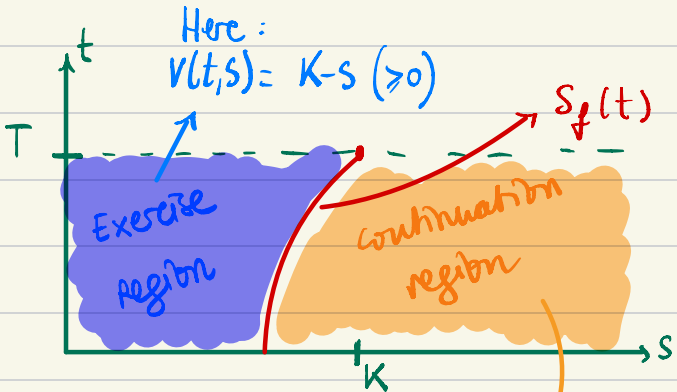
That's why, in the PDE, the condition on S is:

"$\forall S \in (S_f(t), +\infty)$".

So what we have is:



Here, V satisfies the PDE:

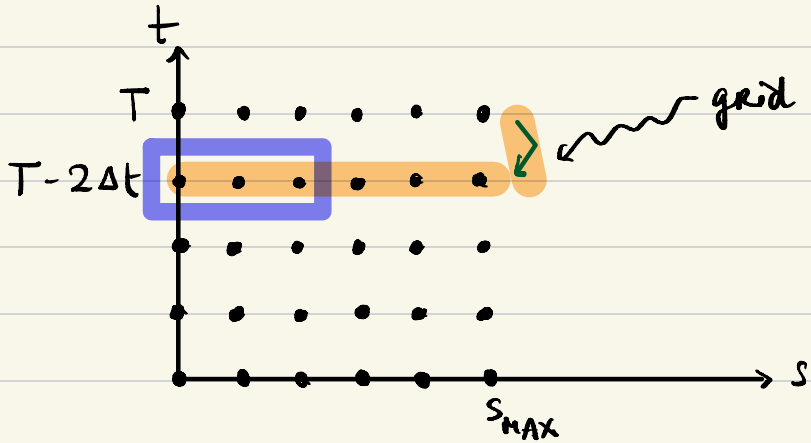
$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0$$

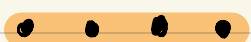
⚠ PROBLEM: we have 2 unknowns.

$V(t,S)$.
"value of the derivative"

$S_f(t)$
"Exercise boundary"

Algorithmic point of view :

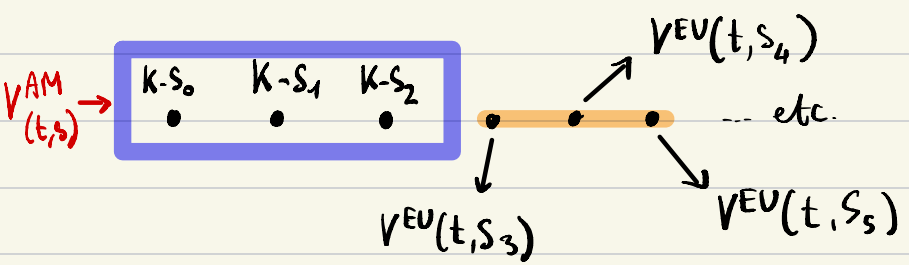


1) Solve Linear System : we find the values in yellow  ; Backward-in-time procedure.

2) Set $V(t,S) \geq (K-S)^+$

exercise value (in blue)

$V^{EU} < (K-S)^+ \longrightarrow$ replace by $(K-S)^+$.



Q1: $V^{AM} \geq (K-S)^+$? YES: when smaller, we replaced it by $(K-S)^+$.

Q2: In 

is it true that the PDE is solved ?

$$\begin{cases} S_3 : \text{NO} \\ S_4 : \text{YES} \\ S_5 : \text{YES} \end{cases}$$

← This is the problem.

$$V^{AM} = \begin{bmatrix} K - S_0 \\ K - S_1 \\ K - S_2 \\ V^{EU}(t, S_3) \\ V^{EU}(t, S_4) \\ \vdots \end{bmatrix}$$

$$\frac{\partial V^{AM/EU}}{\partial S}(t, S_3) = \frac{V^{EU}(t, S_4) - \cancel{V^{EU}(t, S_2)}^{(K-S_2)}}{2\Delta S}$$

When solving the PDE, we used this kind of approximation.

↓

So we have to use an iterative algorithm to solve the linear system.

we are going to use **SOR Algorithm**:

we have to solve the linear system:

$$\underline{\underline{A}}\,\underline{x} = \underline{b} \quad , \quad \underline{\underline{A}} \in \mathbb{R}^{n\times n} \; ; \; \underline{x}, \underline{b} \in \mathbb{R}^{n\times 1}.$$

$\underline{x}^{(0)} \in \mathbb{R}^{n\times 1}$: GUESS SOLUTION (we start from 7).

$w \in (0,2)$

For $k = 1 :$ MAXITER

  For $i = 1 : n$ % n: size of the vector.

$$y = \frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k-1)} \right) \quad (*)$$

$$x_i^{(k)} = x_i^{(k-1)} + w \left(y - x_i^{(k-1)} \right)$$

  end

  if $\| x_i^{(k)} - x_i^{(k-1)} \| < tol$

    break

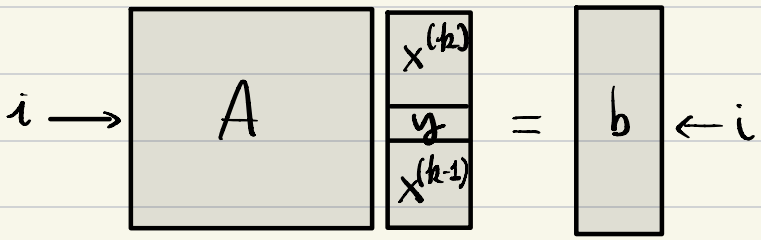
  end

end

This is the SOR algorithm.

$$* \Leftrightarrow \sum_{j=1}^{i-1} A_{ij} x_j^{(k)} + A_{ii} y + \sum_{j=i+1}^{n} A_{ij} x_j^{(k-1)} = b_i$$

$$\Leftrightarrow$$



Let's code it on MATLAB:

→ see the folder PDE_American.

1) File Linear_System.m:

→ Solve a linear system.

1.1)
```
A =
b =
x1 = A\b
```
} classical way.

1.2)
```
Omega = 1.5 ; maxiter = 500; tol = 1e-5;
X_OLD = zeros(size(b))
```
} GUESS

```
for k ...
```

→ implementation of what we saw above.

```
for i ...
    y = (b(i) - sum( ... )
    x_new(i) = omega*y + ...
end
if norm(x_new - x_old, 'inf') < tol
    break
else
    x_old = x_new;
end
end
```

→ So now we can take back one of our code from last time: EU_ImplicitEuler.m and instead of solving via "\" we use the SOR algorithm.

↳ we put the previous algo in a fct that we can call with args (A, rhs)* from our implicit Euler program.

* we can also add "x_old".

⚠ Problem: SOR is very slow.

But we can improve it by recalling that our matrix A is tridiagonal in our case.

↓

Faster, but still slower than "\".

2) Am_PUT_ImplicitEuler.m:

We need the "PSOR_Algorithm_tridiag" function that takes as arguments:

mat A, rhs, V, $\max(K - S_0 * \exp(x), 0)$.

At each time, it ensures that $V(t,S) \geq (K-S)^+$, when computing x_new:

in MATLAB: "x_new(i) = max(payoff(i), (1-w) * x_old(i) + w * y)".