Scientific Computing for Differential Equations 1

Lecture 02B - The Implicit Euler Method

John Bagterp Jørgensen

Department of Applied Mathematics and Computer Science
Technical University of Denmark

02686 Scientific Computing for Differential Equations 1
Spring 2020

Ordinary Differential Equations

Explicit Euler method with fixed step size

The initial value problem (IVP)

$$x(t_a) = x_a \tag{1a}$$

$$\dot{x}(t) = f(t, x(t)) \quad t_a \leq t \leq t_b \tag{1b}$$

can be solved using Euler's explicit method

$$t_0 = t_a, \qquad x_0 = x_a \tag{2a}$$

$$t_{k+1} = t_k + \Delta t, \qquad x_{k+1} = x_k + \Delta t f(t_k, x_k) \tag{2b}$$

with a fixed time-step, Δt, using the Matlab code

```
function [T,X] = ExplicitEulerFixedStepSize(fun,ta,tb,N,xa,varargin)

% Compute step size and allocate memory
dt = (tb-ta)/N;
nx = size(xa,1);
X = zeros(nx,N+1);
T = zeros(1,N+1);

% Eulers Explicit Method
T(:,1) = ta;
X(:,1) = xa;
for k=1:N
    f = feval(fun,T(k),X(:,k),varargin{:});
    T(:,k+1) = T(:,k) + dt;
    X(:,k+1) = X(:,k) + dt*f;
end

% Form a nice table for the result
T = T';
X = X';
```

Newton's Method

We want to find a root of the multivariate function

$$R(x) = 0 \qquad R : \mathbb{R}^n \mapsto \mathbb{R}^n \tag{3}$$

We make a first order Taylor expansion of R around $x^{[k]}$

$$R(x) \approx R(x^{[k]}) + \frac{\partial R}{\partial x}(x^{[k]})(x - x^{[k]}) = 0 \tag{4}$$

and set that first order Taylor approximation to zero.
Newton's method is

$$R(x^{[k+1]}) \approx R(x^{[k]}) + \frac{\partial R}{\partial x}(x^{[k]}) \overbrace{(x^{[k+1]} - x^{[k]})}^{\Delta x} = 0 \tag{5a}$$

$$x^{[k+1]} = x^{[k]} + \Delta x \tag{5b}$$

that can be expressed as

$$b := R(x^{[k]}) \qquad A := \frac{\partial R}{\partial x}(x^{[k]}) \tag{6a}$$

$$\text{Solve for } \Delta x\text{:} \quad A\Delta x = b \tag{6b}$$

$$x^{[k+1]} := x^{[k]} - \Delta x \tag{6c}$$

Numerical Solution of Linear Systems of Equations

$$Ax = b \tag{7}$$

- Backslash in Matlab: $x = A\backslash b$
- LU decomposition

$$PA = LU \tag{8a}$$

$$PAx = L\overbrace{Ux}^{=y} = Pb \tag{8b}$$

1. Compute $\bar{b} = Pb$
2. Solve for y: $Ly = \bar{b}$
3. Solve for x: $Ux = y$

Matlab:

```
[L,U,p] = lu(A,'vector');    % factorization
x = L\(U\b(p));              % back-substitution
```

Newton's Method - Matlab

The solution of

$$R(x) = 0 \tag{9}$$

can be accomplished by the Matlab code

```matlab
function x = NewtonsMethod(ResidualFunJac, x0, tol, maxit, varargin)

k = 0;
x = x0;
[R,dRdx] = feval(ResidualFunJac,x,varargin{:})
while( (k < maxit) & (norm(R,'inf') > tol) )
     k = k+1;
     dx = dRdx\R;
     x = x - dx;
     [R,dRdx] = feval(ResidualFunJac,x,varargin{:});
end
```

This code terminates if $\|R(x)\|_\infty \leq$ `tol` or $k \geq$ `maxit`.

The Implicit Euler's Method

Consider the initial value problem (IVP)

$$x(t_a) = x_a \tag{10a}$$

$$\dot{x}(t) = f(t, x(t)) \qquad t_a \le t \le t_b \tag{10b}$$

It has the solution

$$x_{k+1} - x_k = \int_{x_k}^{x_{k+1}} dx = \int_{t_k}^{t_{k+1}} f(t, x(t))dt \tag{11}$$

using $x_k = x(t_k)$. The integral $\int_{t_k}^{t_{k+1}} f(t, x(t))dt$ can be approximated using the right-side-evaluation. This gives

$$x_{k+1} = x_k + (t_{k+1} - t_k) f(t_{k+1}, x_{k+1}) \tag{12}$$

Let $\Delta t = t_{k+1} - t_k$. Then

$$x_{k+1} = x_k + \Delta t f(t_{k+1}, x_{k+1}) \tag{13}$$

which can be rearranged to

$$R(x_{k+1}) = x_{k+1} - \Delta t f(t_{k+1}, x_{k+1}) - x_k = 0 \tag{14}$$

The Implicit Euler's Method

Consider the initial value problem (IVP)

$$x(t_a) = x_a \tag{15a}$$
$$\dot{x}(t) = f(t, x(t)) \qquad t_a \le t \le t_b \tag{15b}$$

The finite-difference approximation

$$\frac{x_{k+1} - x_k}{\Delta t} \approx \frac{dx}{dt}(t_{k+1}) = \dot{x}(t_{k+1}) = f(t_{k+1}, x_{k+1}) \tag{16}$$

can be rearranged as

$$x_{k+1} = x_k + \Delta t f(t_{k+1}, x_{k+1}) \tag{17}$$

which is equivalent to

$$R(x_{k+1}) = x_{k+1} - \Delta t f(t_{k+1}, x_{k+1}) - x_k = 0 \tag{18}$$

Newton's method in the implicit Euler method

Nonlinear residual equation

$$R(x_{k+1}) = x_{k+1} - \Delta t f(t_{k+1}, x_{k+1}) - x_k = 0 \tag{19}$$

Jacobian

$$M = \frac{\partial R}{\partial x}(x_{k+1}) = I - \Delta t \frac{\partial f}{\partial x}(t_{k+1}, x_{k+1}) \tag{20}$$

Iterations in Newton's method

$$M \Delta x_{k+1} = R(x_{k+1}^{[l]}) \tag{21a}$$

$$x_{k+1}^{[l+1]} = x_{k+1}^{[l]} - \Delta x \tag{21b}$$

Stopping criterion: $\|R(x)\|_\infty \leq \epsilon$
Initial guess (explicit Euler):

$$x_{k+1}^{[0]} = x_k + \Delta t f(t_k, x_k) \tag{22}$$

Newton's Method in the Implicit Euler Method - Matlab

```
function x = NewtonsMethodODE(FunJac, tk, xk, dt, xinit, tol, maxit, varargin)

k = 0;
t = tk + dt;
x = xinit;
[f,J] = feval(FunJac,t,x,varargin{:})
R = x - dt*f - xk;
I = eye(size(xk));
while( (k < maxit) & (norm(R,'inf') > tol) )
    k = k+1;
    dRdx = I - dt*J;
    dx = dRdx\R;
    x = x - dx;
    [f,J] = feval(FunJac,t,x,varargin{:});
    R = x - dt*f - xk;
end
```

The Implicit Euler Method - A first crude implementation

```
function [T,X] = ImplicitEulerFixedStepSize(funJac,ta,tb,N,xa,varargin)

% Compute step size and allocate memory
dt = (tb-ta)/N;
nx = size(xa,1);
X = zeros(nx,N+1);
T = zeros(1,N+1);

tol = 1.0e-8;
maxit = 100;

% Eulers Implicit Method
T(:,1) = ta;
X(:,1) = xa;
for k=1:N
    f = feval(fun,T(k),X(:,k),varargin{:});
    T(:,k+1) = T(:,k) + dt;
    xinit = X(:,k) + dt*f;
    X(:,k+1) = NewtonsMethodODE(funJac,...
                        T(:,k), X(:,k), dt, xinit, tol, maxit, ,varargin{:}));
end

% Form a nice table for the result
T = T';
X = X';
```

Exercises

- Consider the van der Pol problem.
  1. Implement a function that returns the f and $J = \partial f / \partial x$:
     `function [f,J] = VanderPolfunjac(t,x,mu)`
  2. Solve the problem using the explicit Euler method
  3. Solve the problem using the implicit Euler method (make sure you choose a time step that is small enough for the Newton method to converge)
  4. Compare the solutions
- Consider the test equation $\dot{x} = \lambda x$ with $\lambda = -1$ and $x_0 = 1$.
- Compute the global and local error as function of the time-step for both the explicit and the implicit Euler method