Resumo da apresentação

- Parte I
  - A equação do calor
  - Comportamento do erro em métodos de relaxação
- Parte II
  - Método *Multigrid*
  - Aplicações

A equação do calor (discretização)

Utilizando o método de Euler Implícito [1] e diferenças finitas de segunda ordem no espaço temos:

$$\frac{v_{ij}^{n+1} - v_{ij}^{n}}{\Delta t} = c\left(\frac{v_{i-1,j}^{n+1} - 2v_{i,j}^{n+1} + v_{i+1,j}^{n+1}}{\Delta x^2} + \frac{v_{i,j-1}^{n+1} - 2v_{i,j}^{n+1} + v_{i,j+1}^{n+1}}{\Delta y^2}\right) + f_{ij}^{n+1} \quad (1)$$

A equação do calor (discretização)

Utilizando o método de Euler Implícito [1] e diferenças finitas de segunda ordem no espaço temos:

$$\frac{v_{ij}^{n+1} - v_{ij}^{n}}{\Delta t} = c\left(\frac{v_{i-1,j}^{n+1} - 2v_{i,j}^{n+1} + v_{i+1,j}^{n+1}}{\Delta x^2} + \frac{v_{i,j-1}^{n+1} - 2v_{i,j}^{n+1} + v_{i,j+1}^{n+1}}{\Delta y^2}\right) + f_{ij}^{n+1} \quad (1)$$

$$Av = b \quad (1)$$

A é simétrica e tridiagonal em blocos

O método é incondicionalmente estável

Método *Multigrid*

- O que é?
  - Uma metodologia para resolver de forma eficiente sistemas lineares

Método *Multigrid*

- O que é?
  - Uma metodologia para resolver de forma eficiente sistemas lineares

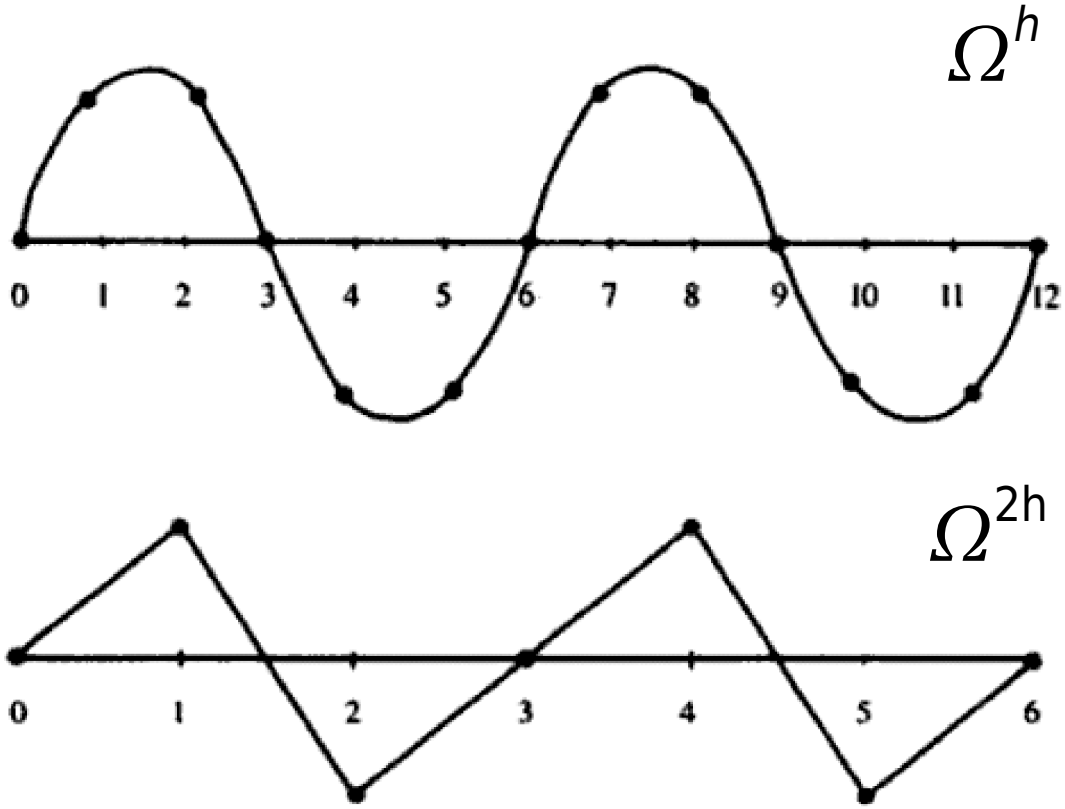
- Por que funciona?
  - Consegue “suavizar” todos os componentes do erro

Método *Multigrid*

- Componentes de baixa frequência em malhas refinadas tornam-se de alta frequência em malhas “grosseiras”

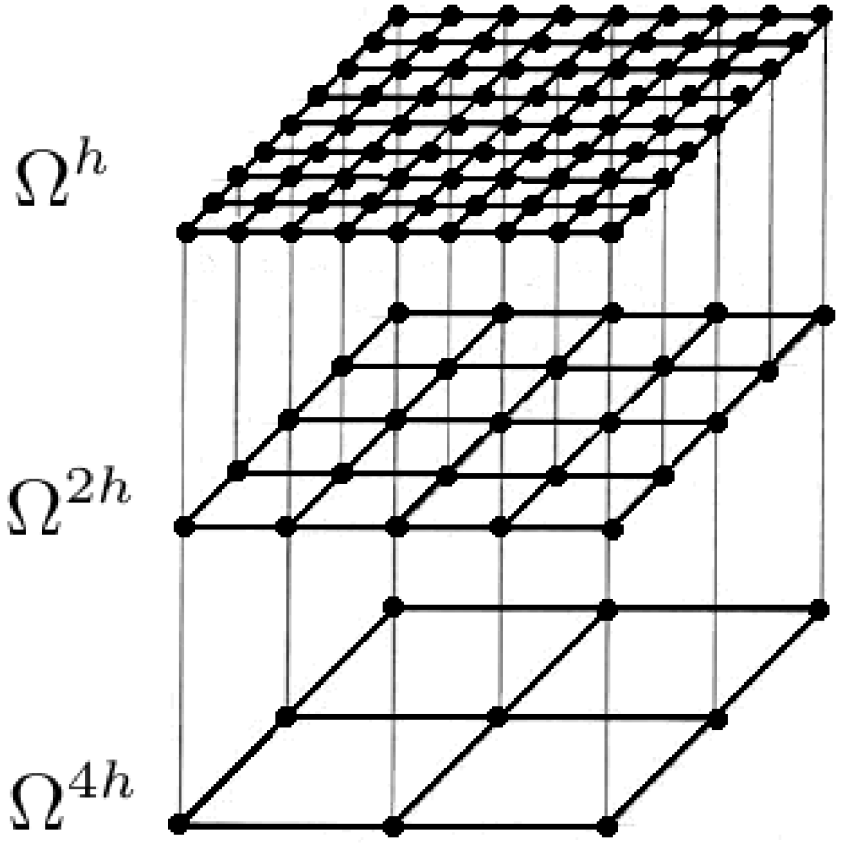
Método *Multigrid*

- Componentes de baixa frequência em malhas refinadas tornam-se de alta frequência em malhas “grosseiras”



Método *Multigrid*

- Exemplo de malhas espaciais sucessivamente mais grosseiras:



Método *Multigrid*

- Seja ***v*** uma aproximação para a solução exata **u** do sistema (1). O erro cometido entre a aproximação ***v*** e a solução exata ***u*** é:

$$C = U - V \quad (2)$$

que chamaremos **correção**.

Método *Multigrid*

- Seja ***v*** uma aproximação para a solução exata **u** do sistema (1). O erro cometido entre a aproximação ***v*** e a solução exata ***u*** é:

$$c = u - v \quad (2)$$

que chamaremos **correção**.
Podemos saber quão bem ***v*** aproxima ***u*** por meio de:

$$r = b - Av \quad (3)$$

que chamaremos **resíduo.**

Método *Multigrid*

- De (1), (2) e (3) pode-se facilmente chegar a:

$$Ac = r \quad (4)$$

que é a chamada equação do resíduo.

Método *Multigrid*

- De (1), (2) e (3) pode-se facilmente chegar a:

$$Ac = r \quad (4)$$

que é a chamada equação do resíduo.

Pode-se mostrar que iterar em (1) com chute inicial qualquer é o mesmo que iterar em (4) com chute inicial c = **0** [2].

Método *Multigrid*

- Esquema de correção:
  Considere duas malhas Ω^h e Ω^{2h}
  - Relaxar em $Av=b$ em Ω^h com chute inicial v_1^h
  - Calcular o residuo em Ω^h
  - Relaxar em $Ac=r$ em Ω^{2h} com chute inicial $c^{2h}=0$
  - Utilizar c^{2h} para melhorar a aproximação v_1^h em Ω^h

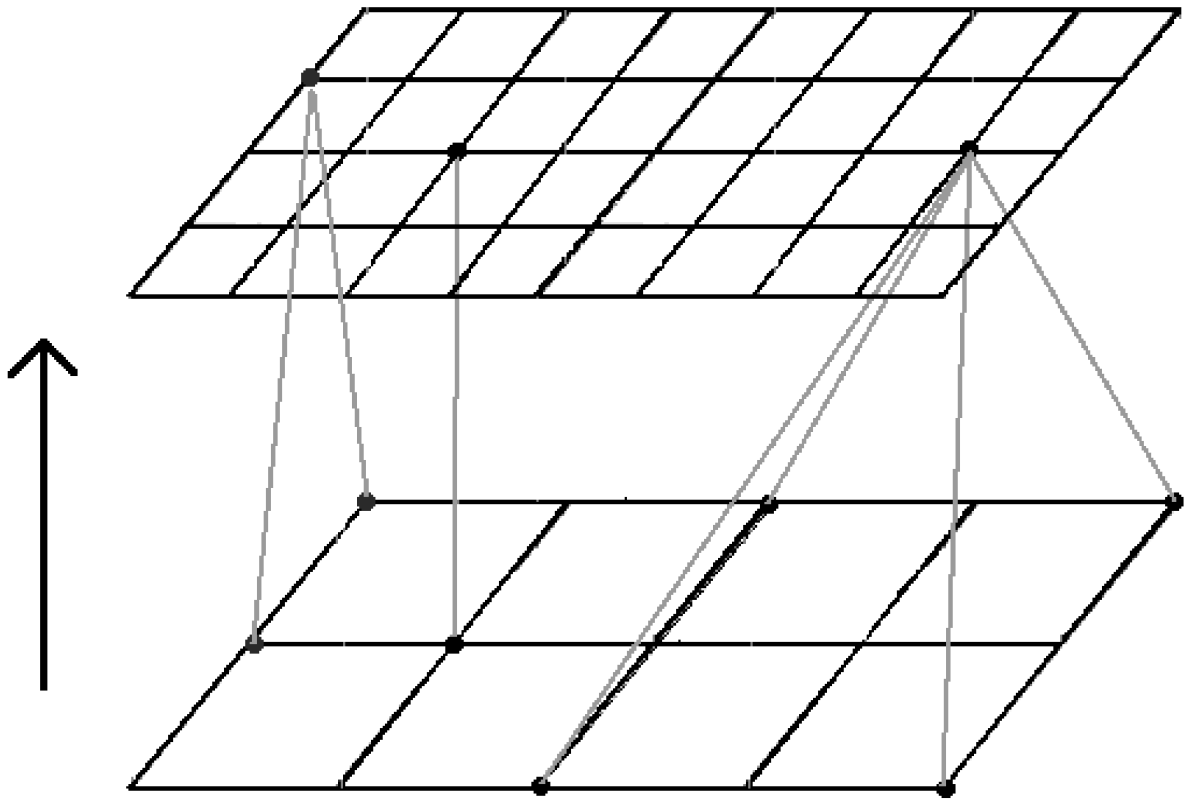
Método *Multigrid*

- Operadores
  - Prolongamento ($\Omega^{2h} \rightarrow \Omega^{h}$)
  - Restrição ($\Omega^{h} \rightarrow \Omega^{2h}$)

Método *Multigrid*

- Operador de prolongamento

  Interpolação Bilinear



Método *Multigrid*

$$v^h_{2i,2j} = v^{2h}_{ij},$$

$$v^h_{2i+1,2j} = \frac{1}{2}(v^{2h}_{ij} + v^{2h}_{i+1,j}),$$

$$v^h_{2i,2j+1} = \frac{1}{2}(v^{2h}_{ij} + v^{2h}_{i,j+1}),$$

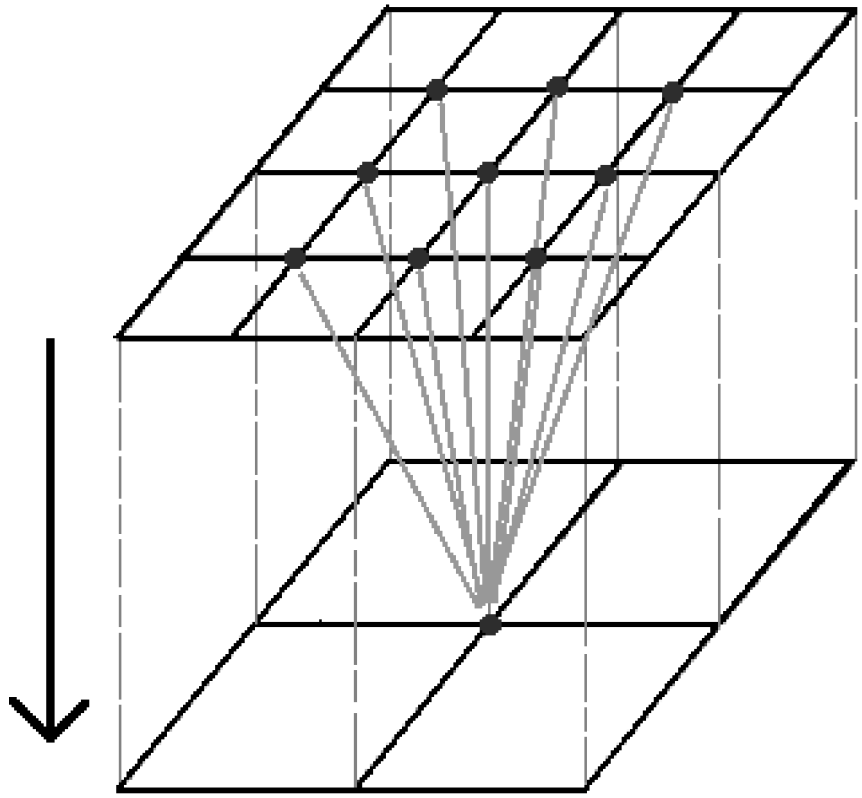
$$v^h_{2i+1,2j+1} = \frac{1}{4}(v^{2h}_{ij} + v^{2h}_{i+1,j} + v^{2h}_{i,j+1} + v^{2h}_{i+1,j+1}),$$

$$0 \leq i, j \leq \frac{n}{2} - 1.$$

Método *Multigrid*

- Operador de restrição

  Full Weighting

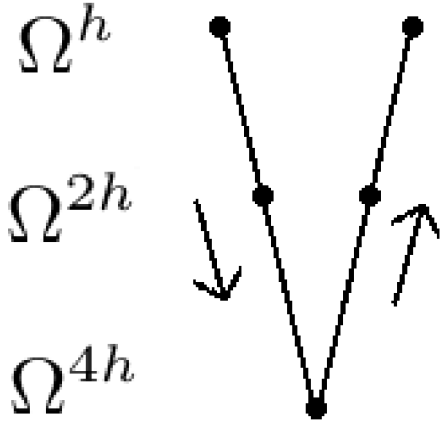


Método *Multigrid*

$$v_{ij}^{2h} = \frac{1}{16}[v_{2i-1,2j-1}^{h} + v_{2i-1,2j+1}^{h} + v_{2i+1,2j-1}^{h} + v_{2i+1,2j+1}^{h} + 2(v_{2i,2j-1}^{h} + v_{2i,2j+1}^{h} + v_{2i-1,2j}^{h} + v_{2i+1,2j}^{h}) + 4v_{2i,2j}^{h}], \quad 1 \leq i, j \leq \frac{n}{2} - 1,$$

Método *Multigrid*

- Ciclo-V
  - Extensão do esquema de correção.



```
template<class T>
class Matriz {
	public:
		Matriz(unsigned _nx, unsigned _ny):nx(_nx),ny(_ny),data_(new T[_nx*_ny]){};
		Matriz(){};
		~Matriz(){delete[] data_;};
		T& operator() (unsigned _nx, unsigned _ny);
		T  operator() (unsigned _nx, unsigned _ny) const;
		Matriz& operator= (const Matriz& m);
		Matriz& operator= (int num);
		bool operator== (const Matriz& m);
		bool operator== (int num);
		Matriz& operator+= (const Matriz& m);
		unsigned int nx, ny;
		void create(unsigned _nx, unsigned _ny){nx=_nx; ny=_ny;data_= new T[_nx*_ny];}
		T max();
		T min();

	private:
		T* data_;
};
```

```
struct par{
	double x,y;
};

class Grid{
	private:
		unsigned int	nx,	ny; //Numero de divisoes em X e Y respectivamente
		double		hx,	hy; //Distancia do espacamento entre 2 variaveis
		bool finest;
	public:
		Matriz<double>	v; //variavel
		Matriz<double>	v_old; //apenas para a malha mais fina
		Matriz<double>	c; //correcao
		Matriz<double>	r; //residuo
		Matriz<par>	X; //valores (xi, yi) em cada ponto da malha

		const unsigned int get_nx(){return nx;}
		const unsigned int get_ny(){return ny;}
		const double get_hx(){return hx;}
		const double get_hy(){return hy;}
		const bool is_finest(){return finest;}
		Grid(int _nx, int _ny);
		~Grid();
		void create_fine_grid();
};
```

```
do{
	tempo += ht;
	multi.atualizar_cc(tempo+ht); //atualiza a condicao de contorno na malha mais fina

	multi->Grids[0]->v_old = multi->Grids[0]->v;

	do //Executa V-Ciclos ate que o maximo residuo na malha fina caia da ordem de ht
	{
		multi.V_cycle(1, 1, tempo);
	} while( fabs(multi.residuo_max(multi->Grids[0]) )  > tolerancia );

	if( ht > (tempo_final - tempo) )  // ht = min( ht , tempo_final - tempo_atual )
			ht = tempo_final – tempo;
			multi->ht = ht;
	}
	while( fabs( tempo_final - tempo ) > 10,0e-14 );
}
```

```
void Multigrid::V_ciclo(int v1, int v2 , double tempo){
	unsigned int i;

	Relaxar_Av(Grids[0] , v1 , tempo);
	calcular_residuo_fine(tempo);

	for( i=0 ; i < Grids.size()-1 ; i++){ //descendo para a malha mais grosseira
		Grids[i]->c = 0;
		Relaxar_Ac(Grids[i], v1); //relaxa em Ac = r , com c = 0
		calcular_residuo_grid(Grids[i]);
		Restricao_fw(Grids[i+1] , Grids[i]); // passa o residuo da malha fina para a grosseira
	}

	Grids[Grids.size()-1]->c = 0;
	Relaxar_Ac(Grids[Grids.size()-1] , 3 );

	for(i = Grids.size() -1 ; i > 0 ; i--){ //subindo para a malha mais fina
		Prolonga_corrige(Grids[i] , Grids[i-1]); // passa a correcao da malha grosseira para a
fina
		Relaxar_Ac(Grids[i-1], v2); // relaxa em Ac = r , com a correcao nova
	}
	corrige_aproximacao(); // v = v + c	corrige a aproximacao inicial da malha mais fina
}
```

Resumo da apresentação

- Parte I
  - A equação do calor
  - Comportamento do erro em métodos de relaxação
- Parte II
  - Método *Multigrid*
  - Aplicações

Aplicações

- Estratégia da “solução manufaturada”

$$\frac{\partial u}{\partial t} = c\nabla^2 u + f$$

Aplicações

- Estratégia da “solução manufaturada”

$$\frac{\partial u}{\partial t} = c\nabla^2 u + f \;\rightarrow\; f = \frac{\partial u}{\partial t} - c\nabla^2 u$$

Aplicações

- Estratégia da “solução manufaturada”

$$\frac{\partial u}{\partial t} = c\nabla^2 u + f \;\rightarrow\; f = \frac{\partial u}{\partial t} - c\nabla^2 u$$

$$u(x, y, t) = \mathrm{e}^{-t}\, sen(2\pi x + 2\pi y)$$

Aplicações

- Estratégia da “solução manufaturada”

$$\frac{\partial u}{\partial t} = c\nabla^2 u + f \;\rightarrow\; f = \frac{\partial u}{\partial t} - c\nabla^2 u$$

$$u(x, y, t) = \mathrm{e}^{-t}\, sen(2\pi x + 2\pi y)$$

$$v(x, y, t_0) = u(x, y, t_0) \text{ em } \Omega$$

$$v(x, y, t) = u(x, y, t + \Delta t) \text{ em } \partial\Omega$$

Aplicações

- Resolvendo a equação em malhas sucessivamente mais finas (com $n, 2n, \ldots, 2^k n$ divisões) observa-se de acordo com a expansão assintótica do erro global [3] que

$$\frac{\|e\|_\infty^k}{\|e\|_\infty^{k+1}}$$

deve se aproximar de 2^p quando ***k*** cresce, onde ***p*** é a ordem do método e $\|e\|_\infty^k$ é a norma do máximo dada por

$$\|e\|_\infty^k = max|v_{ij} - u_{ij}| \quad 1 \leqslant i, j \leqslant 2^k n$$

Aplicações

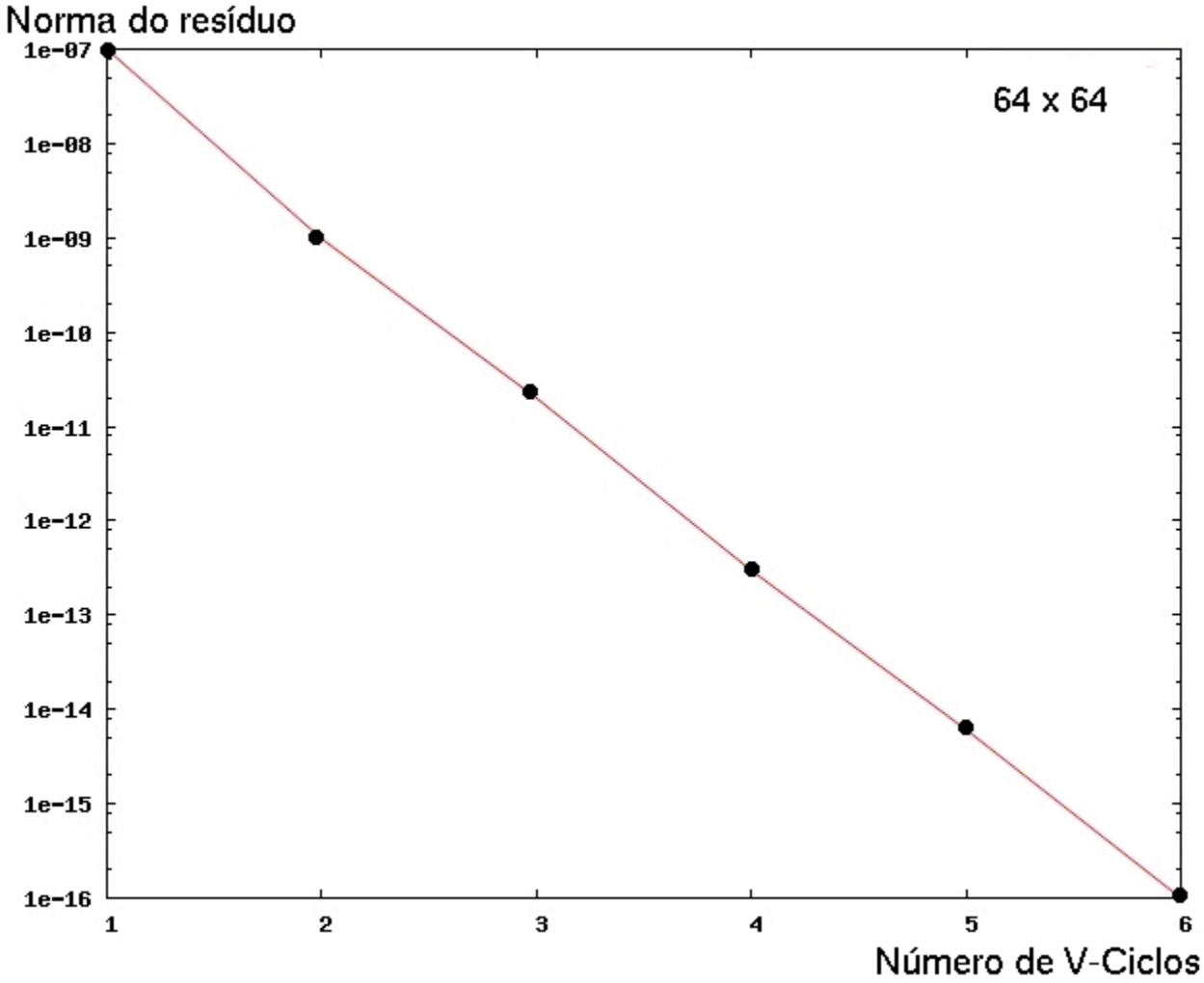
$\Delta t = max(\Delta x, \Delta y)$

Malha $2^k \times 2^k$	$\|e\|_\infty^k$	$\frac{\|e\|_\infty^k}{\|e\|_\infty^{k+1}}$
16×16	0.021144	1.8969
32×32	0.011147	1.9626
64×64	0.005679	1.9855
128×128	0.002860	-

$\Delta t = max(\Delta x^2, \Delta y^2)$

Malha $2^k \times 2^k$	$\|e\|_\infty^k$	$\frac{\|e\|_\infty^k}{\|e\|_\infty^{k+1}}$
16×16	0.004375	3.9167
32×32	0.001117	4,8565
64×64	0.000230	3.1944
128×128	0.000072	-

Aplicações

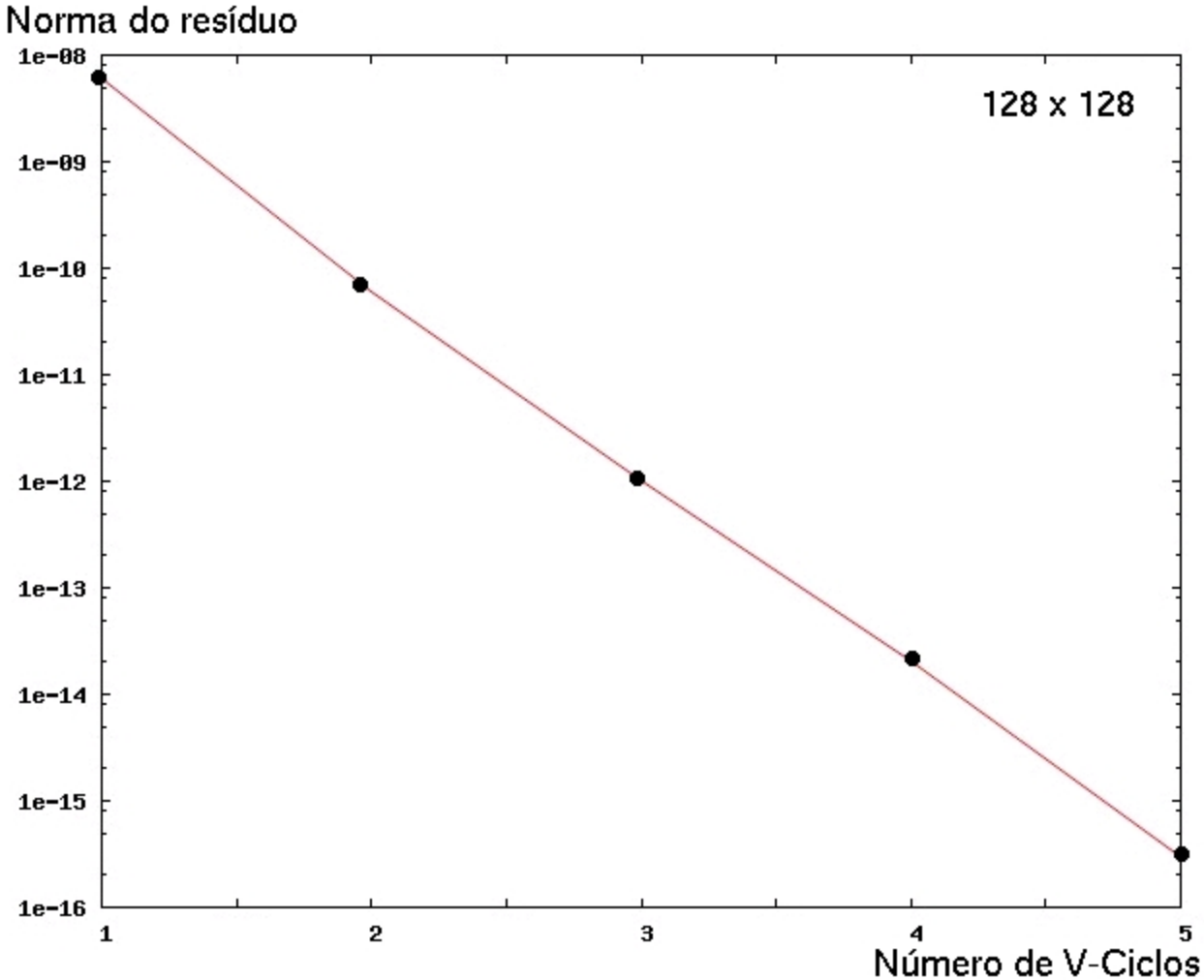


$$\Delta t = max(\Delta x^{2,} \Delta y^2)$$

V(1,1)-Ciclo

Utilizando red-black Gauss-Seidel

Aplicações



$$\Delta t = max(\Delta x^{2,} \Delta y^2)$$

V(1,1)-Ciclo

Utilizando red-black Gauss-Seidel

Referências

- [1] Burden, R. L. ; Faires, J. D., ***Numerical Analysis***, Brooks/Cole, 2001.
- [2] Briggs, W. L. ; Henson, Van Emden ; McCormick, S. F., ***A Multigrid Tutorial***, SIAM, 2000.
- [3] Stoer, J. ; Burlish, R., ***Introduction to Numerical Analysis***, Springer-Verlag, 1980.