ME469A
Handout #8

Gianluca Iaccarino

Geometrical Example

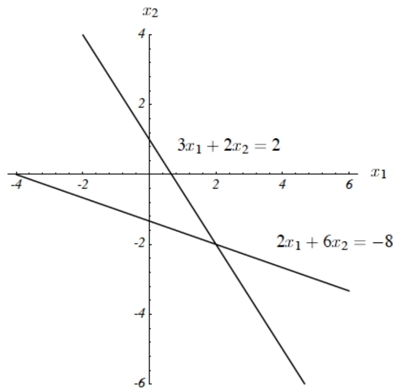
Consider a two-equation system

$$3x_1 + 2x_2 = 2$$
$$2x_1 + 6x_2 = -8$$

$$Ax = b$$

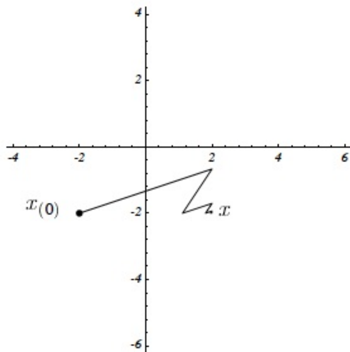
$$A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}, b = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$$

The solution is $x = [2, -2]^T$

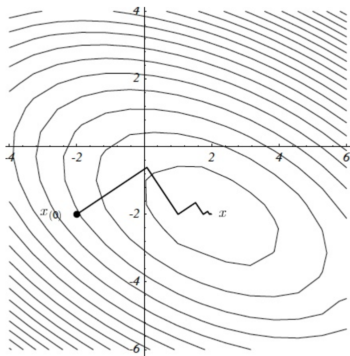


Convergence

- Basic (stationary) iterate is $Mx^{p+1} = Nx^p + b$
- Steepest descent iterate is $x^{p+1} = x^p + \alpha_p \rho_p$



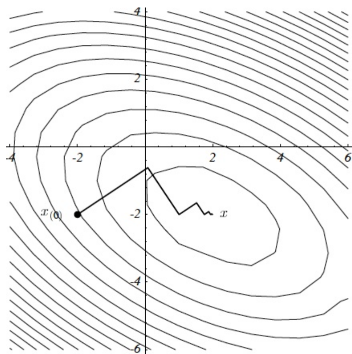
Jacobi



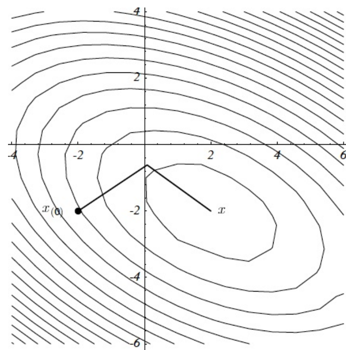
Steepest Descent

Convergence

- The idea of using minimization of the quadratic form to solve the problem can actually be useful...



Steepest Descent



Conjugate Gradient

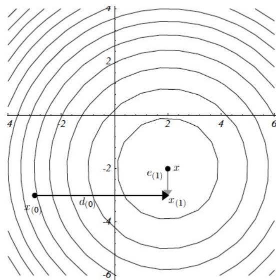
Generalized Descent Methods

- The extension consists in selecting orthogonal directions: never revisit the same direction

  $x^{p+1} = x^p + \alpha \rho_p \rightarrow x^{p+1} = x^p + \alpha d_p$

  - How do we build the d_p directions?
  - d_p: one possible choice is the coordinate directions (x_1 and x_2)

    Note: there are 2 dimensions, and therefore *only* 2 orthogonal directions $\rightarrow$ the method should converge in two steps



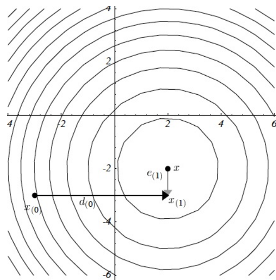
Generalized Descent Methods

- The extension consists in selecting orthogonal directions: never revisit the same direction

$$x^{p+1} = x^p + \alpha \rho_p \rightarrow x^{p+1} = x^p + \alpha d_p$$

- How do we build the d_p directions?
- d_p: one possible choice is the coordinate directions (x_1 and x_2)

  Note: there are 2 dimensions, and therefore *only* 2 orthogonal directions $\rightarrow$ the method should converge in two steps

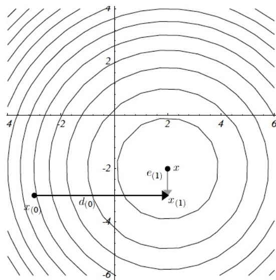


Generalized Descent Methods

- The extension consists in selecting orthogonal directions: never revisit the same direction
  $x^{p+1} = x^p + \alpha \rho_p \rightarrow x^{p+1} = x^p + \alpha d_p$

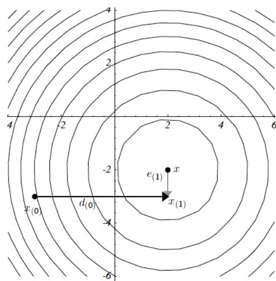
  - How do we build the d_p directions?
  - d_p: one possible choice is the coordinate directions (x_1 and x_2)

    Note: there are 2 dimensions, and therefore *only* 2 orthogonal directions $\rightarrow$ the method should converge in two steps



Generalized Descent Methods

- $x^{p+1} = x^p + \alpha d_p$; we have d_p,
  how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x
  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition
    $(d_p^T \epsilon_{p+1}) = 0$
    $(d_p^T (\epsilon_p + \alpha_p d_p) = 0$
    $$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$

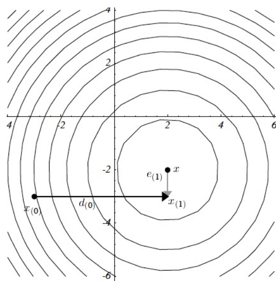


Generalized Descent Methods

- $x^{p+1} = x^p + \alpha d_p$; we have d_p, how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x
  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition $(d_p^T \epsilon_{p+1}) = 0$

$$(d_p^T(\epsilon_p + \alpha_p d_p) = 0$$

$$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

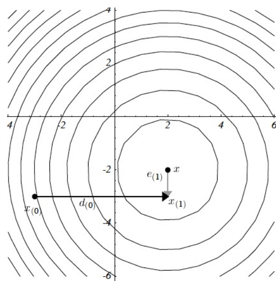
- $x^{p+1} = x^p + \alpha d_p$; we have d_p, how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x

- Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
- We could use the condition

$$(d_p^T \epsilon_{p+1}) = 0$$

$$(d_p^T (\epsilon_p + \alpha_p d_p) = 0$$

$$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

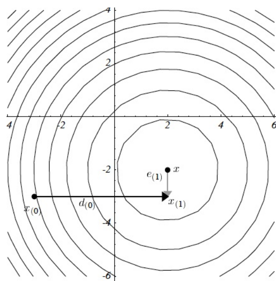
- $x^{p+1} = x^p + \alpha d_p$; we have d_p, how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x

  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition

$$(d_p^T \epsilon_{p+1}) = 0$$

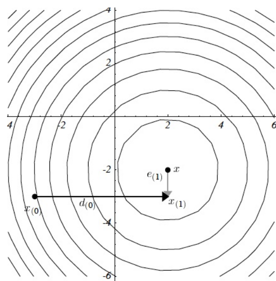
$$(d_p^T (\epsilon_p + \alpha_p d_p)) = 0$$

$$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

- $x^{p+1} = x^p + \alpha d_p$; we have d_p,
  how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x
  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition
    $(d_p^T \epsilon_{p+1}) = 0$
    $(d_p^T(\epsilon_p + \alpha_p d_p) = 0$
    $$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

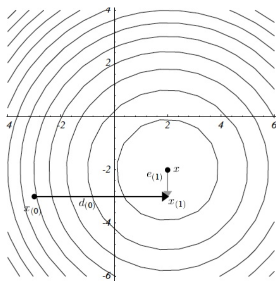
- $x^{p+1} = x^p + \alpha d_p$; we have d_p,
  how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x

  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition

$$(d_p^T \epsilon_{p+1}) = 0$$

$$(d_p^T(\epsilon_p + \alpha_p d_p) = 0$$

$$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

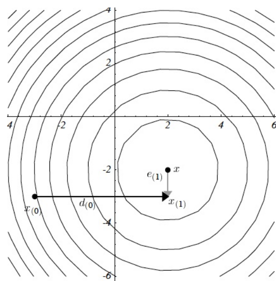
- $x^{p+1} = x^p + \alpha d_p$; we have d_p, how do we compute the α_p weight?
- in each direction, march enough to align the next guess with the exact solution, x

  - Assume $d_0 = \rho_0$, and suppose you know ϵ_0, ϵ_1.
  - We could use the condition

$$(d_p^T \epsilon_{p+1}) = 0$$

$$(d_p^T (\epsilon_p + \alpha_p d_p) = 0$$

$$\alpha_p = -\frac{d_p^T \epsilon_p}{d_p^T d_p}$$



Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  $= f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$ - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  $= f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$
  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  $= f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$ - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$ - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  $= f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error
  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:
  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$
  $= f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$
  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$
  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$
  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$
  - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- Need a definition that does not use the error

  $x^{p+1} = x^p + \alpha d_p$
- Compute the minimum of $f(x^{p+1})$ along the direction d_p:

  $df(x^{p+1})/d\alpha = df(x^{p+1})/dx \quad dx^{p+1}/d\alpha = 0$

  = $f'(x^{p+1})^T d_p = -\rho_{p+1}^T d_p = 0$

  $-\rho_{p+1}^T d_p = (A\epsilon_{p+1})^T d_p = d_p^T A\epsilon_{p+1} = 0$

  $d_p^T A\epsilon_{p+1} = d_p^T A(\epsilon_p + \alpha_p d_p) = 0$

  $d_p^T A\epsilon_p = -d_p^T \rho_p = -\alpha_p d_p^T A d_p$
- $\alpha_p = \dfrac{d_p^T \rho_p}{d_p^T A d_p}$

  Note that we obtained $d_p^T A\epsilon_{p+1} = 0$ instead of $d_p^T \epsilon_{p+1} = 0$ - d_p and ϵ_{p+1} are said to the A-orthogonal.

Generalized Descent Methods

- We have not defined d_p yet, we just said that are A-orthogonal directions.
- It turns out that if $d_p = \rho_p$ we obtain the steepest descent algorithm again
- We can build the d_p vectors using the Graham-Schmidt conjugation process...

Generalized Descent Methods

- We have not defined d_p yet, we just said that are A-orthogonal directions.
- It turns out that if $d_p = \rho_p$ we obtain the steepest descent algorithm again
- We can build the d_p vectors using the Graham-Schmidt conjugation process...

Generalized Descent Methods

- We have not defined d_p yet, we just said that are A-orthogonal directions.
- It turns out that if $d_p = \rho_p$ we obtain the steepest descent algorithm again
- We can build the d_p vectors using the Graham-Schmidt conjugation process...

Descent Methods

- **Steepest Descent**

$$\rho_0 = b - Ax_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{\rho_P^T (A\rho_p)}$$

$$x_{p+1} = x_p + \alpha_p \rho_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (A\rho_p)$$

```
enddo
```

- **Conjugate Gradient**

$$\rho_0 = b - Ax_0 = d_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

$$\beta_{p+1} = \frac{\rho_{p+1}^T \rho_{p+1}}{\rho_P^T \rho_p}$$

$$d_{p+1} = \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

Note: $\rho_{p+1} = b - Ax_{p+1}$ can be computed as $\rho_{p+1} = \rho_p - \alpha_p (A\rho_p)$

Descent Methods

- **Steepest Descent**

$$\rho_0 = b - Ax_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{\rho_P^T (A\rho_p)}$$

$$x_{p+1} = x_p + \alpha_p \rho_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (A\rho_p)$$

```
enddo
```

- **Conjugate Gradient**

$$\rho_0 = b - Ax_0 = d_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

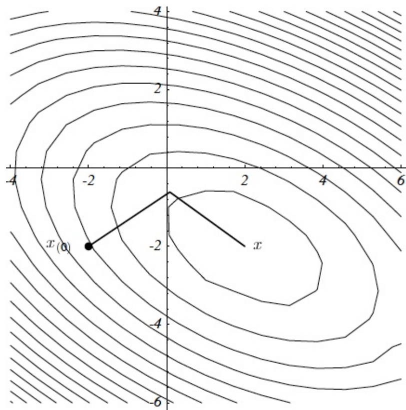
$$\beta_{p+1} = \frac{\rho_{p+1}^T \rho_{p+1}}{\rho_P^T \rho_p}$$

$$d_{p+1} = \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

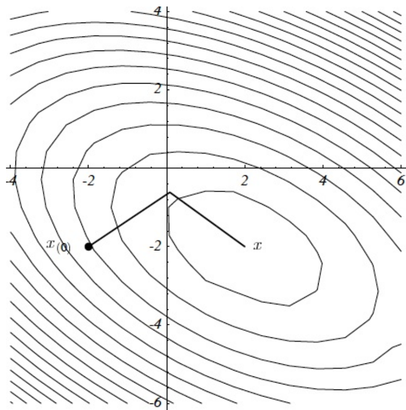
Note: $\rho_{p+1} = b - Ax_{p+1}$ can be computed as $\rho_{p+1} = \rho_p - \alpha_p (A\rho_p)$

Conjugate Gradient



- The convergence in 2 steps is possible but note that the directions d_p are NOT orthogonal, they are *A-orthogonal*

Conjugate Gradient



- The convergence in 2 steps is possible but note that the directions d_p are NOT orthogonal, they are *A-orthogonal*

Convergence comparisons

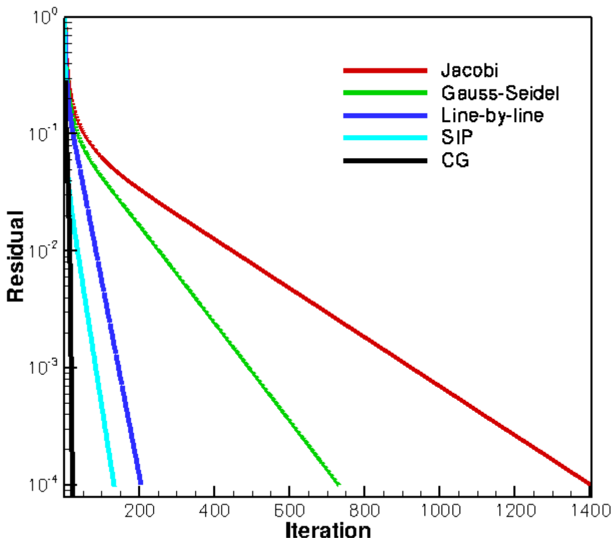
- Let's consider a more realistic problem: solve the Laplace equation in 2D ($[0:1] \times [0:1]$)
- We use uniform grids and a FV central differencing scheme
- Code `lapl2d.f`
- Compare different solver:
  - Jacobi
  - Gauss-Seidel
  - Line-by-Line solver
  - SIP (Stone's method)
  - CG

Convergence comparisons

- Let's consider a more realistic problem: solve the Laplace equation in 2D ($[0:1]\times[0:1]$)
- We use uniform grids and a FV central differencing scheme
- Code `lapl2d.f`
- Compare different solver:
  - Jacobi
  - Gauss-Seidel
  - Line-by-Line solver
  - SIP (Stone's method)
  - CG

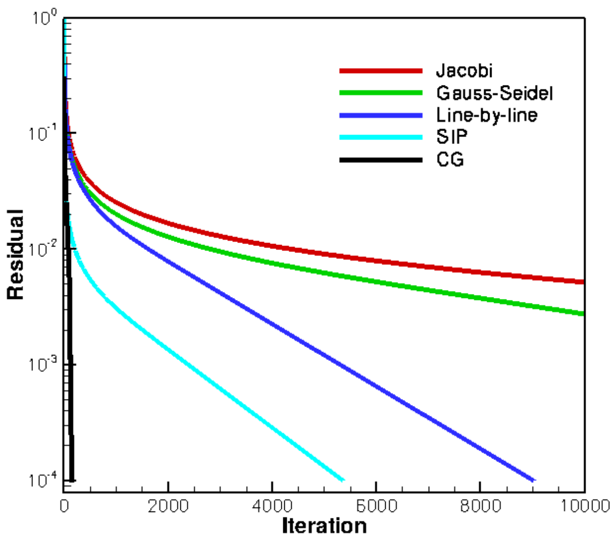
Convergence comparisons

Grid 32×32



Convergence comparisons

Grid 256×256



Convergence comparisons

- Convergence speed measured via iteration count strongly favors SIP and CG.
- This is not necessarily an useful measure, how about computational time?

	252×252	32×32
Jacobi	49.321	0.178
GS	45.583	0.111
LBL	69.698	0.087
SIP	23.619	0.061
CG	1.270	0.045

Time [sec] on a laptop.

Convergence comparisons

- Convergence speed measured via iteration count strongly favors SIP and CG.
- This is not necessarily an useful measure, how about computational time?

	252×252	32×32
Jacobi	49.321	0.178
GS	45.583	0.111
LBL	69.698	0.087
SIP	23.619	0.061
CG	1.270	0.045

Time [sec] on a laptop.

Convergence comparisons

- Convergence speed measured via iteration count strongly favors SIP and CG.
- This is not necessarily an useful measure, how about computational time?

	252×252	32×32
Jacobi	49.321	0.178
GS	45.583	0.111
LBL	69.698	0.087
SIP	23.619	0.061
CG	1.270	0.045

Time [sec] on a laptop.

Convergence of descent methods

- The convergence of *classic* iterative methods (Jacobi, GS, etc.) is controlled by
  $\epsilon^{p+1} = (M^{-1}N)\epsilon^p$
- The convergence rate $\omega = ||\epsilon^{p+1}||/||\epsilon^p||$ is controlled by the spectral radius of $(M^{-1}N)$
- For descent methods it is possible to prove that $\omega = g(\kappa)$ where $\kappa = \lambda^A_{max}/\lambda^A_{min}$
- The convergence rate is controlled by the condition number of A

Convergence of descent methods

- The convergence of *classic* iterative methods (Jacobi, GS, etc.) is controlled by
  $\epsilon^{p+1} = (M^{-1}N)\epsilon^{p}$
- The convergence rate $\omega = ||\epsilon^{p+1}||/||\epsilon^{p}||$ is controlled by the spectral radius of $(M^{-1}N)$
- For descent methods it is possible to prove that $\omega = g(\kappa)$ where $\kappa = \lambda^{A}_{max}/\lambda^{A}_{min}$
- The convergence rate is controlled by the condition number of A

Convergence of descent methods

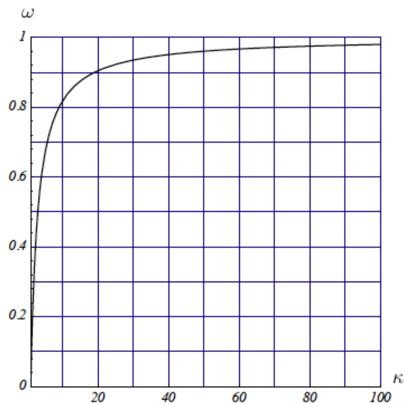
- The convergence of *classic* iterative methods (Jacobi, GS, etc.) is controlled by $\epsilon^{p+1} = (M^{-1}N)\epsilon^{p}$
- The convergence rate $\omega = ||\epsilon^{p+1}||/||\epsilon^{p}||$ is controlled by the spectral radius of $(M^{-1}N)$
- For descent methods it is possible to prove that $\omega = g(\kappa)$ where $\kappa = \lambda^{A}_{max}/\lambda^{A}_{min}$
- The convergence rate is controlled by the condition number of A

Convergence of descent methods

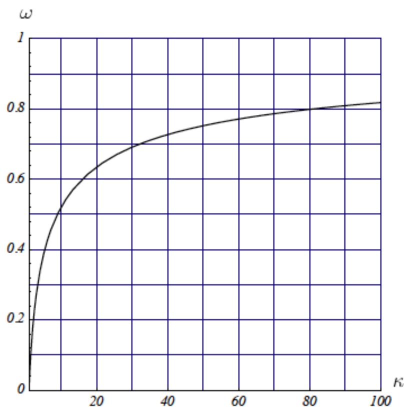
- The convergence of *classic* iterative methods (Jacobi, GS, etc.) is controlled by $\epsilon^{p+1} = (M^{-1}N)\epsilon^{p}$
- The convergence rate $\omega = ||\epsilon^{p+1}||/||\epsilon^{p}||$ is controlled by the spectral radius of $(M^{-1}N)$
- For descent methods it is possible to prove that $\omega = g(\kappa)$ where $\kappa = \lambda^{A}_{max}/\lambda^{A}_{min}$
- The convergence rate is controlled by the condition number of A

Convergence rate

The relation $\omega = \omega(\kappa)$ can be determined theoretically



Steepest Descent



Conjugate Gradient

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioning

- Preconditioning is a general technique for improving the condition number of a matrix
- The basic idea is that if we need to solve $Ax = b$ and assuming we have a matrix P whose inverse is easy to compute...
- we can solve the original problem by solving $P^{-1}Ax = P^{-1}b$
- obviously this is useful only if $\kappa(P^{-1}A) < \kappa(A)$
- there is another requirement: $(P^{-1}A)$ must be symmetric positive-definite
- This is not automatically verified even if P and A are SPD

Preconditioned Conjugate Gradient

- Conjugate Gradient

$$\rho_0 = b - Ax_0 = d_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

$$\beta_{p+1} = \frac{\rho_{p+1}^T \rho_{p+1}}{\rho_P^T \rho_p}$$

$$d_{p+1} = \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

- Preconditioned CG

$$\rho_0 = b - Ax_0$$

$$d_0 = P^{-1} \rho_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T (P^{-1} \rho_p)}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

$$\beta_{p+1} = \frac{\rho_{p+1}^T (P^{-1} \rho_{p+1})}{\rho_P^T (P^{-1} \rho_p)}$$

$$d_{p+1} = P^{-1} \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

Preconditioned Conjugate Gradient

- Conjugate Gradient

$$\rho_0 = b - Ax_0 = d_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T \rho_p}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

$$\beta_{p+1} = \frac{\rho_{p+1}^T \rho_{p+1}}{\rho_P^T \rho_p}$$

$$d_{p+1} = \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

- Preconditioned CG

$$\rho_0 = b - Ax_0$$

$$d_0 = P^{-1} \rho_0$$

```
do p=0...
```

$$\alpha_p = \frac{\rho_p^T (P^{-1} \rho_p)}{d_P^T (Ad_p)}$$

$$x_{p+1} = x_p + \alpha_p d_p$$

$$\rho_{p+1} = \rho_p - \alpha_p (Ad_p)$$

$$\beta_{p+1} = \frac{\rho_{p+1}^T (P^{-1} \rho_{p+1})}{\rho_P^T (P^{-1} \rho_p)}$$

$$d_{p+1} = P^{-1} \rho_{p+1} + \beta_{p+1} d_p$$

```
enddo
```

Preconditioned Conjugate Gradient

- PCG is simple to implement and has good efficiency
- The cost is typically dominated by the matrix-vector multiplication
- The preconditioner used is typically $P = diag(A)$; other options are possible, the ILU(T) is typically one of the best choices
- The nominal convergence in $size(A)$ steps is never achieved because of round-off error

  but for large systems we expect far less steps than $size(A)$

Preconditioned Conjugate Gradient

- PCG is simple to implement and has good efficiency
- The cost is typically dominated by the matrix-vector multiplication
- The preconditioner used is typically $P = diag(A)$; other options are possible, the ILU(T) is typically one of the best choices
- The nominal convergence in $size(A)$ steps is never achieved because of round-off error

  but for large systems we expect far less steps than $size(A)$

Preconditioned Conjugate Gradient

- PCG is simple to implement and has good efficiency
- The cost is typically dominated by the matrix-vector multiplication
- The preconditioner used is typically $P = diag(A)$; other options are possible, the ILU(T) is typically one of the best choices
- The nominal convergence in $size(A)$ steps is never achieved because of round-off error

  but for large systems we expect far less steps than $size(A)$

Preconditioned Conjugate Gradient

- PCG is simple to implement and has good efficiency
- The cost is typically dominated by the matrix-vector multiplication
- The preconditioner used is typically $P = diag(A)$; other options are possible, the ILU(T) is typically one of the best choices
- The nominal convergence in $size(A)$ steps is never achieved because of round-off error

  but for large systems we expect far less steps than $size(A)$

Preconditioned Conjugate Gradient

- PCG is simple to implement and has good efficiency
- The cost is typically dominated by the matrix-vector multiplication
- The preconditioner used is typically $P = diag(A)$; other options are possible, the ILU(T) is typically one of the best choices
- The nominal convergence in $size(A)$ steps is never achieved because of round-off error

  but for large systems we expect far less steps than $size(A)$

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T Ax = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T A x = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T A x = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T Ax = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T Ax = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T Ax = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Non-symmetric matrices?

- (P)CG is only applicable for SPD matrices!
- What can we do for a general matrix?
- Observation: the system $A^T A x = A^T b$ is equivalent to the original system $Ax = b$ and $A^T A$ is a symmetric matrix
- It is possible to derive the same CG algorithm for $A^T A$
- the problem is that $\kappa(A^T A) = \kappa(A)^2$ leading to slow convergence
- Another approach is based on the simultaneous generation of two set of dimensions d_p and $\tilde{d}_p$ corresponding to the two matrices A and A^T (Bi-CG)
- Many variants with pros and cons: GMRES, BiCG, BiCGSTAB, etc.

Summary

- Linear system solver are a major bottleneck;
- Direct solvers
  - Gauss elimination
  - LU decomposition
  - Specialized solvers
    - Tridiagonal solver
- Iterative solvers
  - Classical solvers
    - Jacobi/Seidel/SOR
    - SIP (ILU)
    - Line-by-line/ADI solver
  - Descent methods
    - Steepest descent
    - Conjugate gradient
    - Preconditioning
- Multigrid

Summary

- Linear system solver are a major bottleneck;
- Direct solvers
  - Gauss elimination
  - LU decomposition
  - Specialized solvers
    - Tridiagonal solver
- Iterative solvers
  - Classical solvers
    - Jacobi/Seidel/SOR
    - SIP (ILU)
    - Line-by-line/ADI solver
  - Descent methods
    - Steepest descent
    - Conjugate gradient
    - Preconditioning
- Multigrid

Summary

- Linear system solver are a major bottleneck;
- Direct solvers
  - Gauss elimination
  - LU decomposition
  - Specialized solvers
    - Tridiagonal solver
- Iterative solvers
  - Classical solvers
    - Jacobi/Seidel/SOR
    - SIP (ILU)
    - Line-by-line/ADI solver
  - Descent methods
    - Steepest descent
    - Conjugate gradient
    - Preconditioning
- Multigrid

Summary

- Linear system solver are a major bottleneck;
- Direct solvers
  - Gauss elimination
  - LU decomposition
  - Specialized solvers
    - Tridiagonal solver
- Iterative solvers
  - Classical solvers
    - Jacobi/Seidel/SOR
    - SIP (ILU)
    - Line-by-line/ADI solver
  - Descent methods
    - Steepest descent
    - Conjugate gradient
    - Preconditioning
- Multigrid

Beyond structured grids

- The key element of CG-type solvers is the matrix-vector multiplication, e.g. compute a residual $\rho_p = b - A\phi^p$
- For a 5-point stencil (2D steady convection/diffusion or 2D Laplace) the matrix is pentadiagonal;

  $A_S\phi_S + A_W\phi_W + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P$
- Computing the residual is straightforward:

  $\rho_p = Q_P - (A_S\phi_S^p + A_W\phi_W^p + A_P\phi_P^p + A_N\phi_N^p + A_E\phi_E^p)$
- CG-type algorithms are general, do not require an ordered matrix (structured grid) - should we try an unstructured grid?
- The first question is how does a discretization stencil for the Laplace equation looks like on an unstructured grid?

Beyond structured grids

- The key element of CG-type solvers is the matrix-vector multiplication, e.g. compute a residual $\rho_p = b - A\phi^p$
- For a 5-point stencil (2D steady convection/diffusion or 2D Laplace) the matrix is pentadiagonal;

$$A_S\phi_S + A_W\phi_W + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P$$

- Computing the residual is straightforward:

$$\rho_p = Q_P - (A_S\phi_S^p + A_W\phi_W^p + A_P\phi_P^p + A_N\phi_N^p + A_E\phi_E^p)$$

- CG-type algorithms are general, do not require an ordered matrix (structured grid) - should we try an unstructured grid?
- The first question is how does a discretization stencil for the Laplace equation looks like on an unstructured grid?

Beyond structured grids

- The key element of CG-type solvers is the matrix-vector multiplication, e.g. compute a residual $\rho_p = b - A\phi^p$
- For a 5-point stencil (2D steady convection/diffusion or 2D Laplace) the matrix is pentadiagonal;

  $A_S\phi_S + A_W\phi_W + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P$
- Computing the residual is straightforward:

  $\rho_p = Q_P - (A_S\phi_S^p + A_W\phi_W^p + A_P\phi_P^p + A_N\phi_N^p + A_E\phi_E^p)$
- CG-type algorithms are general, do not require an ordered matrix (structured grid) - should we try an unstructured grid?
- The first question is how does a discretization stencil for the Laplace equation looks like on an unstructured grid?

Beyond structured grids

- The key element of CG-type solvers is the matrix-vector multiplication, e.g. compute a residual $\rho_p = b - A\phi^p$
- For a 5-point stencil (2D steady convection/diffusion or 2D Laplace) the matrix is pentadiagonal;

  $A_S\phi_S + A_W\phi_W + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P$
- Computing the residual is straightforward:

  $\rho_p = Q_P - (A_S\phi_S^p + A_W\phi_W^p + A_P\phi_P^p + A_N\phi_N^p + A_E\phi_E^p)$
- CG-type algorithms are general, do not require an ordered matrix (structured grid) - should we try an unstructured grid?
- The first question is how does a discretization stencil for the Laplace equation looks like on an unstructured grid?

Beyond structured grids

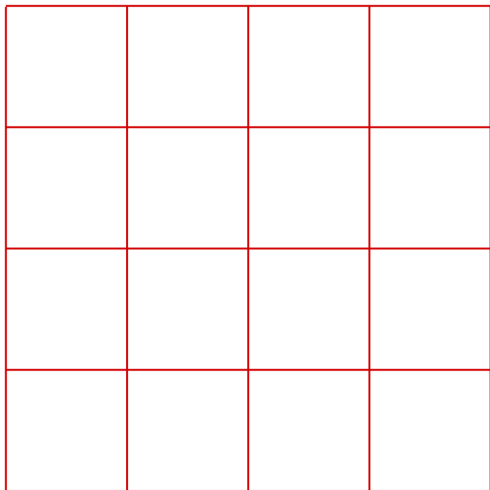
- The key element of CG-type solvers is the matrix-vector multiplication, e.g. compute a residual $\rho_p = b - A\phi^p$
- For a 5-point stencil (2D steady convection/diffusion or 2D Laplace) the matrix is pentadiagonal;

  $A_S\phi_S + A_W\phi_W + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P$
- Computing the residual is straightforward:

  $\rho_p = Q_P - (A_S\phi_S^p + A_W\phi_W^p + A_P\phi_P^p + A_N\phi_N^p + A_E\phi_E^p)$
- CG-type algorithms are general, do not require an ordered matrix (structured grid) - should we try an unstructured grid?
- The first question is how does a discretization stencil for the Laplace equation looks like on an unstructured grid?

Beyond structured grids

To simplify things, let's assume that this grid is unstructured



Beyond structured grids

To simplify things, let's assume that this grid is unstructured

			(4,4)
(1,2)			
(1,1)	(2,1)		

Beyond structured grids

Now it is really unstructured

7

1 6

4 12 14

Beyond structured grids

- Never ever store A as $a_{ij} = [N][N]$
- For the ordered grid we build the matrix (row-by-row) assuming a global CV index

Matrix elements

```
for (int i=0; i<ni-1; i++) {
  for (int j=0; j<nj-1; j++) {
     int icv = i + (j-1)*ni // global index
     ae[icv] = -(y[j]-y[j-1])/(xc[i+1]-xc[i])
     aw[icv] = ...
     as[icv] = ...
     an[icv] = ...
     ap[icv] = ...
  }
 }
```

Residual evaluation

```
for (int icv=0; icv<(ni-1)*(nj-1); icv++) {
   rho[icv] = q[icv] - as[icv]*phi[icv-ni]
                     - aw[icv]*phi[icv-1]
                     - an[icv]*phi[icv+ni]
                     - aw[icv]*phi[icv+1]
                     - ap[icv]*phi[icv]
 }
```

Beyond structured grids

- Never ever store A as $a_{ij} = [N][N]$
- For the ordered grid we build the matrix (row-by-row) assuming a global CV index

(4,4)
(1,2)
(1,1)
(2,1)

Matrix elements

```
for (int i=0; i<ni-1; i++) {
  for (int j=0; j<nj-1; j++) {
     int icv = i + (j-1)*ni // global index
     ae[icv] = -(y[j]-y[j-1])/(xc[i+1]-xc[i])
     aw[icv] = ...
     as[icv] = ...
     an[icv] = ...
     ap[icv] = ...
   }
 }
```

Residual evaluation

```
for (int icv=0; icv<(ni-1)*(nj-1); icv++) {
   rho[icv] = q[icv] - as[icv]*phi[icv-ni]
                     - aw[icv]*phi[icv-1]
                     - an[icv]*phi[icv+ni]
                     - aw[icv]*phi[icv+1]
                     - ap[icv]*phi[icv]
 }
```

Beyond structured grids

- Never ever store A as $a_{ij} = [N][N]$
- For the ordered grid we build the matrix (row-by-row) assuming a global CV index

(4,4)
(1,2)
(1,1)
(2,1)

Matrix elements

```
for (int i=0; i<ni-1; i++) {
  for (int j=0; j<nj-1; j++) {
     int icv = i + (j-1)*ni // global index
     ae[icv] = -(y[j]-y[j-1])/(xc[i+1]-xc[i])
     aw[icv] = ...
     as[icv] = ...
     an[icv] = ...
     ap[icv] = ...
   }
 }
```

Residual evaluation

```
for (int icv=0; icv<(ni-1)*(nj-1); icv++) {
   rho[icv] = q[icv] - as[icv]*phi[icv-ni]
                     - aw[icv]*phi[icv-1]
                     - an[icv]*phi[icv+ni]
                     - aw[icv]*phi[icv+1]
                     - ap[icv]*phi[icv]
 }
```

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

7

1 6

4 12 14

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

			7
1	6		
4	12	14	

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

7
1 6
4 12 14

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

7
1 6
4 12 14

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

			7
1	6		
4	12	14	

- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- For the unstructured grid we need more information
- We need to build a list of neighbors for each CV

			7
1	6		
4	12	14	

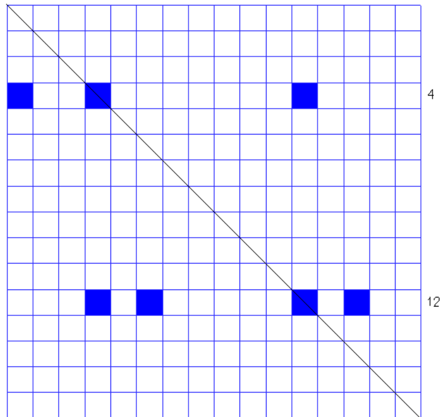
- One possibility is to have a list of neighbors of CV, `nbocv`
- `nbocv[0:15][0:3]` will do...BUT
- What about an unstructured grid with triangles? or polygons?
- How to store the matrix?
- The diagonals are not there anymore!

Beyond structured grids

- Sparse matrix for the unstructured ordering (the discretization is the same as before)

Grid ($N = 16$)

Matrix (16×16)



General sparse matrices

- The idea is to compress all the non-zero matrix elements
- We store the matrix row-by-row as a sequence of entries skipping all the zeros - CRS structure (compressed row storage)

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val	10	-2	3	9	3	7	8	7	3 ··· 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ··· 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

.

- `val`: real array containing all the matrix entries $[0 : NZ - 1]$
- `col_ind`: the column index of the elements in `val` $[0 : NZ - 1]$
  if `val(k)` $= a_{ij} \neq 0$ then `col_ind(k)` $= j$
- `row_ptr`: the location in `val` where a new row starts $[0 : N]$

General sparse matrices

- The idea is to compress all the non-zero matrix elements
- We store the matrix row-by-row as a sequence of entries skipping all the zeros - CRS structure (compressed row storage)

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val	10	-2	3	9	3	7	8	7	3 · · · 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 · · · 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

.

- `val`: real array containing all the matrix entries $[0 : NZ - 1]$
- `col_ind`: the column index of the elements in `val` $[0 : NZ - 1]$

  if `val(k)` $= a_{ij} \neq 0$ then `col_ind(k)` $= j$
- `row_ptr`: the location in `val` where a new row starts $[0 : N]$

General sparse matrices

- The idea is to compress all the non-zero matrix elements
- We store the matrix row-by-row as a sequence of entries skipping all the zeros - CRS structure (compressed row storage)

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val	10	-2	3	9	3	7	8	7	3 ··· 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ··· 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

.

- `val`: real array containing all the matrix entries $[0:NZ-1]$
- `col_ind`: the column index of the elements in `val` $[0:NZ-1]$

  if `val(k)` = $a_{ij} \neq 0$ then `col_ind(k)` $= j$
- `row_ptr`: the location in `val` where a new row starts $[0:N]$

Matrix-Vector multiplication

Full Storage

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

```
for (int icv = 0; icv < ncv; icv++) {
  y[icv] = 0;
  for (int jcv = 0; jcv < ncv; jcv++) {
      y[icv] +=  a[icv][jcv] * phi[jcv];
}}
```

CRS Format

val	10	-2	3	9	3	7	8	7	3 ··· 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ··· 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

.

```
for (int icv = 0; icv < ncv; icv++) {
   y[icv] = 0;
   int j_f = row_ptr[i];
   int j_l = row_ptr[i+1] - 1;
   for (int j = j_f; j <= j_l; j++)
      y[icv] +=  val[j] * phi[col_ind[j]];
}
```

Towards unstructured grids

The same compressed structure is used in handling unstructured grid connectivity