(b)

Use Euler’s method (forward/explicit)

LHS:

(forward difference formula)

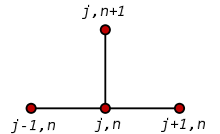
RHS:

(centered three-point formula)

Ignoring error terms and equating the two:

Simplifying:

Given initial values , boundaries , straightforward to solve for .



Computing using matrices (NOT solving linear system):

Use Euler’s method (backward/implicit):

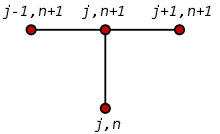
LHS:

(backward difference formula)

RHS:

(centered three-point formula)

Ignoring error terms and equating the two:



Given initial values , boundaries , construct linear system to solve for .

Simplify above equation:

Construct linear system accordingly, for each j:

where is number of partitions.

Initial values: when , . So we should loop starting from j=1. Boundary values: , so we always have .

Use Crank-Nicolson:

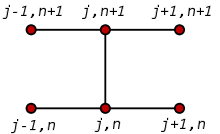
LHS:

(forward difference formula)

RHS:

(centered three-point formula applied twice)

Ignoring error terms and equating the two:



Simplifying:

Construct linear system accordingly, for each j:

(c)

Initial condition: except boundaries. Boundaries: . ?

Use Euler’s method (forward/explicit)

LHS:

(forward difference formula)

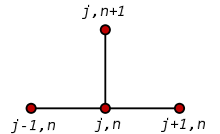
RHS:

(centered three-point formula)

Ignoring error terms and equating the two:

Simplifying:

Given except boundaries and boundaries , straightforward to solve for .



Computing using matrices (NOT solving linear system):+

Use Euler’s method (backward/implicit):

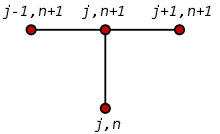
LHS:

(backward difference formula)

RHS:

(centered three-point formula)

Ignoring error terms and equating the two:



Simplifying:

Given except boundaries and boundaries , construct linear system, for each j for each k:

=

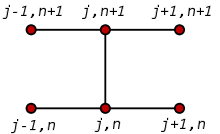
Use Crank-Nicolson:

LHS:

(forward difference formula)

RHS：

Ignoring error terms and equating the two:



Simplifying:

Construct linear system accordingly, for each j for each k: