early, i.e., before all six function evaluations have been computed on the current step, if we suspect trouble, and to accept a lower order solution if appropriate.

We assume that we have computed a numerical solution y_{n-1} at the step point x_{n-1} and that for the current step, from x_{n-1} to $x_n = x_{n-1} + h$, all six function evaluations are computed so that solutions of all orders from 1 to 5 are available. (We guarantee this situation for the first step with $n = 1$). We denote the imbedded solution of order i at x_n by $y_n^{(i)}$, $1 \le i \le 5$, and define

$$\mathrm{ERR}(n, i) = \| y_n^{(i+1)} - y_n^{(i)} \|^{1/(i+1)}, \qquad \text{for} \quad i \in 1, 2, 4. \tag{6}$$

We exclude the case $i = 3$ for two reasons. First, following the approach of Shampine et al. [15], we allow only a few different orders to be used, and we have chosen to allow orders 2, 3, or 5. Second, ERR(n, 3) is of no use in predicting when to quit early since all six k_i's are required before $y_n^{(4)}$ can be computed.

Suppose now that we were to accept the solution of order 5 at x_n. We wish to compute a suitable step length, $\bar{h}_4$, to be used in integrating from x_n to x_{n+1} using a 5(4) formula. A typical step-choosing strategy would compute $\bar{h}_4$ as

$$\bar{h}_4 = \frac{\mathrm{SF} \times h}{\mathrm{E}(n, 4)}, \qquad \text{where} \quad \mathrm{E}(n, 4) = \frac{\mathrm{ERR}(n, 4)}{\epsilon^{1/5}}. \tag{7}$$

Here ϵ is the local accuracy required (as specified by the user) and SF is a safety factor often taken to be 0.9. Similarly, if we were to accept either the second- or third-order solution at x_n, the steplengths $\bar{h}_1$, $\bar{h}_2$, respectively, that would be selected at the next step by our step-control algorithm would be

$$\bar{h}_i = \frac{\mathrm{SF} \times h}{\mathrm{E}(n, i)}, \qquad \text{where} \quad \mathrm{E}(n, i) = \frac{\mathrm{ERR}(n, i)}{\epsilon^{1/(i+1)}}, \quad i \in 1, 2. \tag{8}$$

From relations (7) and (8), we can compute what we call our quitting factors,

$$\mathrm{QUIT}(n, i) = \frac{\bar{h}_4}{\bar{h}_i} \equiv \frac{\mathrm{E}(n, i)}{\mathrm{E}(n, 4)}, \quad i \in 1, 2. \tag{9}$$

To explain how these quitting factors are used, it is convenient to assume that the fifth-order solution $y_n^{(5)}$ is accepted at x_n. We now consider what happens in integrating from x_n to x_{n+1}. The first stage in integrating forward from x_n is to compute two function evaluations k_1 and k_2, and use these to compute first- and second-order solutions, $y_{n+1}^{(1)}$ and $y_{n+1}^{(2)}$, at x_{n+1}. We can now compute

$$\mathrm{ERR}(n + 1, 1) \equiv \| y_{n+1}^{(2)} - y_{n+1}^{(1)} \|^{1/2}. \tag{10}$$

Use of this error estimate allows us to compute the step $\bar{h}_1$, which would be used for the next step if we were to accept the first-order solution:

$$\bar{h}_1 = \frac{\mathrm{SF} \times h}{\mathrm{E}(n + 1, 1)}, \tag{11}$$

where $\mathrm{E}(n + 1, 1)$ is defined as in Eq. (8).

We now have sufficient information to allow us to estimate the step $\tilde{h}_4$ which would be selected for the next step if all six function evaluations defining the current step were computed and a fifth-order solution, $y_{n+1}^{(5)}$, accepted. By definition $\tilde{h}_4 = \mathrm{QUIT}(n + 1, 1) \times \bar{h}_1$. We now assume that $\mathrm{QUIT}(n + 1, 1) \simeq \mathrm{QUIT}(n, 1)$, where $\mathrm{QUIT}(n, 1)$ is available from the previous step. Now we can make the approximation

$$\tilde{h}_4 = \mathrm{QUIT}(n, 1) \times \bar{h}_1 = \frac{\mathrm{QUIT}(n, 1) \times \mathrm{SF} \times h}{\mathrm{E}(n + 1, 1)}. \tag{12}$$

If $\tilde{h}_4 < \mathrm{SF} \times h$, that is,

$$\mathrm{QUIT}(n, 1) < \mathrm{E}(n + 1, 1), \tag{13}$$

then we expect that the fifth-order solution will be rejected, and we should abandon the current step. It is important to realize that this test can be performed after only two function evaluations have been computed. Although this approach forms the basis of our strategy, it is not quite the strategy that is used in practice, as we now explain.

An important part of our strategy is the assumption that the quit factors are changing slowly from step to step (as they normally will with smooth solutions), so a big change in the quit factor for a low-order solution signals trouble and causes us to abandon the current step. However, we must allow the quit factors to increase slowly from step to step, which means that the satisfaction of Eq. (13) is too severe a requirement to impose. We overcome this problem by introducing a *twiddle factor*, so that our code will quit early when

$$\mathrm{E}(n + 1, 1) > \mathrm{QUIT}(n, 1) \times \mathrm{TWIDDLE}(n, 1). \tag{14}$$

We experimented with several twiddle factors using only the information that they should be greater than unity and not "too big," i.e., probably less than 1.5. Our experiments showed clearly that the optimal value of these twiddle factors was problem-dependent, and so in our implementation we allowed them to be chosen dynamically. At the end of this section we list our complete strategy in step-by-step form. Before doing so, we explain what happens when a lower order solution is accepted. We first explain under what circumstances we accept solutions of various orders.

We assume that the current step is successful, i.e., $E(n, i) < 1$ for at least one $i \in [1, 2, 4]$. Then if

(i) $E(n, 1) < 1$, $E(n, 2) > 1$, $E(n, 4) > 1$, accept $y_{n+1}^{(2)}$
(ii) $E(n, 1) < 1$, $E(n, 2) < 1$, $E(n, 4) > 1$, accept $y_{n+1}^{(3)}$
(iii) $E(n, 1) < 1$, $E(n, 2) < 1$, $E(n, 4) < 1$, accept $y_{n+1}^{(5)}$.

Thus, we see that at each step we only accept the second-, third-, or fifth-order solutions. This strategy of allowing only a few different orders in a variable order Runge-Kutta code (i.e., not allowing first- or fourth-order solutions in our case) is in line with the results of Shampine et al. [15], who found this approach to be the most efficient.

Let us now suppose that case (i) holds, and we wish to accept the order 2 solution. The error estimate in this solution is based on the following information.

$$\begin{array}{c|cc} 0 & 0 & \\ \frac{1}{5} & \frac{1}{5} & 0 \\ \hline & & \end{array}$$

Since we have only used values of c_i up to 1/5, and we suspect there may be trouble ahead due to the unacceptability of higher order solutions, it would be dangerous to integrate past $x_n + h/5$ on the current step. Thus, using the above information, the solution we compute at $x_{n+1} = x_n + h/5$ is

$$y_{n+1} = y_n + \frac{h}{10}(k_1 + k_2). \tag{15}$$

The corresponding order 1 solution is

$$\bar{y}_{n+1} = y_n + \frac{h}{5} k_1$$

and the error estimate for this solution is

$$E_n^1(1/5) \equiv y_{n+1} - \bar{y}_{n+1} = \frac{h}{10}(k_2 - k_1).$$

If $\| E_n^1(1/5) \| > \epsilon$, then the step is abandoned and the integration is continued from (x_n, y_n) with a step $h/5$. Otherwise, the solution y_{n+1} is accepted, and the value $h/5$ is used for the next integration step starting from $(x_n + h/5, y_{n+1})$.

A similar strategy is used if the third-order solution, $y_{n+1}^{(3)}$, is accepted. The information on which this solution and its error estimate are based is as follows.

0	0				
$\frac{1}{5}$	$\frac{1}{5}$	0			
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0		
$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$	0	
	$\frac{19}{54}$	0	$-\frac{10}{27}$	$\frac{55}{54}$	Order 3
	$-\frac{3}{2}$	$\frac{5}{2}$			Order 2

Similar arguments to those used for the first-order solution can now be applied. Since the order 3 solution is acceptable, while the order 4 solution is not, we anticipate possible trouble in the range $[x_n + 3h/5, x_n + h]$. As a result, it is only safe to integrate forward a distance $3h/5$ during the current step. We therefore seek a third-order formula of the form

$$y_{n+3/5} = y_n + \frac{3}{5} h(b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4)$$

which uses information that has already been computed to integrate forward a distance $3h/5$ rather than h. It is straightforward to show that the choice $b_1 = b_4 = 1/6$, $b_3 = 2/3$, $b_2 = 0$ satisfies this requirement. Our strategy is to accept the third-order solution with

$$y_{n+3/5} = y_n + h\left(\frac{1}{10} k_1 + \frac{2}{5} k_3 + \frac{1}{10} k_4\right) \tag{16}$$

as the solution at $x_n + 3h/5$, providing that it satisfies an appropriate error criterion. A second-order formula at $x_{n+3/5}$ is

$$\bar{y}_{n+3/5} = y_n + \frac{3}{5} h k_3.$$

An estimate of the error for this solution is

$$\mathrm{E}_n^2(3/5) \equiv y_{n+3/5} - \bar{y}_{n+3/5} = \frac{h}{10}(k_1 - 2k_3 + k_4).$$

If $\| \mathrm{E}_n^2(3/5) \| > \epsilon$, then we abandon the third-order solution and test the acceptability of a second-order solution at $x_n + h/5$, as previously described. Otherwise, the solution $y_{n+3/5}$ is accepted, and we continue the integration from $(x_n + 3h/5, y_{n+3/5})$ using a step $3h/5$.

We see from this approach that important use is made of what Shampine et al. refer to in [15] as "fall back" formulas. The idea is that, if an integration

over the whole step $[x_n, x_n + h]$ using the 5(4) pair fails because of something bad in the region $(x_n + h/5, x_n + h]$, then it is possible to use an independent, lower order solution that uses information taken only from the first part of the step. In our algorithm we have a "fall back" formula that uses information only in the region $[x_n, x_n + h/5]$ if the third-order solution appears bad, and a second "fall back" formula that uses information in the region $[x_n, x_n + 3h/5]$ if the fourth-order solution appears bad.

There are two different interpretations of what might go wrong in a step to cause a higher order solution to be rejected. It is convenient to discuss this point in terms of the lower order formulas. If the (1, 2) pair succeeds, but the (2, 3) pair does not, it may be that the (1, 2) pair has not sampled the function in the range $(x_n + h/5, x_n + h]$ where something bad happens that the third-order formula detects. Alternatively, it may be that the problem "looks" smooth to the second-order formula, but does *not* look smooth to the third-order formula. In the latter case, the third-order formula may be accurately reflecting the true behavior of the solution, and there may be nothing wrong with the last part of the step. In constructing our algorithm we have adopted the first point of view, i.e., that failure of a high-order solution, but success in a low-order solution, indicates bad behavior in the latter part of the step.

We now explain our complete algorithm in more detail. First, we initialize the constants.

(1) Set the twiddle factors.

These factors can be arbitrary, but they should be set just larger than unity. In our code we take TWIDDLE(0, 1) = 1.5 and TWIDDLE(0, 2) = 1.1. As explained earlier, these values will be changed automatically by the code.

(2) Set the quit factors.

These values should be set to be quite large initially to make the code take a full step at the start. In our code we use QUIT(0, 1) = QUIT(0, 2) = 100.

Next, we integrate one step at a time. Assume that an approximate solution y_{n-1} has been computed at the step point x_{n-1}.

(1) Compute the first two function evaluations in VRKF.
(2) From these evaluations obtain an error estimate ERR(n, 1) in the order 1 solution using Fehlberg imbedding with the order 1 and 2 solutions.
(3) Compute E(n, 1) as in (8).
(4) Check the error estimate.

IF E(n, 1) > TWIDDLE($n - 1$, 1) × QUIT($n - 1$, 1),
THEN abandon the step.
a. Put ESTTOL = E(n, 1)/QUIT($n - 1$, 1).
b. Choose the next step $\bar{h}$ as

$$\bar{h} = \max\left(\frac{1}{5}, \frac{SF}{\text{ESTTOL}}\right) \times h. \tag{17}$$

c. Go to step 1.
ENDIF

(5) Compute the third and fourth function evaluations in VRKF.
(6) Compute the order 3 solution.

(7) Compute $\mathrm{ERR}(n, 2) = \| y_n^{(3)} - y_n^{(2)} \|^{1/3}$ and hence $\mathrm{E}(n, 2)$ from (8).
(8) Check the error estimate.

IF $\mathrm{E}(n, 2) > \mathrm{TWIDDLE}(n-1, 2) \times \mathrm{QUIT}(n-1, 2)$,
THEN try a lower order solution.
IF $\mathrm{E}(n, 1) < 1$,
THEN check the error of the second order solution.
IF $\| \mathrm{E}_n^1(1/5) \| < \epsilon$,
THEN accept the second order solution computed from (15).
a. Cut the step from h to $h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.
ELSE abandon the step.
a. Cut the step from h to $h/5$.
b. Go to step 1.
ENDIF
ELSE abandon the step.
a. Set $\mathrm{ESTTOL} = \mathrm{E}(n, 2)/\mathrm{QUIT}(n-1, 2)$.
b. Choose the step $\bar{h}$ as in (17).
c. Go to step 1.
ENDIF
ENDIF

(9) Compute the final two function evaluations in VRKF.
(10) Use all six function evaluations to compute the order 4 and 5 solutions.
(11) Compute $\mathrm{ERR}(n, 4) = \| y_n^{(5)} - y_n^{(4)} \|^{1/5}$ and hence $\mathrm{E}(n, 4)$ from (8).
(12) Check the error estimate.

IF $\mathrm{E}(n, 4) > 1$,
THEN readjust the twiddle factors.
IF $\mathrm{E}(n, i)/\mathrm{QUIT}(n-1, i) < \mathrm{TWIDDLE}(n-1, i), \quad i \in 1, 2$
THEN $\mathrm{TWIDDLE}(n, i) = \mathrm{MAX}(1.1, \mathrm{E}(n, i)/\mathrm{QUIT}(n-1, i))$.
ELSE $\mathrm{TWIDDLE}(n, i) = \mathrm{TWIDDLE}(n-1, i)$.
ENDIF
IF $\mathrm{E}(n, 2) < 1$,
THEN check the accuracy of the third order solution.
IF $\| \mathrm{E}_n^2(3/5) \| < \epsilon$,
THEN accept the order 3 solution computed from (16).
a. Cut the step from h to $3h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.
ENDIF
ELSE try a lower order solution.
IF $\mathrm{E}(n, 1) < 1$,
THEN check the accuracy of the second order solution.
IF $\| \mathrm{E}_n^1(1/5) \| < \epsilon$,
THEN accept the order 2 solution computed from (15).
a. Cut the step from h to $h/5$.
b. Replace n by $n + 1$.
c. Go to step 1.

ELSE abandon the step.
a. Cut the step from h to $h/5$.
b. Go to step 1.
ENDIF
ELSE abandon the current step.
a. Set the step $\bar{h}$ as in (17) where ESTTOL $\equiv$ E$(n, 4)$.
b. Go to step 1.
ENDIF
ENDIF
ELSE accept the order 5 solution for $x_n = x_{n-1} + h$.
a. Choose the new step $\bar{h}$ as

$$\bar{h} = \min\left(5.0, \frac{SF}{\mathrm{E}(n, 4)}\right) \times h.$$

b. Update the quit factors. (We will give our strategy here and explain it later.) Define $Q_1 = \mathrm{E}(n, 1)/\mathrm{E}(n, 4)$ and $Q_2 = \mathrm{E}(n, 2)/\mathrm{E}(n, 4)$. Then for $j \in (1, 2)$,
IF $Q_j >$ QUIT$(n-1, j)$,
THEN $Q_j = \min(Q_j, 10 \times \mathrm{QUIT}(n-1, j))$.
ELSE $Q_j = \max(Q_j, \frac{2}{3} \times \mathrm{QUIT}(n-1, j))$.
ENDIF
QUIT$(n, j) = \max(1.0, \min(10000, Q_j))$.
c. Set TWIDDLE$(n, j) =$ TWIDDLE$(n-1, j)$.
ENDIF

The quit factors are not allowed to vary by arbitrary amounts at each step, since we do not want an isolated, very good solution or an isolated, very poor solution to have a big effect on the quit factors. We prefer to have the quit factors change slowly from step to step. A second reason for limiting the change in the quit factors comes from examining the error test

IF E$(n, j) >$ TWIDDLE$(n-1, j) \times$ QUIT$(n-1, j)$, **THEN** quit.

We see from this test that, if our quit factors are too small, then we are forced to quit early unnecessarily. Such a situation may occur when we move from a smooth region, where the quit factors will normally be large, into a rough region, where the quit factors would become very small if allowed to become so. If we then move back into a smooth region (which would typically happen after passing through a discontinuity), we may not be able to increase the step very quickly, due to the extremely small quit factors.

A final factor is that it is generally better to complete the step by computing all six function evaluations, and then to reject the solution, than to quit early when a full step would have been accepted. When a full step has been completed, we have a complete set of information on which to base our step-choosing strategy; when we quit early, we have much less information available for choosing h.

For all these reasons, it is much more desirable for the quit factors to be too large than too small. (However, for smooth problems the quit factors normally do an extremely good job of judging when to quit early.) In view of these observations, we limit the change in quit factors from step to step, but allow

Table II. Distance Forward per Function Evaluation

Order	Function evaluations (F)	Distance forward (D)	D/F
2	2	$h/5$	$h/10$
3	4	$3h/5$	$3h/20$
5	6	h	$h/6$

them to increase much more rapidly than they are allowed to decrease. In our program we limit the decrease to a factor of $\frac{2}{3}$ while allowing an increase of up to a factor of 10 per step.

Finally, in this section we wish to explain why we do not necessarily accept a lower order solution early and abandon the rest of the step if it passes the error test. A strategy of accepting early could well be used in a variable order code designed for smooth solutions, but we are particularly interested in problems where there is a distinct lack of smoothness. If we were to accept early and adopt the step-choosing strategy described earlier in this section, the distance travelled forward per function evaluation is as shown in Table II. We see that, on a step where solutions of all orders pass the error test, it is the order 5 solution that goes the farthest forward per function evaluation.

There is an obvious way to increase the distance travelled for the low-order solutions if we believe the solution to be smooth. Suppose the order 1 solution passes the error test. We accept this solution and compute

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{18}$$

Accepting the solution y_{n+1} can be regarded as a bit suspect because we have only used information in the range $[x_n, x_n + h/5]$, and we have not sampled f at all in the range $(x_n + h/5, x_{n+1}]$. However, we go on to the next step and compute $f(x_{n+1}, y_{n+1})$. Using this extra function evaluation allows us to compute a second estimate of the error in y_{n+1} as

$$\begin{aligned} \mathrm{E} &\equiv y(x_{n+1}) - y_{n+1} \\ &\simeq \underbrace{\left\{ y_{n+1} - y_n - \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right] \right\}}_{\text{Order 2 solution}} - \underbrace{\left\{ y_{n+1} - y_n - hf(x_n, y_n) \right\}}_{\text{Order 1 solution}} \\ &= \frac{h}{2}\left[f(x_n, y_n) - f(x_{n+1}, y_{n+1})\right]. \end{aligned}$$

This new error estimate uses information at x_n and x_{n+1}. If this estimate is less than the prescribed tolerance, then we continue with the next step (the cost of computing the additional error estimate will be negligible). Otherwise, we accept

$$y_{n+1} = y_n + \frac{h}{5} f(x_n, y_n).$$

The cost of having to reject the solution defined by (18) will be one function evaluation. We have not examined this approach because we are interested mainly in nonsmooth solutions.

4. NUMERICAL RESULTS

In this section we give some numerical results that were obtained by applying a code based on Eq. (5) and the ideas explained in Section 3 to some test problems. We consider first the performance of our code on some smooth problems. Then we look at how our code does with some rough problems.

The smooth problems chosen are the well-known DETEST test set which contains five classes of smooth problems. Table III compares the results of three codes, RKF45, RKFNC, and VRKF. RKFNC is identical to RKF45, except that it uses the coefficients from Eq. (5); VRKF uses the coefficients from Eq. (5) and the variable order scheme described in Section 3.

While RKF45 is no longer the "state of the art" 5(4) Runge-Kutta formula, it is very widely used (it appears in several subroutine libraries), and we feel it is a suitable code to compare to ours.

Although we have been very careful to derive our code with quality in mind, as explained in Section 2, we were very surprised at how well RKFNC performed on the DETEST test set compared to RKF45. As can be seen from Table III, there is a gain in efficiency of about 25% in terms of time and function evaluations. This leads us to believe that, for the DETEST test set at least, the code based on RKFNC is a considerable improvement over RKF45. We see no reason why this conclusion should not extend in general to problems with smooth solutions.

We also see that VRKF is somewhat better than RKF45 in terms of both function evaluations and time. However, VRKF is only superior to RKFNC in terms of function evaluations at low accuracy. The time required to check the errors at the intermediate steps makes VRKF less efficient than RKFNC, even at low orders, if function evaluations are fast. We believe this conclusion will hold for other problems, making RKFNC the method of choice for smooth problems.

We now consider the performance of our code on some problems with non-smooth solutions. Our primary interest is in problems having solutions with sharp fronts but which are not discontinuous. However, in our numerical experiments, we consider problems with discontinuities that are triggered by a condition on x. Problems with discontinuities triggered by y, or one of its derivatives, are rather more specialized. We consider them to be beyond the scope of this paper. Such problems are of considerable interest, and we hope to extend our algorithm to deal with these problems at a future time.

The class of discontinuous problems we are interested in are those where the discontinuities appear without warning. An example is where the right-hand side is supplied by some black box code that hides the switching from the user. Such problems are much harder than those for which we know the x-value where f changes, or we are given a switching function which, on reaching a certain known value, triggers the discontinuity. It is important to use information regarding the switching function, if it is known, and several authors have proposed algorithms for dealing with this problem [2, 6, 9, 11, 12].

The case of interest to us, where the switching function is not known, has been considered by Gear and Osterby [8] and by Enright et al. [7]. The approach adopted by Gear and Osterby attempts to identify the nature of the singularity.

Table III. Summary of Results on the 25 (unscaled) DETEST Test Set

$\log_{10}$TOL	Time	Function calls	Number of steps	Maximum local error	Fraction deceived	Fraction bad deceived
Results for RKF45						
−2.00	0.498	4123	548	50.869	0.224	0.038
−3.00	0.565	4587	637	9.300	0.148	0.006
−4.00	0.745	6344	884	3.442	0.070	0.000
−5.00	1.014	8935	1270	1.439	0.012	0.000
−6.00	1.453	12948	1893	6.938	0.005	0.001
−7.00	2.117	18741	2886	2.144	0.002	0.000
−8.00	3.136	27552	4477	1.491	0.001	0.000
−9.00	4.849	42648	7023	1.270	0.000	0.000
Overall	14.377	125878	19618	50.869	0.017	0.001
Results for RKFNC						
−2.00	0.408	3272	437	7.762	0.156	0.005
−3.00	0.478	3965	555	10.875	0.138	0.004
−4.00	0.626	5432	752	6.355	0.092	0.001
−5.00	0.828	7508	1068	5.285	0.050	0.001
−6.00	1.145	10710	1555	3.471	0.027	0.000
−7.00	1.628	15349	2349	1.485	0.013	0.000
−8.00	2.370	22419	3609	1.480	0.004	0.000
−9.00	3.601	34086	5636	1.560	0.001	0.000
Overall	11.083	102741	15961	10.875	0.024	0.000
Results for VRKF						
−2.00	0.495	3171	465	6.495	0.099	0.009
−3.00	0.591	3919	580	10.875	0.064	0.003
−4.00	0.760	5327	782	3.283	0.051	0.000
−5.00	1.029	7488	1115	2.196	0.025	0.000
−6.00	1.407	10688	1604	2.760	0.017	0.000
−7.00	2.033	15645	2392	1.783	0.008	0.000
−8.00	2.953	22940	3674	1.628	0.004	0.000
−9.00	4.457	34524	5697	1.209	0.000	0.000
Overall	13.726	103702	16309	10.875	0.014	0.000

It is not clear how their approach will perform when the function is rapidly varying but does not actually contain a discontinuity (see [8, p. 41]).

All of the algorithms that we have mentioned for discontinuities and rapidly varying functions are at a rather preliminary stage of development. Few numerical results have appeared in the literature. For this reason we will not attempt to compare different methods for dealing with discontinuities. Instead we compare the performance of our code with that of a standard 5(4) Runge-Kutta code used with Fehlberg imbedding and local extrapolation. Our aim is to demonstrate the superior performance of a modified 5(4) code compared to that of a conventional code of the same type.

In what follows we consider four test problems. The first two have sharp fronts but no discontinuities, while the second two have discontinuities triggered on x. As explained earlier, we assume that the analytic form of f is unknown to us, so that we are not able to use information concerning the exact location of any discontinuous points of f or its derivatives in deriving our algorithm.

Problem 1.

$$\begin{aligned} y' &= z, & y(0) &= 10 \\ z' &= z^2 - \frac{3}{(A + y^2)}, & z(0) &= 0 \end{aligned} \qquad 0 \le x \le 50.$$

This problem[1] does not have a known, analytic solution. However, it was found to be a very good model for testing ODE solvers for the oil-reservoir problems one of us was studying. The solution is smooth almost everywhere except near $x = 35$, where it develops a sharp front. We believe it to be a very interesting and illuminating test problem for codes designed to deal with nonsmooth solutions. The values of the parameter A, which we considered for this problem, are 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}. Errors are computed at the end of the range only ($x = 50$), where the solution was found using the code with a very small ε. Thus, the "exact" solution, used to compute the errors, itself has an error of about 10^{-9}.

Problem 2.

$$\begin{aligned} y' &= z, & y(-1) &= -1, \\ z' &= \frac{-(1 + \pi^2 A)\cos \pi x - \pi x \sin \pi x - xz + y}{A}, & z(-1) &= 0.0017, \end{aligned} \qquad -1 \le x \le 1.$$

The true solution to this problem from Hemker [10, p. 138] is

$$y(x) = \cos \pi x + x + \frac{\sqrt{2A}\, \exp(-x^2/(2A))\exp(1/(2A)) + \exp(1/(2A))\sqrt{\pi}\, erf\,(\pi/\sqrt{2A})}{\sqrt{2A} + \exp(1/(2A))\sqrt{\pi}\, erf\,(1/\sqrt{2A})}.$$

This problem has a smooth solution $x + \cos \pi x$ coupled to rapidly varying solutions at $x = \pm 1$. For a description of the behavior of the complementary functions, the reader is referred to [10, p. 16]. In Table V we give results for this problem with $A = 0.1$.

Problem 3.

$$y' = \begin{cases} 0 & x < 0 \\ x^A & x \ge 0 \end{cases} \qquad -1 \le x \le 1, \quad y(-1) = 0.$$

This linear problem from [15, p. 15] has a discontinuity in the $(A + 1)$th derivative at the point $x = 0$. In Table VI we give the results obtained for this problem in the cases $A = 0, 1, 2, 3$.

Problem 4.

$$y' = \begin{cases} 55 - 1.5y & \text{if } [x] \text{ even} \\ 55 - 0.5y & \text{if } [x] \text{ odd} \end{cases} \qquad 0 \le x \le 20, \quad y(0) = 110.$$

[1] This problem was taken from the literature, but the reference can no longer be found. We apologize to the author.

Table IV. Results for Problem 1

A	10^{-1}		10^{-2}		10^{-3}		10^{-4}		10^{-5}	
TOL	Evals	Error	Evals	Error	Evals	Error	Evals	Error	Evals	Error
Results for RKF45										
10^{-3}	276	0.24E-2	344	0.17E-2	417	0.18E-2	486	0.22E-2	566	0.23E-2
10^{-4}	390	0.12E-3	481	0.13E-3	567	0.15E-3	686	0.15E-3	795	0.18E-3
10^{-5}	578	0.71E-5	698	0.14E-4	846	0.10E-4	1016	0.11E-4	1177	0.12E-4
10^{-6}	783	0.48E-6	981	0.73E-7	1198	0.12E-6	1442	0.23E-7	1704	0.17E-6
10^{-7}	991	0.20E-8	1257	0.34E-7	1547	0.33E-7	1841	0.27E-7	2144	0.35E-7
10^{-8}	1546	0.85E-8	1887	0.10E-7	2259	0.91E-8	2661	0.10E-7	3111	0.10E-7
10^{-9}	2398	0.76E-9	2910	0.10E-8	3508	0.82E-9	4145	0.82E-9	4887	0.89E-9
Results for RKFNC										
10^{-3}	213	0.21E-3	283	0.92E-4	346	0.15E-3	420	0.19E-3	480	0.17E-3
10^{-4}	293	0.30E-4	358	0.21E-4	454	0.19E-4	519	0.39E-5	632	0.14E-4
10^{-5}	411	0.11E-5	538	0.29E-6	663	0.59E-7	799	0.22E-6	920	0.25E-6
10^{-6}	586	0.28E-7	734	0.13E-6	920	0.15E-6	1118	0.16E-6	1317	0.23E-6
10^{-7}	801	0.24E-7	1027	0.44E-7	1272	0.56E-7	1576	0.75E-7	1867	0.70E-7
10^{-8}	1185	0.56E-8	1446	0.90E-8	1715	0.11E-7	2011	0.12E-7	2364	0.12E-7
10^{-9}	1771	0.11E-8	2110	0.16E-8	2549	0.97E-9	3053	0.17E-8	3598	0.12E-8
Results for VRKF										
10^{-3}	211	0.26E-3	243	0.29E-3	297	0.27E-3	346	0.26E-3	396	0.25E-3
10^{-4}	281	0.21E-4	342	0.15E-4	405	0.16E-4	474	0.24E-4	549	0.18E-4
10^{-5}	430	0.20E-6	504	0.73E-6	596	0.54E-6	707	0.54E-6	823	0.10E-5
10^{-6}	599	0.50E-7	724	0.16E-6	857	0.15E-6	1049	0.21E-6	1255	0.31E-6
10^{-7}	800	0.34E-7	1042	0.46E-7	1303	0.50E-7	1605	0.54E-7	1902	0.60E-7
10^{-8}	1196	0.55E-8	1478	0.88E-8	1748	0.11E-7	2043	0.12E-7	2391	0.12E-7
10^{-9}	1788	0.10E-8	2124	0.15E-8	2573	0.16E-8	3087	0.15E-8	3623	0.16E-8

Evals: The number of function evaluations.
Error: The computed error at the endpoint.

Table V. Results for Problem 2

	Number of function evaluations			Modulus of error at endpoint		
TOL	RKF45	RKFNC	VRKF	RKF45	RKFNC	VRKF
10^{-3}	94	87	89	0.26E-0	0.48E-1	0.48E-1
10^{-4}	131	107	109	0.31E-1	0.28E-2	0.28E-2
10^{-5}	203	156	163	0.37E-2	0.20E-3	0.20E-3
10^{-6}	316	251	252	0.12E-3	0.14E-4	0.14E-4
10^{-7}	484	384	391	0.84E-5	0.98E-6	0.25E-6
10^{-8}	770	610	617	0.11E-5	0.50E-7	0.63E-7
10^{-9}	1191	952	964	0.10E-6	0.16E-8	0.15E-8

This problem is F2 of the DETEST test set. For this problem, the function f has a discontinuity wherever x is an integer. The results obtained are given in Table VII.

Before discussing our numerical results in detail, we briefly discuss the sort of numerical results we might expect to obtain. We first note that Runge-Kutta methods are one-step in nature. We therefore expect them to have less trouble with singularities than would multistep methods such as Adams

Table VI. Results for Problem 3

A	0		1		2		3	
TOL	Evals	Error	Evals	Error	Evals	Error	Evals	Error
Results for RKF45								
10^{-3}	67	0.72E-2	23	0.33E-2	18	0.84E-2	18	0.35E-2
10^{-4}	121	0.11E-3	40	0.16E-3	18	0.12E-4	24	0.12E-3
10^{-5}	155	0.24E-3	62	0.88E-4	46	0.18E-4	30	0.17E-4
10^{-6}	190	0.33E-4	79	0.34E-5	46	0.69E-6	52	0.36E-5
10^{-7}	246	0.20E-5	128	0.79E-6	64	0.29E-5	58	0.28E-7
10^{-8}	278	0.69E-7	155	0.18E-6	107	0.82E-7	81	0.34E-8
10^{-9}	327	0.45E-8	225	0.19E-7	112	0.94E-8	85	0.10E-9
Results for RKFNC								
10^{-3}	67	0.22E-2	29	0.11E-2	23	0.36E-3	18	0.68E-3
10^{-4}	90	0.92E-3	51	0.17E-3	24	0.12E-3	24	0.67E-4
10^{-5}	161	0.20E-3	56	0.12E-3	41	0.38E-4	24	0.11E-7
10^{-6}	174	0.11E-4	96	0.28E-6	52	0.11E-5	46	0.43E-6
10^{-7}	240	0.41E-7	107	0.20E-6	79	0.19E-6	58	0.99E-7
10^{-8}	254	0.19E-6	130	0.13E-7	97	0.12E-7	52	0.13E-9
10^{-9}	330	0.49E-8	153	0.37E-8	107	0.97E-7	69	0.14E-9
Results for VRKF								
10^{-3}	47	0.77E-2	19	0.31E-2	19	0.35E-3	19	0.17E-3
10^{-4}	67	0.15E-2	37	0.51E-3	25	0.21E-3	25	0.15E-3
10^{-5}	98	0.91E-4	54	0.16E-3	1ᵹ	0.72E-5	19	0.88E-4
10^{-6}	116	0.24E-4	67	0.85E-7	49	0.19E-5	55	0.42E-5
10^{-7}	126	0.55E-6	66	0.69E-7	64	0.76E-7	40	0.75E-6
10^{-8}	148	0.40E-6	68	0.12E-6	63	0.11E-8	61	0.89E-9
10^{-9}	159	0.13E-7	98	0.61E-9	73	0.22E-9	64	0.19E-10

Evals: The number of function evaluations.
Error: The computed error at the endpoint.

Table VII. Results for Problem 4

	Number of function evaluations			Modulus of the error at the endpoint		
TOL	RKF45	RKFNC	VRKF	RKF45	RKFNC	VRKF
10^{-3}	1013	1046	936	0.14E-1	0.50E-2	0.13E-1
10^{-4}	1704	1606	1213	0.16E-2	0.33E-3	0.60E-3
10^{-5}	2087	1983	1530	0.11E-3	0.10E-3	0.15E-3
10^{-6}	2601	2443	1918	0.58E-4	0.70E-5	0.87E-5
10^{-7}	3027	3011	2198	0.21E-5	0.61E-6	0.15E-5
10^{-8}	4176	3822	2853	0.12E-6	0.65E-8	0.86E-8
10^{-9}	5125	4640	3425	0.51E-7	0.16E-8	0.17E-7

predictor-corrector formulas, for example. Indeed, if the "natural" step sequence chosen by the Runge-Kutta code nearly hits the singularity, then the unmodified code may experience very little difficulty in passing through the singular point. In view of this fact, we expect the variable order code to generally perform better than RKF45. Of course, there may be cases when solving discontinuous problems where RKF45 performs particularly well, due to a fortunate choice of step-size sequence.

The situation is quite different for problems where f is rapidly varying but not discontinuous. In such cases the rough behavior in the solution will occur over a range of x rather than at a particular point. The unmodified code will not be able to step through the rough part of the solution trajectory without noticing it. For such problems, we would expect the modified code based on VRKF to be superior to the unmodified RKF45, especially for relaxed tolerances when the codes normally try to use a large step-length.

We first consider the results obtained for Problem 1. This problem has a sharp front near $x = 35$, which steepens as A decreases. It is very noticeable from the results that, as the front gets steeper, the steplength of integration has to be cut back more and more, resulting in many rejected steps. This is an example of a problem where there is not just one point where bad (singular) behavior occurs. Instead, the difficult region extends over a range of x.

As mentioned earlier, for such problems we expect the variable order code VRKF to perform better than either RKF45 or RKFNC. This is borne out by the results given in Table IV, which show that VRKF is generally 25%–35% more efficient than RKF45. A large part of this gain is due to the use of the new coefficients, as can be seen from the results of RKFNC. At a modest tolerance, VRKF needs about 10% fewer function evaluations than RKFNC. When TOL is less than 10^{-6}, RKFNC is more efficient than VRKF. This effect occurs because, at such small tolerances, the step-size is small compared to the width of the front. In other words, the function appears smooth at the resolution of the step-size. However, the difference is small, and VRKF can be used without great loss of efficiency, especially if the function evaluations are expensive.

Results for Problem 2 are rather harder to predict, since the problem has boundary layers at both ends of the range of integration. In fact, this equation was used as an experiment to see how the codes would perform on such a problem. As can be seen from Table V, RKFNC is about 20% more efficient in terms of function evaluations than RKF45 and achieves much better accuracy. Indeed, if we compare function evaluations against accuracy, we see that RKFNC needs about half the function evaluations of RKF45.

For this problem, VRKF and RKFNC need almost the same number of function evaluations. As before, this phenomenon is due to the rough part being confined to a relatively small section of the domain. Clearly, RKF45 and RKFNC do not have any trouble handling the roughness near the endpoints. This problem shows that RKFNC should be used for such problems, but that not much is lost by using VRKF.

Problem 3 is an example of a discontinuous initial value problem where we have just a single trouble spot, i.e., at $x = 0$. We expect that the fixed order methods would experience the greatest trouble in the case $A = 0$ when f is discontinuous. As A increases, the singularity becomes less severe, and we expect RKF45 and RKFNC to perform more efficiently. This expectation is borne out by the results of Table VI. We see that in the case $A = 0$, VRKF is considerably superior to RKFNC; in some cases, needing half as many function evaluations. As A increases the difference becomes less apparent.

Differences in implementation can have a major effect on the performance of a code on this problem, as discussed in some detail in [15]. An important point brought out by this problem is that a code must have some restriction on its

allowable step-size change if it is to perform efficiently. Different step-size restrictions can lead to dramatically different performance. However, since the three codes we are comparing have exactly the same limits imposed on their step-size change, we feel that our comparison is a fair one.

Finally, we consider Problem 4. Here we expect a large difference in the performance of the fixed-order and variable order codes, since there are twenty singularities in the range of integration instead of just one. The chances of a fixed-order code picking a natural step sequence, suitable for passing through all these singular points, is very small, so we do not expect RKF45 or RKFNC to have abnormally good behavior. We see from Table VII that these expectations are borne out in practice. VRKF is considerably more efficient than either RKFNC or RKF45, with the gain in efficiency generally being about 20% over RKFNC and 30% over RKF45.

5. CONCLUSIONS

In this paper we have developed a modified Runge-Kutta code that contains imbedded formulas of all orders. This code is suitable for dealing with initial value problems where the function f is changing rapidly in some part of the region of integration. We were careful to derive a "high quality" formula that would perform well on problems with smooth solutions. The results given in Table III for the DETEST test set indicate that this goal has been achieved. At least for these smooth problems, our code is superior to RKF45.

Problems with rough solutions are handled by noting that our Runge-Kutta code computes f at several, reasonably uniformly spaced points in $[x_n, x_{n+1}]$, and trouble spots can be recognized early by looking for nonsmooth behavior in f. This strategy allows us to quit early when a high-order solution may not be acceptable or to accept a low-order solution when it is appropriate to do so.

Our belief is that this approach for dealing with nonsmooth solutions is a very powerful and general one. We are at present seeking to extend this idea to the case where the switching function is driven by a condition on y rather than on x. However, we feel that, in common with other approaches for dealing with discontinuous problems, our approach is in an embryonic stage. More understanding, both computational and theoretical, of this problem class is needed. Despite this, we believe that the results we have presented are sufficiently good to show that this approach shows great promise and deserves further attention.

ACKNOWLEDGMENT

The authors are grateful to L. F. Shampine for many useful suggestions concerning this paper.

REFERENCES

1. BETTIS, D. G. Efficient embedded Runge-Kutta methods. In *Numerical Treatment of Differential Equations: Proceedings Oberwolfach, 1976, Lecture Notes in Mathematics, 631*. R. Bulirsch, R. D. Grigorieff, and J. Shröder, Eds., Springer, Berlin, 1978, 9–18.
2. CARVER, M. B. Efficient integration over discontinuities in ordinary differential equation simulation. *Math. Comput. Simul. 20,* 3 (1978), 190–196.
3. CASH, J. R. A block 6(4) Runge-Kutta formula for nonstiff initial value problems. *ACM Trans. Math. Softw. 15*, 1 (1989), 15–28.

4. CELLIER, F. E. Stiff computation: where to go? In *Stiff Computation*, R. C. Aiken, Ed., OUP, 1985, 386–392.
5. DORMAND, J. R., AND PRINCE, P. J. A family of imbedded Runge-Kutta formulae. *J. Comput. Appl. Math. 6* (1980), 19–26.
6. ELLISON, D. Efficient automatic integration of ordinary differential equations with discontinuities. *Math. Comput. Simul. 23*, 1 (1981), 12–20.
7. ENRIGHT, W. H., JACKSON, K. R., NORSETT, S. P., AND THOMSEN, P. G. Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants. Numerical Analysis Rep. 113, Univ. of Manchester, Jan. 1986.
8. GEAR, C. W., AND ØSTERBY, O. Solving ordinary differential equations with discontinuities. *ACM Trans. Math. Softw. 10* (1984), 23–44.
9. HAY, J. L., CROSBIE, R. E., AND CHAPLIN, R. I. Integration routines for systems with discontinuities. *Comput. J. 17*, 3 (1974), 275–278.
10. HEMKER, P. W. A numerical study of stiff two-point boundary value problems. Mathematical Centre Tracts 80, Mathematisch Centrum, Amsterdam, 1977.
11. MANNSHARDT, R. One step methods of any order for ordinary differential equations with discontinuous right hand sides. *Numer. Math. 31*, 2 (1978), 131–152.
12. O'REGAN, P. G. Step size adjustment at discontinuities for fourth order Runge-Kutta methods. *Comput. J. 13*, 4 (1970), 401–404.
13. SHAMPINE, L. F. Some practical Runge-Kutta formulas. *Math. Comput. 46* (1986), 135.
14. SHAMPINE, L. F., AND GORDON, M. K. *Solution of Ordinary Differential Equations—The Initial Value Problem*. W. H. Freeman, San Francisco, Calif., 1975.
15. SHAMPINE, L. F., GORDON, M. K., AND WISNIEWSKI, J. A. Variable order Runge-Kutta codes. In *Computational Techniques for Ordinary Differential Equations Conference* (Univ. of Manchester, 1978), I Gladwell and D. K. Sayers, Eds. Academic Press, London, 1980, 83–101.
16. SHAMPINE, L. F., AND WATTS, H. A. Subroutine RKF45. In *Computer Methods for Mathematical Computations*. G. E. Forsythe, M. A. Malcolm, and C. B. Moler, Eds. Prentice-Hall, Englewood Cliffs, N.J., 1977, 135–147.
17. VERNER, J. H. Families of imbedded Runge-Kutta methods. *SIAM J. Numer. Anal. 16*, 5 (1979), 857–875.

Received November 1988; revised June 1989; accepted August 1989

APÊNDICE IV

LOW-ORDER CLASSICAL RUNGE-KUTTA FORMULAS WITH STEPSIZE CONTROL AND THEIR APPLICATION TO SOME HEAT TRANSFER PROBLEMS

Erwin Fehlberg

NASA Marshall Space Flight Center, Huntsville, Alabama, United States

NASA TR R-315

July 01, 1969

NASA TECHNICAL REPORT

NASA TR R-315

NASA TR R-315

LOW-ORDER CLASSICAL RUNGE-KUTTA FORMULAS WITH STEPSIZE CONTROL AND THEIR APPLICATION TO SOME HEAT TRANSFER PROBLEMS

by Erwin Fehlberg

George C. Marshall Space Flight Center
Marshall, Ala.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JULY 1969

NASA TR R-315

LOW-ORDER CLASSICAL RUNGE-KUTTA FORMULAS WITH STEPSIZE CONTROL AND THEIR APPLICATION TO SOME HEAT TRANSFER PROBLEMS

By Erwin Fehlberg

George C. Marshall Space Flight Center
Marshall, Ala.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

For sale by the Clearinghouse for Federal Scientific and Technical Information
Springfield, Virginia 22151 – CFSTI price $3.00

TABLE OF CONTENTS

LIST OF TABLES

LOW-ORDER CLASSICAL RUNGE-KUTTA FORMULAS WITH STEPSIZE CONTROL AND THEIR APPLICATION TO SOME HEAT TRANSFER PROBLEMS

INTRODUCTION

1. In an earlier report [1], the author derived fifth- to eighth-order RUNGE-KUTTA formulas with stepsize control. In this paper, similar first- to fourth-order formulas are developed.

2. Such low-order RUNGE-KUTTA formulas are of interest in some heat transfer problems. It is well-known that the parabolic partial differential equations of such problems can be reduced to ordinary differential equations. For instance, by a discretization of the space variable(s) of the problem, we obtain a system of ordinary differential equations with the time as the independent variable. Such a system can then be integrated by RUNGE-KUTTA methods.

3. However, it is also well-known that the application of RUNGE-KUTTA methods to such problems is often very time-consuming. Higher-order RUNGE-KUTTA formulas do not offer advantages in this respect, since stability considerations, resulting from the exponential character of the solution, exclude an increase of the integration stepsize that would make such high-order formulas meaningful. Therefore, low-order RUNGE-KUTTA formulas (second- or third-order) can be expected to solve such problems more efficiently than any high-order formula. On the other hand, they are potentially more efficient than the standard difference formulas obtained by discretization of the space variable(s) as well as the time variable.

4. For the efficiency of RUNGE-KUTTA formulas, it is essential that their truncation errors be as small as possible, since the permissable integration stepsize is strongly dependent upon the magnitude of these errors. Therefore, we have tried to establish RUNGE-KUTTA formulas with small truncation errors.

SECTION I. FOURTH-ORDER FORMULAS

5. We consider the (vector) differential equation

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

and write for our RUNGE-KUTTA formula ,

$$f_o = f(x_o, y_o)$$

$$f_\kappa = f\left(x_o + \alpha_\kappa h, \sum_{\lambda=0}^{\kappa-1} \beta_{\kappa\lambda} f_\lambda\right) \quad (\kappa = 1, 2, 3, 4, 5) \qquad (2)$$

and

$$y = y_o + h \sum_{\kappa=0}^{4} c_\kappa f_\kappa + 0(h^5)$$

$$\hat{y} = y_o + h \sum_{\kappa=0}^{5} \hat{c}_\kappa f_\kappa + 0(h^6) \qquad (3)$$

with h as integration stepsize and (x_o, y_o) as initial values. Equations (3) imply that we try to determine the coefficients α_κ, $\beta_{\kappa\lambda}$, c_κ, $\hat{c}_\kappa$ in such a way that the first formula (3) represents a fourth-order, and the second formula (3), a fifth-order RUNGE-KUTTA formula. The difference $y - \hat{y}$ then represents an approximation for the leading (fifth- order) truncation error term of our fourth-order RUNGE-KUTTA formula and can be used easily for establishing a reliable stepsize control procedure for this formula.

6. The coefficients α_κ, $\beta_{\kappa\lambda}$, c_κ, and $\hat{c}_\kappa$ have to satisfy certain equations of condition that can be obtained by TAYLOR expansions. These equations of condition are well-known in the literature, see for example J.C. BUTCHER's paper ([2], Table 1), or a paper by this author ([1], Table I). For the convenience of the reader, we list these equations here for a fifth-order formula like the second equation (3).

Introducing the abbreviations:

$$\beta_{\kappa 1}\alpha_1^\lambda + \beta_{\kappa 2}\alpha_2^\lambda \ldots\ldots + \beta_{\kappa\kappa-1}\alpha_{\kappa-1}^\lambda = P_{\kappa\lambda}$$

$$(\kappa = 2, 3, 4, 5; \ \lambda = 1, 2, 3) \qquad (4)$$

Table I contains the 17 equations of condition for our fifth-order formula.

TABLE I. EQUATIONS OF CONDITION FOR FIFTH-ORDER FORMULA

$$(\mathrm{I}, \hat{1}) \qquad \sum_{\kappa=1}^{5} \hat{c}_{\kappa} - 1 = 0$$

$$(\mathrm{II}, \hat{1}) \qquad \sum_{\kappa=1}^{5} \hat{c}_{\kappa} \alpha_{\kappa} - \frac{1}{2} = 0$$

$$(\mathrm{III}, \hat{1}) \qquad \sum_{\kappa=2}^{5} \hat{c}_{\kappa} P_{\kappa 1} - \frac{1}{6} = 0$$

$$(\mathrm{III}, \hat{2}) \qquad \frac{1}{2} \sum_{\kappa=1}^{5} \hat{c}_{\kappa} \alpha_{\kappa}^{2} - \frac{1}{6} = 0$$

$$(\mathrm{IV}, \hat{1}) \qquad \sum_{\kappa=3}^{5} \hat{c}_{\kappa} \left(\sum_{\lambda=2}^{\kappa-1} \beta_{\kappa\lambda} P_{\lambda 1} \right) - \frac{1}{24} = 0$$

$$(\mathrm{IV}, \hat{2}) \qquad \frac{1}{2} \sum_{\kappa=2}^{5} \hat{c}_{\kappa} P_{\kappa 2} - \frac{1}{24} = 0$$

$$(\mathrm{IV}, \hat{3}) \qquad \sum_{\kappa=2}^{5} \hat{c}_{\kappa} \alpha_{\kappa} P_{\kappa 1} - \frac{1}{8} = 0$$

$$(\mathrm{IV}, \hat{4}) \qquad \frac{1}{6} \sum_{\kappa=1}^{5} \hat{c}_{\kappa} \alpha_{\kappa}^{3} - \frac{1}{24} = 0$$

$$(\mathrm{V}, \hat{1}) \qquad \sum_{\kappa=4}^{5} \hat{c}_{\kappa} \left[\sum_{\lambda=3}^{\kappa-1} \beta_{\kappa\lambda} \left(\sum_{\mu=2}^{\lambda-1} \beta_{\lambda\mu} P_{\mu 1} \right) \right] - \frac{1}{120} = 0$$

$$(\mathrm{V}, \hat{2}) \qquad \frac{1}{2} \sum_{\kappa=3}^{5} \hat{c}_{\kappa} \left(\sum_{\lambda=2}^{\kappa-1} \beta_{\kappa\lambda} P_{\lambda 2} \right) - \frac{1}{120} = 0$$

TABLE I. (Concluded)

$$(V,\hat{3}) \qquad \sum_{\kappa=3}^{5} \hat{c}_\kappa \left(\sum_{\lambda=2}^{\kappa-1} \beta_{\kappa\lambda} \alpha_\lambda P_{\lambda 1} \right) - \frac{1}{40} = 0$$

$$(V,\hat{4}) \qquad \frac{1}{6} \sum_{\kappa=2}^{5} \hat{c}_\kappa P_{\kappa 3} - \frac{1}{120} = 0$$

$$(V,\hat{5}) \qquad \sum_{\kappa=3}^{5} \hat{c}_\kappa \alpha_\kappa \left(\sum_{\lambda=2}^{\kappa-1} \beta_{\kappa\lambda} P_{\lambda 1} \right) - \frac{1}{30} = 0$$

$$(V,\hat{6}) \qquad \frac{1}{2} \sum_{\kappa=2}^{5} \hat{c}_\kappa \alpha_\kappa P_{\kappa 2} - \frac{1}{30} = 0$$

$$(V,\hat{7}) \qquad \frac{1}{2} \sum_{\kappa=2}^{5} \hat{c}_\kappa P_{\kappa 1}^2 - \frac{1}{40} = 0$$

$$(V,\hat{8}) \qquad \frac{1}{2} \sum_{\kappa=2}^{5} \hat{c}_\kappa \alpha_\kappa^2 P_{\kappa 1} - \frac{1}{20} = 0$$

$$(V,\hat{9}) \qquad \frac{1}{24} \sum_{\kappa=1}^{5} \hat{c}_\kappa \alpha_\kappa^4 - \frac{1}{120} = 0$$

The Roman numerals in front of the equations in Table I indicate the order of the terms in the TAYLOR expansion.

A similar table holds for a fourth-order formula such as the first formula (3). We obtain this table from Table I by omitting the fifth-order equations $(V,\hat{1})$ through $(V,\hat{9})$ and replacing, in the remaining equations, $\hat{c}_\kappa$ by c_κ and the upper limit 5 of the k-sums by 4. These remaining eight equations might be denoted by (I, 1) through (IV, 4).

All eight equations of this new table for a fourth-order formula and the 17 equations of Table I for a fifth-order formula have to be satisfied simultaneously.

7. For the following we assume

$$c_1 = 0\,, \qquad \hat{c}_1 = 0\,, \qquad \alpha_4 = 1 \tag{5}$$

We further assume that α_2, α_3, α_5 are different from one another and from 0 and 1.

Equations (II, 1), (III, 2), (IV, 4) then yield

$$\left.\begin{aligned} c_2 &= \frac{1}{12}\,\frac{2\alpha_3 - 1}{\alpha_2(\alpha_3 - \alpha_2)(1 - \alpha_2)} \\ c_3 &= \frac{1}{12}\,\frac{2\alpha_2 - 1}{\alpha_3(\alpha_2 - \alpha_3)(1 - \alpha_3)} \\ c_4 &= \frac{1}{12}\,\frac{6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3}{(1 - \alpha_2)(1 - \alpha_3)} \end{aligned}\right\} \tag{6}$$

and equations (II, $\hat{1}$), (III, $\hat{2}$), (IV, $\hat{4}$), (V, $\hat{9}$)

$$\left.\begin{aligned} \hat{c}_2 &= \frac{1}{60}\,\frac{10\alpha_3\alpha_5 - 5(\alpha_3 + \alpha_5) + 3}{\alpha_2(\alpha_3 - \alpha_2)(1 - \alpha_2)(\alpha_5 - \alpha_2)} \\ \hat{c}_3 &= \frac{1}{60}\,\frac{10\alpha_2\alpha_5 - 5(\alpha_2 + \alpha_5) + 3}{\alpha_3(\alpha_2 - \alpha_3)(1 - \alpha_3)(\alpha_5 - \alpha_3)} \\ \hat{c}_4 &= \frac{1}{60}\,\frac{30\alpha_2\alpha_3\alpha_5 - 20(\alpha_2\alpha_3 + \alpha_2\alpha_5 + \alpha_3\alpha_5) + 15(\alpha_2 + \alpha_3 + \alpha_5) - 12}{(\alpha_2 - 1)(\alpha_3 - 1)(\alpha_5 - 1)} \\ \hat{c}_5 &= \frac{1}{60}\,\frac{10\alpha_2\alpha_3 - 5(\alpha_2 + \alpha_3) + 3}{\alpha_5(\alpha_2 - \alpha_5)(\alpha_3 - \alpha_5)(1 - \alpha_5)} \end{aligned}\right\} \tag{7}$$

8. Furthermore, we make the following assumptions that greatly reduce the number of equations of condition

$$(A1)\quad \begin{cases} P_{21} = \frac{1}{2}\alpha_2^2 \\ P_{31} = \frac{1}{2}\alpha_3^2 \\ P_{41} = \frac{1}{2} \\ P_{51} = \frac{1}{2}\alpha_5^2 \end{cases} \qquad (A2)\quad \begin{cases} P_{22} = \frac{1}{3}\alpha_2^3 \\ P_{32} = \frac{1}{3}\alpha_3^3 \\ P_{42} = \frac{1}{3} \\ P_{52} = \frac{1}{3}\alpha_5^3 \end{cases}$$

$$(B)\quad c_2\beta_{21} + c_3\beta_{31} + c_4\beta_{41} = 0$$

$$(\hat{B})\quad \hat{c}_2\beta_{21} + \hat{c}_3\beta_{31} + \hat{c}_4\beta_{41} + \hat{c}_5\beta_{51} = 0$$

$$(\hat{C})\quad \hat{c}_2\alpha_2\beta_{21} + \hat{c}_3\alpha_3\beta_{31} + \hat{c}_4\beta_{41} + \hat{c}_5\alpha_5\beta_{51} = 0$$

From (A1) and (A2) the following identities result:

$$\left.\begin{array}{l} (III,1) \equiv (III,2)\ ;\quad (IV,2) \equiv (IV,4)\ ;\quad (IV,3) \equiv 3(IV,4) \\ (III,\hat{1}) \equiv (III,\hat{2})\ ;\quad (IV,\hat{2}) \equiv (IV,\hat{4})\ ;\quad (IV,\hat{3}) \equiv 3(IV,4)\ ; \\ (V,\hat{6}) \equiv 4(V,\hat{9})\ ;\quad (V,\hat{7}) \equiv 3(V,\hat{9})\ ;\quad (V,\hat{8}) \equiv 6(V,\hat{9}) \end{array}\right\} \quad (8)$$

By also using (B) and $(\hat{B})$ we find the following identities:

$$\begin{array}{l} (IV,1) \equiv (IV,2) \\ (IV,\hat{1}) \equiv (IV,\hat{2})\ ;\quad (V,\hat{2}) \equiv (V,\hat{4})\ ;\quad (V,\hat{3}) \equiv 3(V,\hat{4}) \end{array} \quad (9)$$

and finally by also taking into account assumption $(\hat{C})$

$$(V,\hat{5}) \equiv (V,\hat{6}) \quad (10)$$

Therefore, equations $(I,1)$; $(I,\hat{1})$; $(V,\hat{1})$; $(V,\hat{4})$ are the only remaining equations of condition.

The first two equations determine the coefficients c_o and $\hat{c}_o$ that otherwise do not enter our equations of condition.

The remaining equations (V, $\hat{1}$) and (V, $\hat{4}$) have to be solved together with our assumptions (A1), (A2), (B), ($\hat{B}$), and ($\hat{C}$).

9. From the first equations (A1) and (A2), the following relations are obtained:

$$\alpha_1 = \frac{2}{3}\alpha_2 \tag{11}$$

$$\beta_{21} = \frac{3}{4}\alpha_2 \tag{12}$$

The second equations (A1) and (A2) yield

$$\left.\begin{aligned} \beta_{31} &= \frac{3}{4}\left(\frac{\alpha_3}{\alpha_2}\right)^2 (3\alpha_2 - 2\alpha_3) \\ \beta_{32} &= \left(\frac{\alpha_3}{\alpha_2}\right)^2 (\alpha_3 - \alpha_2) \end{aligned}\right\} \tag{13}$$

The third equations (A1) and (A2), together with (B), determine the coefficients β_{41}, β_{42}, β_{43}. We find the following expressions for these coefficients:

$$\left.\begin{aligned} \beta_{41} &= \frac{3}{4} \cdot \frac{1}{\alpha_2^2} \cdot \frac{6\alpha_2^2\alpha_3 - 6\alpha_2\alpha_3 + 2\alpha_3 - \alpha_2}{6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3} \\ \beta_{42} &= -\frac{1}{\alpha_2^2} \cdot \frac{1 - \alpha_2}{\alpha_3 - \alpha_2} \cdot \frac{2\alpha_2^2\alpha_3 - 4\alpha_2\alpha_3 + \alpha_3^2 + \alpha_2}{6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3} \\ \beta_{43} &= -\frac{1}{\alpha_3} \cdot \frac{(1 - \alpha_2)(1 - \alpha_3)}{\alpha_3 - \alpha_2} \cdot \frac{2\alpha_2 - 1}{6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3} \end{aligned}\right\} \tag{14}$$

Elimination of $\hat{c}_5$ from ($\hat{B}$) and ($\hat{C}$) leads to the following relation:

$$\hat{c}_2(\alpha_5 - \alpha_2)\beta_{21} + \hat{c}_3(\alpha_5 - \alpha_3)\beta_{31} + \hat{c}_4(\alpha_5 - 1)\beta_{41} = 0 \tag{15}$$

Introducing the above computed values for $\hat{c}_2$, $\hat{c}_3$, $\hat{c}_4$, β_{21}, β_{31}, and β_{41} into (15) leads to a relation between α_2, α_3, and α_5.

$$M\alpha_5 = N \tag{16}$$

with

$$\left.\begin{aligned} M &= (6\alpha_2\alpha_3 - 4\alpha_2 - 4\alpha_3 + 3)(30\alpha_2\alpha_3^2 - 30\alpha_2^2\alpha_3^2 - 10\alpha_3^2 + 5\alpha_2\alpha_3) \\ &\quad + (6\alpha_2^2\alpha_3 - 6\alpha_2\alpha_3 + 2\alpha_3 - \alpha_2)(30\alpha_2\alpha_3^2 - 20\alpha_2\alpha_3 - 20\alpha_3^2 + 15\alpha_3) \\ N &= (6\alpha_2\alpha_3 - 4\alpha_2 - 4\alpha_3 + 3)(16\alpha_2\alpha_3^2 - 15\alpha_2^2\alpha_3^2 - 6\alpha_3^2 + 3\alpha_2\alpha_3) \\ &\quad + (6\alpha_2^2\alpha_3 - 6\alpha_2\alpha_3 + 2\alpha_3 - \alpha_2)(20\alpha_2\alpha_3^2 - 15\alpha_2\alpha_3 - 15\alpha_3^2 + 12\alpha_3) \end{aligned}\right\} \tag{17}$$

It is easily verified that

$$M = 0 \tag{18}$$

for any value of α_2 and α_3 .

Because of (16), only such values as α_2 and α_3 are possible that also lead to

$$N = 0 \tag{19}$$

Equation (19) represents a restrictive relation between α_2 and α_3. This relation can be reduced to

$$\alpha_3 = \frac{1}{2} \cdot \frac{\alpha_2}{5\alpha^2 - 4\alpha_2 + 1} \tag{20}$$

10. We use relation (20) to eliminate α_3 from the expressions for our RUNGE-KUTTA coefficients as obtained in Nos. 7, 8, and 9. The elimination results in

$$\left.\begin{aligned} \alpha_1 &= \frac{2}{3}\alpha_2 \\ \alpha_3 &= \frac{1}{2} \cdot \frac{\alpha_2}{5\alpha_2^2 - 4\alpha_2 + 1} \\ c_2 &= \frac{1}{6} \cdot \frac{1}{\alpha_2^2(1 - \alpha_2)} \cdot \frac{5\alpha_2^2 - 5\alpha_2 + 1}{10\alpha_2^2 - 8\alpha_2 + 1} \end{aligned}\right\} \tag{21}$$

$$
\left.
\begin{aligned}
c_3 &= \frac{2}{3} \cdot \frac{1}{\alpha_2^2(5\alpha_2 - 2)} \cdot \frac{(5\alpha_2^2 - 4\alpha_2 + 1)^3}{10\alpha_2^2 - 8\alpha_2 + 1} \\
c_4 &= \frac{1}{6} \cdot \frac{10\alpha_2^2 - 12\alpha_2 + 3}{(1 - \alpha_2)(5\alpha_2 - 2)} \\
\beta_{21} &= \frac{3}{4}\alpha_2 \\
\beta_{31} &= \frac{3}{16}\alpha_2 \cdot \frac{15\alpha_2^2 - 12\alpha_2 + 2}{(5\alpha_2^2 - 4\alpha_2 + 1)^3} \\
\beta_{32} &= -\frac{1}{8}\alpha_2 \cdot \frac{10\alpha_2^2 - 8\alpha_2 + 1}{(5\alpha_2^2 - 4\alpha_2 + 1)^3} \\
\beta_{41} &= \frac{3}{4} \cdot \frac{1}{10\alpha_2^2 - 12\alpha_2 + 3} \\
\beta_{42} &= -\frac{1}{2} \cdot \frac{1 - \alpha_2}{\alpha_2^2} \cdot \frac{60\alpha_2^3 - 78\alpha_2^2 + 31\alpha_2 - 4}{(10\alpha_2^2 - 12\alpha_2 + 3)(10\alpha_2^2 - 8\alpha_2 + 1)} \\
\beta_{43} &= -2 \cdot \frac{(1 - \alpha_2)(5\alpha_2 - 2)(2\alpha_2 - 1)(5\alpha_2^2 - 4\alpha_2 + 1)^2}{\alpha_2^2(10\alpha_2^2 - 12\alpha_2 + 3)(10\alpha_2^2 - 8\alpha_2 + 1)}
\end{aligned}
\right\} \quad (21)\ \text{(continued)}
$$

11. The weight factors $\hat{c}_2$ through $\hat{c}_5$ of the fifth-order formula can now be expressed by α_2 and α_5:

$$
\left.
\begin{aligned}
\hat{c}_2 &= \frac{1}{60} \cdot \frac{10\alpha_5(5\alpha_2^2 - 5\alpha_2 + 1) - (30\alpha_2^2 - 29\alpha_2 + 6)}{\alpha_2^2(10\alpha_2^2 - 8\alpha_2 + 1)(1 - \alpha_2)(\alpha_5 - \alpha_2)} \\
\hat{c}_3 &= \frac{4}{15} \cdot \frac{(5\alpha_2^2 - 4\alpha_2 + 1)^4 [5\alpha_5(2\alpha_2 - 1) - (5\alpha_2 - 3)]}{\alpha_2^2(10\alpha_2^2 - 8\alpha_2 + 1)(5\alpha_2^2 - 2)(2\alpha_2 - 1)[2\alpha_5(5\alpha_2^2 - 4\alpha_2 + 1) - \alpha_2]} \\
\hat{c}_4 &= \frac{1}{60} \cdot \frac{10(2\alpha_2 - 1)(10\alpha_2^2 - 12\alpha_2 + 3)\alpha_5 - (150\alpha_2^3 - 260\alpha_2^2 + 141\alpha_2 - 24)}{(1 - \alpha_2)(5\alpha_2 - 2)(2\alpha_2 - 1)(1 - \alpha_5)} \\
\hat{c}_5 &= \frac{1}{60} \cdot \frac{(5\alpha_2 - 2)(10\alpha_2^2 - 12\alpha_2 + 3)}{\alpha_5(1 - \alpha_5)(\alpha_2 - \alpha_5)[2(5\alpha_2^2 - 4\alpha_2 + 1)\alpha_5 - \alpha_2]}
\end{aligned}
\right\} \quad (22)
$$

12. We still have to determine the coefficients β_{51}, β_{52}, β_{53}, β_{54} of our fifth-order formula. From equation $(\hat{B})$ we find β_{51}. The fourth equations (A1) and (A2) together with equation $(V, \hat{4})$, determine the coefficients β_{52}, β_{53}, β_{54}. It can be shown that equation $(V, \hat{1})$ is then also satisfied for any value of α_2 and α_5.

This concludes the computation of our RUNGE-KUTTA coefficients, since c_o and $\hat{c}_o$ can be determined from (I, 1) or $(I, \hat{1})$, respectively, and the coefficients $\beta_{\kappa o}$ $(\kappa = 1, 2, 3, 4, 5)$ can be obtained from the standard equations

$$\sum_{\lambda=0}^{\kappa-1} \beta_{\kappa\lambda} = \alpha_\kappa \qquad (\kappa = 1, 2, 3, 4, 5) \tag{23}$$

13. Our RUNGE-KUTTA coefficients contain two arbitrary parameters, α_2 and α_5. We shall show that we can reduce the truncation error of our fourth-order formula by a proper choice of the parameter α_2. From Table I, it follows that the leading term (the fifth-order term) of the truncation error consists of nine sub-terms. These sub-terms are certain expressions built up by the partial derivatives of the right-hand sides of the differential equation (1). These sub-terms are multiplied by certain numerical factors T_1 through T_9. For these factors, we find from equations $V, \hat{1})$ through $(V, \hat{9})$ of Table I, replacing $\hat{c}_\kappa$ by c_κ and the upper limit 5 of the κ-sums by 4,

$$\left.\begin{aligned} T_1 &= c_4\beta_{43}\beta_{32}P_{21} - \frac{1}{120} \\ T_2 &= \frac{1}{2}\sum_{\kappa=3}^{4} c_\kappa\left(\sum_{\lambda=2}^{\kappa-1}\beta_{\kappa\lambda}P_{\lambda 2}\right) - \frac{1}{120} \\ &\;\;\vdots \\ T_9 &= \frac{1}{24}\sum_{\kappa=1}^{4} c_\kappa\alpha_\kappa^4 - \frac{1}{120} \end{aligned}\right\} \tag{24}$$

Naturally, it is desirable to find RUNGE-KUTTA formulas with small numerical factors T_1 through T_9 to make the leading term of the local truncation error small.

14. Since we can express all RUNGE-KUTTA coefficients that enter the right-hand sides of (24) by α_2 alone, the factors $T_1, \ldots, T_9$ can finally be written as functions of α_2. The computation results in the following values for these factors:

$$\left.\begin{aligned} T_1 = -T_5 &= -\frac{1}{240}\,\frac{(5\alpha_2 - 2)\,(4\alpha_2 - 1)}{5\alpha_2^2 - 4\alpha_2 + 1} \\ T_2 = \frac{1}{3}T_3 = T_4 = -T_6 = -\frac{4}{3}T_7 = -\frac{2}{3}T_8 &= 4T_9 \\ = \frac{1}{720}\,\frac{(5\alpha_2 - 2)\,(10\alpha_2^2 - 12\alpha_2 + 3)}{5\alpha_2^2 - 4\alpha_2 + 1} & \end{aligned}\right\} \quad (25)$$

15. We see from (25) that all factors $T_1, \ldots, T_9$ would become zero for $\alpha_2 = \frac{2}{5}$. Because of (20), this value α_2 leads to $\alpha_3 = 1$. It can, however, be shown easily that $\alpha_3 = \alpha_4 = 1$ leads to contradictions in the equations (II, 1), (III, 2), (IV, 4) and (II, $\hat{1}$), (III, $\hat{2}$), (IV, $\hat{4}$), (V, $\hat{9}$). Therefore, we have to exclude the value $\alpha_2 = \frac{2}{5}$.

Another interesting choice of α_2 would result from

$$10\alpha_2^2 - 12\alpha_2 + 3 = 0 \quad (26)$$

If (26) would hold, all error factors in the second group of (25) would become zero.

Because of (26), it would follow from (21),

$$c_4 = 0 \quad (27)$$

It can be shown easily that for the values α_2 resulting from (26) and for $c_4 = 0$, equation (B) cannot be satisfied.

Therefore, we also have to discard the values $\alpha_2 = \frac{1}{10}(6 \pm \sqrt{6}) \approx \begin{cases} 0.845 \\ 0.355 \end{cases}$ resulting from (26).

However, by choosing for α_2 a value close to one of the above values, say close to 0.355, we can expect that at least the error factors in the second group of (25) will become small.

We shall consider two choices for α_2 that are reasonably close to 0.355 and lead to relatively simple RUNGE-KUTTA coefficients, namely $\alpha_2 = \frac{1}{3}$ and $\alpha_2 = \frac{3}{8}$. The choice of α_5 in our formulas remains arbitrary.

16. Choosing $\alpha_2 = \frac{1}{3}$ leads to the RUNGE-KUTTA coefficients of Table II.

TABLE II. COEFFICIENTS FOR RK4(5), FORMULA 1

κ \ λ	α_κ	$\beta_{\kappa\lambda}$					c_κ	$\hat{c}_\kappa$
		0	1	2	3	4		
0	0	0					$\frac{1}{9}$	$\frac{47}{450}$
1	$\frac{2}{9}$	$\frac{2}{9}$					0	0
2	$\frac{1}{3}$	$\frac{1}{12}$	$\frac{1}{4}$				$\frac{9}{20}$	$\frac{12}{25}$
3	$\frac{3}{4}$	$\frac{69}{128}$	$-\frac{243}{128}$	$\frac{135}{64}$			$\frac{16}{45}$	$\frac{32}{225}$
4	1	$-\frac{17}{12}$	$\frac{27}{4}$	$-\frac{27}{5}$	$\frac{16}{15}$		$\frac{1}{12}$	$\frac{1}{30}$
5	$\frac{5}{6}$	$\frac{65}{432}$	$-\frac{5}{16}$	$\frac{13}{16}$	$\frac{4}{27}$	$\frac{5}{144}$		$\frac{6}{25}$

Subtracting the last two columns of Table II from one another, one finds as approximation for the leading term of the local truncation error of our fourth-order formula

$$TE = \left(\frac{1}{150} f_o - \frac{3}{100} f_2 + \frac{16}{75} f_3 + \frac{1}{20} f_4 - \frac{6}{25} f_5 \right) h \qquad (28)$$

We also list the error factors T_1 through T_9 of our formula 1:

$$\left.\begin{aligned} T_1 &= \frac{1}{480}, \quad T_2 = -\frac{1}{4320}, \quad T_3 = -\frac{1}{1440}, \quad T_4 = -\frac{1}{4320}, \\ T_5 &= -\frac{1}{480}, \quad T_6 = \frac{1}{4320}, \quad T_7 = \frac{1}{5760}, \quad T_8 = \frac{1}{2880}, \\ T_9 &= \frac{1}{17280} \end{aligned}\right\} \qquad (29)$$

17. For our second choice, $\alpha_2 = \frac{3}{8}$, we find the RUNGE-KUTTA coefficients of Table III.

TABLE III. COEFFICIENTS FOR RK4(5), FORMULA 2

κ \ λ	α_κ	$\beta_{\kappa\lambda}$					c_κ	$\hat{c}_\kappa$
		0	1	2	3	4		
0	0	0					$\frac{25}{216}$	$\frac{16}{135}$
1	$\frac{1}{4}$	$\frac{1}{4}$					0	0
2	$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				$\frac{1408}{2565}$	$\frac{6656}{12825}$
3	$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			$\frac{2197}{4104}$	$\frac{28561}{56430}$
4	1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		$-\frac{1}{5}$	$-\frac{9}{50}$
5	$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$		$\frac{2}{55}$

For the leading term of the local truncation error we obtain from Table III the approximation

$$TE = \left(-\frac{1}{360} f_o + \frac{128}{4275} f_2 + \frac{2187}{75240} f_3 - \frac{1}{50} f_4 - \frac{2}{55} f_5 \right) h \qquad (30)$$

This formula 2 has the following error factors:

$$\left.\begin{aligned} T_1 &= \frac{1}{780}, \quad T_2 = \frac{1}{12480}, \quad T_3 = \frac{1}{4160}, \quad T_4 = \frac{1}{12480}, \\ T_5 &= -\frac{1}{780}, \quad T_6 = -\frac{1}{12480}, \quad T_7 = -\frac{1}{16640}, \\ T_8 &= -\frac{1}{8320}, \quad T_9 = -\frac{1}{49920} \end{aligned}\right\} \qquad (31)$$

We notice that the error factors (31) of our second formula are somewhat smaller than the corresponding terms (29) of our first formula.

18. We should like to mention that another RK4(5)-formula was derived by D. SARAFYAN ([3], p. 4). His fourth-order formula is based upon only four (instead of five) evaluations of the differential equations. Therefore, his fourth-order formula has larger error terms than our formulas.

Since SARAFYAN's formula is published in an internal Technical Report and therefore is not easily accessible, we present SARAFYAN's formula as Table IV.

From Table IV it follows for the leading term of the local truncation error

$$TE = \left(\frac{1}{8} f_o + \frac{2}{3} f_2 + \frac{1}{16} f_3 - \frac{27}{56} f_4 - \frac{125}{336} f_5 \right) h \qquad (32)$$

The error factors T_1 through T_9 for SARAFYAN's formula read as follows:

$$\left.\begin{aligned} T_1 &= -\frac{1}{120}, \quad T_2 = \frac{1}{480}, \quad T_3 = -\frac{1}{240}, \quad T_4 = -\frac{1}{720}, \\ T_5 &= \frac{1}{120}, \quad T_6 = -\frac{1}{480}, \quad T_7 = \frac{1}{960}, \quad T_8 = \frac{1}{480}, \\ T_9 &= \frac{1}{2880} \end{aligned}\right\} \qquad (33)$$

TABLE IV. COEFFICIENTS FOR SARAFYAN'S RK4(5)-FORMULA

κ \ λ	α_κ	$\beta_{\kappa\lambda}$					c_κ	$\hat{c}_\kappa$
		0	1	2	3	4		
0	0	0					$\frac{1}{6}$	$\frac{1}{24}$
1	$\frac{1}{2}$	$\frac{1}{2}$					0	0
2	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$				$\frac{2}{3}$	0
3	1	0	-1	2			$\frac{1}{6}$	$\frac{5}{48}$
4	$\frac{2}{3}$	$\frac{7}{27}$	$\frac{10}{27}$	0	$\frac{1}{27}$			$\frac{27}{56}$
5	$\frac{1}{5}$	$\frac{28}{625}$	$-\frac{1}{5}$	$\frac{546}{625}$	$\frac{54}{625}$	$-\frac{378}{625}$		$\frac{125}{336}$

If we compare (33) with (31), we notice that in (31) the factors T_1 and T_5 are only $\frac{2}{13}$ of the corresponding factors of (33); the other factors of (31) are even smaller compared with the corresponding factors of (33).

For our RK4(5)-formula number 1, the error factors are $\frac{1}{4}$ (or better) of the corresponding error factors of SARAFYAN.

Because of the smaller error factors we may expect that our RK4(5) formulas 1 and 2 operate somewhat more economically than SARAFYAN's formula. Numerical experiments on an electronic computer have confirmed these expectations.

19. It is interesting to compare the error factors of SARAFYAN's formula with those of KUTTA's ([4], p. 443) standard fourth-order formula of Table V.

TABLE V. COEFFICIENTS FOR KUTTA'S RK4-FORMULA

κ \ λ	α_κ	$\beta_{\kappa\lambda}$			c_κ
		0	1	2	
0	0	0			$\frac{1}{6}$
1	$\frac{1}{2}$	$\frac{1}{2}$			$\frac{1}{3}$
2	$\frac{1}{2}$	0	$\frac{1}{2}$		$\frac{1}{3}$
3	1	0	0	1	$\frac{1}{6}$

The computation shows that KUTTA's error factors are identical with the error factors (33) of SARAFYAN, except for T_7, which in the case of KUTTA's formulas has to be replaced by $\frac{1}{160}$.

Therefore, SARAFYAN's formula, in general, will not reduce substantially the number of integration steps required by KUTTA's formula. However, it will reduce the computer time, since it requires only six evaluations per step compared to seven in KUTTA's formula, if the latter one is applied with the standard stepsize control procedure (recomputation of two steps as one step with double stepsize).

20. We might mention that we have presented general RUNGE-KUTTA transformation formulas with stepsize control in two earlier papers [5], [6]. In the special case of a fourth-order formula these general formulas do not require any differentiation and can also be written in the form of classical RUNGE-KUTTA formulas.

SECTION II. THIRD-ORDER FORMULAS

21. Since a fourth-order RUNGE-KUTTA formula requires four evaluations (per step) of the differential equations, one might expect a pair of RUNGE-KUTTA formulas RK3(4) to require four evaluations also.

However, it can be shown easily that four evaluations are not sufficient to define the pair RK3(4). The equations of condition lead to contradictions in the case of four evaluations. We therefore allow for five evaluations per step.

However, it is possible to choose the fifth evaluation in such a way that this evaluation can be taken over as the first evaluation for the next step. Thereby the number of evaluations per step again will be reduced to four, except for the very first step, when the integration is started.

Since the derivation of the RK3(4)-formulas is very similar to the derivation of the RK4(5)-formulas of Section I, we may omit some details and present the main results only.

22. The equations of condition, as they hold for a fourth-order formula, are listed in Table VI.

TABLE VI. EQUATIONS OF CONDITION FOR FOURTH-ORDER FORMULA

$$(\mathrm{I},\hat{1}) \qquad \sum_{\kappa=0}^{4} \hat{c}_\kappa - 1 = 0$$

$$(\mathrm{II},\hat{1}) \qquad \sum_{\kappa=1}^{4} \hat{c}_\kappa \alpha_\kappa - \frac{1}{2} = 0$$

$$(\mathrm{III},\hat{1}) \qquad \sum_{\kappa=2}^{4} \hat{c}_\kappa P_{\kappa 1} - \frac{1}{6} = 0$$

$$(\mathrm{III},\hat{2}) \qquad \frac{1}{2} \sum_{\kappa=1}^{4} \hat{c}_\kappa \alpha_\kappa^2 - \frac{1}{6} = 0$$

$$(\mathrm{IV},\hat{1}) \qquad \sum_{\kappa=3}^{4} \hat{c}_\kappa \left(\sum_{\lambda=2}^{\kappa-1} \beta_{\kappa\lambda} P_{\lambda 1} \right) - \frac{1}{24} = 0$$

$$(\mathrm{IV},\hat{2}) \qquad \frac{1}{2} \sum_{\kappa=2}^{4} \hat{c}_\kappa P_{\kappa 2} - \frac{1}{24} = 0$$

TABLE VI. (Concluded)

$$(\mathrm{IV},\hat{3}) \qquad \sum_{\kappa=2}^{4} \hat{c}_\kappa \alpha_\kappa P_{\kappa 1} - \frac{1}{8} = 0$$

$$(\mathrm{IV},\hat{4}) \qquad \frac{1}{6} \cdot \sum_{\kappa=1}^{4} \hat{c}_\kappa \alpha_\kappa^3 - \frac{1}{24} = 0$$

A similar table for a third-order formula is obtained from Table VI by omitting the fourth-order equations of condition and replacing, in the remaining equations, $\hat{c}_\kappa$ by c_κ and the upper limit 4 of the κ-sums by 3.

We denote the remaining four equations for the third-order RUNGE-KUTTA formula by (I, 1), (II, 1), (III, 1), and (III, 2).

23. Again assuming that the conditions (5) hold and that α_2, α_3 are different from one another and from 0 and 1, we find from (II, 1) and (III, 2):

$$\left.\begin{aligned} c_2 &= \frac{1}{6} \cdot \frac{3\alpha_3 - 2}{\alpha_2(\alpha_3 - \alpha_2)} \\ c_3 &= \frac{1}{6} \cdot \frac{3\alpha_2 - 2}{\alpha_3(\alpha_2 - \alpha_3)} \end{aligned}\right\} \tag{34}$$

and from $(\mathrm{II},\hat{1})$, $(\mathrm{III},\hat{2})$, $(\mathrm{IV},\hat{4})$:

$$\left.\begin{aligned} \hat{c}_2 &= \frac{1}{12} \cdot \frac{2\alpha_3 - 1}{\alpha_2(\alpha_3 - \alpha_2)(1 - \alpha_2)} \\ \hat{c}_3 &= \frac{1}{12} \cdot \frac{2\alpha_2 - 1}{\alpha_3(\alpha_2 - \alpha_3)(1 - \alpha_3)} \\ \hat{c}_4 &= \frac{1}{12} \cdot \frac{6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3}{(1 - \alpha_2)(1 - \alpha_3)} \end{aligned}\right\} \tag{35}$$

24. To bring the remaining equations of condition into a form that can be handled more easily, we make, similar to Section I, the further assumptions

$$
(A)\quad \begin{cases} P_{21} = \frac{1}{2}\alpha_2^2 \\ P_{31} = \frac{1}{2}\alpha_3^2 \\ P_{41} = \frac{1}{2} \end{cases}
$$

$$
(\hat{B})\qquad \hat{c}_2\beta_{21} + \hat{c}_3\beta_{31} + \hat{c}_4\beta_{41} = 0
$$

$$
(\hat{D})\qquad \beta_{40} = c_0,\quad \beta_{41} = c_1 = 0,\quad \beta_{42} = c_2,\quad \beta_{43} = c_3
$$

The assumption $(\hat{D})$ is required if the fifth evaluation is to be taken over as the first evaluation for the next step.

From the assumptions (A), $(\hat{B})$, $(\hat{D})$, it follows immediately that the remaining equations of condition reduce to

$$
\left.\begin{array}{l}
(IV,\hat{2})\quad \hat{c}_3\beta_{32}\alpha_2^2 + \hat{c}_4 \cdot \frac{1}{3} = \frac{1}{12} \\[2ex]
(A)\quad \begin{cases} \beta_{21}\alpha_1 = \frac{1}{2}\alpha_2^2 \\ \beta_{31}\alpha_1 + \beta_{32}\alpha_2 = \frac{1}{2}\alpha_3^2 \end{cases} \\[2ex]
(\hat{B})\quad \hat{c}_2\beta_{21} + \hat{c}_3\beta_{31} = 0
\end{array}\right\} \qquad (36)
$$

25. The first equation (A) expresses $\beta_{21}\alpha_1$ by α_2. From equation $(IV,\hat{2})$ we obtain β_{32} as a function of α_2 and α_3, since $\hat{c}_3$ and $\hat{c}_4$ are given as functions of α_2 and α_3 by (35).

The second equation (A) can then serve to find $\beta_{31}\alpha_1$ as function of α_2 and α_3.

The equation $(\hat{B})$, finally, represents a restrictive condition that must hold between α_2 and α_3 to make the equations of condition compatible. The computation results in the following restrictive condition:

$$\alpha_3 = \frac{1}{2} \cdot \frac{\alpha_2}{3\alpha_2^2 - 3\alpha_2 + 1} \tag{37}$$

Eliminating α_3 from the coefficients of our third-order formula, these coefficients become functions of α_2 only. We find the following expressions for them:

$$\left.\begin{aligned} c_2 &= \frac{1}{6} \cdot \frac{1}{\alpha_2^2} \cdot \frac{12\alpha_2^2 - 15\alpha_2 + 4}{6\alpha_2^2 - 6\alpha_2 + 1} \\ c_3 &= \frac{2}{3} \cdot \frac{1}{\alpha_2^2} \frac{(3\alpha_2 - 2)(3\alpha_2^2 - 3\alpha_2 + 1)^2}{6\alpha_2^2 - 6\alpha_2 + 1} \\ \beta_{21}\alpha_1 &= \frac{1}{2}\alpha_2^2 \\ \beta_{31}\alpha_1 &= \frac{1}{8}\alpha_2^2 \frac{(3\alpha_2 - 1)(3\alpha_2 - 2)}{(3\alpha_2^2 - 3\alpha_2 + 1)^3} \\ \beta_{32} &= -\frac{1}{8}\alpha_2 \frac{6\alpha_2^2 - 6\alpha_2 + 1}{(3\alpha_2^2 - 3\alpha_2 + 1)^3} \end{aligned}\right\} \tag{38}$$

26. We now consider the error factors for our third-order formula and try to make these error factors small by a proper choice of α_2 .

From Table V we find the following four error factors:

$$\left.\begin{aligned} T_1 &= c_3\beta_{32}P_{21} - \frac{1}{24} \\ T_2 &= \frac{1}{2}(c_2P_{22} + c_3P_{32}) - \frac{1}{24} \\ T_3 &= c_2\alpha_2P_{21} + c_3\alpha_3P_{31} - \frac{1}{8} \\ T_4 &= \frac{1}{6}(c_2\alpha_2^3 + c_3\alpha_3^3) - \frac{1}{24} \end{aligned}\right\} \tag{39}$$

or, if we insert (38) into (39):

$$\left.\begin{aligned} T_1 &= -\frac{1}{24}\,\frac{(2\alpha_2-1)(3\alpha_2-1)}{3\alpha_2^2-3\alpha_2+1} \\ T_2 &= \frac{1}{8}\alpha_1\cdot\frac{\alpha_2(2\alpha_2-1)}{3\alpha_2^2-3\alpha_2+1}-\frac{1}{24}\,\frac{(3\alpha_2-1)(2\alpha_2-1)}{3\alpha_2^2-3\alpha_2+1} \\ T_3 &= \frac{1}{8}\,\frac{(\alpha_2-1)(2\alpha_2-1)^2}{3\alpha_2^2-3\alpha_2+1} \\ T_4 &= \frac{1}{24}\,\frac{(\alpha_2-1)(2\alpha_2-1)^2}{3\alpha_2^2-3\alpha_2+1} \end{aligned}\right\} \qquad (40)$$

From the second equation (40), it follows that we can make $T_2 = 0$ by choosing

$$\alpha_1 = \frac{3\alpha_2 - 1}{3\alpha_2} \qquad (41)$$

27. All four error factors would become zero for $\alpha_2 = \frac{1}{2}$. This means for this choice of α_2, our third-order formula would actually become a fourth-order formula. Our pair RK3(4) of RUNGE-KUTTA formulas of the third- and of the fourth-order would degenerate into one fourth-order formula.

However, by choosing for α_2 a value close to $\frac{1}{2}$, we obtain RK3(4)-formulas with small error factors (40). We give two examples for such formulas with small error factors.

28. Choosing $\alpha_2 = \frac{4}{9}$, we obtain the RUNGE-KUTTA coefficients of Table VII.

For the leading term of the local truncation error we find from Table VII

$$TE = \left(\frac{5}{288}f_o - \frac{27}{416}f_2 + \frac{245}{1872}f_3 - \frac{1}{12}f_4\right)h \qquad (42)$$

For the four error factors of the formula of Table VII we obtain

$$T_1 = \frac{1}{168}, \quad T_2 = 0, \quad T_3 = -\frac{5}{1512}, \quad T_4 = -\frac{5}{4536} \qquad (43)$$

TABLE VII. COEFFICIENTS FOR RK3(4), FORMULA 1

κ \ λ	α_κ	$\beta_{\kappa\lambda}$				c_κ	$\hat{c}_\kappa$
		0	1	2	3		
0	0	0				$\frac{1}{6}$	$\frac{43}{288}$
1	$\frac{1}{4}$	$\frac{1}{4}$				0	0
2	$\frac{4}{9}$	$\frac{4}{81}$	$\frac{32}{81}$			$\frac{27}{52}$	$\frac{243}{416}$
3	$\frac{6}{7}$	$\frac{57}{98}$	$-\frac{432}{343}$	$\frac{1053}{686}$		$\frac{49}{156}$	$\frac{343}{1872}$
4	1	$\frac{1}{6}$	0	$\frac{27}{52}$	$\frac{49}{156}$		$\frac{1}{12}$

29. Another suitable choice for α_2 would be $\alpha_2 = \frac{7}{15}$. For this value of α_2 we list the RUNGE-KUTTA coefficients in Table VIII.

TABLE VIII. COEFFICIENTS FOR RK3(4), FORMULA 2

κ \ λ	α_κ	$\beta_{\kappa\lambda}$				c_κ	$\hat{c}_\kappa$
		0	1	2	3		
0	0	0				$\frac{79}{490}$	$\frac{229}{1470}$
1	$\frac{2}{7}$	$\frac{2}{7}$				0	0
2	$\frac{7}{15}$	$\frac{77}{900}$	$\frac{343}{900}$			$\frac{2175}{3626}$	$\frac{1125}{1813}$
3	$\frac{35}{38}$	$\frac{805}{1444}$	$-\frac{77175}{54872}$	$\frac{97125}{54872}$		$\frac{2166}{9065}$	$\frac{13718}{81585}$
4	1	$\frac{79}{490}$	0	$\frac{2175}{3626}$	$\frac{2166}{9065}$		$\frac{1}{18}$

For the formula of Table VIII we obtain

$$TE = \left(\frac{4}{735}f_o - \frac{75}{3626}f_2 + \frac{5776}{81585}f_3 - \frac{1}{18}f_4\right)h \qquad (44)$$

and the four error factors

$$T_1 = \frac{1}{228}, \quad T_2 = 0, \quad T_3 = -\frac{1}{855}, \quad T_4 = -\frac{1}{2565} \qquad (45)$$

30. For comparison, we list KUTTA's third-order formula ([4], p. 440) in Table IX. His formula reads:

TABLE IX. COEFFICIENTS FOR KUTTA'S RK3-FORMULA

κ \ λ	α_κ	$\beta_{\kappa\lambda}$		c_κ
		0	1	
0	0	0		$\frac{1}{6}$
1	$\frac{1}{2}$	$\frac{1}{2}$		$\frac{2}{3}$
2	1	-1	2	$\frac{1}{6}$

It has the following error factors:

$$T_1 = -\frac{1}{24}, \quad T_2 = 0, \quad T_3 = \frac{1}{24}, \quad T_4 = 0 \qquad (46)$$

Comparing (43) and (45) with (46), we see that the largest value of (43) is only $\frac{1}{7}$ and the largest value of (45) only $\frac{2}{19}$ of the largest value of (46).

Therefore, our formulas of Table VII and Table VIII can be expected to be more economical than KUTTA's RK3-formula.

Since KUTTA's formula does not provide for a stepsize control, it requires five evaluations per step if operated with the standard stepsize procedure (recomputation of two steps as one step with double stepsize).

SECTION III. SECOND-ORDER FORMULAS

31. Allowing for four evaluations per step for an RK2(3)-formula, we list in Table X the equations of condition for a third-order formula.

TABLE X. EQUATIONS OF CONDITION FOR THIRD-ORDER FORMULA

$$(\mathrm{I},\hat{1}) \qquad \sum_{\kappa=0}^{3} \hat{c}_\kappa - 1 = 0$$

$$(\mathrm{II},\hat{1}) \qquad \sum_{\kappa=1}^{3} \hat{c}_\kappa \alpha_\kappa - \frac{1}{2} = 0$$

$$(\mathrm{III},\hat{1}) \qquad \sum_{\kappa=2}^{3} \hat{c}_\kappa P_{\kappa 1} - \frac{1}{6} = 0$$

$$(\mathrm{III},\hat{2}) \qquad \frac{1}{2} \sum_{\kappa=1}^{3} \hat{c}_\kappa \alpha_\kappa^2 - \frac{1}{6} = 0$$

32. We want to use the fourth evaluation of the differential equations as the first evaluation for the next step. This requires the conditions

$$\alpha_3 = 1\,, \quad \beta_{30} = c_0\,, \quad \beta_{31} = c_1\,, \quad \beta_{32} = c_2 \tag{47}$$

where the c_ν's are the weight factors of the second-order formula, which is obtained from the first two equations of Table IX, replacing $\hat{c}_\nu$ by c_ν and the upper index 3 of κ-sums by 2. We denote those equations for the second-order formula by (I, 1) and (II, 1).

Furthermore, we assume

$$\hat{c}_1 = 0 \tag{48}$$

and

$$(A)\quad \begin{cases} P_{21} = \frac{1}{2}\alpha_2^2 \\ P_{31} = \frac{1}{2} \end{cases}$$

33. Because of (A), equations (III, $\hat{1}$) and (III, $\hat{2}$) become identical. Therefore, we can omit equation (III, $\hat{1}$) from Table X.

From equations (II, $\hat{1}$) and (III, $\hat{2}$) we obtain, because of $\hat{c}_1 = 0$ and $\alpha_3 = 1$,

$$\left.\begin{aligned} \hat{c}_2 &= \frac{1}{6}\cdot\frac{1}{\alpha_2(1-\alpha_2)} \\ \hat{c}_3 &= \frac{1}{6}\cdot\frac{2-3\alpha_2}{1-\alpha_2} \end{aligned}\right\} \qquad (49)$$

34. Let us now consider the error factors of our second-order formula. From the last two equations of Table X, we obtain for the two error factors the expressions

$$\left.\begin{aligned} T_1 &= c_2\beta_{21}\alpha_1 - \frac{1}{6} \\ T_2 &= \frac{1}{2}(c_1\alpha_1^2 + c_2\alpha_2^2) - \frac{1}{6} \end{aligned}\right\} \qquad (50)$$

Because of the first equation (A), we can write for the first equation (50),

$$T_1 = \frac{1}{2}c_2\alpha_2^2 - \frac{1}{6} = T_2 - \frac{1}{2}c_1\alpha_1^2 \qquad (51)$$

or,

$$T_2 - T_1 = \frac{1}{2}c_1\alpha_1^2 \qquad (52)$$

From (52), it follows that $T_2 \neq T_1$, assuming $c_1 \neq 0$.

For the following, we might assume

$$T_2 = -T_1 \tag{53}$$

By a proper choice of α_1 and α_2 we try to make T_1 and $T_2 = -T_1$ sufficiently small.

From (51), (52), and (53) we find

$$T_1 = \frac{1}{2} c_2 \alpha_2^2 - \frac{1}{6} = -\frac{1}{4} c_1 \alpha_1^2$$

or

$$c_1 \alpha_1^2 + 2 c_2 \alpha_2^2 = \frac{2}{3} \tag{54}$$

Equation (54) and equation (II, 2) represent a system of two linear equations for c_1 and c_2 .

The system has the solution

$$\left.\begin{aligned} c_1 &= \frac{1}{3} \cdot \frac{2 - 3\alpha_2}{\alpha_1(\alpha_1 - 2\alpha_2)} \\ c_2 &= \frac{1}{6} \cdot \frac{3\alpha_1 - 4}{\alpha_2(\alpha_1 - 2\alpha_2)} \end{aligned}\right\} \tag{55}$$

Introducing the expression (55) for c_2 into (51) yields

$$T_1 = \frac{1}{12} \alpha_1 \frac{3\alpha_2 - 2}{\alpha_1 - 2\alpha_2} \tag{56}$$

35. From equation (56), it follows that we could make $T_1 = T_2 = 0$ by choosing $\alpha_2 = \frac{2}{3}$. However, as in the case of our third-order formula RK3(4), our pair of RUNGE-KUTTA formulas RK2(3) would then degenerate into a single third-order formula.

However, by choosing α_2 close to $\frac{2}{3}$, we might obtain a suitable pair of RK2(3)-formulas with small error factors T_1 and T_2 .

Choosing $\alpha_2 = \frac{27}{40}$ and $\alpha_1 = \frac{1}{4}$, we find the RUNGE-KUTTA coefficients of Table XI as follows.

TABLE XI. COEFFICIENTS FOR RK2(3)

κ \ λ	α_κ	$\beta_{\kappa\lambda}$			c_κ	$\hat{c}_\kappa$
		0	1	2		
0	0	0			$\frac{214}{891}$	$\frac{533}{2106}$
1	$\frac{1}{4}$	$\frac{1}{4}$			$\frac{1}{33}$	0
2	$\frac{27}{40}$	$-\frac{189}{800}$	$\frac{729}{800}$		$\frac{650}{891}$	$\frac{800}{1053}$
3	1	$\frac{214}{891}$	$\frac{1}{33}$	$\frac{650}{891}$		$-\frac{1}{78}$

From Table XI we find for the leading term of the local truncation error the approximation

$$TE = \left(-\frac{23}{1782} f_0 + \frac{1}{33} f_1 - \frac{350}{11583} f_2 + \frac{1}{78} f_3 \right) h \qquad (57)$$

The error factors for our RK2(3)-formula would read

$$T_1 = -\frac{1}{2112}, \quad T_2 = +\frac{1}{2112} \qquad (58)$$

36. It is well-known that second-order RUNGE-KUTTA formulas RK2 can be obtained by two evaluations only. It is also possible to provide a stepsize control procedure by a third evaluation. We list in Table XII an example for such a RK2(3)-formula.

For the truncation error we find from Table XII the approximation

$$TE = \left(\frac{1}{3} f_0 + \frac{1}{3} f_1 - \frac{2}{3} f_2 \right) h \qquad (59)$$

TABLE XII. COEFFICIENTS FOR RK2(3), BASED ON THREE EVALUATIONS

κ \ λ	α_κ	$\beta_{\kappa\lambda}$		c_κ	$\hat{c}_\kappa$
		0	1		
0	0	0		$\frac{1}{2}$	$\frac{1}{6}$
1	1	1		$\frac{1}{2}$	$\frac{1}{6}$
2	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$		$\frac{2}{3}$

The error factors for the formula of Table XII would read

$$T_1 = -\frac{1}{6}, \quad T_2 = \frac{1}{12} \tag{60}$$

It is interesting to note that KUTTA's third-order formula (Table IX) can also be operated as a second-order formula RK2(3) with stepsize control. We have to change only the weight factors of the formula, considering the c_κ-values of Table IX as $\hat{c}_\kappa$-values of the RK2(3)-formula and using $c_0 = 0$ $c_1 = 1$ as its c_κ-values.

Comparing (58) with (60), we notice that the error factors of our formula of Table XI are only $\frac{1}{352}$ of the larger error factor (60). Therefore, we may again expect our formula of Table XI to be more economical than the formula of Table XII, since the formula of Table XI also requires only three evaluations per step (except for the very first integration step).

SECTION IV. FIRST-ORDER FORMULA

37. If we base our RUNGE-KUTTA formula RK1(2) upon three evaluations per step, we obtain for the second-order formula the following equations of condition.

TABLE XIII. EQUATIONS OF CONDITION FOR SECOND-ORDER FORMULA

$$(\mathrm{I},\hat{1}) \qquad \sum_{\kappa=0}^{2} \hat{c}_\kappa - 1 = 0$$

$$(\mathrm{II},\hat{1}) \qquad \sum_{\kappa=1}^{2} \hat{c}_\kappa \alpha_\kappa - \frac{1}{2} = 0$$

Since we intend to use the third evaluation again as the first evaluation for the next step, we require

$$\alpha_2 = 1, \quad \beta_{20} = c_0, \quad \beta_{21} = c_1 \tag{61}$$

where c_0 and c_1 are the weight factors of the first-order formula, which is obtained from the first equation of Table XIII, replacing $\hat{c}_\nu$ by c_ν and the upper index 2 of the κ-sum by 1. Let us denote this equation for the first-order formula by **(I, 1)**.

Obviously, there is only one error factor T_1, obtained from $(\mathrm{II},\hat{1})$ as:

$$T_1 = c_1\alpha_1 - \frac{1}{2} \tag{62}$$

We have to make this error factor small to obtain an efficient RK1(2)-formula. However, we should not make T_1 zero, since our pair of formulas RK1(2) would degenerate for $T_1 = 0$ into one second-order formula. We choose as coefficients the values of Table XIV:

TABLE XIV. COEFFICIENTS FOR RK1(2)

κ \ λ	α_κ	$\beta_{\kappa\lambda}$		c_κ	$\hat{c}_\kappa$
		0	1		
0	0			$\frac{1}{256}$	$\frac{1}{512}$
1	$\frac{1}{2}$	$\frac{1}{2}$		$\frac{255}{256}$	$\frac{255}{256}$
2	1	$\frac{1}{256}$	$\frac{255}{256}$		$\frac{1}{512}$

which clearly satisfy equations (I, 1), $(\mathrm{I},\hat{1})$, and $(\mathrm{II},\hat{1})$.

For the approximate truncation error we find

$$TE = \frac{1}{512}(f_0 - f_2)\,h, \tag{63}$$

and the error factor T_1 becomes

$$T_1 = -\frac{1}{512} \tag{64}$$

38. EULER-CAUCHY's method can be considered as a first-order RUNGE-KUTTA formula, requiring only one evaluation per step. The method can even be operated as a RUNGE-KUTTA formula RK1(2) without additional evaluations by adding a second evaluation that can again be taken over as the first evaluation for the next step.

The resulting pair of formulas is shown in Table XV.

TABLE XV. EULER-CAUCHY'S METHOD AS RK1(2)

κ \ λ	α_κ	$\beta_{\kappa\lambda}$	c_κ	$\hat{c}_\kappa$
		0		
0	0		1	$\frac{1}{2}$
1	1	1		$\frac{1}{2}$

The second-order formula RK2 of Table XV is obviously the so-called modified EULER-CAUCHY method.

Table XV yields as approximate truncation error

$$TE = \frac{1}{2}(f_0 - f_1)\,h \tag{65}$$

The error factor for the RK1(2)-formula of Table XV becomes

$$T_1 = -\frac{1}{2} \tag{66}$$

Comparison of (64) and (66) clearly shows that our formula of Table XIV will be more efficient than the formula of Table XV. Even the one additional evaluation per step required by our formula of Table XIV, in general, will not outweigh the advantage of the much smaller error factor.

We shall present a numerical example in Section V.

SECTION V. A NUMERICAL EXAMPLE (ORDINARY DIFFERENTIAL EQUATIONS)

39. In this section we present the numerical results of our new formulas for the same example that we considered in our earlier NASA report ([1], p. 30):

$$\left.\begin{aligned} &y' = -2xy \cdot \log z\ , \quad z' = 2xz \cdot \log y \\ &\text{Initial values: } x_0 = 0\ , \quad y_0 = e\ , \quad z_0 = 1 \\ &\text{Exact solution: } y = e^{\cos(x^2)}\ , \quad z = e^{\sin(x^2)} \end{aligned}\right\} \qquad (67)$$

For reasons of comparison, we also include the results for some formulas of other authors. Our results are presented in Table XVI, the results of our new formulas being marked by an asterisk (*).

The numerical integration of problem (67) was executed on an IBM-7094 computer. All methods listed in Table XVI were run in single precision (eight decimal digits) and with the same tolerance (10^{-8}) for the local truncation error. Since the first-order formulas require a very small stepsize, we ran these formulas only up to $x = 5$; for the higher-order formulas the integration was performed up to $x = 25$.

40. Table XVI shows that the small error factors of our new formulas (*) make themselves felt in a considerable reduction of the number of integration steps and, therefore, of the computer time required for our problem.

The greatest reduction gained was for our first- and second-order formulas. For these formulas we required only about $\frac{1}{9}$ and about $\frac{1}{6}$ of the computer time of the corresponding conventional formulas. In the case of the third-order formula, we could cut the computer time to less than $\frac{1}{2}$, compared

TABLE XVI. COMPARISON OF THE VARIOUS METHODS FOR EXAMPLE (67)

	Method	Order of Method	Number of Evaluations Per Step	x_{max}	Results for $x = x_{max}$ (Tolerance: 10^{-8}) Number of Steps	Total Number of Evaluations	Running Time (min) on IBM-7094	Accumulated Errors in y and z Δy	Δz
	EULER-CAUCHY (Table XV)	1	1	5	269 956	269 956	2.90	$+0.3018 \cdot 10^{-2}$	$-0.2945 \cdot 10^{-3}$
(*)	RK1(2) (Table XIV)	1	2	5	16 871	33 742	0.32	$+0.1926 \cdot 10^{-3}$	$-0.1543 \cdot 10^{-4}$
	RK2(3) (Table XII)	2	3	25	243 510	730 530	6.43	$+0.1458 \cdot 10^{-4}$	$-0.1781 \cdot 10^{-3}$
(*)	RK2(3) (Table XI)	2	3	25	37 493	112 479	1.13	$-0.1874 \cdot 10^{-4}$	$-0.8330 \cdot 10^{-5}$
	KUTTA (Table IX)	3	5	25	41 862	209 310	1.91	$+0.2190 \cdot 10^{-5}$	$-0.4664 \cdot 10^{-5}$
(*)	RK3(4) (Table VII)	3	4	25	23 225	92 900	0.90	$-0.2611 \cdot 10^{-5}$	$+0.1639 \cdot 10^{-4}$
(*)	RK3(4) (Table VIII)	3	4	25	22 054	88 216	0.85	$-0.2578 \cdot 10^{-5}$	$+0.1474 \cdot 10^{-4}$
	KUTTA (Table V)	4	7	25	16 010	112 070	0.95	$+0.1881 \cdot 10^{-5}$	$+0.2207 \cdot 10^{-4}$
	SARAFYAN (Table IV)	4	6	25	14 746	88 476	0.81	$-0.1546 \cdot 10^{-5}$	$-0.2086 \cdot 10^{-4}$
(*)	RK4(5) (Table II)	4	6	25	11 059	66 354	0.68	$+0.1222 \cdot 10^{-5}$	$+0.2015 \cdot 10^{-4}$
(*)	RK4(5) (Table III)	4	6	25	9 947	59 682	0.58	$+0.2041 \cdot 10^{-5}$	$+0.2512 \cdot 10^{-4}$

with KUTTA's formula. For fourth-order formulas our gains are more modest. Out second fourth-order formula (Table III), however, still requires only about $\frac{3}{5}$ of the computer time of KUTTA's formula and about $\frac{3}{4}$ of the time of SARAFYAN's formula.

The accuracy of our new formulas is about the same as that of the conventional formulas, except for the case of the first-order formulas. In this case we gain one more decimal digit with our new formula. Because the number of steps required by our new formula is only about $\frac{1}{16}$ of the number of steps of EULER-CAUCHY's formula, we do not accumulate round-off errors as heavily as the latter formula.

Compared with the conventional RUNGE-KUTTA formulas our new formulas offer results of the same accuracy in a fraction of the computer time. Therefore, our new formulas might be of interest for the numerical integration of ordinary differential equations.

Moreover, we shall show in the following sections of this paper that they can also be applied successfully to certain partial differential equations of the parabolic type.

SECTION VI. APPLICATION TO HEAT TRANSFER PROBLEMS

41. Let us consider the one-dimensional heat transfer problem

$$\left.\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \\ &\text{Initial Condition:} \quad t = 0 : u(x,0) = u_0(x) \\ &\text{Boundary Conditions:} \begin{cases} x = 0 : u(0,t) = b_0(t) \\ x = 1 : u(1,t) = b_1(t) \end{cases} \end{aligned}\right\} \qquad (68)$$

The first equation of (68) represents the simplest partial differential equation of the parabolic type.

Problem (68) requires finding a quantity u (usually the temperature) as function of the space variable x and the time variable t.

The existence of a solution of the problem (68) can be shown, under proper assumptions, by Fourier series methods. See, for example, the textbook by CARSLAW and JAEGER ([7], pp. 76-88). A quite elementary proof of the uniqueness of the solution of (68) is given in a textbook by BIEBERBACH ([8], pp. 352-353).

42. Replacing both derivatives of the first equation (68) by finite differences, one obtains the well-known difference equation

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{u_{i+1,j} - 2\,u_{i,j} + u_{i-1,j}}{h^2} \qquad (69)$$

with $k = \Delta t$ and $h = \Delta x$.

Equation (69) is widely used for obtaining numerical solutions of problem (68), since this is the simplest explicit approach to the problem.

In applying (69) one has to pay attention to the fact that the mesh sizes h and k have to satisfy the well-known stability condition

$$r = \frac{k}{h^2} \leqq \frac{1}{2} \qquad (70)$$

The condition (70) represents a certain restriction for the time mesh k if the space mesh h is given.

To preserve a reasonable accuracy of the results one has to apply a small space mesh h. In some problems the time mesh k, resulting from (70), may then become prohibitively small.

Furthermore, one has to consider that the mesh size k, resulting from (70), will guarantee the stability of the results, but no estimate of the local truncation error can be obtained from (70).

43. There exist other difference methods without stability restrictions, for example, the explicit method of DU FORT-FRANKEL [9]:

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} = \frac{u_{i+1,j} - (u_{i,j+1} + u_{i,j-1}) + u_{i-1,j}}{h^2} , \quad (71)$$

or the implicit method of CRANK-NICOLSON [10]:

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{1}{2}\left\{ \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{h^2} + \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \right\} \quad (72)$$

However, other difficulties occur in the application of these two methods. The method of DU FORT-FRANKEL is a three-level method (time levels j-1, j, j+1) requiring a special starting procedure and additional considerations for a stepsize change of the time step. In the case of the implicit method of CRANK-NICOLSON, one has to solve a system of linear equations of the triple-diagonal form for each time step.

Although these difficulties might not be considered too serious, these methods again do not furnish an estimate of the local truncation error. Since they have no stability restrictions, an increased danger exists in that the integration could be performed with too large a time step. If this is the case, one would lose any accuracy, and, after a certain number of time steps, the computed u-values would bear little resemblence to the solution of the problem.

Equations (72) and (69) have identical left-hand sides; they approximate the time derivative $\frac{\partial u}{\partial t}$ by the same first-order accurate, finite difference. Therefore, the time integration of the CRANK-NICOLSON method cannot be more accurate than in the case of the explicit method (69).

44. If one wants to take full advantage of the accuracy that modern electronic computers are capable of, a more accurate approximation of the partial derivatives in the heat transfer equation — first equation (68) — and a current stepsize control for the integration become absolute necessities. A convenient way to achieve these objectives consists of converting the heat transfer equation into a system of ordinary differential equations by replacing the space derivative only by a finite difference. The resulting system of ordinary differential equations can then be integrated by RUNGE-KUTTA methods.

This is not a new procedure; it was suggested long ago by D. R. HARTREE and collaborators ([11] and [12], p. 254). They suggest replacing the first equation (68) by the system

$$\frac{du_i(\tau)}{d\tau} = u_{i+1}(\tau) - 2u_i(\tau) + u_{i-1}(\tau) = \delta^2 u_i(\tau) \qquad (73)$$

with $\tau = \frac{t}{h^2}$.

Naturally, one can apply a better approximation to $\frac{\partial^2 u}{\partial x^2}$ by using higher-order central differences, resulting in

$$\frac{du_i(\tau)}{d\tau} = \delta^2 u_i(\tau) - \frac{1}{12}\delta^4 u_i(\tau) + \frac{1}{90}\delta^6 u_i(\tau) - + \ldots . \qquad (74)$$

Formula (74) is suggested in GOODWIN's book ([13], p. 113).

In equations (73) and (74) the authors used the standard fourth-order RUNGE-KUTTA formula (Table V) for the numerical integration. GOODWIN ([13], pp. 114-115) and FOX ([14], pp. 240-241) point out that the fourth-order RUNGE-KUTTA method offers few advantages compared with the explicit difference formula (69), since the stability restriction of the RUNGE-KUTTA method is only slightly better ($r < 0.7$ instead of $r < 0.5$). On the other hand, the RUNGE-KUTTA method requires considerably more computer time per time step than formula (69). Therefore, it might seem questionable whether RUNGE-KUTTA methods are really worthwhile for application to problems such as (68).

45. We feel that the explicit difference method (69) certainly has its merits when applied to the computationally simple problem (68) and when requiring a moderate accuracy only.

However, the new low-order RUNGE-KUTTA formulas with stepsize control that are presented in this paper show the application of RUNGE-KUTTA formulas to parabolic partial differential equations in a different light, especially in connection with the use of highly accurate electronic computers.

For our numerical computations we used an eight-decimal digit computer (IBM-7094), and, when applying our new RUNGE-KUTTA formulas to heat

transfer problems, we required that the local truncation error became negligible. This means we required that the local truncation error not contribute to the eight leading digits that the computer carries.

By doing so, we were able, when using higher-order central differences for the space derivatives, to obtain by means of our new RUNGE-KUTTA formulas, five to six good digits, even after a considerable number of integration steps. (See the examples in Section VII.) We found that these somewhat severe accuracy requirements could be well-satisfied by our low-order RUNGE-KUTTA methods and that we did not run into stability problems with these low-order methods since our accuracy requirements are much sharper than the stability requirements of our formulas.

46. Comparing the explicit difference method (69) with EULER-CAUCHY's method (Table XV), we see that both methods are identical as far as the time integration is concerned. Since the method of Table XV yields a convenient stepsize control procedure we have substituted EULER-CAUCHY's method of Table XV, for the explicit difference method (69).

47. Naturally, the explicit difference method and any RUNGE-KUTTA method can be run in a fraction of the computer time if we apply less severe accuracy restrictions, for example, if we apply the stability restriction only. However, such a relaxation of the accuracy restriction would result in a severe loss of accuracy.

48. The RUNGE-KUTTA methods are not restricted to the simple problem (68). They can be applied to more involved one-dimensional heat transfer problems, as outlined in the next section. Multi-dimensional heat transfer problems can also be made amenable to RUNGE-KUTTA methods when replacing all space derivatives by proper finite differences.

SECTION VII. TWO NUMERICAL EXAMPLES (HEAT TRANSFER PROBLEMS)

49. In this section we present numerical results of our new RUNGE-KUTTA formulas for two one-dimensional heat transfer problems. In both problems the exact solution is available so that the errors of our formulas can be stated.

First problem:

$$\left.\begin{aligned} &\frac{\partial u}{\partial t} = \frac{1}{4} \cdot \frac{e^2}{2 + x^2} \cdot e^{-u} \cdot \frac{\partial^2 u}{\partial x^2} \\ &\text{Initial Condition:} \quad t = 0 : u = 2\left[1 - \log(2 - x^2)\right] \\ &\text{Boundary Conditions:} \begin{cases} x = 0 : \dfrac{\partial u}{\partial x} = 0 \\ x = 1 : u = 2 + \log(1 + t) \end{cases} \\ &\text{Exact Solution:} \quad u = 2 + \log(1 + t) - 2\log(2 - x^2) \end{aligned}\right\} \tag{75}$$

Using higher-order central differences when replacing $\frac{\partial^2 u}{\partial x^2}$ by finite differences, the partial differential equation of our problem is converted into the following system of ordinary differential equations:

$$\left.\begin{aligned} \frac{du_i(\tau)}{d\tau} = \frac{1}{4} \cdot \frac{e^2}{2 + x_i^2} \cdot e^{-u_i(\tau)} \Big\{ &\delta^2 u_i(\tau) - \frac{1}{12}\delta^4 u_i(\tau) \\ &+ \frac{1}{90}\delta^6 u_i(\tau) - + \ldots \Big\} \end{aligned}\right\} \tag{76}$$

The introduction of the fourth-, sixth-, . . . order differences in (76) makes necessary some extrapolation in the vicinity of the boundary line $x = 1$. When including the fourth-order differences only, we have assumed that the sixth-order differences are constant in the vicinity of $x = 1$. Correspondingly, the eighth-order differences are assumed to be constant if sixth-order differences are still carried in (76). For $x = 0$ no extrapolation is needed since the problem is symmetric with respect to $x = 0$.

For our computation we divided the x-interval $< 0, 1 >$ into 16 equal parts ($h = \frac{1}{16}$). The numerical integration of (76) with respect to $\tau = \frac{t}{h^2}$ was performed with a tolerance of 10^{-8} on an eight-decimal digit computer. The stepsize control procedure was applied for $x = 0$ only.

Table XVII shows the results for $t = 100$, obtained with some of our new formulas. Under the heading "Method" of Table XVII we have also indicated the order of the highest central difference δ^2, δ^4, or δ^6 carried in (76).

TABLE XVII. COMPARISON OF VARIOUS RUNGE-KUTTA METHODS FOR PROBLEM (75)

Results for t = 100 (Tolerance: 10^{-8}; $h = \frac{1}{16}$)

Method	Order of Method	Number of Steps	Running Time (min) on IBM-7094	Accumulated Error (maximum) in u
EULER-CAUCHY (Table XV) — δ^2	1	30 721	1.75	$0.1408 \cdot 10^{-2}$
RK1(2) (Table XIV) — δ^2	1	1 924	0.22	$0.1452 \cdot 10^{-2}$
RK2(3) (Table XI) — δ^2	2	822	0.23	$0.1425 \cdot 10^{-2}$
RK3(4) (Table VIII) — δ^2	3	1 036	0.33	$0.1424 \cdot 10^{-2}$
EULER-CAUCHY (Table XV) — δ^4	1	30 715	2.07	$-0.3767 \cdot 10^{-4}$
RK1(2) (Table XIV) — δ^4	1	1 998	0.28	$-0.1991 \cdot 10^{-4}$
RK2(3) (Table XI) — δ^4	2	1 070	0.36	$-0.1961 \cdot 10^{-4}$
RK3(4) (Table VIII) — δ^4	3	1 305	0.50	$-0.2086 \cdot 10^{-4}$
EULER-CAUCHY (Table XV) — δ^6	1	30 712	2.33	$-0.2068 \cdot 10^{-4}$
RK1(2) (Table XIV) — δ^6	1	2 072	0.39	$-0.1848 \cdot 10^{-5}$
RK2(3) (Table XI) — δ^6	2	1 227	0.45	$-0.3636 \cdot 10^{-5}$
RK3(4) (Table VIII) — δ^6	3	1 520	0.61	$-0.4947 \cdot 10^{-5}$

From Table XVII one recognizes immediately the gain of accuracy obtained by taking into account the fourth- and sixth-order central differences in (76). The results clearly suggest that at least the fourth-order central differences should be included in the computation.

The table also shows that our new RUNGE-KUTTA methods give about the same or more accurate results in a fraction of the time required by EULER-CAUCHY's method, which is identical with the explicit difference formula (69).

However, our third-order formula is already slower on the computer than our first- and second-order formulas. The trend continues: our fourth-order formula is again slower than our third-order formula, etc.

This confirms the experience that other authors have had with fourth-order RUNGE-KUTTA formulas, which are too slow for heat transfer problems and cannot compete with our new lower-order RUNGE-KUTTA formulas.

50. We present another example:

Second problem:

$$\left.\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + 2t \cdot \frac{\partial u}{\partial x} + u\left[(\log u)^2 + \log u - 1\right] \\ &\text{Initial Condition:} \quad t = 0 : u = e^{\cos x} \\ &\text{Boundary Conditions:} \quad \begin{cases} x = 0 : u = e^{\cos(t^2)} \\ x = 1 : u = e^{\cos(1+t^2)} \end{cases} \\ &\text{Exact Solution:} \quad u = e^{\cos(x+t^2)} \end{aligned}\right\} \quad (77)$$

In this example we have to replace $\frac{\partial u}{\partial x}$ by finite differences also. Taking into account higher-order central differences, we replace the partial differential equation of our problem with the following system of ordinary differential equations:

$$\left.\begin{aligned} \frac{du_i(\tau)}{d\tau} = &\left[\delta^2 u_i(\tau) - \frac{1}{12}\delta^4 u_i(\tau) + \frac{1}{90}\delta^6 u_i(\tau)\right] \\ &+ h^3 \cdot \tau\left[\left(u_{i+1}(\tau) - u_{i-1}(\tau)\right) - \frac{1}{6}\left(\delta^2 u_{i+1}(\tau) - \delta^2 u_{i-1}(\tau)\right)\right. \\ &\left. + \frac{1}{30}\left(\delta^4 u_{i+1}(\tau) - \delta^4 u_{i-1}(\tau)\right)\right] \\ &+ h^2 \cdot u_i(\tau)\left[\left(\log u_i(\tau)\right)^2 + \log u_i(\tau) - 1\right] \end{aligned}\right\} \quad (78)$$

The numerical integration of (78) is performed in the same way as in our first problem. However, our new problem requires some extrapolating for $x = 0$ also; and the stepsize control procedure is now performed for $x = \frac{1}{2}$.

Table XVIII shows our results for t = 5. These results are quite similar to those for our first problem. The computer time for our new RUNGE-KUTTA formulas is, however, here a considerably smaller fraction of the computer time of the explicit difference formula (EULER-CAUCHY's formula).

TABLE XVIII. COMPARISON OF VARIOUS RUNGE-KUTTA METHODS FOR PROBLEM (77)

Results for t = 5 (Tolerance: 10^{-8}; h = $\frac{1}{16}$)

Method	Order of Method	Number of Steps	Running Time (min) on IBM-7094	Accumulated Error (maximum) in u
EULER-CAUCHY (Table XV) — δ^2	1	235 354	19.46	$0.6313 \cdot 10^{-3}$
RK1(2) (Table XIV) — δ^2	1	14 737	2.47	$0.6711 \cdot 10^{-3}$
RK2(3) (Table XI) — δ^2	2	2 142	0.87	$0.7065 \cdot 10^{-3}$
RK3(4) (Table VIII) — δ^2	3	2 519	1.03	$0.6745 \cdot 10^{-3}$
EULER-CAUCHY (Table XV — δ^4	1	235 388	22.87	$-0.4032 \cdot 10^{-4}$
RK1(2) (Table XIV) — δ^4	1	14 906	2.93	$-0.3516 \cdot 10^{-5}$
RK2(3) (Table XI) — δ^4	2	2 695	1.08	$0.3994 \cdot 10^{-5}$
RK3(4) (Table VIII) — δ^4	3	3 334	1.91	$0.3129 \cdot 10^{-5}$
EULER-CAUCHY (Table XV) — δ^6	1	235 372	25.32	$-0.5391 \cdot 10^{-5}$
RK1(2) (Table XIV) — δ^6	1	14 971	3.25	$-0.4351 \cdot 10^{-5}$
RK2(3) (Table XI) — δ^6	2	2 812	1.50	$-0.3427 \cdot 10^{-5}$
RK3(4) (Table VIII) — δ^6	3	3 677	2.38	$-0.1132 \cdot 10^{-5}$

51. In both examples, our new low-order RUNGE-KUTTA formulas proved to be considerably faster and of equal or better accuracy than the conventional explicit difference formula (69). This is not too suprising if one realizes that (69) is a formula of first-order accuracy only, and that its truncation error factor is 256 times as large as the error factor of our first-order formula RK1(2) of Table XIV.

George C. Marshall Space Flight Center
National Aeronautics and Space Administration
Marshall Space Flight Center, Alabama 35812, April 15, 1969
129-04-03-00-62

REFERENCES

[1] FEHLBERG, E.: Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control. NASA TR R-287, October 1968.

[2] BUTCHER, J. C.: Coefficients for the Study of Runge-Kutta Integration Processes. J. Austral. Math. Soc., vol. 3, 1963, pp. 185-201.

[3] SARAFYAN, D.: Error Estimation for Runge-Kutta Methods Through Pseudo-Iterative Formulas. Technical Report No. 14, Louisiana State University in New Orleans, May 1966.

[4] KUTTA, W.: Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. Z. Math. Phys. Band 46, 1901, pp. 435-453.

[5] FEHLBERG, E.: New High-Order Runge-Kutta Formulas with Stepsize Control for Systems of First- and Second-Order Differential Equations. Z. angew. Math. Mech., Band 44, 1964, Sonderheft, T 17-T 29.

[6] FEHLBERG, E.: New High-Order Runge-Kutta Formulas with an Arbitrarily Small Truncation Error. Z. angew. Math. Mech., Band 46, 1966, pp. 1-16.

[7] CARSLAW, H. S.; and JAEGER, J. C.: Conduction of Heat in Solids. Clarendon Press (Oxford), 1947.

[8] BIEBERBACH, L.: Theorie der Differentialgleichungen. Second ed., Verlag Springer (Berlin), 1926.

[9] DU FORT, E.C.; and FRANKEL, S. P.: Stability Conditions in the Numerical Treatment of Parabolic Differential Equations. Math. Tables Aids Comput., vol. 7, 1953, pp. 135-152.

[10] CRANK, J.; and NICOLSON, P.: A Practical Method for Numerical Evaluation of Solutions of Partial Differential Equations of the Heat-Conduction Type. Proc. Cambridge Philos. Soc., vol. 43, 1947, pp. 50-67.

REFERENCES (Concluded)

[11] HARTREE, D. R.; and WOMERSLEY, J. R.: A Method for the numerical or Mechanical Solution of Certain Types of Partial Differential Equations. Proc. Royal Soc. London, vol. 161, 1937, pp. 353-366.

[12] HARTREE, D. R.: Numerical Analysis. Second ed., Clarendon Press (Oxford), 1958.

[13] GOODWIN, E. T., ed.: Modern Computing Methods. Second ed., Philosophical Library (New York), 1961.

[14] FOX, L., ed.: Numerical Solution of Ordinary and Partial Differential Equations. Pergamon Press (New York), 1962.

NASA-Langley, 1969 —— 19 M-526