Jurnal Matematika UNAND
Vol. **11** No. **1** Hal. 32 – 46
Edisi Januari 2022
ISSN : 2303–291X
e-ISSN : 2721–9410

AKURASI DAN EFISIENSI SOLUSI PERSAMAAN DIFERENSIAL BIASA DENGAN MASALAH NILAI BATAS PADA JULIA DAN OCTAVE

NGAKAN KOMANG KUTHA ARDHANA, SRI NURDIATI*,
MOHAMAD KHOIRUN NAJIB, SYAHID AHMAD MUKRIM

Departemen Matematika, FMIPA, IPB University, Bogor 16680, Indonesia.
email : kutha@apps.ipb.ac.id, nurdiati@apps.ipb.ac.id,
mkhoirun_najib@apps.ipb.ac.id, syahid_ahmad@apps.ipb.ac.id

Diterima 1 Desember 2021 Direvisi 24 Februari 2022 Dipublikasikan 7 April 2022

Abstrak. Salah satu program yang andal untuk menyelesaikan masalah nilai batas secara numerik adalah MATLAB. Namun, program tersebut bersifat komersial, sehingga tidak semua pengguna dapat menggunakannya. Adapun program lain yang bersifat open source seperti Octave, yang sering digunakan karena kemiripannya dengan MATLAB. Selain itu, ada pula Julia, yang diklaim dinamis dan cepat. Keduanya menyediakan rutin untuk menyelesaikan masalah nilai batas menggunakan metode kolokasi. Oleh karena itu, penelitian ini bertujuan untuk menguji dan membandingkan akurasi serta efisiensi dari rutin pencarian solusi masalah nilai batas pada Octave dan Julia. Hasil yang diperoleh menunjukkan bahwa pencarian solusi masalah nilai batas pada Julia jauh lebih akurat dan efisien dibandingkan Octave berdasarkan beberapa kasus yang diberikan. Julia menyelesaikan masalah nilai batas dengan waktu komputasi rata-rata 2500 kali lebih cepat dibandingkan Octave. Dari sisi akurasi, Julia memiliki relatif *error* rata-rata 100000 kali lebih kecil dibandingkan Octave.

Kata Kunci: akurasi, efisiensi, julia, masalah nilai batas, octave

1. Pendahuluan

Masalah nilai batas atau *boundary value problem* (BVP) merupakan suatu permasalahan dalam matematika maupun terapannya yang sering kali tidak dapat ditentukan solusi analitiknya. Oleh sebab itu, salah satu cara alternatif untuk mencari solusi dari suatu masalah nilai batas adalah menggunakan pendekatan numerik [1]. Banyak bahasa pemrograman yang telah menyediakan rutin untuk menyelesaikan masalah nilai batas ini. Masalah nilai batas jauh lebih sulit untuk dipecahkan daripada masalah nilai awal dan *solver* manapun mungkin bisa gagal untuk menemukan solusi numerik, bahkan dengan tebakan yang bagus untuk solusi dan parameter yang tidak diketahui. Salah satu *solver* yang telah terbukti akurat dalam menyelesaikan masalah nilai batas adalah fungsi bvp4c pada MATLAB yang menerapkan metode

*penulis korespondensi

kolokasi ordo keempat [2]. Namun, MATLAB adalah program komersial sehingga diperlukan lisensi untuk menggunakan fasilitas yang disediakan oleh MATLAB.

Salah satu alternatif bahasa pemrograman yang bersifat *free* dan *open source* serta mirip dan kompatibel dengan MATLAB adalah Octave dan FreeMat karena keduanya memiliki sintaks yang sama dan memiliki kemampuan untuk menjalankan *m-file*. Di antara kedua program ini, Octave jauh lebih matang dan memiliki lebih banyak fungsi yang tersedia untuk digunakan [3]. Oleh karena itu, banyak perguruan tinggi di Indonesia yang menggunakan Octave sebagai bahasa pemrogramannya.

Beberapa tahun terakhir telah dikembangkan bahasa pemrograman baru yang bersifat *free* dan *open source* serta diklaim memiliki kecepatan hampir setingkat C, dinamis seperti Ruby, terasa seperti Lisp, familiar dengan notasi matematika seperti MATLAB, mudah digunakan seperti Python dan R, yaitu bahasa Julia [4]. Selain itu, Julia terintegrasi dengan baik dengan bahasa lain termasuk memanggil C secara langsung, Python melalui PyCall, dan R melalui RCall [5]. Versi stabil dari Julia yang pertama yaitu Julia v.1.0 pertama kali rilis pada tanggal 8 Agustus 2018, sehingga bahasa pemrograman ini masih tergolong baru.

Baik Octave maupun Julia, keduanya menyediakan rutin untuk menyelesaikan masalah nilai batas yaitu fungsi bvp4c pada paket odepkg untuk Octave dan fungsi BVProblem pada paket DifferentialEquations.jl pada Julia. Kedua rutin tersebut juga menyediakan metode penyelesaian yang sama yaitu metode kolokasi. Oleh karena itu, tujuan penelitian ini adalah untuk menguji akurasi dan efisiensi dari rutin pencarian solusi masalah nilai batas pada Octave dan Julia. Tujuan lainnya ialah memberikan teladan penggunaan *open source* di lingkungan komputasi matematika Indonesia, yang secara tidak langsung memberi dukungan pada proyek nasional IGOS (*Indonesia Goes Open Source*).

2. Metode Penelitian

Metodologi penelitian yang digunakan berupa simulasi numerik dari kedua bahasa pemrograman yaitu Julia versi 1.5.1 dan Octave versi 5.1.0 untuk menyelesaikan Masalah Nilai Batas (MNB) standar yang digunakan untuk menguji suatu algoritma penyelesaian. Simulasi numerik dijalankan menggunakan komputer dengan prosesor AMD A4 dan RAM 8GB. Beberapa kondisi yang setara diterapkan pada kedua bahasa pemrograman seperti metode yang digunakan yaitu metode kolokasi, nilai tebakan awal solusi, dan ukuran langkah. Selanjutnya, waktu komputasi dan akurasi (jika kasus tersebut diketahui solusi analitiknya) dihitung. Terdapat lima kasus yang digunakan, yaitu

(1) MNB sederhana,
(2) MNB dengan banyak peubah,
(3) MNB dengan parameter yang ditentukan,
(4) MNB dengan parameter yang tidak ditentukan, dan
(5) terapan MNB pada masalah kontrol optimum

dengan rincian masing-masing kasus seperti berikut.

2.1. *Kasus 1 (MNB Sederhana)*

Diberi masalah nilai batas [6] dengan persamaan

$$y'' = \frac{1}{x}y' - \frac{1}{x^2}y + 1 \tag{2.1}$$

dengan nilai batas $y(0.5) = 1$ dan $y(4.5) = 2$ pada interval $x \in [0.5, 4.5]$ serta solusi eksak yaitu $y(x) = x^2 - 0.252582649x - 2.5284423x\ln(x)$.

2.2. *Kasus 2 (MNB Peubah Banyak)*

Diketahui masalah nilai batas dengan sistem persamaan seperti berikut.

$$\begin{aligned} u' &= 0.5u(w-u) \\ v' &= -0.5(w-u) \\ w' &= \frac{0.9 - 1000(w-y) - 0.5w(w-u)}{z} \\ z' &= 0.5(w-u) \\ y' &= -100(y-w) \end{aligned} \tag{2.2}$$

dengan nilai batas yaitu $u(0) = v(0) = w(0) = 1$, $z(0) = -10$, dan $w(1) = y(1)$ pada interval $t \in [0, 1]$. Sebagai perbandingan, solusi MUSN kasus 2 dapat diperoleh dari https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/3819/versions/6/previews/BVP_tutorial/BVP_examples_70/ex1bvp.m/index.html [7].

2.3. *Kasus 3 (MNB Dengan Parameter Tetap)*

Diberikan persamaan diferensial dengan suatu parameter p yaitu

$$y'' + \frac{3py}{(p+t^2)^2} = 0 \tag{2.3}$$

dengan solusi eksak $y(t) = \dfrac{t}{\sqrt{p+t^2}}$. Uji standar yang digunakan untuk menyelesaikan MNB di atas adalah menyelesaikan pada interval $[-0.1, 0.1]$ dengan kondisi batas

$$\begin{aligned} y(-0.1) &= \frac{-0.1}{\sqrt{p+0.01}} \\ y(0.1) &= \frac{0.1}{\sqrt{p+0.01}} \end{aligned} \tag{2.4}$$

Pada kasus ini, digunakan nilai parameter $p = 10^{-5}$ dengan nilai tolerasnsi relatif *error* yaitu 10^{-3} dan 10^{-4} [8].

2.4. *Kasus 4 (MNB Dengan Parameter Bebas)*

Persamaan yang akan diselesaikan adalah Persamaan Mathieu yaitu

$$y'' + (\lambda - 2q\cos 2x)y = 0 \tag{2.5}$$

pada interval $[0, \pi]$ dengan nilai batas $y'(0) = 0$ dan $y'(\pi) = 0$ ketika $q = 5$. Solusi dinormalisasikan sedemikian sehingga $y(0) = 1$. Solusi penyelesaian membutukan nilai eigen λ yang sesuai, sehingga nilai batas $y'(\pi) = 0$ dapat terpenuhi [2].

2.5. *Kasus 5 (Masalah Kontrol Optimum)*

Perhatikan sistem terkontrol $y' = y^2 + v$ dan kontrol v mengendalikan lintasan dari $y(0) = 2$ ke $y(10) = 1$. Tujuan kasus ini adalah mencari nilai kontrol v yang meminimumkan ongkos kuadratik

$$J(y, v) = \int_0^{10} (y^2 + 10v^2)\ dt \tag{2.6}$$

Fungsi ongkos ini menyebabkan peubah keadaan y dan peubah kontrol v terpenalisasi (jika ada pelanggaran kendala, fungsi dikenakan sanksi dengan penambahan nilai positif [9]), sehingga menyebabkan peubah keadaan dan kontrol menjadi kecil. Syarat cukup masalah kontrol optimum tersebut dapat diperoleh dengan mendefinisikan fungsi Hamiltonian

$$H = (y^2 + 10v^2) + \lambda(y^2 + v) \tag{2.7}$$

Dengan menurunkan H terhadap λ, y, dan v, akan diperoleh masalah nilai batas

$$\begin{aligned} y' &= y^2 - \frac{\lambda}{20} \\ \lambda' &= -2y - 2\lambda y \end{aligned} \tag{2.8}$$

dengan $v = -\lambda/20$ serta nilai batas $y(0) = 2$ dan $y(10) = 1$ [10].

3. Hasil dan Pembahasan

3.1. *Kasus 1 (MNB Sederhana)*

Pada kasus 1, diberikan masalah nilai batas dengan persamaan diferensial orde kedua. Untuk menyelesaikan masalah tersebut pada Octave maupun Julia, persamaan ditransformasi menjadi sistem persamaan diferensial orde pertama, yaitu

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \frac{1}{x} y_2 - \frac{1}{x^2} y_1 + 1 \end{aligned} \tag{3.1}$$

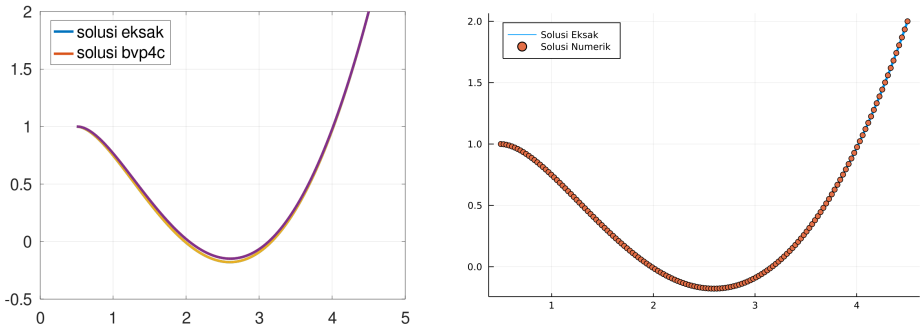
dengan nilai batas $y_1(0.5) = 1$ dan $y_1(4.5) = 2$. Adapun sintaks untuk menyelesaikan masalah tersebut pada Octave maupun Julia hampir sama, yaitu mendefinisikan fungsi yang berisi sistem persamaan diferensial dan kondisi batas yang diberikan. Perbedaannya, Octave meminta inisiasi banyak sub-interval awal sedangkan Julia meminta panjang langkah akhir. Dengan nilai tebakan solusi awal $[0, 0]$, berikut adalah sintaks Octave dan Julia untuk menyelesaikan masalah nilai batas kasus 1.

Sintaks 1: Kasus 1 pada Octave

```
function dydx = case1ode(x,y)
    dydx = [ y(2)
             1+(1/x)*y(2)-(1/(x^2))*y(1)];
endfunction
function res = case1bc(ya,yb)
    res = [ ya(1)-1
            yb(1)-2];
endfunction
Nint = 9
solinit.x = linspace(0.5,4.5,Nint+1);
solinit.y = repmat([0,0]',1,Nint+1);
sol = bvp4c(@case1ode,@case1bc,solinit)
```

Sintaks 2: Kasus 1 pada Julia

```
function case1!(du,u,p,x)
    y = u[1]
    dy = u[2]
    du[1] = dy
    du[2] = (1/x)*dy - (1/x^2)*y + 1
end
function bc1!(residual, u, p, t)
    residual[1] = u[1][1] - 1
    residual[2] = u[end][1] - 2
end
tspan = (0.5,4.5)
init = [0,0]
bvp1 = BVProblem(case1!, bc1!, init, tspan)
sol1 = solve(bvp1, GeneralMIRK4(),dt = (tspan[2]-tspan[1])/144);
```



(a) (b)

Gambar 1: Solusi numerik dan perbandingannya terhadap solusi eksak masalah nilai batas pada kasus 1 menggunakan (a) Octave dan (b) Julia

Karena Octave memerlukan inisiasi untuk sub-interval awal, simulasi numerik dikerjakan pada Octave terlebih dulu. Dari 9 sub-interval awal, proses simulasi pada Octave menghasilkan sebanyak 144 sub-interval untuk memenuhi estimasi relatif *error* yaitu 10^{-3}. Waktu rata-rata dari 5 kali percobaan pada Octave yaitu 1288.303 detik. Dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = (4.5-0.5)/144$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia hanya 2.149 detik. Gambar 1 menunjukkan solusi yang dihasilkan oleh Octave dan Julia untuk menyelesaikan kasus 1.

Dengan diketahuinya solusi eksak, relatif *error* dari solusi Octave dan Julia dapat dihitung, yaitu 0.0019152 dan 2.38×10^{-9}. Dengan membandingkan waktu komputasi dan akurasi dari solusi Octave dan Julia pada kasus 1 ini, Julia jauh lebih akurat dan efisien dibandingkan Octave dalam menyelesaikan masalah nilai batas sederhana pada kasus 1.

3.2. *Kasus 2 (MNB Peubah Banyak)*

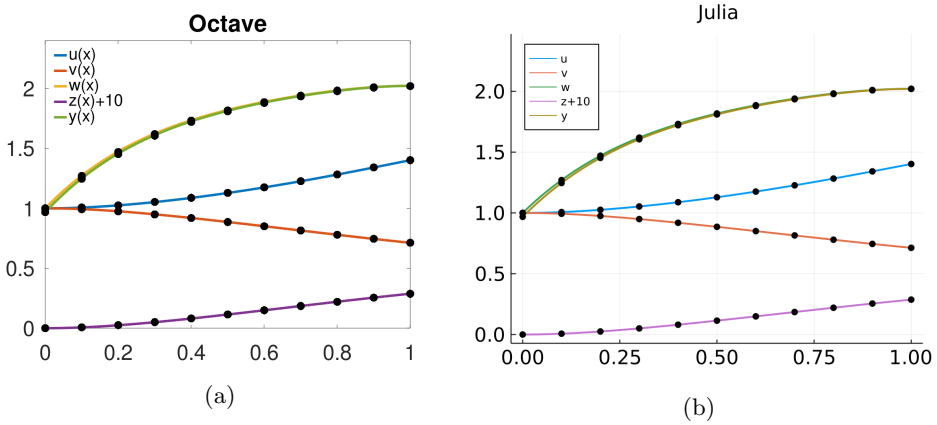
Pada kasus 2, masalah nilai batas yang diberikan sudah dalam bentuk sistem persamaan diferensial. Dengan cara yang sama seperti pada kasus 1, berikut adalah sintaks penyelesaian masalah nilai batas kasus 2 pada Octave dan Julia dengan tebakan solusi awal $[1, 1, 1, -10, 0.91]$.

Sintaks 3: Kasus 2 pada Octave

```
function dydx = case2ode(x,y)
    dydx = [ 0.5*y(1)*(y(3) - y(1))/y(2)
             -0.5*(y(3) - y(1))
             (0.9 - 1000*(y(3) - y(5)) - 0.5*y(3)*(y(3) - y(1)))/y(4)
             0.5*(y(3) - y(1))
             100*(y(3) - y(5))];
end
function res = case2bc(ya,yb)
    res = [ ya(1) - 1
            ya(2) - 1
            ya(3) - 1
            ya(4) + 10
            yb(3) - yb(5)];
end
Nint = 4
x = linspace(0,1,Nint+1);
solinit.x = x;
solinit.y = repmat([1 1 1 -10 0.91]',1,Nint+1);
sol = bvp4c(@case2ode,@case2bc,solinit);
```

Sintaks 4: Kasus 2 pada Julia

```
function case2!(du,U,p,x)
    u = U[1]; v = U[2]; w = U[3]; z = U[4]; y = U[5]
    du[1] = 0.5*u*(w - u)/v
    du[2] = -0.5*(w - u)
    du[3] = (0.9 - 1000*(w - y) - 0.5*w*(w - u))/z
    du[4] = 0.5*(w - u)
    du[5] = 100*(w - y)
end
function bc2!(residual, u, p, t)
    residual[1] = u[1][1] - 1
    residual[2] = u[1][2] - 1
    residual[3] = u[1][3] - 1
    residual[4] = u[1][4] + 10
    residual[5] = u[end][5] - u[end][3]
end
tspan = (0,1)
init = [1,1,1,-10,0.91]
bvp2 = BVProblem(case2!, bc2!, init, tspan)
sol2 = solve(bvp2, GeneralMIRK4(),dt = 1/128);
```



Gambar 2: Solusi numerik dan perbandingannya terhadap solusi dari MUSN untuk masalah nilai batas pada kasus 2 menggunakan (a) Octave dan (b) Julia

Seperti sebelumnya, simulasi dikerjakan pada Octave terlebih dulu dengan banyak sub-interval awal 4. Proses simulasi pada Octave menghasilkan sub-interval akhir sebanyak 128 sub-interval untuk memenuhi estimasi relatif *error* yaitu 10^{-3} dalam waktu rata-rata dari 5 kali percobaan yaitu 1543.7 detik. Dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = 1/128$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia yaitu 9.2 detik.

Karena solusi analitik tidak diketahui, relatif *error* dari solusi yang dihasilkan oleh Octave dan Julia tidak dapat dihitung. Akan tetapi, solusi dapat dibandingkan dengan solusi yang diperoleh dari MUSN. Perbandingan solusi masalah nilai batas dari Octave dan Julia terhadap solusi MUSN dapat dilihat pada Gambar 2. Solusi yang dihasilkan Octave maupun Julia memiliki akurasi yang baik jika dibandingkan dengan solusi MUSN. Namun, Julia mampu menghasilkan solusi kasus 2 jauh lebih cepat dibandingkan Octave.

3.3. *Kasus 3 (MNB Dengan Parameter Tetap)*

Pada kasus 3 ini, diberikan suatu masalah nilai batas dengan suatu parameter yang diketahui. Seperti kasus 1, kasus 3 ini memiliki persamaan diferensial orde kedua (Persamaan 2.3). Oleh karena itu, persamaan tersebut harus diubah menjadi sistem persamaan diferensial orde pertama, yaitu

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \frac{3py_1}{(p+t^2)^2} \end{aligned} \tag{3.2}$$

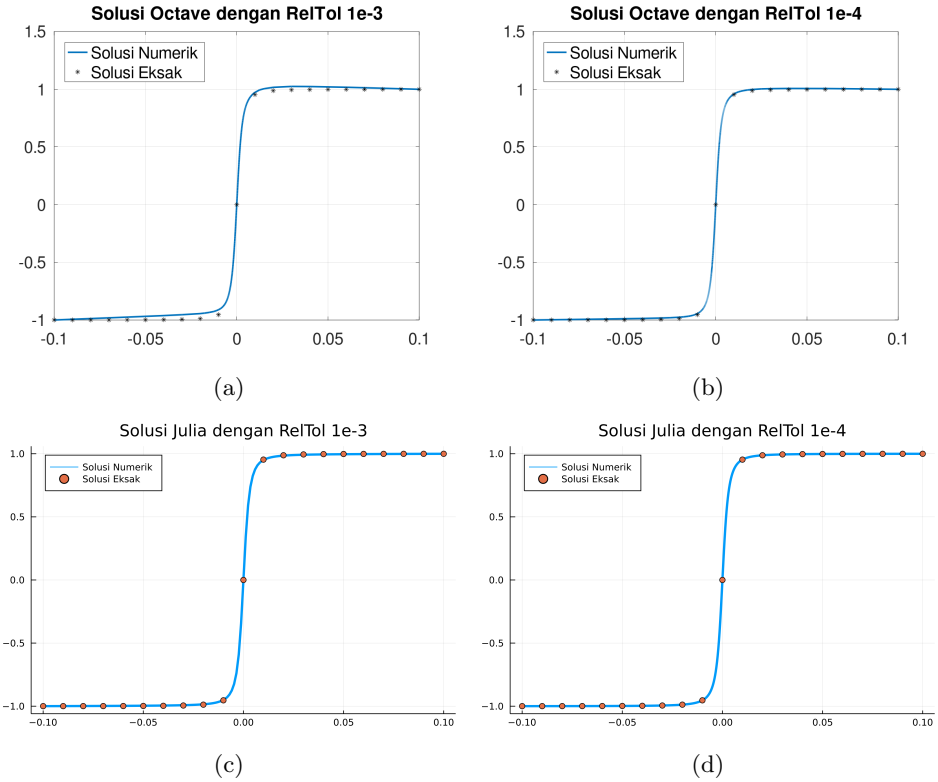
dengan nilai batas seperti pada Persamaan 2.4 serta parameter $p = 10^{-5}$. Selain itu, simulasi numerik akan dikerjakan untuk memenuhi dua toleransi relatif *error* yang berbeda yaitu 10^{-3} dan 10^{-4}. Secara *default*, Octave akan mencari solusi masalah nilai batas dengan toleransi relatif *error* yaitu 10^{-3}. Namun, nilai toleransi tersebut dapat diubah sesuai dengan kebutuhan. Berikut merupakan cara menuliskan fungsi persamaan diferensial dan kondisi batas dengan gaya berbeda untuk menyelesaikan masalah nilai batas kasus 3 dengan toleransi relatif *error* 10^{-3} dan 10^{-4}.

Sintaks 5: Kasus 3 pada Octave.

```
p = 1e-5;
case3ode = @(t,y) [ y(2)
                    -3*p*y(1)/(p+t^2)^2];
case3bc = @(ya,yb) [ ya(1) + 0.1/sqrt(p + 0.01)
                     yb(1) - 0.1/sqrt(p + 0.01) ];
Nint = 10

# untuk relatif error 1e-3
x = linspace(-0.1,0.1,Nint+1);
solinit.x = x;
solinit.y = repmat([0 10]',1,Nint+1);
sol_a = bvp4c(case3ode,case3bc,solinit);

# untuk relatif error 1e-4
x = linspace(-0.1,0.1,Nint+1);
solinit.x = x;
solinit.y = repmat([0 10]',1,Nint+1);
options.RelTol = 1e-4;
sol_b = bvp4c(case3ode,case3bc,solinit,options);
```

Sintaks 6: Kasus 3 pada Julia.

```
function case3!(du,U,p,t)
    y  = U[1]
    dy = U[2]
    du[1] = dy
    du[2] = -3*p*y/(p+t^2)^2
end
function bc3!(residual, u, p, t)
    residual[1] = u[1][1] + 0.1/sqrt(p + 0.01)
    residual[2] = u[end][1] - 0.1/sqrt(p + 0.01)
end
tspan = (-0.1,0.1)
init = [0, 10]
p = 10^-5
bvp3 = BVProblem(case3!, bc3!, init, tspan, p)

# untuk relatif error 1e-3
sol3_a = solve(bvp3b,GeneralMIRK4(),dt=(tspan[2]-tspan[1])/171)

# untuk relatif error 1e-4
sol3_b = solve(bvp3b,GeneralMIRK4(),dt=(tspan[2]-tspan[1])/630)
```



Gambar 3: Solusi masalah nilai batas pada kasus 3 menggunakan Octave dengan toleransi (a) 10^{-3} dan (b) 10^{-4} serta Julia dengan toleransi (c) 10^{-3} dan (d) 10^{-4}.

Gambar 3 menunjukkan solusi masalah nilai batas pada kasus 3 menggunakan Octave dengan toleransi 10^{-3} dan 10^{-4} serta Julia dengan toleransi 10^{-3} dan 10^{-4}. Dengan toleransi relatif *error* 10^{-3} dan 10 sub-interval awal, proses simulasi pada

Octave menghasilkan sub-interval akhir sebanyak 171 sub-interval untuk memenuhi estimasi relatif *error* yaitu 10^{-3}. Waktu yang diperlukan untuk menyelesaikan kasus tersebut yaitu 18621.2 detik. Karena waktu komputasi pada Octave sangat lama, proses simulasi hanya dijalankan sekali. Dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = (0.1 - (-0.1))/171$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia yaitu 3.2 detik. Jauh lebih cepat dibandingkan Octave. Dengan diketahuinya solusi eksak dari masalah nilai batas, relatif *error* dari solusi Octave dan Julia secara berturut turut dapat dihitung, yaitu 0.0048526 dan 0.000242. Berdasarkan waktu komputasi dan akurasi untuk kasus 3, Julia jauh lebih efisien dibandingkan Octave dengan mengurangi waktu komputasi dari 5 jam menjadi 3.2 detik. Selain itu, *error* yang dihasilkan oleh solusi Julia 20 kali lebih rendah daripada *error* solusi Octave.

Dengan toleransi relatif *error* 10^{-4} dan 10 sub-interval awal, proses simulasi pada Octave menghasilkan sub-interval akhir sebanyak 630 sub-inteval untuk memenuhi estimasi relatif *error* yaitu 10^{-4}. Waktu yang diperlukan untuk menyelesaikan kasus tersebut yaitu 187434.2 detik atau sekitar 52 jam atau lebih dari 2 hari. Dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = (0.1 - (-0.1))/630$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia yaitu 30.5 detik. Dengan diketahui solusi eksak dari masalah nilai batas, relatif *error* dari solusi Octave dan Julia secara berturut turut adalah 0.00073713 dan 7.5×10^{-7}. Berdasarkan waktu komputasi dan akurasi, Julia jauh lebih efisien dibandingkan Octave dengan mengurangi waktu komputasi yang lebih dari 2 hari menjadi 30.5 detik. Selain itu, *error* yang dihasilkan oleh solusi Julia jauh lebih rendah daripada *error* solusi Octave.

3.4. *Kasus 4 (MNB Dengan Parameter Bebas)*

Pada kasus ini, diberikan suatu masalah nilai batas dengan suatu parameter yang tidak diketahui. Baik Octave maupun Julia, keduanya tidak menyediakan penyelesaian masalah nilai batas dengan suatu parameter yang tidak diketahui nilainya seperti fungsi bvp4c pada MATLAB. Oleh karena itu, diperlukan sedikit modifikasi pada masalah nilai batas supaya nilai parameter yang tidak diketahui tersebut diperoleh. Modifikasi yang dimaksud adalah memisalkan parameter λ menjadi suatu peubah sehingga nilai turunan dari λ bernilai nol. Dengan memecah persamaan differensial orde 2 menjadi sistem persamaan diferensial orde pertama, diperoleh sistem persamaan dari kasus 4, yaitu

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= (\lambda - 2q\cos 2x)y \\ p' &= 0 \end{aligned} \tag{3.3}$$

dengan parameter $q = 5$ serta nilai batas $y'(0) = 0$, $y'(\pi) = 0$ dan $y(0) = 1$. Nilai parameter λ merupakan vektor solusi dari peubah ketiga dari masing-masing *solver*. Karena nilai turunan p bernilai nol, solusi peubah ketiga akan bernilai sama untuk semua titik x solusi yang diperoleh. Nilai parameter λ tersebut bergantung pada nilai tebakan awal yang digunakan. Beberapa penelitian sebelumnya menggunakan

$\lambda_0 = 15$ sebagai nilai tebakan awal dan hasil nilai parameter λ yang diperoleh adalah 17.0966. Dengan nilai tebakan awal untuk peubah y dan y' masing-masing yaitu 1 dan 0 serta nilai tebakan parameter, yaitu $\lambda_0 = 15$, berikut merupakan sintaks untuk menyelesaikan masalah nilai batas kasus 4 dengan estimasi toleransi relatif *error* sebesar 10^{-5} pada Octave dan Julia.

Sintaks 7: Kasus 4 pada Octave

```
function res = case4bc(ya,yb)
    res = [ ya(2)
            yb(2)
            ya(1) - 1];
endfunction
function du = case4ode(x,u)
    q = 5;
    y = u(1);dy = u(2);p = u(3);
    du = [dy; -(p - 2*q*cos(2*x))*y; 0];
endfunction
Nint = 10
solinit.x = linspace(0,pi,Nint+1);
solinit.y = repmat([1,0,15]',1,Nint+1);
options.RelTol = 1e-5;
sol = bvp4c(@case4ode,@case4bc,solinit,options)
```

Sintaks 8: Kasus 4 pada Julia

```
function case4!(du,U,pp,x)
    q = 5
    y, dy, p = U
    du = [dy; -(p - 2*q*cos(2*x))*y; 0]
end
function bc4!(residual, u, p, t)
residual[1] = u[1][2]
residual[2] = u[end][2]
residual[3] = u[1][1] - 1
end
tspan = (0,Float64(pi))
init = [1,0,15]
bvp4 = BVProblem(case4!, bc4!, init, tspan)
sol4 = solve(bvp4, GeneralMIRK4(),dt = (tspan[2]-tspan[1])/61)
```

Dengan toleransi relatif *error* 10^{-5} dan banyak sub-interval awal 10, proses simulasi pada Octave menghasilkan sub-interval akhir sebanyak 1276 sub-interval untuk memenuhi estimasi relatif *error* yaitu 10^{-5} dalam waktu yaitu 140813.1 detik atau lebih dari 39 jam. Hasil yang diperoleh adalah nilai λ konvergen menuju $\lambda = 7.4362$. Selanjutnya, dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = \pi/1276$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia dengan banyak sub-interval 1276 yaitu 304.8 detik. Dari proses tersebut, diperoleh nilai parameter λ konvergen menuju $\lambda = 17.096582$. Selain dengan toleransi relatif *error* 10^{-5}, sebelumnya simulasi telah dilakukan dengan beberapa nilai toleransi pada Octave dan Julia dengan hasil masing-masing dapat dilihat pada Tabel 1 dan 2 berikut.

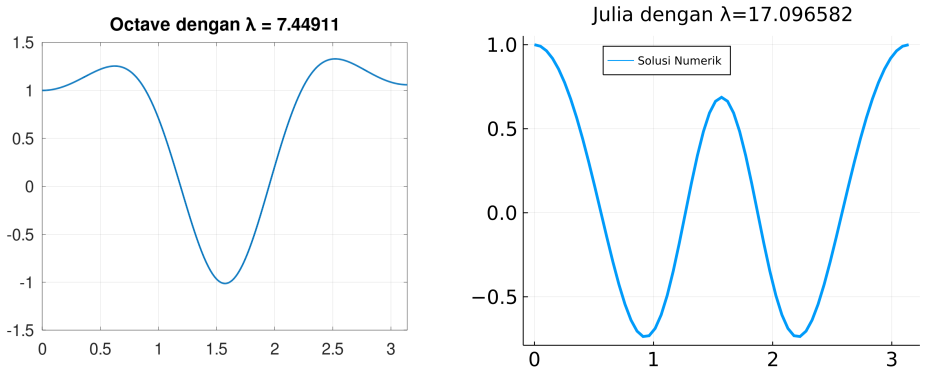
Tabel 1: Solusi Kasus 4 untuk beberapa nilai toleransi relatif *error* pada Octave.

Toleransi	Waktu komputasi (detik)	Banyak sub-interval	Parameter λ
10^{-2}	646.7	61	7.5250
10^{-3}	2115.6	153	7.4666
10^{-4}	36399.8	630	7.4520
10^{-5}	140813.1	1276	7.4362

Tabel 2: Solusi Kasus 4 untuk beberapa sub-interval pada Julia. Waktu komputasi adalah rata-rata dari 5 kali percobaan.

Banyak sub-interval	Waktu komputasi (detik)	Parameter λ
61	1.0	17.096686
153	4.2	17.096584
630	73.5	17.096582
1276	304.8	17.096582

Dari kedua percobaan, nilai parameter λ yang diperoleh dari Octave dan Julia konvergen ke dua nilai yang berbeda, masing-masing $\lambda = 7.4362$ dan $\lambda = 17.46854$. Berdasarkan nilai komputasi pada MATLAB [2], beberapa nilai parameter yang memenuhi masalah nilai batas pada kasus 4 adalah $\lambda = 17.096582$ untuk tebakan awal $\lambda_0 = 15$, $\lambda = 36.3609$ untuk tebakan awal $\lambda_0 = 5$, dan $\lambda = 7.44911$ untuk tebakan awal $\lambda_0 = 9$. Meskipun berbeda, kedua nilai parameter yang dihasilkan oleh Octave dan Julia merupakan nilai parameter yang memenuhi masalah nilai batas pada Kasus 4. Namun, nilai parameter yang dihasilkan oleh Octave memiliki akurasi yang rendah meskipun menggunakan toleransi relatif *error* 10^{-5}, yaitu nilai parameter memiliki *error* sekitar 0.01 terhadap solusi MATLAB. Sementara itu, nilai parameter yang dihasilkan oleh Julia memiliki nilai *error* yang jauh lebih rendah dibanding Octave. Dengan sub-interval 1276, parameter yang dihasilkan oleh Julia memiliki kemiripan paling sedikit 6 angka desimal dengan solusi MATLAB. Selain itu, waktu komputasi pada Julia jauh lebih cepat dibandingkan Octave yaitu sebesar 400-600 kali lebih cepat. Adapun solusi $y(x)$ yang diperoleh dari Octave dan Julia dengan nilai parameter λ yang bersesuaian dapat dilihat pada Gambar 4 berikut.



(a) (b)

Gambar 4: Solusi numerik masalah nilai batas kasus 4 menggunakan (a) Octave dan (b) Julia

3.5. *Kasus 5 (Masalah Kontrol Optimum)*

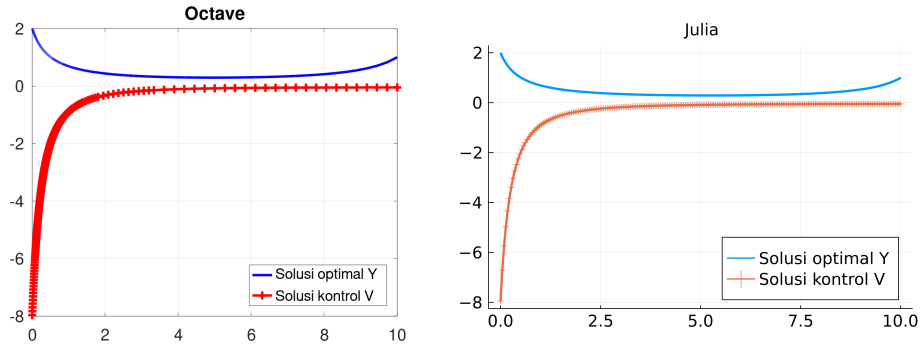
Pada kasus ini, diberikan suatu masalah kontrol optimum. Melalui prinsip maksimum Pontryagin [11], masalah kontrol optimum pada kasus 5 dapat diubah menjadi masalah nilai batas seperti Persamaan 2.8. Berikut merupakan sintaks penyelesaian masalah nilai batas kasus 5 pada Octave dan Julia.

Sintaks 9: Kasus 5 pada Octave

```
function dydx = case5ode(x,z)
    dydx = [z(1)^2-z(2)/20
            2*z(1) - 2*z(1)*z(2)];
end
function res = case5bc(ya,yb)
    res = [ ya(1)-2
            yb(1)-1];
end
Nint = 9
x = linspace(0,10,Nint+1);
solinit.x = linspace(0,10,Nint+1);
solinit.y = repmat([2,1]',1,Nint+1);
sol = bvp4c(@case5ode,@case5bc,solinit);
```

Sintaks 10: Kasus 5 pada Julia

```
function case5!(du,U,p,x)
    y,z = U
    du[1] = y^2-(z/20)
    du[2] = 2*y - 2*y*z
end
function bc5!(residual, u, p, t)
    residual[1] = u[1][1] - 2
    residual[2] = u[end][1] - 1
end
tspan = (0,10)
init = [2,1]
bvp5 = BVProblem(case5!, bc5!, init, tspan)
sol5 = solve(bvp5, GeneralMIRK4(),dt = (tspan[2]-tspan[1])/225);
```



(a) (b)

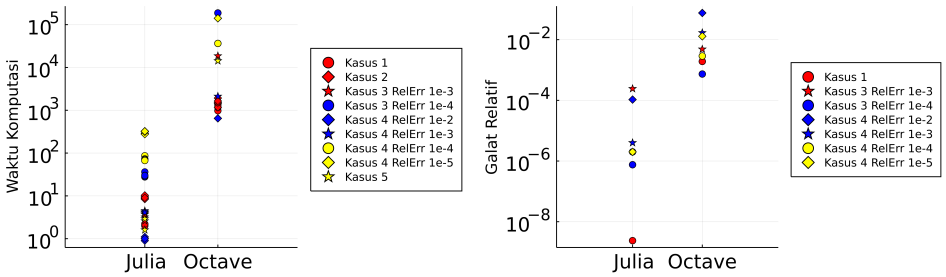
Gambar 5: Solusi numerik masalah nilai batas kasus 5 menggunakan (a) Octave dan (b) Julia

Seperti sebelumnya, simulasi dikerjakan pada Octave terlebih dulu dengan banyak sub-interval awal 9. Proses simulasi pada Octave menghasilkan sub-interval akhir sebanyak 225 sub-interval untuk memenuhi estimasi relatif *error* yaitu 10^{-3}

dalam waktu 14263.5 detik. Dengan kondisi yang sama, simulasi dikerjakan pada Julia dengan ukuran langkah $dt = 10/225$. Waktu rata-rata dari 5 kali percobaan simulasi pada Julia yaitu 1.9 detik. Sama seperti sebelumnya, untuk menyelesaikan masalah kontrol optimum, Julia menghasilkan solusi jauh lebih cepat dibandingkan Octave. Solusi optimum y dan kontrol v dari kasus 5 yang dihasilkan dapat dilihat pada Gambar 5. Adapun solusi yang diperoleh Octave serupa dengan solusi yang diperoleh Julia.

3.6. *Akurasi dan Efisiensi*

Pada kelima kasus yang dibahas, semua kasus menunjukkan bahwa waktu komputasi dari Julia jauh lebih cepat dibandingkan Octave untuk menyelesaikan masalah nilai batas. Selain itu, akurasi dari solusi yang dihasilkan oleh Julia lebih baik dibandingkan Octave, yang ditunjukkan oleh nilai relatif *error* terhadap solusi sebenarnya. Rangkuman waktu komputasi dan relatif *error* penyelesaian masalah nilai batas pada Octave dan Julia dapat dilihat pada Gambar 6 berikut.



(a) (b)

Gambar 6: Perbandingan waktu komputasi dan relatif *error* penyelesaian masalah nilai batas pada Octave dan Julia

Dari Gambar 6 dapat dilihat bahwa waktu komputasi Julia yang paling lama yaitu pada kasus 4 dengan estimasi *error* 10^{-5} bahkan lebih cepat daripada waktu komputasi Octave yang paling cepat yaitu pada kasus 4 dengan estimasi *error* 10^{-2}. Jika dibandingkan untuk setiap kasus, waktu komputasi Julia rata-rata sekitar 2500 kali lebih cepat dibandingkan Octave untuk menyelesaikan masalah nilai batas. Dari sisi akurasi, relatif *error* yang dihasilkan Julia juga jauh lebih kecil dibandingkan Octave. Relatif *error* Julia yang terbesar yaitu pada kasus 3 dengan estimasi *error* 10^{-3} lebih kecil daripada relatif *error* Octave yang terkecil yaitu pada kasus 3 dengan estimasi *error* 10^{-4}. Jika dibandingkan untuk setiap kasus, relatif *error* Julia rata-rata sekitar 10^5 kali lebih kecil dibandingkan Octave untuk menyelesaikan masalah nilai batas.

4. Kesimpulan

Bahasa pemrograman Julia dan Octave memiliki rutin untuk menyelesaikan masalah nilai batas dengan sintaks yang serupa menggunakan metode kolokasi.

Sedikit berbeda dengan MATLAB, rutin pada Octave maupun Julia tidak menyediakan fasilitas untuk menghitung masalah nilai batas dengan suatu parameter yang tidak diketahui. Namun, masalah semacam itu dapat diselesaikan dengan memodifikasi masalah nilai batas. Dari lima kasus yang diberikan, Julia mampu menyelesaikan masalah nilai batas dengan lebih akurat dan efisien dibanding Octave. Hal ini didasarkan pada waktu komputasi Julia yang memiliki kecepatan rata-rata sekitar 2500 kali lebih cepat dibandingkan Octave. Sementara itu dari sisi akurasi, Julia lebih akurat dengan relatif *error* rata-rata sekitar 10^5 kali lebih kecil dibandingkan Octave.

Berbeda dengan Julia yang sedang berkembang pesat, rutin penyelesaian masalah nilai batas yang tersedia pada Octave sudah tidak dilanjutkan (*discontinued*). Dengan demikian, meskipun fungsi `bvp4c` dapat digunakan pada Octave, waktu yang diperlukan cukup lama karena fungsi tersebut tidak mengalami perkembangan lebih lanjut. Jadi sebagai saran, jika diperlukan penyelesaian masalah nilai batas pada Octave, lebih baik untuk mengembangkan fungsi dengan metode lain yang lebih sederhana seperti *shooting method* [12,13], maupun *finite difference* [14,15]. Akan tetapi, jika diperlukan rutin yang sudah andal dalam menyelesaikan masalah nilai batas dengan lebih cepat, dapat digunakan rutin yang tersedia pada Julia.

5. Ucapan Terima kasih

Penulis mengucapkan terima kasih kepada Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Institut Pertanian Bogor atas segala dukungan dan fasilitas yang telah diberikan untuk menyelesaikan penelitian ini.

Daftar Pustaka

[1] Erliana, W., Garnadi, A.D., Nurdiati, S. Julianto, M.T., 2015. Penyelesaian Masalah Syarat Batas Persamaan Diferensial Biasa dalam Software R dengan Menggunakan BVPSolve. *Journal of Mathematics and Its Applications*, **14**(2): 9–18.

[2] Shampine, L.F., Kierzenka, J., Reichelt, M.W., 2000. Solving boundary value problems for ordinary differential equations in MATLAB with bvp4c. *Tutorial notes*, **2000**: 1–27

[3] Sharma, N. and Gobbert, M.K., 2010. A comparative evaluation of Matlab, Octave, FreeMat, and Scilab for research and teaching. *UMBC Faculty Collection*

[4] Joshi, A. and Lakhanpal, R., 2017. *Learning Julia: Build high-performance applications for scientific computing*. Packt Publishing Ltd.

[5] Schissler, A.G., Nguyen, H., Nguyen, T., Petereit, J. and Gardeux, V., 2014. Statistical Software. *Wiley StatsRef: Statistics Reference Online*, 1–11.

[6] Mathews, J.H. and Fink, K.D., 2004. *Numerical methods using MATLAB*. Pearson prentice hall Upper Saddle River, NJ.

[7] Ascher, U.M., Mattheij, R.M. and Russell, R.D., 1995. *Numerical solution of boundary value problems for ordinary differential equations*. SIAM.

[8] Scott, M.R. and Watts, H.A., 1977. Computational solution of linear two-point boundary value problems via orthonormalization. *SIAM Journal on Numerical Analysis*, **14**(1): 40–70.

[9] Arora, J. S., 2017. More on Numerical Methods for Unconstrained Optimum Design. Di dalam *Introduction to Optimum Design*. Chapter 11: 455509.

[10] Garnadi, A.D., Ayatullah, F., Ekastrya, D., Julianto, M., Nurdiati, S. and Erliana, W., 2015. Penyelesaian Masalah Syarat Batas Persamaan Diferensial Biasa Dalam Scilab Dengan Menggunakan Bvode. *Journal of Mathematics and Its Applications*, **14**(1): 55–68

[11] Kopp, R.E., 1962. Pontryagin maximum principle. Di dalam *Mathematics in Science and Engineering*, **5**: 255–279. Elsevier.

[12] Morrison, D.D., Riley, J.D., Zancanaro, J.F., 1962. Multiple shooting method for two-point boundary value problems. *Communications of the ACM*. **5**(12): 613-614

[13] Roberts, S.M., Shipman, J.S., 1967. Continuation in shooting methods for two-point boundary value problems. *Journal of Mathematical Analysis and Applications*. **18**(1): 45-58

[14] Fujita, H., 1956. On the Error of the Finite Difference Method in 1-dimensional Boundary Value Problems. *Journal of the Physical Society of Japan*. **11**(2): 160-169

[15] Mitchell, A.R., 1959. The influence of critical boundary conditions on finite difference solutions of two point boundary value problems. *Mathematics of Computation*. **13**(68): 252-260