

AMSC 607 / CMSC 764 Advanced Numerical Optimization  
Fall 2001  
UNIT 4: Special Topics  
PART 2: Large-Scale Problems  
Dianne P. O'Leary  
©2001

---

### Large-Scale Optimization Problems

#### The plan:

- Reminders of what you already know about this
- The ubiquitous subproblem: the linear system in Newton's method
- Solving linear systems
  - Sparse iterative methods: preconditioning
  - Sparse direct methods
- Handling sparsity in linear programming
  - Sparsity in the Simplex algorithm
  - Sparsity in IPMs for LP

**Warning:** These notes are a bit sketchy. **So ask questions if they don't make sense!**

---

#### Reminders of what you already know about this

We'll consider a problem with  $n$  variables to be **large scale** if we cannot afford to store a matrix of size  $n \times n$  unless we can take advantage of its **sparsity**.

---

#### Large-Scale unconstrained optimization

We have already studied a lot of algorithms for handling large-scale unconstrained optimization problems:

- nonlinear conjugate gradient algorithm.
  - truncated Newton.
  - sparse quasi-Newton.
- 

#### The ubiquitous subproblem

Note that whether we solve constrained or unconstrained problems, whenever we employ **Newton's method**, we obtain a system of **linear equations** to solve. So we need to exploit sparsity in the **matrix** of this equation.

---

## Solving linear systems

---

### Sparse iterative methods: preconditioning

We already know a pretty good way to solve sparse linear equations: **the (linear) conjugate gradient algorithm**.

---

### The conjugate gradient method for linear systems

Suppose we want to solve the linear system

$$Ax = b$$

where  $A$  is a **symmetric positive definite** matrix.

Recall that we only need the matrix in forming **matrix-vector** products; in contrast to direct methods, **we never modify the matrix**.

1. Let  $x^{(0)}$  be an initial guess.  
Let  $r^{(0)} = b - Ax^{(0)}$  and  $p^{(0)} = Mr^{(0)}$ .
2. For  $k = 0, 1, 2, \dots$ , until convergence,
  - (a) Compute the search parameter  $\alpha_k$  and the new iterate and residual

$$\begin{aligned}\alpha_k &= \frac{r^{(k)T} M r^{(k)}}{p^{(k)T} A p^{(k)}} \\ x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)}, \\ r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)},\end{aligned}$$

- (b) Compute the new search direction

$$\begin{aligned}\beta_k &= \frac{r^{(k+1)T} M r^{(k+1)}}{r^{(k)T} M r^{(k)}}, \\ p^{(k+1)} &= M r^{(k+1)} + \beta_k p^{(k)},\end{aligned}$$

End For.

---

### How much work?

So the work-per-iteration is quite low for cg:  
 $O(n)$ , if matrix-vector product by  $A$  and by  $M$  is  $O(n)$ .

But we must ensure that we don't take a lot of steps.

In particular, if  $n$  is large, then knowing that cg terminates with the exact solution in  $n$  steps is no help at all.

We need a good approximation in a very small number of iterations.

### How can we achieve this?

---

### Choosing preconditioners

- For fast iterations, we want to be able to apply  $M$  very quickly .
- To make the number of iterations small, we want  $M$  to be an approximate inverse of  $A$ .

An approximate inverse is effective if **the eigenvalues of  $MA$  fall into a small number of small clusters**

In practice, we often aim for  
 $MA = I + (\text{a matrix of small rank}) + (\text{a matrix of small norm})$ .

---

### Some common choices of $M^{-1}$

- $M^{-1} =$  the diagonal of  $A$ .
- $M^{-1} =$  a banded piece of  $A$ .
- $M^{-1} =$  an incomplete factorization of  $A$ , leaving out inconvenient elements.

- $M^{-1}$  = a related matrix; e.g., if  $A$  is a discretization of a differential operator,  $M$  might be a discretization of a related operator that is easier to solve.
- $M$  might be the matrix  $B$  from our favorite stationary iterative method (SIM).

---

### A spectrum

We see that there is an entire spectrum of methods here, ranging from

- $M = I$ , giving a **purely iterative method**, to
- $M = A^{-1}$ , giving a **direct method**.

We need to choose an efficient alternative from this range of options.

---

### Before we leave iterative methods ...

#### What if $A$ fails to be symmetric and positive definite?

There are still preconditioned iterative methods, relatives of cg, that can be applied. Two examples:

- `Symmlq` of Paige and Saunders, if  $A$  is symmetric but not positive definite.
- `Gmres` if  $A$  is not symmetric.

#### References:

- Anne Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM Press, 1997.
- R. Barrett et al, *Templates for the Solution of Linear Systems*, SIAM, 1995, with implementations at Netlib.

---

### Sparse direct methods

#### A motivating Unquiz:

- Consider the Cholesky (or LU) factors of the matrix

$$\begin{bmatrix} x & x & x & x & x & x \\ x & x & 0 & 0 & 0 & 0 \\ x & 0 & x & 0 & 0 & 0 \\ x & 0 & 0 & x & 0 & 0 \\ x & 0 & 0 & 0 & x & 0 \\ x & 0 & 0 & 0 & 0 & x \end{bmatrix}.$$

- Now suppose we reorder the rows from bottom to top, and also reorder the columns from last to first:

$$\begin{bmatrix} x & 0 & 0 & 0 & 0 & x \\ 0 & x & 0 & 0 & 0 & x \\ 0 & 0 & x & 0 & 0 & x \\ 0 & 0 & 0 & x & 0 & x \\ 0 & 0 & 0 & 0 & x & x \\ x & x & x & x & x & x \end{bmatrix}.$$

Does this have any effect on the number of nonzeros in the Cholesky factors?

- If you could compute the QR factors of this second matrix, you would find that they have less sparsity than the LU.
- Also note that the **inverse** of each of these sparse matrices is **totally full**.
- If we replace one column of this matrix by another (as we would in the simplex method), the **sparsity** of the triangular factors can **change** rather completely!
- What happens if the main diagonal entries are all of order  $10^{-5}$  while the entries in the last row are of order 1?

□

---

### Some basic conclusions:

- The order of the equations and unknowns is **crucial** to maintaining **sparsity** in the factors.
- Unfortunately, if the matrix is not **symmetric positive definite**, then the order is also crucial to the **stability** of the triangular factorization.
- **Never** try to store the inverse matrix on a sparse problem.
- **QR** is generally more dense than **LU**, **but reordering can be done solely to maintain sparsity**, since **stability** is guaranteed.
- IPMs have the great advantage that the **sparsity** of the matrix **stays the same** from iteration to iteration. The simplex algorithm does not have this nice property.

Let's think about the simplex algorithm first, and then consider a way to solve IPMs.

---

## Handling sparsity in linear programming

---

### Sparsity in the Simplex algorithm

---

#### Standard programs ...

Many standard algorithms for the simplex method update the LU factors of the basis.

They do this by a variant of the Sherman-Morrison formula, equivalent to **no pivoting for stability**.

If they detect trouble (iterative refinement fails to converge), they throw away the current factorization and recompute. This is called **reinversion**.

This way of doing things is **popular** but not what a numerical analyst would want.

---

#### More enlightened programs...

More enlightened programmers update the LU factors of the basis using **pivoting**, even if this hurts sparsity some.

#### References:

- P. E. Gill, G. H. Golub, W. Murray, and M.A. Saunders, "Methods for Modifying Matrix Factorizations", *Mathematics of Computation* 28 (1974) 505–535
- P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981, Chapter 4.

---

## Sparsity in IPMs for LP

**Reference:** Weichung Wang and Dianne P. O'Leary, "Adaptive Use of Iterative Methods in Predictor-Corrector Interior Point Methods for Linear Programming," *Numerical Algorithms* 25 (2000) 387-406.

**Problem:** How can we effectively use iterative methods in IPMs?

---

### The linear programming problem

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b, \\ & && x \geq 0, \end{aligned}$$

$$\begin{aligned} c, x &\in \mathcal{R}^n \\ b &\in \mathcal{R}^m, \\ A &\in \mathcal{R}^{m \times n}, \text{ rank } m, \quad m < n. \end{aligned}$$

---

### The Foundation of Interior Point Methods

Barrier problem

$$\min c^T x - \mu \sum \ln x_i, \quad Ax = b.$$

Soln  $x(\mu)$  converges to lp solution as  $\mu \rightarrow 0$ .

Lagrangian:

$$f(x, y) = c^T x - \mu \sum \ln x_i + y^T (b - Ax)$$

Setting derivatives to zero

$$\begin{aligned} Ax &= b \\ c - \mu X^{-1}e - A^T y &= 0. \end{aligned}$$

Now let  $z = \mu X^{-1}e$  ( $> 0$ ).

The **central path** (barrier trajectory) is defined by

$$\begin{aligned} XZe - \mu e &= 0, \\ Ax - b &= 0, \\ A^T y + z - c &= 0. \end{aligned}$$

---

So at each iteration we want to solve the Newton system

Newton's method:

$$\begin{bmatrix} Z & 0 & X \\ A & 0 & 0 \\ 0 & A^T & I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \mu e - XZe \\ b - Ax \\ c - A^T y - z \end{bmatrix}.$$

Eliminating  $\Delta z$  gives the **KKT (Karush-Kuhn-Tucker) system**:

$$\begin{bmatrix} X^{-1}Z & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = \begin{bmatrix} \mu X^{-1}e - c + A^T y \\ b - Ax \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$$

And eliminating  $\Delta x$  gives the **normal equations**,

$$(AD^2A^T)\Delta y = s_2 - AD^2s_1.$$

with  $D^2 = Z^{-1}X$ .

- Normal eqns matrix is positive definite and symmetric, smaller ( $m \times m$ ), and more dense.
- KKT matrix is symmetric indefinite and more sparse.

---

### Previous Use of Iterative Methods

Several for KKT system.

For normal equations:

- Gill, Murray, Saunders, Tomlin, Wright 1986
- Goldfarb and Mehrotra 1988
- Karmarkar and Ramakrishnan 1991
- Mehrotra 1992
- Carpenter and Shanno 1993
- Nash and Sofer 1993
- Portugal, Resende, Veiga, and Júdice 1994
- Mehrotra and Wang 1995

Limited success.

- Approximate solutions allowed early in the Newton iterations but can fail when iterates are near the boundary.

- $D$  changes quite rapidly and becomes highly ill-conditioned in the final iterations.

---

### Characteristics of Direct Methods

Assume that the columns of  $A$  have been permuted to improve sparsity in the Cholesky factor of  $AD^2A^T$ .

- Direct methods rely on sparse Cholesky factorization of  $AD^2A^T$  as  $LPL^T$ , where  $L$  is a unit lower triangular matrix and  $P$  is a diagonal matrix.
- Iterative refinement used as necessary.
- Dense columns handled separately.

#### Disadvantages:

- Failure of iterative refinement if  $AD^2A^T$  is very ill-conditioned.
- Fill-in.
- Form and refactor  $AD^2A^T$  each Newton iteration.

---

### Characteristics of Iterative Methods

For definiteness, preconditioned conjugate gradient method.

Work per PCG iteration:

- one product of  $AD^2A^T$  with a vector,
- one solution of a linear system involving preconditioner,
- several vector operations.

Advantages:

- somewhat better stability,
- low storage,
- accuracy requirements in the beginning phase quite low.

**Crucial issue: find an effective preconditioner.**

---

### The Preconditioners

**1 : Cholesky factorization.**

**2 : QR decomposition.** (mathematically identical)

**3 : Cholesky factorization of sparse part.**

$$A = [A_S, A_D], \quad A_S D_S^2 A_S^T = L_S P_S L_S^T$$

Preconditioned matrix is the identity plus a rank  $k$

**4 : Incomplete factorization.**

**5 : Updated Cholesky factorization.**

$$\begin{aligned} A \hat{D}^2 A^T &= A D^2 A^T + A \Delta D A^T \\ &= L P L^T + \sum_{i=1}^n \Delta d_{ii} a_i a_i^T \end{aligned}$$

---

### The Interior Point Algorithm

**Initialize**  $k \leftarrow 1$ ;  $\mu_0 > 0$ ;  $x_0, y_0, z_0 > 0$ .

**while** (not convergent)

Solve the normal equations using a direct or iterative solver.

Update the variables:

$$\begin{aligned} x_{k+1} &\leftarrow x_k + \alpha_p \Delta x; \\ y_{k+1} &\leftarrow y_k + \alpha_d \Delta y; \\ z_{k+1} &\leftarrow z_k + \alpha_d \Delta z. \end{aligned}$$

**Choose**  $\mu_{k+1} < \mu_k$ .

**Set**  $k \leftarrow k + 1$ .

**end while**

---

### The Preconditioned CG Solver

**Determine the preconditioner:**

```

if (prev_cost > .8 × drct_cost) or (drct_cost < pred_cost)
then
    Form the matrix  $AD^2A^T$ .
    Factor  $AD^2A^T$  to get the preconditioner.
else
    Perform updt_nمبر rank-one updates to get the new
    preconditioner.
end if

```

### Solve the linear system:

```

if (the diagonal of the preconditioner is singular)
then use the direct method.
else Iterate the PCG method:
    pcg_itn ← 0
    while (not convergent)
        Execute a PCG iteration.
        if (pcg_itn > max_pcg_itn) then
            Factor  $AD^2A^T$  to reinitialize the
            preconditioner.
            Restart the PCG iteration.
        end if
    end while
end if

```

---

### Determining the Preconditioner

Determine whether to update the current preconditioner or refactor. Base the decision on the cost of the preceding iteration, including the cost of updates.

$$\text{prev\_cost} = (\text{updt\_cost} \times \text{updt\_nمبر}) + (\text{pcgi\_cost} \times \text{pcgi\_nمبر}) + (\text{overhead}),$$

- If previous **cost** was **high**, we **reinitialize** the preconditioner.

$$\text{prev\_cost} > .8 \times \text{drct\_cost}.$$

- If previous **cost was not high**, base the decision on a **prediction** of the cost of the current iteration:

Fit a straight line to the number of iterations required to determine two preceding search directions.

$$\text{pred\_cost} = (\text{updt\_cost} \times \text{updt\_nمبر}) + (\text{pcgi\_cost} \times \text{predi\_nمبر}).$$

---

### The Adaptive Updating Strategy

Update the Cholesky factors using the `updt_nbmr` **largest** outer product matrices as determined by  $|\Delta d_{ii}|$ .

Change the number of Cholesky updates adaptively over the course of the algorithm in order to improve efficiency.

---

### The PCG Convergence Test

We start from an initial guess of zero, and iterate until the computed residual norm is less than  $\varepsilon_{pcg}$  times the norm of the right-hand side.

$$\varepsilon_{pcg} = \begin{cases} 10^{-8}, & \text{if } \text{relgap} > 10^{-2}; \\ 10^{-8} \times (\text{relgap})^{\frac{1}{2}}, & \text{otherwise,} \end{cases}$$

---

### Numerical Results

We modified the code OB1-R to adaptively choose the linear system solver, and we performed numerical experiments comparing the results of this modified version of OB1-R to the standard OB1-R code of Lustig, Marsten, and Shanno.

---

### Computational Results for the Larger Test Problems from NETLIB.

| Problem  | Iter. | Time     |          |      |
|----------|-------|----------|----------|------|
|          |       | OB1-R    | Adp      | Diff |
| 80bau3b  | 78    | 46.15    | 48.32    | -2   |
| d2q06c   | 55    | 257.13   | 253.25   | 4    |
| d6cube   | 77-78 | 113.90   | 100.52   | 13   |
| degen3   | 30    | 66.22    | 65.57    | 1    |
| df1001   | 98    | 19844.37 | 16644.35 | 3200 |
| fit2d    | 54    | 46.80    | 47.85    | -1   |
| greenbea | 52    | 52.03    | 54.30    | -2   |
| greenbeb | 74    | 69.15    | 72.12    | -3   |
| maros-r7 | 29    | 1952.93  | 1414.20  | 539  |
| pilot    | 77    | 485.08   | 441.42   | 44   |
| pilot87  | 82    | 1948.82  | 1584.77  | 364  |
| stocfor3 | 87    | 142.22   | 157.28   | -15  |
| truss    | 30    | 19.55    | 22.22    | -3   |
| wood1p   | 18    | 12.95    | 13.77    | -1   |
| woodw    | 37    | 25.30    | 27.65    | -2   |

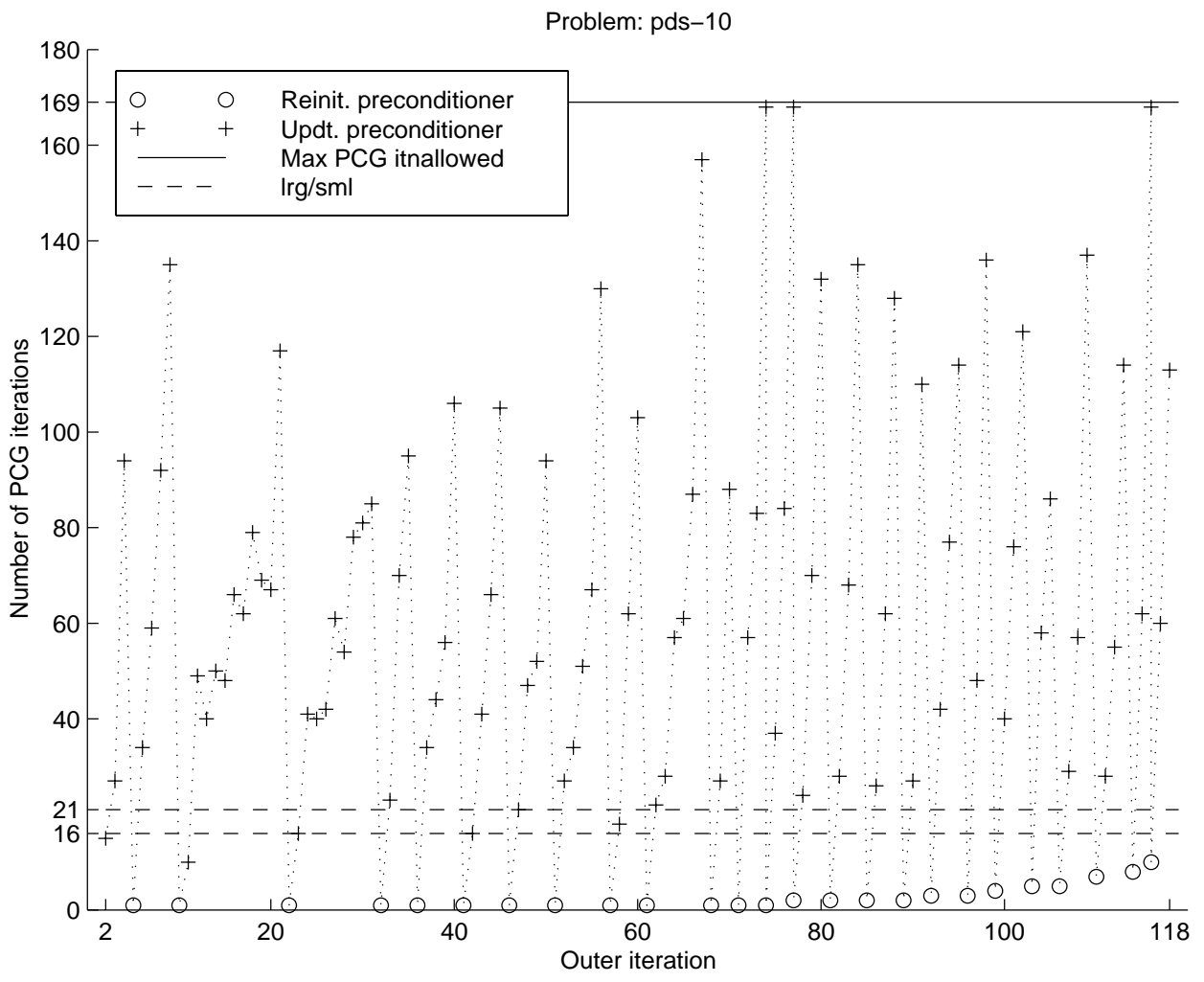


Figure 1: Number of PCG iterations for the adaptive algorithm

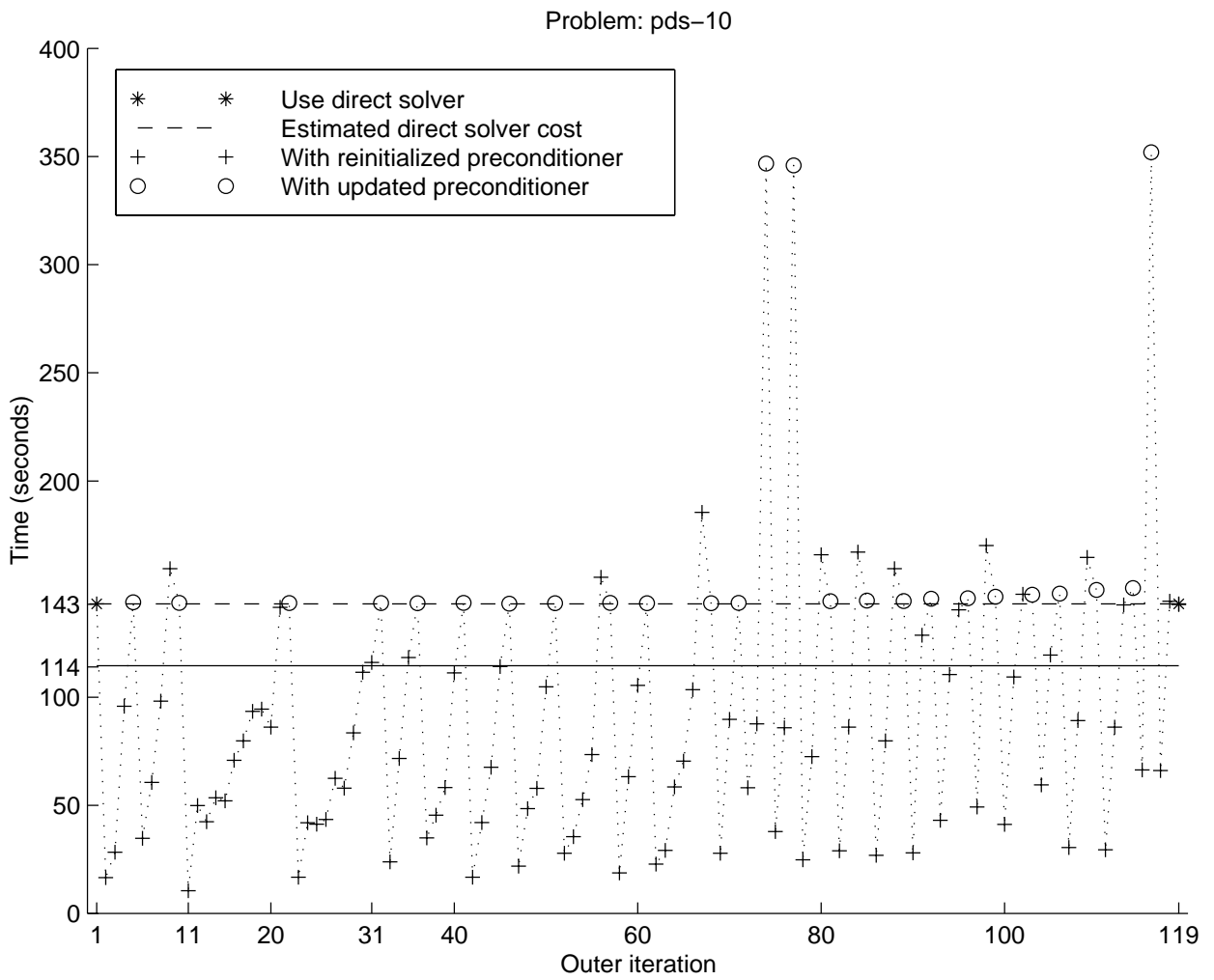


Figure 2: Timing performance for the adaptive algorithm

---

**The Network Problems.**

| Problem | LP size & nonzeros |         |       | Density |     |
|---------|--------------------|---------|-------|---------|-----|
|         | Row/Node           | Col/Arc | Nzros | $AA^T$  | $L$ |
| NET0102 | 999                | 2000    | 3999  | .00     | .08 |
| NET0104 | 1000               | 4000    | 8000  | .01     | .23 |
| NET0108 | 1000               | 8000    | 16000 | .02     | .41 |
| NET0116 | 1000               | 16000   | 32000 | .03     | .56 |
| NET0408 | 4000               | 8000    | 16000 | .00     | .07 |
| NET0416 | 4000               | 16000   | 32000 | .00     | .22 |

| Problem | Iter.   | Time     |         |      |
|---------|---------|----------|---------|------|
|         |         | OB1-R    | Adp     | Diff |
| NET0102 | 43 - 40 | 34.02    | 32.58   | 1    |
| NET0104 | 41 - 41 | 171.23   | 135.52  | 36   |
| NET0108 | 43 - 43 | 461.05   | 335.58  | 125  |
| NET0116 | 58 - 59 | 1005.77  | 718.75  | 287  |
| NET0408 | 43 - 42 | 2099.43  | 1371.17 | 728  |
| NET0416 | 53 - 53 | 16674.13 | 9265.85 | 7408 |

---

**Some advantages of this algorithm**

- **Decisions have been automated:**
  - direct or iterative solver,
  - reinitialize or update the preconditioner,
  - how many updates to apply.
- **Performance** of interior point algorithms on large sparse problems has been **enhanced**.
- Our **preconditioning** strategy is **based on recomputing or updating** the previous preconditioner.
- Open questions:
  - effective termination criteria for the iterative method.
  - a block implementation of the matrix updating and downdating to reduce overhead.
  - the end game.

---

## Final Words

- For more information about iterative methods for solving linear systems, take AMSC/CMSC 666.
- For more information about direct methods for sparse linear systems, take AMSC 600 / CMSC 760.
- The Wang-O'Leary idea of tuning the algorithm to the architecture has been used (independently) in other contexts:
  - FFTW (Matteo Frigo and Steven G. Johnson)
  - Atlas dense linear system software (Jack Dongarra)