Chapter 6 lab lecture

David M. Rosenberg

Committee on Neurobiology
University of Chicago

November 18, 2009

Outline

Admin

- Homework
- Office hours
- LaTeX and TeX interest

Admin

- Homework
- Office hours
- LaTeX and TeX interest

Admin

- Homework
- Office hours
- LaTeX and TeX interest

Outline

Backward Euler

Solving the implicit equation

Remember

There is no "generalized" formula for running backward-Euler. Each model will require a different function to be applied iteratively. Finding this function is *solving the implicit equation*.

- This differs from the *forward* Euler method.
- Typically solved analytically.
- The example in the text *only works for that model* ($\dot{x} = at$)
- The computational / cerebral costs of solving the implicit equation first should be considered when choosing a numerical method.

Remember

There is no "generalized" formula for running backward-Euler. Each model will require a different function to be applied iteratively. Finding this function is *solving the implicit equation*.

- This differs from the *forward* Euler method.
- Typically solved analytically.
- The example in the text *only works for that model* ($\dot{x} = at$)
- The computational / cerebral costs of solving the implicit equation first should be considered when choosing a numerical method.

Backward Euler

Solving the implicit equation

Remember

There is no "generalized" formula for running backward-Euler. Each model will require a different function to be applied iteratively. Finding this function is *solving the implicit equation*.

- This differs from the *forward* Euler method.
- Typically solved analytically.
- The example in the text *only works for that model* ($\dot{x} = at$)
- The computational / cerebral costs of solving the implicit equation first should be considered when choosing a numerical method.

Remember

There is no "generalized" formula for running backward-Euler. Each model will require a different function to be applied iteratively. Finding this function is *solving the implicit equation*.

- This differs from the *forward* Euler method.
- Typically solved analytically.
- The example in the text *only works for that model* ($\dot{x} = at$)
- The computational / cerebral costs of solving the implicit equation first should be considered when choosing a numerical method.

Example

Consider the model $\dot{x} = 5x - 3$

$$
\begin{aligned}
f(x,t) &= \dot{x} = 5x - 3 \\
x_{n+1} &= x_n + \Delta t f(x_{n+1}) \\
x_{n+1} &= x_n + \Delta t(5x_{n+1} - 3) \\
x_{n+1}(1 - 5\Delta t) &= x_n - 3\Delta t \\
x_{n+1} &= \left(\frac{x_n - 3\Delta t}{1 - 5\Delta t}\right)
\end{aligned}
$$

Example

Consider the model $\dot{x} = 5x - 3$

```
>bEulerStepFun <- function(xn, dt) {
    return( (xn - 3 * dt) / (1 - 5 * dt) );
  }
>dt <- 5;
>tRange <- c(0, 40);
>x0 <- 2
>x <- t <- seq(tRange[1], tRange[2], by=dt);
>x[1] <- x0;
>for (ii in 1:(length(t) - 1)) {
    x[ii + 1] <- bEulerStepFun(x[ii], dt);
  }
```

```
>data.frame(t=t, x=x);
   t         x
1  0 2.0000000
2  5 0.5416667
3 10 0.6024306
4 15 0.5998987
5 20 0.6000042
6 25 0.5999998
7 30 0.6000000
8 35 0.6000000
9 40 0.6000000
```

Outline

Eigenvalues/eigenvectors
Predicting long-term behavior

Stability:	$Re(\lambda_1, \lambda_2) > 0$	$Re(\lambda_1, \lambda_2) < 0$
real:	unstable node	stable node
complex:	unstable spiral	stable spiral

Stability:	$Re(\lambda_1) < 0, Re(\lambda_2) > 0$	$Re(\lambda_1 \ or \ \lambda_2) = 0$
real:	saddle point	degenerate node
complex:	N/A	center point

$$A = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-\mathbf{1}}$$

Eigenvalues/eigenvectors
Spectral decomposition

- Exponential matrix operations
- Matrix inversion
- Functional calculus

$$\begin{aligned} A^2 &= A \cdot A \\ &= Q\Lambda Q^{-1} \cdot Q\Lambda Q^{-1} \\ &= Q\Lambda^2 Q^{-1} \end{aligned}$$

By similar argument:

$$A^k = Q\Lambda^k Q^{-1}$$

Eigenvalues/eigenvectors
Spectral decomposition

- Exponential matrix operations
- Matrix inversion
- Functional calculus

Consider $\hat{A} = Q\Lambda^{-1}Q^{-1}$

$$\begin{aligned} A \cdot \hat{A} &= Q\Lambda Q^{-1} \cdot Q\Lambda^{-1}Q^{-1} \\ &= I \end{aligned}$$

Thus we have:

$$A^{-1} = Q\Lambda^{-1}Q^{-1} \tag{1}$$

Furthermore if $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$,

then $\Lambda^{-1} = \begin{bmatrix} \frac{1}{\lambda_1} & 0 \\ 0 & \frac{1}{\lambda_2} \end{bmatrix}$.

- Exponential matrix operations
- Matrix inversion
- Functional calculus

$$
\begin{aligned}
f(x) &= a + bx + cx^2 + \ldots \\
f(A) &= f(Q\Lambda Q^{-1}) \\
&= Qf(\Lambda)Q^{-1} \\
\Lambda &= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \\
f(\Lambda) &= \begin{bmatrix} f(\lambda_1) & 0 \\ 0 & f(\lambda_2) \end{bmatrix}
\end{aligned}
$$

Outline

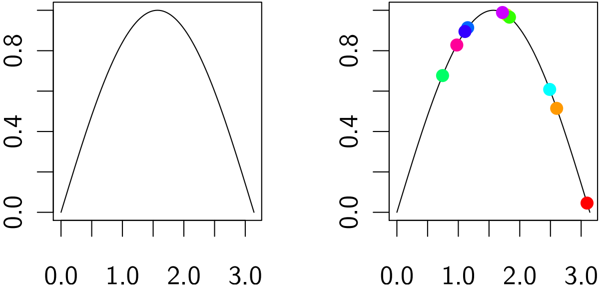
Complex functions

Argument branching

```
>pointCurve <- function(expr, from=0, to=1, xList=NULL, ...) {
   sexpr <- substitute(expr);          ## Don't worry about this
   curve(expr, from=from, to=to, ...)
   if(!is.null(xList)) {               ## This is KEY -
         Identifying whether
                                       ##   or not to do the
                                             optional
                                       ##   'thing'
     y <- numeric(length=length(xList));
     for (ii in 1:length(xList)) {
       y[ii] <- eval(call(eval(as.character(sexpr)), xList[ii]));
     }
     points(xList, y, col=rainbow(n=length(xList)), pch=19);
   }
 }
>layout(matrix(c(1, 2), nrow=1));
>par(mar=c(2,2,2,2));
>pointCurve(sin, 0, pi);                 ## One behavior without
       the arg.
>pointCurve(sin, 0, pi, xList=runif(n=10, ## And one with it.
                         min=0, max=pi));
```

Complex functions
Argument branching

```
>layout(matrix(c(1, 2), nrow=1));
>par(mar=c(2,2,2,2));
>pointCurve(sin, 0, pi);                    ## One behavior without
      the arg.
>pointCurve(sin, 0, pi, xList=runif(n=10, ## And one with it.
                        min=0, max=pi));
```



Complex functions
Sanity checks

- TODO: Example needed