Iterative Solvers for Linear Systems using a Minimization Approach

Martin Berzins
School of Computing

Aim to introduce better methods for solving sub-classes of $Ax=b$

Consider Gradient Descent - very widely used in deep learning

Introduce Conjugate Gradient methods

Symmetric Positive Definite Matrices

Symmetric matrices entries $A_{ij} = A_{ji}$

Positive definite matrices $x^T Ax \geq 0$ for all x

Minimize $\frac{1}{2} x^T Ax - x^T b$ instead of solving $Ax - b = 0$

at a minimum $Ax - b$

Differentiate $\frac{1}{2}$ $x^T Ax - x^T b$ wrt components

of x and set to zero giving $Ax - b$.

Quadratic Form Example

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

Quadratic form

$$Q(x_1, x_2) = \frac{1}{2}[x_1, x_2] \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - [2x_1 - 8x_2] = 0$$

Expanding

$$Q(x_1, x_2) = \frac{1}{2}[3x_1^2 + 4x_1x_2 + 6x_2^2] - [2x_1 - 8x_2] = 0$$

Differentiate $Q(x_1, x_2)$ wrt x_1 and $Q(x_1, x_2)$ wrt x_2 and set to zero to get original eqns at minimum

$$\frac{1}{2}[6x_1 + 4x_2] - 2 = 0 \qquad \frac{1}{2}[4x_1 + 12x_2] + 8 = 0$$

Gradient Vector

Suppose that we have a function that depends on two or more variables

$$Q(x_1, x_2) = \frac{1}{2}[3x_1^2 + 4x_1x_2 + 6x_2^2] - [2x_1 - 8x_2] = 0$$

Then the gradient vector is defined by

$$\nabla Q = \begin{bmatrix} \frac{\partial Q}{\partial x_1} \\ \frac{\partial Q}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}[6x_1 + 4x_2] - 2 \\ \frac{1}{2}[4x_1 + 12x_2] + 8 \end{bmatrix} = Ax - b$$

The operation $\frac{\partial Q}{\partial x_1}$ means differentiate Q with respect to x_1

The operation $\frac{\partial Q}{\partial x_2}$ means differentiate Q with respect to x_2

Stationary and non Stationary Iterative Methods

The problem is still to solve for **Ax** = **b**

Stationary (or relaxation) methods:

$$\mathbf{x}^{(i+1)} = \mathbf{G}\mathbf{x}^{(i)} + \mathbf{c}$$

where **G** and **c** do not depend on iteration count i

Non-stationary methods:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)}\,\mathbf{p}^{(i)}$$

where computation involves information that changes at each iteration

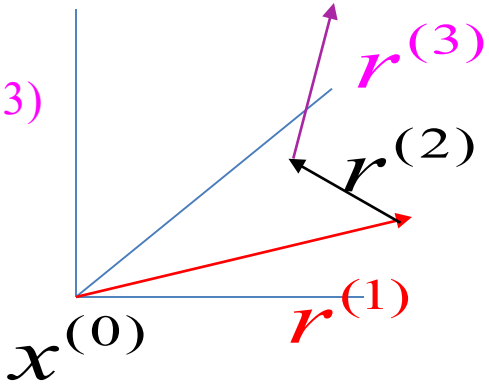
Main Idea – Search Directions

- Construct Search Directions based on residuals to be orthogonal.

$$(r^{(i)})^T . r^{(i+1)} = 0$$

- This means that the residual vectors may span the vector space that the solution is in
- E.g. in three dimensions the solution may be expressed as

$$x_{solution} = x^{(0)} + \alpha_1 r^{(1)} + \alpha_2 r^{(2)} + \alpha_3 r^{(3)}$$



- We then search along these directions
- For the steepest descent method the search directions are only locally orthogonal.

Steepest Descent Algorithm

Iteratively update **x** along the gradient direction

$$r(x) = \mathbf{b} - \mathbf{A}x$$

The *stepsize* is selected to minimize *f*(**x**) along $-r(x)$

Set i=0, $\varepsilon > 0$, $\mathbf{x}^{(0)} = \mathbf{0}$, so $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} = \mathbf{b}$

While $\|\mathbf{r}^{(i)}\| \geq \varepsilon$ Do

(a) calculate the best stepsize $\alpha^{(i)} = \dfrac{\left[r^{(i)}\right]^T r^{(i)}}{\left[r^{(i)}\right]^T A r^{(i)}}$

(b) $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)} \mathbf{r}^{(i)}$

(c) $\mathbf{r}^{(i+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(i+1)}$

(d) i := i + 1

End While

This version has two matrix-vector multiplications per step

Eliminating a matrix multiplication

Matrix vector multiplications are the main work of an iterative algorithm,

$$r^{i+1} = b - Ax^{i+1}$$

as

$$x^{i+1} = x^{i} + \alpha^{(i)} r^{(i)}$$

$$r^{i+1} = \underbrace{b - A(x^{i}}_{r^{(i)}} + \alpha^{(i)} r^{(i)})$$

$$r^{i+1} = r^{(i)} - \alpha^{(i)} A r^{(i)}$$

As we need $Ar^{(i)}$ elsewhere we only need one matrix multiplication per step.

Choosing step α

The current gradient is $b - Ax^{i+1}$

$$r^{(i+1)} = r^{(i)} - \alpha^{(i)} A r^{(i)}$$

As we require that the gradients are orthogonal at each step

$$(r^{(i)})^T r^{(i+1)} = 0$$

and so we require

$$(r^{(i)})^T r^{(i+1)} = (r^{(i)})^T r^{(i)} - \alpha^{(i)} (r^i)^T A r^{(i)} = 0$$

or that

$$(r^{(i)})^T r^{(i)} = \alpha^{(i)} (r^i)^T A r^{(i)} \Rightarrow \alpha^{(i)} = \frac{(r^{(i)})^T r^{(i)}}{(r^i)^T A r^{(i)}}$$

Steepest Descent Algorithm

Iteratively update **x** along the gradient direction

$$\boldsymbol{r}(\boldsymbol{x}) = \mathbf{b} - \mathbf{A}\mathbf{x}$$

The *stepsize* is selected to minimize *f(***x**) along $-\boldsymbol{r}(\boldsymbol{x})$

Set i=0, $\varepsilon > 0$, $\mathbf{x}^{(0)} = \mathbf{0}$, so $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} = \mathbf{b}$

While $\|\mathbf{r}^{(i)}\| \geq \varepsilon$ Do

(a) calculate the best stepsize $\alpha^{(i)} = \dfrac{\left[\boldsymbol{r}^{(i)}\right]^T \boldsymbol{r}^{(i)}}{\left[\boldsymbol{r}^{(i)}\right]^T \boldsymbol{A}\boldsymbol{r}^{(i)}}$

(b) $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)} \mathbf{r}^{(i)}$

(c) $\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} - \alpha^{(i)} \mathbf{A}\mathbf{r}^{(i)}$

(d) i := i + 1

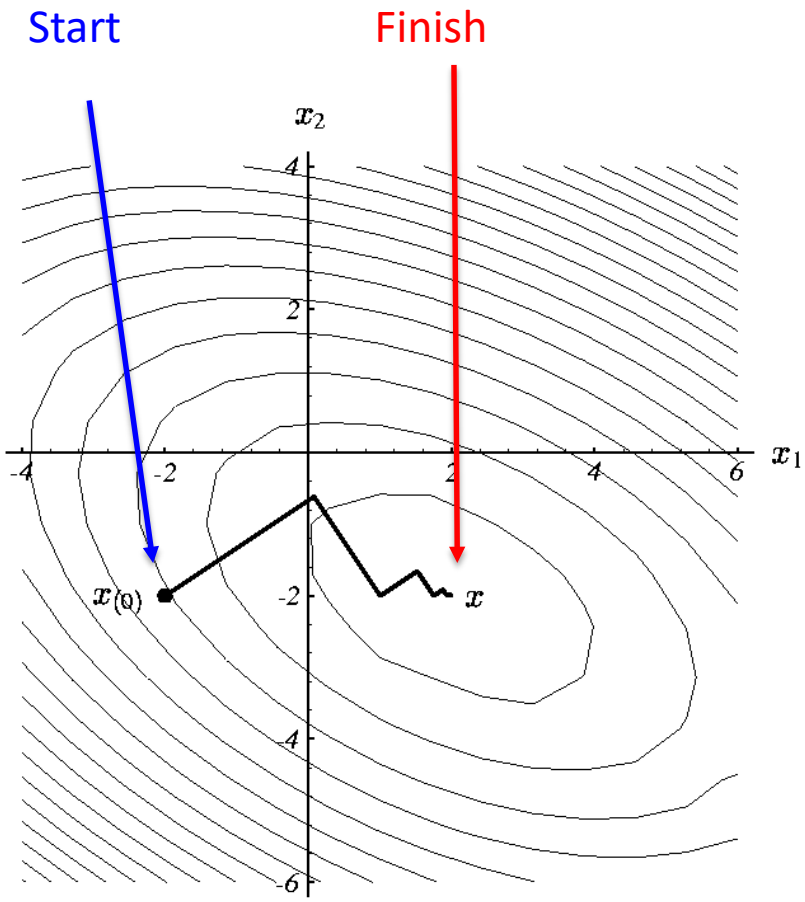
End While

Note there is only one matrix, vector multiply per iteration

SD Example

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

iter	x_1	x_2	r_1	r_2	α
0	0.000	0.000	2.000	-8.000	0.243
1	0.410	-1.639	4.048	1.012	0.205
2	1.393	-1.393	0.607	-2.429	0.243
3	1.517	-1.890	1.229	0.307	0.205
4	1.816	-1.816	0.184	-0.737	0.243
5	1.853	-1.967	0.373	0.093	0.205
6	1.944	-1.944	0.056	-0.224	0.243
7	1.955	-1.990	0.113	0.028	0.205
8	1.983	-1.983	0.017	-0.068	0.243



Steepest Descent Code

```
%% Steepest Descent example
A=[3 2;2 6]
b=[2 -8]'
x=[0 0]'
r = b;
normVal=Inf;
itr = 0;
tol = 0.1e-1;
%% Algorithm%%
while normVal>tol
    xold=x;
    y = A*r;
    alpha  = (r'*r)/(r'*y);
    x = x + alpha*r;
    r = r - alpha* y;
    itr=itr+1;
    normVal=abs(xold-x);
end
```

Additional Material on Machine Learning Applications of Gradient Descent

The following material is to give you an overview of some aspects of gradient descent use in machine learning. The material is not related to any of the assignments

Improvements in Gradient Descent

- Back-tracking is used to make sure that the function decreases at each step
- The gradient may be evaluated at different points – this is Nesterov's method
- The Lasso Method may be used to make sure the solution is in a valid solution space
- This is a very fast moving and active research area right now
- These ideas are all discussed in Linear Algebra and Learning from Data Gilbert Strang MIT Press 2019

Stochastic Gradient Descent

Widely used in deep learning

Only randomly makes use of parts of the residual

Very slow but reliable …. Eventually get to a solution

Original method – Robbins and Munro Annals of Math. Stats vol.22 1951 pp400-407

Modern GPU architectures make large amounts of computation routine

Stochastic Gradient Descent

- Machine learning: usually minimizing the in-sample loss (training loss)

$$\min_{\boldsymbol{w}}\{\frac{1}{N}\sum_{n=1}^{N}\ell(\boldsymbol{w}^T\boldsymbol{x}_n, y_n)\} := E_{\text{in}}(\boldsymbol{w}) \text{ (linear model)}$$

$$\min_{\boldsymbol{w}}\{\frac{1}{N}\sum_{n=1}^{N}\ell(h_{\boldsymbol{w}}(\boldsymbol{x}_n), y_n)\} := E_{\text{in}}(\boldsymbol{w}) \text{ (general hypothesis)}$$

  ℓ: loss function (e.g., $\ell(a,b) = (a-b)^2$)

- Gradient descent:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \underbrace{\nabla E_{\text{in}}(\boldsymbol{w})}_{\text{Main computation}}$$

- In general, $E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\boldsymbol{w})$,

  each $f_n(\boldsymbol{w})$ only depends on $(\boldsymbol{x}_n, y_n)$

Stochastic Gradient Descent

- Gradient:

$$\nabla E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(\boldsymbol{w})$$

- Each gradient computation needs to go through all training samples

  slow when millions of samples

- Faster way to compute "approximate gradient"?

Stochastic Gradient Descent

- Gradient:

$$\nabla E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(\boldsymbol{w})$$

- Each gradient computation needs to go through all training samples

  slow when millions of samples

- Faster way to compute "approximate gradient"?
- Use stochastic sampling:
  - Sample a small subset $B \subseteq \{1, \cdots, N\}$
  - Estimate gradient

$$\nabla E_{\text{in}}(\boldsymbol{w}) \approx \frac{1}{|B|}\sum_{n\in B}\nabla f_n(\boldsymbol{w})$$

  $|B|$: batch size

Stochastic Gradient Descent

- Input: training data $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$
- Initialize $\boldsymbol{w}$ (zero or random)
- For $t = 1, 2, \cdots$
  - Sample a small batch $B \subseteq \{1, \cdots, N\}$
  - Update parameter

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta^t \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\boldsymbol{w})$$

Stochastic Gradient Descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers
- If $\boldsymbol{w}^*$ is the minimizer, $\nabla f(\boldsymbol{w}^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(\boldsymbol{w}^*) = 0$,

$$\text{but} \quad \frac{1}{|B|}\sum_{n\in B} \nabla f_n(\boldsymbol{w}^*) \neq 0 \quad \text{if } B \text{ is a subset}$$

(Even if we got minimizer, SGD will move away from it)

- To make SGD converge:

  Step size should decrease to 0

  $\eta^t \to 0$

  Usually with polynomial rate: $\eta^t \approx t^{-a}$ with constant a

Stochastic Gradient Descent as Used Today

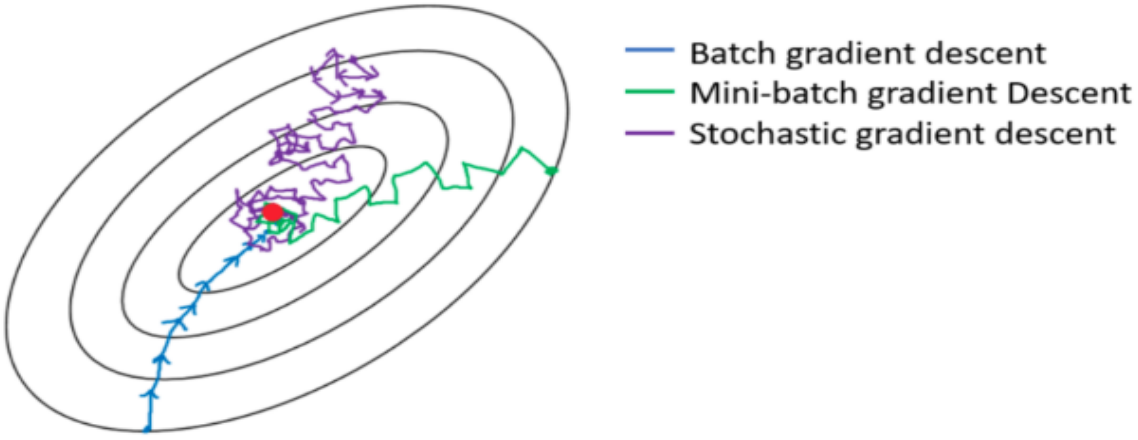
- Gradients are computed using back propagation – a previously computed gradient is used. – ADAGRAD and ADAM methods
- Instead of least squares objective function – hinge loss or cross entropy loss are used
- Randomized Kaczmarz method is a very efficient update method.
- Averaging values over multiple iterations holds promise, - Stocastiv Weight Averaging SWA.
- Again this is a very fast moving and active research area right now

- These ideas are all discussed in Linear Algebra and Learning from Data Gilbert Strang MIT Press 2019

Gradient Descent vs Stochastic Gradient Descent

Stochastic gradient descent:

- pros:
  - cheaper computation per iteration
  - faster convergence in the beginning
- cons:
  - less stable, slower final convergence
  - hard to tune step size



(Figure from https://medium.com/@ImadPhd/gradient-descent-algorithm-and-its-variants-10f652806a3)

Main Material restarts here

Conjugate Direction Methods

An improvement over the steepest descent is to take the exact number of steps using a set of search directions and obtain the solution after *n* such steps

This is the basic idea in the conjugate direction methods

Image compares steepest descent with a conjugate direction approach

An example of a conjugate directions type Method is the Conjugate Gradient method

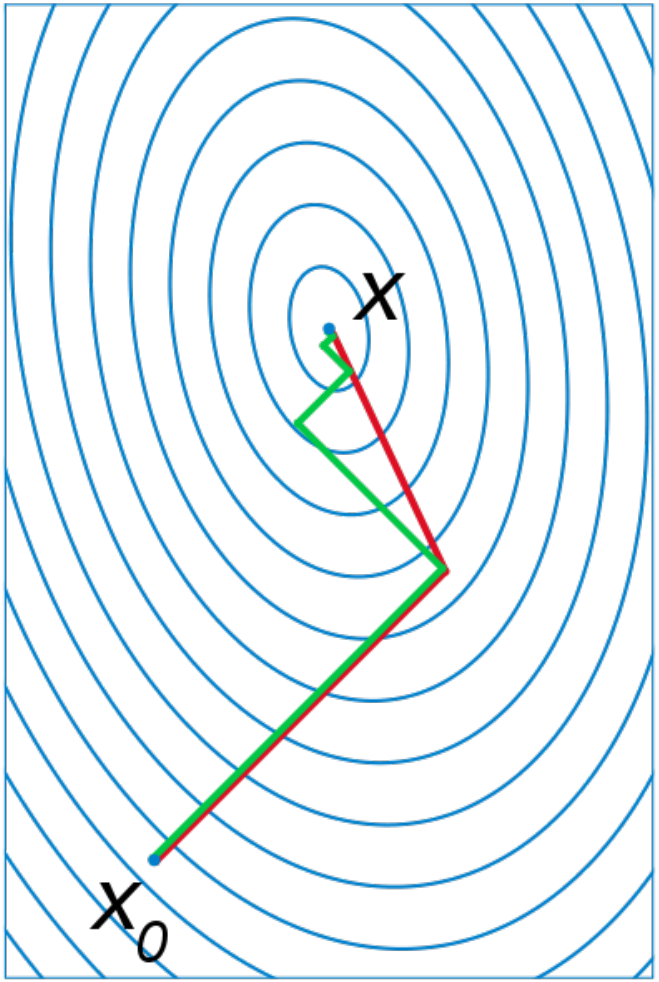
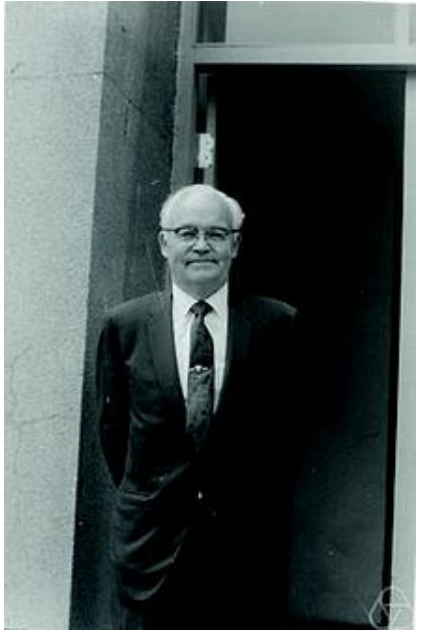


Image Source: http://en.wikipedia.org/wiki/File:Conjugate_gradient_illustration.svg

Conjugate Gradient Method



Magnus Hestenes

A is positive definite if for every nonzero vector v and its transpose v^T, the product $v^T Ax > 0$

If *A* is symmetric and positive definite, then the function

$$q(v) = \frac{1}{2} v^T Av - v^T b + c$$

has a unique minimizer that is solution to $Av = b$

Conjugate gradient is an iterative method that solves $Av = b$ by minimizing $q(x)$.

Minimization involves picking orthogonal search directions

$$d_k$$

Conjugate Gradient Algorithm

Pick v_1

Let $r_1 = b - Av_1$ first residual

$d_1 = r_1$ pick first direction

For $k = 1, 2, 3 \ldots., n$

$q_k = Ad_k$

$\alpha_k = \dfrac{r_k . r_k}{d_k . q_k}$ pick distance along search direction

$v_{k+1} = v_k + \alpha_k d_k$ new value

$r_{k+1} = r_k - \alpha_k q_k$ new residual

$\beta_k = \dfrac{r_{k+1} . r_{k+1}}{r_k . r_k}$

$d_{k+1} = r_{k+1} + \beta_k d_k$ new search direction

end

```
function x=cgm(A,b,sol)
% set CGM parameters
tol=1000*eps;
nm=length(b);
 x=zeros(nm,1);
Tic
 % start iteration
r=b-A*x; d=r;
rr=dot(r,r);
counter=0; err=1;
while err>tol
   counter=counter+1;
   iter(counter)=counter;
   if counter==1
      beta=0;
   else
      beta=rr/rr0;
   end;
   d=r+beta*d;
   q=A*d;
   alpha=rr/dot(d,q);
   x=x+alpha*d;
   r0=r;
   r=r-alpha*q;
   rr0=rr;
   rr=dot(r,r);
   error2(counter)=norm(alpha*d,inf);
   error3(counter)=norm(r,inf);
   error(counter)=norm(x-sol,inf);
   err=error(counter);
end;
toc
```

“Simple” Example Code

Conjugate Gradient Example

- Using the same system as before, let

$$\mathbf{A} = \begin{bmatrix} 10 & -5 & -4 \\ -5 & 12 & -6 \\ -4 & -6 & 10 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 10 \\ -20 \\ 15 \end{bmatrix} \quad \text{We are solving for } \mathbf{x} = \begin{bmatrix} 3.354 \\ 1.645 \\ 3.829 \end{bmatrix}$$

- Select i=0, $\mathbf{x}^{(0)} = \mathbf{0}$, $\varepsilon = 0.1$, then $\mathbf{r}^{(0)} = \mathbf{b}$
- With i = 0, $\mathbf{d}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b}$

Conjugate Gradient Example

$$\alpha^{(0)} = \frac{(\boldsymbol{d}^{(0)})' \boldsymbol{r}^{(0)}}{(\boldsymbol{d}^{(0)})' \boldsymbol{A}\, \boldsymbol{d}^{(0)}} = 0.0582$$

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha^{(0)}\mathbf{d}^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + 0.0582 \times \begin{bmatrix} 10 \\ -20 \\ 15 \end{bmatrix} = \begin{bmatrix} 0.582 \\ -1.165 \\ 0.873 \end{bmatrix}$$

$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha^{(0)}\mathbf{A}\mathbf{d}^{(1)} = \begin{bmatrix} 10 \\ -20 \\ 15 \end{bmatrix} - 0.0582 \times \begin{bmatrix} 10 & -5 & -4 \\ -5 & 12 & -6 \\ -4 & -6 & 10 \end{bmatrix} \begin{bmatrix} 10 \\ -20 \\ 15 \end{bmatrix} = \begin{bmatrix} 1.847 \\ 2.129 \\ 1.606 \end{bmatrix}$$

$$i = i + 1 = 1$$

This first step exactly matches Steepest Descent

Conjugate Gradient Example

With i=1 solve for $\beta^{(1)}$

$$\beta^{(2)} = \frac{\left[\mathbf{r}^{(1)}\right]^T \mathbf{r}^{(1)}}{\left[\mathbf{r}^{(0)}\right]^T \mathbf{r}^{(0)}} = \frac{10.524}{725} = 0.01452$$

$$\mathbf{d}^{(2)} = \mathbf{r}^{(1)} + \beta^{(2)}\mathbf{d}^{(1)} = \begin{bmatrix} 1.847 \\ 2.128 \\ 1.606 \end{bmatrix} + 0.01452 \times \begin{bmatrix} 10 \\ -20 \\ 15 \end{bmatrix} = \begin{bmatrix} 1.992 \\ 1.838 \\ 1.824 \end{bmatrix}$$

Then

$$\alpha^{(1)} = \frac{(\boldsymbol{d}^{(1)})' \boldsymbol{r}^{(1)}}{(\boldsymbol{d}^{(1)})' \boldsymbol{A}\, \boldsymbol{d}^{(1)}} = \frac{725}{12450} = 1.388$$

Conjugate Gradient Example

- And

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha^{(1)}\mathbf{d}^{(2)} = \begin{bmatrix} 0.582 \\ -1.165 \\ 0.873 \end{bmatrix} + 1.388 \times \begin{bmatrix} 1.993 \\ 1.838 \\ 1.824 \end{bmatrix} = \begin{bmatrix} 3.348 \\ 1.386 \\ 3.405 \end{bmatrix}$$

$$\mathbf{r}^{(2)} = \mathbf{r}^{(1)} - \alpha^{(1)}\mathbf{A}\mathbf{d}^{(2)} = \begin{bmatrix} 1.847 \\ 2.129 \\ 1.606 \end{bmatrix} - 1.388 \times \begin{bmatrix} 10 & -5 & -4 \\ -5 & 12 & -6 \\ -4 & -6 & 10 \end{bmatrix} \begin{bmatrix} 1.993 \\ 1.838 \\ 1.824 \end{bmatrix} = \begin{bmatrix} -2.923 \\ 0.532 \\ 2.658 \end{bmatrix}$$

$$i = 1 + 1 = 2$$

Conjugate Gradient Example

- With i=2 solve for $\beta^{(2)}$

$$\beta^{(2)} = \frac{\left[\mathbf{r}^{(2)}\right]^T \mathbf{r}^{(2)}}{\left[\mathbf{r}^{(1)}\right]^T \mathbf{r}^{(1)}} = \frac{15.897}{10.524} = 1.511$$

$$\mathbf{d}^{(2)} = \mathbf{r}^{(2)} + \beta^{(2)}\mathbf{d}^{(1)} = \begin{bmatrix} -2.924 \\ 0.531 \\ 2.658 \end{bmatrix} + 1.511 \times \begin{bmatrix} 1.992 \\ 1.838 \\ 1.824 \end{bmatrix} = \begin{bmatrix} 0.086 \\ 3.308 \\ 5.413 \end{bmatrix}$$

- Then

$$\alpha^{(2)} = \frac{(\boldsymbol{d}^{(2)})' \boldsymbol{r}^{(2)}}{(\boldsymbol{d}^{(2)})' \boldsymbol{A}\, \boldsymbol{d}^{(2)}} = 0.078$$

Conjugate Gradient Example

- And

$$\mathbf{x}^{(3)} = \mathbf{x}^{(2)} + \alpha^{(2} \mathbf{d}^{(3)} = \begin{bmatrix} 3.348 \\ 1.386 \\ 3.405 \end{bmatrix} + 0.783 \times \begin{bmatrix} 0.086 \\ 3.308 \\ 5.413 \end{bmatrix} = \begin{bmatrix} 3.354 \\ 1.646 \\ 3.829 \end{bmatrix}$$

$$\mathbf{r}^{(3)} = \mathbf{r}^{(2)} - \alpha^{(2)} \mathbf{A}\mathbf{d}^{(3)} = \begin{bmatrix} -2.923 \\ 0.532 \\ 2.658 \end{bmatrix} - 0.783 \times \begin{bmatrix} 10 & -5 & -4 \\ -5 & 12 & -6 \\ -4 & -6 & 10 \end{bmatrix} \begin{bmatrix} 0.086 \\ 3.308 \\ 5.413 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$i = 2 + 1 = 3$$

Done in 3 = n iterations!

A comparison against Steepest Descent

iter	x_1	x_2	x_3	r_1	r_2	r_3	α
0	0.000	0.000	0.000	10.000	-20.00	15.00	0.111
1	0.582	-1.165	0.873	1.847	2.129	1.606	0.058
2	2.487	1.030	2.530	0.398	-4.746	5.830	1.031
3	2.511	0.745	2.880	0.135	0.892	0.717	0.060
4	2.545	0.972	3.062	1.656	-0.565	0.392	0.254
5	2.680	0.926	3.094	0.210	0.848	0.336	0.081

Lots of iterations here

68	3.352	1.644	3.827	0.002	-0.002	0.003	0.192
69	3.353	1.644	3.827	0.001	0.002	0.001	0.078
70	3.353	1.644	3.827	0.002	-0.001	0.003	0.192
71	3.353	1.644	3.828	0.001	0.002	0.001	0.078
72	3.353	1.644	3.828	0.002	-0.001	0.002	0.192

In general SD need $O(\kappa(A))$ iterations where $O(\kappa(A))$ is the condition number of A while CG needs only the square root of this.

Preconditioned Conjugate Gradients

One problem is that the Conjugate Gradient method often fails on real problems. The solution is to use a preconditioner to transform the system to one that is more easily solved.

A simple approach uses $M^{-1}Ax = M^{-1}b$

Choices for Preconditioners

- Use Jacobi Method diagonal Matrix D as M inverse is straightforward
- Use Gauss Seidel Iteration

Preconditioned Conjugate Gradient Algorithm

Pick v_1

Let $r_1 = (b - Av_1)$ first residual

$d_1 = M^{-1} r_1$ pick first direction

For $k = 1, 2, 3 \ldots., n$

$q_k = Ad_k$

$$\alpha_k = \frac{r_k . M^{-1} r_k}{d_k . q_k}$$

$v_{k+1} = v_k + \alpha_k d_k$ new value

$r_{k+1} = r_k - \alpha_k q_k$ new residual

$$\beta_k = \frac{r_{k+1} . M^{-1} r_{k+1}}{r_k . M^{-1} r_k}$$

$d_{k+1} = M^{-1} r_{k+1} + \beta_k d_k$ new search direction

end

Note

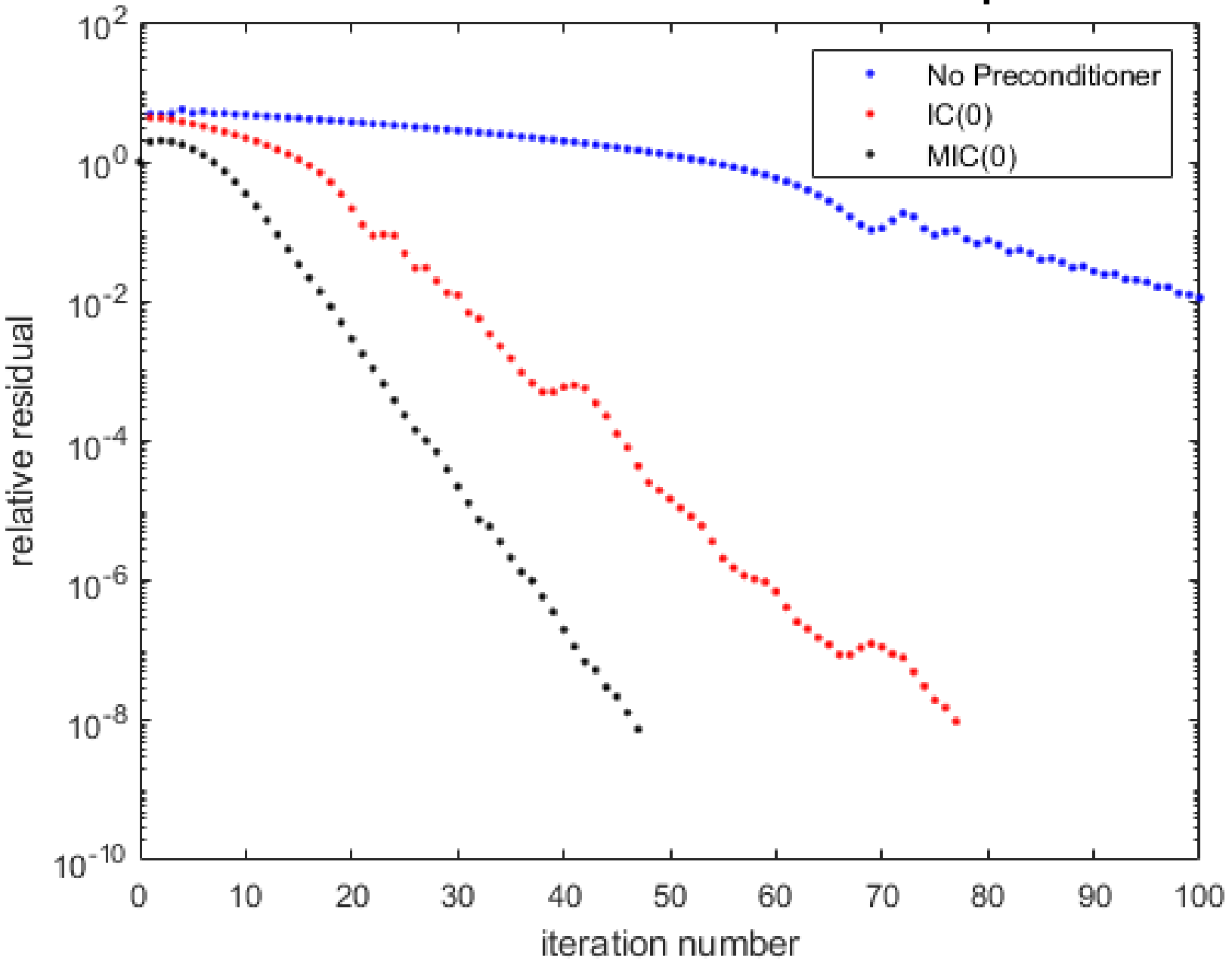
One each step only need

one new evaluation of M^{-1}

to calculate $M^{-1} r_{k+1}$

which is saved for future steps

Effect of Preconditioners Matlab Example



Summary

Iterative methods when they work well may use many fewer floating point operations than direct methods and much less storage

Convergence is not assured and the methods may stall.

Solutions to this include preconditioners

Preconditioners may be very problem specific.

Nevertheless with the best iterative methods today we can solve very large systems on the largest parallel machines.
Conjugate Gradient and gradient descent approaches are very important in data and AI applications .