Neumann and insulated boundary conditions

Introduction

• In this topic, we will
  – Review Neumann boundary conditions
  – Approximate derivatives using the formula from calculus
  – Author a solution in MATLAB using this approximation
  – Observe there are weaknesses in such an approach
  – Consider a superior approach
Dirichlet boundary conditions

- Up to this point, we’ve used Dirichlet boundary conditions:
  \[ u(a) = u_a \]
  \[ u(b) = u_b \]
- Recall that this affected the first and last equations:
  \[ p_1 u_b + q_1 u_1 + r_1 u_2 = 2 g(x_i) h^2 \]
  \[ p_{n-1} u_{n-2} + q_{n-2} u_{n-1} + r_{n-1} u_n = 2 g(x_{n-1}) h^2 \]

Neumann and insulated boundary conditions

- What happens if a boundary has an insulated or more generally a Neumann boundary condition?
  \[ u^{(1)}(a) = 0 \quad u^{(1)}(a) = u^{(1)}_a \]
  \[ u^{(1)}(b) = 0 \quad u^{(1)}(b) = u^{(1)}_b \]
Neumann and insulated boundary conditions

• Suppose we have a Neumann boundary condition at \( x = a \): 
  \[ u^{(1)}(a) = u^{(1)}_a \]
  
  – How do we eliminate the unknown \( u_0 \)?
  \[ p_1 u_0 + q_0 u_1 + r_0 u_2 = 2 g (x_1) h^2 \]
  
  – If we don’t eliminate it, we will have fewer equations than unknowns...

Approximating the derivative

• Recall we used a divided-difference approximation of the derivative:
  \[ \frac{u(a + h) - u(a)}{h} \approx u^{(1)}(a) = u^{(1)}_a \]
  
  – But \( u(a) \approx u_0 \) and \( u(a + h) \approx u_1 \):
    \[ \frac{u_1 - u_0}{h} \approx u^{(1)}_a \]
  
  – Thus, we have an equation
    \[ u_0 = u_1 - u^{(1)}_a h \]
Approximating the derivative

• Thus, if we have a left-hand Neumann condition, we have
  \[ p_1u_0 + q_1u_1 + r_1u_2 = 2g(x_1)h^2 \]
  \[ u_0 = u_1 - u_a^{(1)}h \]
  
  – Substituting the second into the first yields
  \[ p_1(u_1 - u_a^{(1)}h) + q_1u_1 + r_1u_2 = 2g(x_1)h^2 \]
  \[ (p_1 + q_1)u_1 + r_1u_2 = 2g(x_1)h^2 + p_1u_a^{(1)}h \]
  
  – Thus, entry (1,1) of the matrix needs to be updated and
  entry 1 of the target vector needs to be updated

Approximating the derivative

• Similarly, we can find on the right-hand boundary value:
  \[ p_{n-1}u_{n-2} + q_{n-1}u_{n-1} + r_{n-1}u_n = 2g(x_{n-1})h^2 \]
  \[ u_n = u_{n-1} + u_a^{(1)}h \]
  
  – This yields
  \[ p_{n-1}u_{n-2} + (q_{n-1} + r_{n-1})u_{n-1} = 2g(x_{n-1})h^2 - r_{n-1}u_a^{(1)}h \]
  
  – Thus, entry \((n-1, n-1)\) of the matrix needs to be updated and
  entry \(n-1\) of the target vector needs to be updated
Implementation

function [xs, us] = bvp( a2, a1, a0, g, x_rng, u_bndry, dirichlet, n )
    h = (x_rng(2) - x_rng(1))/n;
    p = @(x)( 2.0*a2(x) - a1(x)*h );
    q = @(x)(-4.0*a2(x) + 2.0*a0(x)*h^2);
    r = @(x)( 2.0*a2(x) + a1(x)*h );
    xs = linspace( x_rng(1) + h, x_rng(2) - h, n - 1 )';
    A = zeros( n - 1, n - 1 );
    for k = 1:(n - 1)
        A(k, k) = q(xs(k));
    end
    for k = 1:(n - 2)
        A(k + 1, k  ) = p(xs(k + 1));
        A(k,  k + 1) = r(xs(k));
    end
    v = 2.0*g(xs)*h^2;
    if dirichlet( 1 )
        v(1) = v(1) - p(xs(1))*u_bndry(1);
    else
        A(1, 1) = A(1, 1) + p(xs(1));
        v(1) = v(1) + p(xs(1))*u_bndry(1)*h;
    end
    if dirichlet( 2 )
        v(end) = v(end) - r(xs(end))*u_bndry(2);
    else
        A(end, end) = A(end, end) + r(xs(end));
        v(end) = v(end) - r(xs(end))*u_bndry(2)*h;
    end
    p_n-1*u_n-2 + (q_n-1 + r_n-1)*u_n-2 = 2*g(x_n-1)*h^2 - r_n-1*u_n-1*h

Neumann and insulated boundary conditions

( ) ( ) ( )12
1 1 1 1 2 1 1 2 ap q u ru g x h p u h+ + = +

( ) ( ) ( )12
1 2 1 1 1 1 1 2n n n n n n n bp u q r u g x h r u h− − − − − − −+ + = −
Implementation

\[ \text{us} = A \setminus v; \]
\[ \text{xs} = [x_{\text{rng}(1)}; \text{xs}; x_{\text{rng}(2)}]; \]
\[ \text{if \ dirichlet(1)} \]
\[ \quad \text{us} = [u_{\text{bdry}(1)}; \text{us}]; \]
\[ \quad \text{else} \]
\[ \quad \quad \text{us} = [\text{us}(1) - u_{\text{bdry}(1)}*h; \text{us}]; \]
\[ \qquad u_0 = u_1 - u_a^{(1)} h \]
\[ \text{end} \]
\[ \text{if \ dirichlet(2)} \]
\[ \quad \text{us} = [\text{us}; u_{\text{bdry}(2)}]; \]
\[ \quad \text{else} \]
\[ \quad \quad \text{us} = [\text{us}; \text{us(\text{end}) + u_{\text{bdry}(2)}*h}]; \]
\[ \qquad u_n = u_{n-1} + u_a^{(1)} h \]
\[ \text{end} \]
\[ \text{end} \]

Example

- Let us examine this BVP:
  \[ 13x^2 u^{(2)}(x) - 5u^{(1)}(x) + 8xu(x) = \sin(x) \]
  \[ u^{(1)}(-1) = -0.4738221764482897 \]
  \[ u(1) = 2 \]
- If \( n = 10 \), then \( h = 0.2 \), so
  \[ p_i = 2a_i(x_i) - a_i(x_i) h = 2 \cdot 13x_i^2 - (-5) \cdot 0.2 \]
  \[ q_i = -4a_i(x_i) + 2a_i(x_i) h^2 = -4 \cdot 13x_i^2 + 2 \cdot 8x \cdot 0.04 \]
  \[ r_i = 2a_i(x_i) + a_i(x_i) h = 2 \cdot 13x_i^2 + (-5) \cdot 0.2 \]
- As before,
  the \( x \)-values are \(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1\)
Example

• We can call our function with the appropriate arguments:
  \[
  \begin{align*}
  &>> [xs, us] = \text{bvp}( @(x)(x^2*13), @(x)(-5.0), @(x)(8*x), \ldots \\
  &\quad @\sin, [-1 1], [-0.4738221764482897 2], \ldots \\
  &\quad [\text{false, true}, 10) 
  \end{align*}
  \]

Example

• Solving this system of linear equations yields:

\[
\begin{bmatrix}
0.975469169822180 \\
0.896850697262553 \\
0.835215756372538 \\
0.792375715630198 \\
0.745797266197350 \\
0.792375715630198 \\
1.029613008278919 \\
1.343447491612049 \\
1.676056679222552
\end{bmatrix} \approx u(-0.2) 
\]

\[
\begin{bmatrix}
0.975469169822180 \\
0.896850697262553 \\
0.835215756372538 \\
0.792375715630198 \\
0.745797266197350 \\
0.792375715630198 \\
1.029613008278919 \\
1.343447491612049 \\
1.676056679222552
\end{bmatrix} \approx u(0.8) 
\]

\[
u(1) = 2 \quad u(-1) \approx 1.070233605111838
\]
Example

- Here is a plot of the solution and the approximations:

\[
\begin{align*}
u(-1) &= 1 \\
u(1) &= 2
\end{align*}
\]

\[
\begin{align*}
u^{(1)}(-1) &= -0.4738221764482897 \\
u^{(1)}(1) &= 2
\end{align*}
\]

Error analysis

- Problem:
  - When we substituted the derivative and second derivative, we used \(O(h^2)\) approximations
  - When we approximated the derivatives, we used \(O(h)\) approximations

\[
\frac{u(a+h) - u(a)}{h} \approx u^{(1)}(a) = u_2^{(1)}
\]

- Consequently, the overall error will now be \(O(h)\)
An O($h^2$) approximation

- Recall we used a divided-difference approximation of the derivative:

\[
\begin{align*}
-u_0 + 4u_1 - 3u_2 &= \frac{u^{(1)}_n}{2h} \\
u_{n-2} - 4u_{n-1} + 3u_n &= \frac{u^{(1)}_n}{2h}
\end{align*}
\]

\[
\begin{align*}
u_0 &= -\frac{2}{3}u^{(1)}_n + \frac{4}{3}u_1 - \frac{1}{3}u_2 \\
u_n &= \frac{2}{3}u^{(1)}_n + \frac{4}{3}u_{n-1} - \frac{1}{3}u_{n-2}
\end{align*}
\]

\[
p_0u_0 + q_0u_1 + r_0u_2 = 2g(x_1)h^2 \\
p_{n-1}u_{n-2} + q_{n-1}u_{n-1} + r_{n-1}u_n = 2g(x_{n-1})h^2
\]

\[
\begin{align*}
\left(q_1 + \frac{4}{3}p_1\right)u_1 + \left(r_1 - \frac{1}{3}p_1\right)u_2 &= 2g(x_1)h^2 + \frac{2}{3}p_{0}u^{(1)}_n \\
\left(p_{n-1} - \frac{1}{3}r_{n-1}\right)u_{n-2} + \left(q_{n-1} + \frac{4}{3}r_{n-1}\right)u_{n-1} &= 2g(x_{n-1})h^2 + \frac{2}{3}r_{n-1}u^{(1)}_n
\end{align*}
\]

Implementation

```matlab
function [xs, us] = bvp( a2, a1, a0, g, x_rng, u_bndry, dirichlet, n )
    h = (x_rng(2) - x_rng(1))/n;

    p = @(x)( 2.0*a2(x) - a1(x)*h );
    q = @(x)(-4.0*a2(x) + 2.0*a0(x)*h^2);
    r = @(x)( 2.0*a2(x) + a1(x)*h );

    xs = linspace( x_rng(1) + h, x_rng(2) - h, n - 1 );

    A = zeros( n - 1, n - 1 );
    for k = 1:(n - 1)
        A(k, k) = q(xs(k));
    end

    for k = 1:(n - 2)
        A(k + 1, k    ) = p(xs(k + 1));
        A(k,     k + 1) = r(xs(k));
    end
```
Implementation

\[ v = 2.0 \cdot g \cdot (xs) \cdot h^2; \]

if dirichlet(1)
\[ v(1) = v(1) - p(xs(1)) \cdot u_{bndry}(1); \]
else
\[
\begin{align*}
A(1, 1) &= A(1, 1) + (4.0/3.0) \cdot p(xs(1)); \\
A(1, 2) &= A(1, 2) - (1.0/3.0) \cdot p(xs(1)); \\
v(1) &= v(1) + (2.0/3.0) \cdot p(xs(1)) \cdot u_{bndry}(1) \cdot h;
\end{align*}
\]
end

if dirichlet(2)
\[ v(end) = v(end) - r(xs(end)) \cdot u_{bndry}(2); \]
else
\[
\begin{align*}
A(end, end-1) &= A(end, end-1) - (1.0/3.0) \cdot r(xs(end)); \\
A(end, end) &= A(end, end) + (4.0/3.0) \cdot r(xs(end)); \\
v(end) &= v(end) - (2.0/3.0) \cdot r(xs(end)) \cdot u_{bndry}(2) \cdot h;
\end{align*}
\]
end

us = \( A \backslash v; \)
xs = \([x_{rng}(1); xs; x_{rng}(2)];\)

if dirichlet(1)
\[ us = [u_{bndry}(1); us]; \]
else
\[
\begin{align*}
us &= [(\cdot1.0/3.0) \cdot us(2) + (4.0/3.0) \cdot us(1) \ldots \\
    - (2.0/3.0) \cdot u_{bndry}(1) \cdot h; us];
\end{align*}
\]
end

if dirichlet(2)
\[ us = [us; u_{bndry}(2)]; \]
else
\[
\begin{align*}
us &= [us; (-\cdot1.0/3.0) \cdot us(end-1) + (4.0/3.0) \cdot us(end) \ldots \\
    + (2.0/3.0) \cdot u_{bndry}(2) \cdot h];
\end{align*}
\]
end
Example

• Solving this system of linear equations yields:

\[
\begin{bmatrix}
0.908106526012137 \\
0.835681556241632 \\
0.778912134340048 \\
0.739455748895171 \\
0.696099826080860 \\
0.739455748895171 \\
0.981688077550621 \\
1.307561538118296 \\
1.656738191074670 \\
\end{bmatrix}
\]

\[u = \begin{bmatrix}
u(1) = 2 \\
u(-1) \approx 0.995424472795410
\end{bmatrix}\]

Example

• The O(h) approximation is on the left, the O(h^2) approximation is on the right
Insulated boundary conditions

- Recall that insulated boundary conditions are when the derivatives are zero
  - Consequently, only the matrix is modified, as the change to the target vector is zero

\[
\begin{align*}
  \left( q_1 + \frac{4}{3} p_1 \right) u_1 + \left( r_1 - \frac{1}{3} p_i \right) u_2 &= 2 g (x_i) h^2 + \frac{2}{3} p_i n_i^0 h \\
  \left( p_{n-1} - \frac{1}{3} r_{n-1} \right) u_{n-2} + \left( q_{n-1} + \frac{4}{3} r_{n-1} \right) u_{n-1} &= 2 g (x_{n-1}) h^2 - \frac{2}{3} r_{n-1} n_{b}^0 h
\end{align*}
\]

Summary

- Following this topic, you now
  - Understand better what Neumann conditions are
  - Understand that better approximations cannot compensate for poorer approximations
  - Know how to approximation a BVP with Neumann conditions
  - Have gone through an implementation in MATLAB
  - Have seen an example
References


Acknowledgments

None so far.
These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using Consolas. Mathematical equations are prepared in MathType by Design Science, Inc. Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see https://www.rbg.ca/ for more information.

Disclaimer

These slides are provided for the ECE 204 Numerical methods course taught at the University of Waterloo. The material in it reflects the author’s best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.