Computational NETLIB experience with a dense projected gradient sagitta method

Ángel Santos-Palomo[a,*] Pablo Guerrero-García[a]

[a] *Department of Applied Mathematics, University of Málaga, 29071 Málaga, Spain*

Abstract

Computational results obtained when solving a subset of NETLIB problems by using a dense projected gradient implementation of the non-simplex active-set sagitta method presented in [12] are summarized. Two different addition rules for its initial phase are considered and, for each problem solved, two corresponding graphs are reported to illustrate the variations of the objective value along the active-set path. The comparison of our code for the sagitta method versus MATLAB code `linprog` shows that this sagitta method outperforms the simplex method in number of iterations and reliability and can be competitive in overall speed.

Key words: linear programming, non-simplex active-set method, projected gradient
1991 MSC: 90C05, 65K05

1 Preliminaries

Let us consider the usual unsymmetric primal-dual pair of linear programs using a non-standard notation (we have deliberately exchanged the usual roles of b and c, x and y, n and m, and (P) and (D), as in e.g. [9, §2]):

$$(P) \quad \min \ \ell(x) \doteq c^T x \ , \ x \in \mathbb{R}^n \qquad\qquad (D) \quad \max \ \mathcal{L}(y) \doteq b^T y \ , \ y \in \mathbb{R}^m$$
$$\text{s.t. } A^T x \geq b \qquad\qquad\qquad\qquad \text{s.t. } Ay = c \ , \ y \geq \mathbf{0}$$

* Corresponding author.
Email addresses: `santos@ctima.uma.es` (Ángel Santos-Palomo), `pablito@ctima.uma.es` (Pablo Guerrero-García).
URL: `http://www.satd.uma.es/matap/personal/pablito/` (Pablo Guerrero-García).

Preprint submitted to Optimization Online *14 September 2005*

where $A \in \mathbb{R}^{n \times m}$ with $m \geq n$. The condition $c \neq \mathbf{0}$ is added. (In this work the dimension of the null vector $\mathbf{0}$ depends on the context, and $\| \cdot \|$ denotes the euclidean vector norm.)

The original sagitta method (see [12,15] and also [6]) is a non-simplex active-set method that starts without any iteration point, selects successive foreactive or active sets (in [12] the foreactive-set term is used because initially the constraints in this set do not have to be active constraints) and, as long as possible, computes corresponding null-space descent directions, because the method attempts to find a *direction of the primal feasible region* (see definition in [9, p. 48] or in [5, p. 82]). An iteration point is computed if, for the current active set, there is not a corresponding null-space descent direction. Then, while violated constraints exist, a primal-feasibility search loop is carried out. When suitable strategies are used, the first primal feasible point obtained by the method is generally an optimal solution; but, if it does not occur, the process restarts. The method convergence is not theoretically guaranteed and it is not possible to rule out the cycling possibility. We point out that modifications of the sagitta method with guaranteed convergence under nondegeneracy assumption have been recently presented in [16,18].

In our opinion, the original sagitta method has some features which would entail a revision of some old topics with regard to non-simplex active-set methods. Thus:

- It works with active-set techniques and, if $\mathcal{A}$ is the foreactive or active set, then $\mid \mathcal{A} \mid < n$ for a large percentage of the iterative process (unlike simplex-type methods for which always $\mid \mathcal{A} \mid = n$); it even can end up with a non-square optimal solution, i.e. with $\mid \mathcal{A}_* \mid < n$ for $\mathcal{A}_*$ being a subset of the set $\mathcal{A}(x^*)$ of active constraints at an optimal solution x^* for the primal P.
- Using a global viewpoint in an unusual initial phase, it starts without any iteration point and attempts to find a descent direction of the primal feasible region or, alternatively, a solution of the system $Ay = c$.
- Once a —generally non-square and sometimes infeasible— solution of $Ay = c$ is obtained, the search of primal feasibility is triggered even if dual feasibility has not been achieved yet (unlike two-phase (simplex or non-simplex) active-set methods). Whereas dual feasibility is maintained once achieved, it does not occur so with primal feasibility. (Accurately speaking, the method carries out a simultaneous search of optimality and feasibility because it assembles primal and dual feasibility searchs.)
- It works with the original linear program thorought the whole process, with no artificial variables in contrast to usual two-phase (simplex or non-simplex) active-set methods.

As we detail in [17], the sagitta methods can be implemented, using projected-gradient techniques. Then, if the subscript j denotes the iteration counter and $P_1 = I$ for I being the identity matrix, the null-space descent direction used in the initial phase of the method is

$$d^{(j)} = P_j(-c), \quad (1)$$

where

$$P_j = I - A_{\mathcal{A}}(A_{\mathcal{A}}^T A_{\mathcal{A}})^{-1} A_{\mathcal{A}}^T = I - A_{\mathcal{A}} A_{\mathcal{A}}^\dagger$$

is the orthogonal projector onto the null-space of $A_{\mathcal{A}}^T$ and $A_{\mathcal{A}}^\dagger$ is the *Moore-Penrose pseudoinverse* of $A_{\mathcal{A}}$, with the active-set matrix $A_{\mathcal{A}}$ of full column rank. This null-space descent direction (1) is a *steepest-descent* direction [4, pp. 377–378]. When the active-set matrix $A_{\mathcal{A}}$ is such that $d^{(j)} = \mathbf{0}$, the system $A_{\mathcal{A}}\mu = c$ is compatible and it is possible to compute a dual point $y^{(j)}$, and also a primal point

$$x^{(j)} = A_{\mathcal{A}}(A_{\mathcal{A}}^T A_{\mathcal{A}})^{-1} b_{\mathcal{A}} = (A_{\mathcal{A}}^\dagger)^T b_{\mathcal{A}}, \quad (2)$$

for $b_{\mathcal{A}}$ being the subvector of b corresponding to the indices in the current active-set. This point $x^{(j)}$ is the minimum-norm solution of the underdetermined system $A_{\mathcal{A}}^T x = b_{\mathcal{A}}$.

Update formulae for the current primal and dual points and current steepest-descent direction are used in our dense implementation of the original sagitta method (see [17]). This implementation is based on the QR factorization of the matrix $A_{\mathcal{A}}$, using the classical Gram-Schmidt method with reorthogonalization [1, §2.4]. Details and computational considerations of practical interest are included in [17].

The aim of this paper is to summarize the computational results obtained when solving a subset of NETLIB problems by using the aforementioned dense implementation of the original sagitta method. This problem subset —limited in accordance with the capabilities of the system used to obtain the computational results— is selected to test the sagitta method on small and medium practical instances. The viability of a sparse implementation of the original sagitta method on top of the static data structure of a Cholesky factor of $P^T A^T A P$, where P is a permutation matrix determined for $P^T A^T A P$ to have a Cholesky factor as sparse as possible, was established in [6,14]. Details and computational results using this sparse implementation are also included in [6,7,17].

The paper is organized as follows. In section 2 we give details of our computational experience, whereas the computational results are summarized in section 3. The comparison of computational results obtained using the same sagitta method but with different addition rule at start in its initial phase is carried out in section 4 and final remarks are included in section 5. Additional tables of numerical results and illustrative graphs are furnished in the appendix.

2 Details of our computational experience

Theoretical and practical issues of our dense implementation of the sagitta method are detailed in [17]. Based on them, in this section we state precise details of our computational experience.

A) Index sets: The foreactive or active set $\mathcal{A}_j$ and its complement set are implemented as unsorted vectors of indices in our dense implementation of the sagitta method. A constraint index is always added on the right and, then, ties are solved by selecting the leftmost from amongst the tied elements, in accordance with the order maintained in both sets.

B) Addition strategy in the initial phase: The strategy used to add constraints to the current foreactive set is crucial for the behaviour of the initial phase of the sagitta method (see [17]). The practical index set of contrary constraints to the current descent direction is $\mathcal{C} = \{\, i \notin \mathcal{A}_j \mid \; a_i^T d^{(j)} < -\epsilon_c \,\}$, for $\epsilon_c = 1.06n\epsilon$ and $\epsilon \approx 2.22 \times 10^{-16}$ being the machine epsilon. We have considered in this computational experience the following two plain addition strategies:

- **Addition strategy A** (*Most-Obtuse-Angle Rule*).- A most contrary constraint to the current null-space descent direction $d^{(j)}$, that is to say,

$$p = \arg\min_{i \in \mathcal{C}} \left\{ \frac{a_i^T d^{(j)}}{\| \, a_i \, \|} \right\}.$$

- **Addition strategy B** (*Corrected Sagitta Rule*).- Amongst the contrary constraints to the current null-space descent direction $d^{(j)}$, a most contrary to the "arrow" $-c$, that is to say,

$$p = \arg\min_{i \in \mathcal{C}} \left\{ \frac{-a_i^T c}{\| \, a_i \, \|} \right\};$$

  but if $\cos(a_p, \; d^{(j)}) > -tol_1$, where $tol_1 = 0.01$, then the most-obtuse-angle rule is used to select a new constraint p again.

We note that ties are broken according to the order of the constraint indices in $\mathcal{C}$ and that, when a restarting event occurs, the addition strategy A is the only one used.

C) Dependency check: We determine that a_p is in the range-space of $A_\mathcal{A}$ if the following inequality is satisfied,

$$\| a_p - A_\mathcal{A}\eta_\mathcal{A} \| \leq \epsilon_r(1+ \| a_p \|),$$

where $\eta_\mathcal{A}$ is the least-squares solution of the system $A_\mathcal{A}\eta = a_p$ and $\epsilon_r = \sqrt{\epsilon}$ in our computational experience.

D) Addition strategy in the primal-feasibility-search loop: We select the incoming constraint by using the Brown–Koopmans rule, namely the normalized version of the classical Dantzig or textbook rule,

$$p = \arg\min \left\{ \frac{r_i(x^{(j)})}{\| a_i \|} | \ r_i(x^{(j)}) < -\epsilon_P, \ i \notin \mathcal{A}_j \right\},$$

where $\epsilon_P = \sqrt{\epsilon}$ in our computational experience.

E) Min-ratio rule: An exchange occurs in the primal-feasibility-search loop if a_p is in the range-space of $A_\mathcal{A}$. In this case, the leaving constraint is selected by using the min-ratio rule; but, in accordance with the practical considerations detailed in [17], we use different primal iterations in the same loop depending on the suitable implementation of this min–ratio. Let us denote with $y^{(j)}$ and $\delta^{(j)}$ the dual vectors whose elements are zeros barring those corresponding to the respective solutions of the compatible systems $A_\mathcal{A}\mu = c$ and $A_\mathcal{A}\eta = a_p$, with $\mathcal{S}_1 \doteq \{i \in \mathcal{A}_j \mid \delta_i^{(j)} \geq \epsilon_D\}$ and with $\mathcal{V}_D \doteq \{i \in \mathcal{A}_j \mid y_i^{(j)} < -\epsilon_D\}$, where $\epsilon_D = \sqrt{\epsilon}$ in our computational experience. Thus, when $\mathcal{S}_1 \neq \varnothing$:

- If dual feasibility is not achieved, i.e. $\mathcal{V}_D \neq \varnothing$,

$$q = \arg\min \left\{ \frac{y_i^{(j)}}{\delta_i^{(j)}} | \ i \in \mathcal{S}_1 \quad \text{if } \mathcal{S}_2 = \varnothing, \text{ or } i \in \mathcal{S}_2 \quad \text{if } \mathcal{S}_2 \neq \varnothing \right\}$$

  where $\mathcal{S}_2 \doteq \{i \in \mathcal{A}_j \mid \delta_i^{(j)} \geq tol_2\}$ with $tol_2 = 0.001$ in our computational experience.
- Once dual feasibility is achieved, i.e. $\mathcal{V}_D = \varnothing$,

$$q = \begin{cases} \arg\min \left\{ \dfrac{y_i^{(j)}}{\delta_i^{(j)}} \mid i \in \mathcal{S}_1 \right\} & \text{if } \mathcal{N} = \varnothing \\ \arg\max \left\{ \delta_i^{(j)} \mid i \in \mathcal{N} \right\} & \text{if } \mathcal{N} \neq \varnothing \quad \text{and} \ \max\left\{\delta_i^{(j)} \mid i \in \mathcal{N}\right\} > \epsilon_D \\ \arg\min \left\{ \dfrac{y_i^{(j)}}{\delta_i^{(j)}} \mid i \in \mathcal{S}_1 \backslash \mathcal{N} \right\} & \text{if } \mathcal{N} \neq \varnothing \quad \text{and} \ \max\left\{\delta_i^{(j)} \mid i \in \mathcal{N}\right\} \leq \epsilon_D \end{cases}$$

  where $\mathcal{N} \doteq \{i \in \mathcal{S}_1 \mid \quad -\epsilon_D \leq y_i^{(j)} \leq \epsilon_D\}$.

F) Restarting: If the primal-feasibility-search loop ends because a primal feasible point is reached but the optimality condition $y \geq \mathbf{0}$ is not satisfied by the associated dual point $y^{(j)}$, then the method restarts after deleting a single constraint of the current active set $\mathcal{A}_j$ that is selected according to the following plain deletion strategy:

$$q = \mathcal{A}_j(k) = \arg\min \left\{ y_i^{(j)} \mid y_i^{(j)} < -\epsilon_D,\ i \in \mathcal{A}_j \right\}.$$

Possibly, when a restarting event occurs, the steepest-descent direction (1) is not null for several successive iterations and, then, the current dual point $y^{(j)}$ cannot be computed because the corresponding system $A_{\mathcal{A}}\mu = c$ is incompatible. However we have computed the objective value by using the minimum-norm solution (2) of the current system $A_{\mathcal{A}}^T x = b_{\mathcal{A}}$, and such value has been used in the graphs of the objective function. Note that in its initial phase, at start, the sagitta method does not compute neither primal or dual points, nor objective value.

3 Computational results

As an illustrative sample of the performance of the sagitta method we summarize the computational results obtained for the duals of 36 NETLIB problems [3] —the first 36 smallest problems in which neither BOUNDS nor RANGES sections occur and with less than 10000 nonzeros— using the dense projected-gradient implementation described in [17] and in accordance with the details pointed out in the previous section.

All the test problems have been solved with our code without previous scaling nor preprocessing; they have been read as linear programs in standard form and then dualized to obtain a problem P [6, §5.3]. The computational results were obtained with an Intel Pentium IV at 3.00 GHz with 512MB RAM, using MATLAB release 14 and **interpreted** code, at least with regard to our source code.

We have compared our code against the MATLAB code `linprog` of the MATLAB Optimization Toolbox release 3.0. This code `linprog` solves linear programs by using, in accordance with the documentation of the distributor, three different options:

- **Simplex-on** option, where the code `linprog` uses the two-phase simplex algorithm, with the same preprocessing steps as in the large-scale option.
- **Medium-scale** option, where the code `linprog` uses an active set projection method which is a variation of the well-known simplex method for linear programming. The algorithm finds an initial feasible solution by first solving another linear programming problem.
- **Large-scale** option, where the code `linprog` uses a primal-dual interior-

point method, based on (compiled) LIPSOL which is a variant of Mehrotra's predictor-corrector algorithm. A number of preprocessing steps occur before the algorithm begins to iterate.

TABLE 1. Computational results when solving NETLIB *problems by using* MATLAB *code* `linprog` *with Simplex-on Option*

#	*Name*	*n*	*m*	*Optimal value*	*Iter*	*Time*	*MinRes*
1	AFIRO	27	51	4.647531428571E+2	32	0.05	-5.6E-17
2	SC50B	50	78	7.000000000000E+1	48	0.09	-2.2E-16
3	SC50A	50	78	6.457507705856E+1	75	0.13	-5.6E-17
4	SC105	105	163	5.220206121171E+1	187	0.42	-1.1E-16
6	ADLITTLE	56	138	-2.254949631623E+5	205	0.61	-3.7E-10
7	SCAGR7	129	185	2.331389824331E+6	113	0.30	-2.5E-12
8	STOCFOR1	117	165	3.8296896695E+21(*)	0	0.16	-1.2E+20
9	BLEND	74	114	3.081214984583E+1	68	0.45	-3.6E-15
10	SC205	205	317	5.220206121171E+1	362	1.38	-2.5E-16
12	SHARE2B	96	162	4.157322407414E+2	198	0.53	-4.2E-14
14	LOTFI	153	366	2.526470606188E+1	1140	3.98	-3.3E-16
15	SHARE1B	117	253	7.733802172937E+4	338	1.63	-1.7E+01
17	SCORPION	388	466	-1.878124822738E+3	399	1.92	-3.1E+02
19	SCAGR25	471	671	1.475343306077E+7	1160	7.34	-1.2E-10
20	SCTAP1	300	660	-1.412250000000E+3	2930	19.97	-5.7E-14
22	BRANDY	220	303	-1.600244678227E+3	2915	15.48	-1.1E+03
23	ISRAEL	174	316	8.966448218630E+5	550	3.72	-4.8E-13
26	SCSD1	77	760	-8.666666674333E+0	291	4.17	-3.7E-09
28	AGG	488	615	3.1701910085E+10(‡)	0	60.25	-2.6E+01
29	BANDM	305	472	2.0347063101E+3(‡)	0	48.44	-9.6E+01
30	E226	223	472	-1.037028095E+1(‡)	0	48.47	-2.9E+01
31	SCFXM1	330	600	2.6713249014E+4(‡)	0	90.47	-3.3E+00
34	SCRS8	490	1275	-9.042969538008E+2	980	15.42	-2.3E-12
35	BEACONFD	173	295	-3.398085982900E+4	118	0.23	-2.4E+00
40	DEGEN2	444	757	1.435178000000E+3	3318	57.41	-5.3E-14
41	AGG2	516	758	2.816416170E+9(‡)	0	22.89	-7.8E+01
42	AGG3	516	758	9.9449537076E+19(*)	0	1.00	-1.7E+02
43	SCSD6	147	1350	-5.050000007714E+1	2669	65.92	-3.4E-09
44	SHIP04S	402	1506	-1.798714700445E+6	602	4.03	-1.9E-12
48	BNL1	643	1586	1.552547771E+0(‡)	0	33.98	-2.8E-01
50	SCFXM2	660	1200	6.9085787326E+4(‡)	0	117.34	-3.4E+00
53	FFFFF800	524	1028	-7.351927600E+18(†)	11	0.89	-1.0E+16
54	SHIP04L	402	2166	-1.793324537970E+6	887	6.48	-5.1E-12
55	SCTAP2	1090	2500	-1.724807142857E+3	3505	76.97	-2.1E-14
57	SHIP08S	778	2735	-1.920098210535E+6	977	13.72	-5.5E-12
59	SCFXM3	990	1800	6.9085787325E+4(‡)	0	154.66	-3.4E+00

Notes: (*) No feasible point was found. (†) Problem is unbounded.
(‡) Number of iterations exceeded in Phase 1; increase options. (MaxIter=10000)

However, a comparison of computational results using this large-scale option would not be fair because our source code is interpreted. Moreover, it is well-known [8, p. 10] that simplex methods outperform interior-point ones when the problem is not large enough.

In tables below we use a first column labeled # to hold a number to recognize each NETLIB problem. This number was assigned to each NETLIB problem by Bixby in [2], according to the number of its nonzeros. The name, the number n of variables and the number m of constraints of each NETLIB problem solved are also given in Table 1.

TABLE 2. Computational results when solving NETLIB *problems by using* MATLAB *code* `linprog` *with Medium-Scale Option*

#	*Optimal value*		*Iter*	*Time*	*MinRes*
1	4.647531428571E+2		36	0.45	-3.5E-15
2	7.000000000000E+1		48	0.16	-2.6E-14
3	6.457507705854E+1		49	0.13	-2.3E-13
4	5.220206121095E+1		153	1.17	-1.4E-11
6	-2.254869494679E+5		114	0.36	-1.5E-11
7	2.331389647646E+6		298	6.47	-2.8E-05
8	4.113197621944E+4		134	0.86	-5.7E-13
9	3.081214983612E+1		101	2.19	-8.6E-10
10	5.220206115002E+1		312	9.33	-9.6E-10
12	4.157322407411E+2		185	1.86	-3.2E-11
14	2.523476221909E+1		320	8.06	-2.3E-06
15	7.658931857919E+4		275	2.75	-5.7E-12
17	-1.878124822738E+3		394	64.45	-5.1E-12
19	-1.542545322695E+8	(†)	1124	665.66	-7.4E+04
20	-1.412250000000E+3		649	129.61	-3.8E-12
22	0.000000000000E+0	(‡)	273	6.84	0.0E+00
23	8.966579621649E+5		519	8.67	-2.1E-07
26	-8.666666674333E+0		126	2.09	-8.0E-15
28	3.599176728660E+7		678	231.91	-2.9E-10
29	-3.925389270571E+9	(†)	1378	349.36	-6.8E+08
30	5.589897323428E+0	(†)	557	109.08	-5.3E+00
31	-4.374337958081E+9	(†)	888	416.20	-2.0E+07
34	-9.042969538008E+2		956	2717.89	-2.0E-12
35	-3.359248580720E+4		270	9.00	-1.6E-14
40	1.206130292277E+3	(†)	941	1787.64	-1.5E+02
41	2.023925235569E+7		585	41.23	-1.7E-07
42	-1.031359656446E+7		606	79.13	-8.6E-02
43	-5.050000007714E+1		356	6.13	-1.1E-14
44	-1.798714700445E+6		639	337.00	-4.7E-11
48	-1.977629561541E+3		2985	10401.17	-9.8E-10
50	-1.499191537245E+15	(†)	2020	6812.83	-2.4E+12
53	-5.556795648331E+5		1894	4393.23	-6.7E-08
54	-1.793324537970E+6		763	676.67	-1.1E-11
55	-1.724807142857E+3		1160	7269.59	-3.2E-13
57	-1.920098210535E+6		839	3034.77	-4.5E-11
59	-1.194681939107E+21	(†)	2908	27973.06	-9.7E+18

Notes: (†) The problem is badly conditioned; the solution may not be reliable.
(‡) Exiting: the search direction is close to zero; the problem is ill-posed.

Tables 1–2 sum up the computational results obtained using MATLAB code `linprog` with simplex-on or medium– scale option, respectively. Total number of iterations and running time required to solve each problem are displayed in two columns labeled *Iter* and *Time*, along with two additional columns (labeled *Optimal value* and *MinRes*, respectively) with the computed optimal value of the objective and the minimum element of the residual vector at the optimal solution obtained. The code `linprog` warns about its difficulties to solve (eleven or eight, according to the option used) NETLIB problems by issuing displayed notes, which are incorporated as a footnote.

TABLE 3. Computational results for the Sagitta Method when solving NETLIB *problems by using Most-Obtuse-Angle Rule at start*

#	*Optimal value*	*Its*	*Scs*	*MinRes*	$\lvert \mathcal{A}_* \rvert$	*IPh*	*R*	*%Itb*
1	4.647531428571E+2	23	0.03	-1.8E-15	20	7	0	0.0
2	7.000000000000E+1	67	0.08	0.0E+00	48	5	0	0.0
3	6.457507705856E+1	64	0.08	-1.2E-16	49	17	0	0.0
4	5.220206121171E+1	141	0.30	-1.9E-16	104	33	0	0.0
6	-2.254949631624E+5	153	0.31	-3.8E-12	56	39	0	41.8
7	2.331389824331E+6	188	0.61	-8.9E-13	129	113	3	16.5
8	4.113197621943E+4	127	0.16	-3.1E-13	117	98	0	8.7
9	3.081214984583E+1	127	0.25	0.0E+00	74	8	0	28.3
10	5.220206121171E+1	312	2.03	-6.5E-17	203	60	0	0.0
12	4.157322407414E+2	258	0.63	-4.8E-13	96	47	5	26.7
14	2.526470606237E+1	313	1.78	-9.4E-15	153	85	0	29.4
15	7.658931857919E+4	228	0.80	-9.8E-11	117	117	0	49.1
17	-1.878124822738E+3	382	4.84	-1.4E-12	336	260	1	0.0
19	1.475343306077E+7	694	23.03	-8.2E-12	448	419	0	0.0
20	-1.412250000000E+3	511	9.05	-4.0E-11	279	223	2	0.0
22	-1.518509896488E+3	504	3.67	-3.1E-13	170	159	0	0.0
23	8.966448218631E+5	401	2.83	-1.7E-11	174	171	0	25.9
26	-8.666666674333E+0	123	0.52	-1.3E-08	77	7	0	14.6
28	3.599176728658E+7	563	14.47	-2.6E-09	486	459	1	0.0
29	1.586280184499E+2	754	13.42	-1.4E-12	304	288	0	0.0
30	1.875192906630E+1	831	9.02	-2.8E-14	213	139	0	0.0
31	-1.841675902836E+4	554	10.50	-3.4E-13	320	273	0	0.0
34	-9.042969538008E+2	1164	57.38	-3.6E-10	479	154	1	0.0
35	-3.359248580720E+4	141	0.39	-4.2E-14	122	122	0	0.0
40	1.435178000000E+3	4211	245.08	-4.4E-14	440	309	0	0.0
41	2.023925235598E+7	607	21.02	-1.5E-11	516	485	0	7.2
42	-1.031211593509E+7	602	20.33	-1.1E-12	516	486	0	5.3
43	-5.050000007714E+1	564	7.86	-1.2E-08	147	43	0	20.9
44	-1.798714700445E+6	560	24.41	-8.2E-12	325	300	24	0.0
48	0.000000000E+0(†)	0	0.00	0.0E+00	0	0	0	0.0
50	-3.666026156503E+4	1291	99.47	-5.2E-12	649	548	0	0.0
53	-5.556795648175E+5	764	31.92	-2.9E-09	470	336	6	0.0
54	-1.793324537970E+6	661	36.05	-3.1E-12	323	299	16	0.0
55	-1.724807142857E+3	1053	167.61	-4.8E-12	869	777	2	0.0
57	-1.920098210535E+6	742	79.47	-4.5E-12	503	463	14	0.0
59	-5.490125454861E+4	1930	325.98	-6.7E-10	974	822	0	0.0
Total		21608	1215.34			8171		

Note: (†) Number of iterations exceeded.

Tables 3–4 sum up the computational results obtained for the original sagitta method, using the same initial phase with the *most-obtuse-angle rule* or the *corrected sagitta rule*, respectively, to choose the incoming constraint. In these tables, apart from the five columns labeled #, *Optimal value*, *Iter*, *Time* and *MinRes* (displaying the Bixby's number and the corresponding computational results obtained with the respective rule), other columns with additional numerical information of interest are included, on which we comment now:

TABLE 4. *Computational results for the Sagitta Method when solving* NETLIB *problems by using Corrected Sagitta Rule at start*

#	*Optimal value*	*Its*	*Scs*	*MinRes*	$\mid \mathcal{A}_* \mid$	*IPh*	*R*	*%Itb*
1	4.647531428571E+2	25	0.03	-2.0E-14	22	9	0	0.0
2	7.000000000000E+1	67	0.06	0.0E+000	48	5	0	0.0
3	6.457507705856E+1	59	0.06	-2.1E-18	47	25	0	0.0
4	5.220206121171E+1	128	0.22	-1.3E-18	99	50	0	0.0
6	-2.254949631624E+5	161	0.30	-2.3E-11	56	41	0	42.2
7	2.331389824331E+6	197	0.59	-9.1E-13	129	115	0	12.2
8	4.113197621944E+4	132	0.17	-3.3E-13	117	101	0	11.4
9	3.081214984583E+1	126	0.23	0.0E+000	74	12	0	31.0
10	5.220206121171E+1	282	1.80	-5.6E-18	199	95	0	0.0
12	4.157322407414E+2	258	0.64	-4.8E-13	96	47	5	26.7
14	2.526470605672E+1	317	1.80	-1.7E-14	153	86	0	23.7
15	7.658931857919E+4	228	0.83	-9.8E-11	117	117	0	49.1
17	-1.878124822738E+3	375	4.72	-1.2E-12	339	278	0	0.0
19	1.475343306077E+7	969	36.25	-1.0E-11	448	421	0	0.0
20	-1.412250000000E+3	457	7.75	-2.9E-10	281	227	8	0.0
22	-1.518509896488E+3	504	3.25	-3.1E-13	170	159	0	0.0
23	8.966448218631E+5	402	2.69	-6.1E-11	174	172	0	16.7
26	-8.666666674333E+0	161	0.72	-1.2E-14	77	43	0	20.5
28	3.599176728665E+7	582	15.64	-4.6E-09	486	461	1	0.0
29	1.586280184071E+2	1008	19.19	-6.7E-14	304	297	0	0.0
30	1.875192906629E+1	818	8.70	-4.8E-13	214	139	0	0.0
31	-1.841675902834E+4	573	11.22	-1.4E-13	319	274	0	0.0
34	-9.042969538008E+2	961	46.00	-3.8E-11	478	154	2	0.0
35	-3.359248580720E+4	141	0.36	-4.2E-14	122	122	0	0.0
40	1.435178000000E+3	11386	702.64	-5.3E-14	442	313	0	0.0
41	2.023925235598E+7	607	20.98	-1.5E-11	516	485	0	7.2
42	-1.031211593509E+7	602	20.38	-1.1E-12	516	486	0	5.3
43	-5.050000007764E+1	503	6.86	-1.4E-08	147	46	0	11.9
44	-1.798714700445E+6	538	21.44	-6.0E-10	327	307	17	0.0
48	0.000000000E+0(†)	0	0.00	0.0E+00	0	0	0	0.0
50	-3.666026156505E+4	1207	90.88	-2.1E-09	647	545	0	0.0
53	-5.556795648176E+5	854	36.69	-2.9E-10	475	346	12	0.0
54	-1.793324537970E+6	602	35.78	-6.0E-10	327	306	7	0.0
55	-1.724807142857E+3	1088	173.89	-1.2E-11	879	792	1	0.0
57	-1.920098210535E+6	727	76.84	-6.8E-12	506	465	7	0.0
59	-5.490125454933E+4	1881	306.39	-1.9E-09	977	814	0	0.0
Total		28926	1655.98			8355		

Note: (†) Number of iterations exceeded.

- Column labeled $\mid \mathcal{A}_* \mid$ shows the cardinal of the final active set $\mathcal{A}_*$, subset of the active set $\mathcal{A}(x^*)$ of active constraints at the computed optimal solution x^*. We can check that $\mid \mathcal{A}_* \mid < n$ for 23 out of the 35 (66.66%) problems solved, or in other words, that such problems are solved by sagitta methods working with basis deficiency (see [10]) throughout the whole process.
- Column labeled *IPh* shows the number of iterations performed in the initial phase, which coincides with the cardinal of the active set at the end of such phase and the beginning of the primal-feasibility-search loop. Note that only

for problem SHARE1B (#15), the cardinal of the active set at the end of the initial phase is equal to n.

- Column labeled *RS* shows the number of restarting events.
- Column labeled *%Itb* shows the percentage of square basis iterations, i.e. iterations with $|\mathcal{A}_j| = n$. It is worth noting that such percentage is zero or less than 50% for all problems solved.

The comparison of the computational results obtained using the sagitta method, most-obtuse-angle (MOA) rule versus corrected sagitta (CS) rule, is deferred to the following section.

TABLE 5. Comparison for code `linprog` *versus original sagitta method when solving* NETLIB *problems*

Method option	*Solved problems*	*Iter*	*Time*	*Opt. sol. quality*
`Linprog` *simplex-on*	25/36	24067	302.36	*Deficient for 4 lp*(†)
`Linprog` *med.-scale*	28/36	15444	29436.33	*Deficient for 4 lp*(‡)
Sagitta MOA rule	35/36	21608	1215.34	*Good*
Sagitta CS rule	35/36	28926	1655.98	*Good*

(†) Problems SHARE1B, SCORPION, BRANDY, BEACONFD.
(‡) Problems SCAGR7, LOTFI, ISRAEL, AGG3.

The comparison of computational results obtained using code `linprog` versus our dense code for the original sagitta method is summarized in tables 5–7. Table 5 shows clearly that the sagitta method **outperforms** `linprog` in reliability, both simplex-on and medium-scale option, solving with good quality 35 out of 36 test problems against 25 or 28 problems solved (four with deficient quality) using code `linprog` with simplex-on or medium-scale option, respectively. Totals of iterations and running time in table 5 are not comparable because they correspond to a different number of problems solved.

TABLE 6. Totals for code `linprog` *simplex-on versus original sagitta method when solving 25* NETLIB *problems*

	`Linprog` *simplex-on*		*Sagitta MOA rule*		*Sagitta CS rule*	
Problem size	*Iter*	*Time*	*Iter*	*Time*	*Iter*	*Time*
Small(17)	9969	55.06	3936	28.19	3888	26.05
Medium(8)	14098	247.30	9649	640.87	16774	1099.70
Total(25)	24067	302.36	13585	669.06	20662	1125.75

The comparison by only taking the problems solved by both codes into account is summarized in tables 6–7. We have distinguished between small problems, those where the number of variables plus the number of constraints (rows plus columns) is less than 1000, and medium problems, where such addition is higher than 1000. Note that the number of variables (n) for medium problems is higher than 400 (except for problem SCSD6), which entails that, when solving a problem of this kind, our dense implementation works with a dense orthonormal matrix of considerable size.

Considering only those problems solved by using the code `linprog` with simplex-on option, table 6 shows that our code sagitta outperforms `linprog` in total of number of iterations, largely if the most-obtuse-angle (MOA) rule is used. However, the code `linprog` outperfoms globally our code sagitta in running time, even though the reverse occurs when considering uniquely small problems. Note, nevertheless, that the code `linprog` with simplex-on option:

- Uses an evolved *preprocessing*.
- Does not compute the multipliers (a fatal error occurs if the multipliers are requested!) and, according to the MATLAB documentation, it "might save some time computationally".
- Computes an optimal solution with deficient quality for some problems.

Moreover, with some bias towards `linprog`, note that in this comparison we have left aside all problems not solved by using `linprog` with simplex-on option. We point out that problems DEGEN2 and BNL1 are degenerate problems (see [3, p. 7]) whose solution raises special difficulties to the sagitta method and, nevertheless, problem DEGEN2 has been included in the comparison. (Problem DEGEN2 can be solved in only 43.30 seconds using a dual-then-primal sagitta method [18].) Summing up, the sagitta method can be competitive in running time with the simplex method.

TABLE 7. Totals for code `linprog` *medium-scale versus original sagitta method when solving 27* NETLIB *problems*

	`Linprog` *medium-scale*		*Sagitta MOA rule*		*Sagitta CS rule*	
Problem size	*Iter*	*Time*	*Iter*	*Time*	*Iter*	*Time*
Small(17)	3983	247.61	3559	24.67	3516	22.97
Medium(10)	8476	18787.55	7280	460.50	7064	454.50
Total(27)	12459	19035.16	10839	485.17	10580	477.47

Considering only those problems solved by using the code `linprog` with medium-scale option except problem BNL1, table 7 shows clearly that our code sagitta outperforms code `linprog` with medium-scale option, slightly in number of iterations but **very largely** (more than 90%!) in running time.

4 Most-obtuse-angle rule versus corrected sagitta rule

Tables A.1–A.35 and Figures 1–35 in the appendix, one for each problem solved, are aimed to complement the computational results obtained by using the sagitta method. Each table or figure collects two blocks of results or two graphs of the objective function value corresponding to the respective use at start of strategy A or B.

Tables in the appendix also complement Tables 3–4 by showing additionally, for each solved problem, the iteration number j, the number of constraints in current working or active set $\mathcal{A}_j$ and the objective value for each of the following events: a) first computed point after the initial phase, b) first feasible dual point, c) first square basis, d) first feasible primal point, and e) computed optimal solution.

The independent variable in the graphs is the iteration counter j of the method. The graphs start with the minimum j for which $d^{(j)} = \mathbf{0}$, that is the iteration from which both a primal and dual point, $x^{(j)}$ and $y^{(j)}$ respectively, are available.

Note that the comparison of the totals of both number of iterations and running time in Tables 3–4, most-obtuse-angle rule versus corrected sagitta rule, shows apparently advantage for the most-obtuse-angle rule; however, problem DEGEN2 (a *stalling* event) is clearly a definite factor in this conclusion. The performance of the sagitta method using both rules is generally similar, even though it can vary with the value of the tolerance tol_1. Globally, in view of all these results, we can point out:

- For the method to obtain the first zero projection of the negative gradient, the rule used at start for which more indices have to be generally added to the working set is the corrected sagitta rule.
- The objective value for the first computed primal point using both rules is often (but not always, see figures 16 –SCTAP1 problem–, 23 –SCRS8 problem– and 33 –SCTAP2 problem–) lower than the optimal objective value.
- A first feasible dual point is obtained generally before, and with fewer constraints in the active set $\mathcal{A}_j$, by using the most-obtuse-angle rule at start.
- If the most-obtuse-angle rule is used at start, the final set $\mathcal{A}_*$ has generally the same or fewer number of constraints.
- Restarting events occur nearly for the same problems, independently of the used rule used.
- The "basic" iterations, namely those with square basis or $|\mathcal{A}_j| = n$, are relatively few, if any, for the problems solved. This implies that the greatest computational effort is done with $|\mathcal{A}_j| < n$.

5 Final remarks

We have used a dense implementation of the sagitta method for solving a set of 36 NETLIB problems. The computational results obtained show that this code sagitta outperfoms the MATLAB code `linprog` (both simplex-on and medium-scale option) in number of iterations and reliability; furthermore it outperfoms code `linprog` with medium-scale option very largely in speed. Although, by comparing the total running time for only those problem solved by using code `linprog` with simplex-on option, its speed is lower than (roughly 50%) that of this code `linprog`, our opinion is that, globally, the code sagitta is competitive in running time with the simplex method. Note that, because of reliability, the sagitta method deserves to be considered a suitable alternative.

We know that the method performance using a sparse implementation is an important matter. We have developed two sparse techniques [13,14] that lead to interesting sparse implementations of the sagitta method with encouraging computational results (see [6,7,17]). We can carry out both a projected or a reduced implementation; but we are still working in the development of a compiled code to be able to compare against alternative commercial implementations of the simplex method. Moreover, the computational results recently obtained by Pan [11] with an sparse implementation of a basis-deficiency-allowing simplex algorithm using an LU-decomposition strengthen our opinion about the competitivity of the non-simplex active-set methods.

References

[1] BJÖRCK, Å. (1996). *Numerical Methods for Least Squares Problems.* SIAM Publications, Philadelphia, USA.

[2] BIXBY, R.E. (1992). Implementing the simplex method: The initial basis. *ORSA J. Computing* 14(3), pp. 670–676.

[3] GAY, D.M. (1985). Electronic mail distribution of linear programming test problems. *COAL Newsletter* 13, 10–12.

[4] GILL, P.E., W. MURRAY and M.H. WRIGHT (1991). *Numerical Linear Algebra and Optimization*, Vol. 1. Addison-Wesley Publishing, Redwood City, California.

[5] GOLDFARB, D., and M.J. TODD (1989). Linear Programming. Chap. II in: NEMHAUSER, G.L., et al. (eds.), *Optimization*, pp. 73–170.

[6] GUERRERO-GARCÍA, P. (2002) *Range-Space Methods for Sparse Linear Programs* (Spanish). Ph.D. thesis, Dept. App. Math., Univ. Málaga, Spain.

[7] GUERRERO-GARCÍA, P., and A. SANTOS-PALOMO (2003). A comparison of three sparse linear program solvers. Submitted for publication.

[8] LUSTIG, I.J., R.E. MARSTEN and D.F. SHANNO (1994). Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing* 6(1), 1–14.

[9] OSBORNE, M.R. (1985). *Finite Algorithms in Optimization and Data Analysis.* Wiley, Chichester.

[10] PAN, P.-Q. (1998). A basis-deficiency-allowing variation of the simplex method for linear programming. *Computers Math. Applic.* 36(3), 33–53.

[11] PAN, P.-Q. (2004). Revised basis-deficiency-allowing simplex algorithm using LU-decomposition for linear programming. Presented at 6th ICOTA Conf.

[12] SANTOS-PALOMO, A. (2004). The *sagitta* method for solving linear programs. *European Journal of Operational Research* 157(3), 527–539.

[13] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2001a). Updating and downdating an upper trapezoidal sparse orthogonal factorization. To be published in IMA Journal of Numerical Analysis.

[14] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2001b). Solving a sequence of sparse least squares problems. Technical Report, Dept. Appl. Math., Univ. Málaga. Submitted for publication.

[15] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2004). A fresh view on the sagitta method. Submitted for publication.

[16] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2005). Sagitta method with guaranteed convergence. Submitted for publication.

[17] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2005). Computational experiences with dense and sparse implementations of the sagitta method. Submitted for publication.

[18] SANTOS-PALOMO, A., and P. GUERRERO-GARCÍA (2005). Dual-then-primal sagitta method. Submitted for publication.

6 Appendix

This appendix contains Tables A.1–A.35 and Figures 1–35 that complement the computational results obtained by using the sagitta method. Each table or figure collects, for each problem solved, two blocks of results or two graphs of the objective function value corresponding to the respective use at start of strategy A or B.

This tables show, for each problem solved, the iteration number j, the number of constraints in current working or active set $\mathcal{A}_j$ and the objective value for each of the following events: a) first computed point after the initial phase, b) first feasible dual point, c) first square basis, d) first feasible primal point, and e) computed optimal solution. The independent variable in the graphs is the iteration counter j of the method. The graphs start with the minimum j for which $d^{(j)} = \mathbf{0}$, that is the iteration from which both a primal and dual point, $x^{(j)}$ and $y^{(j)}$ respectively, are available.

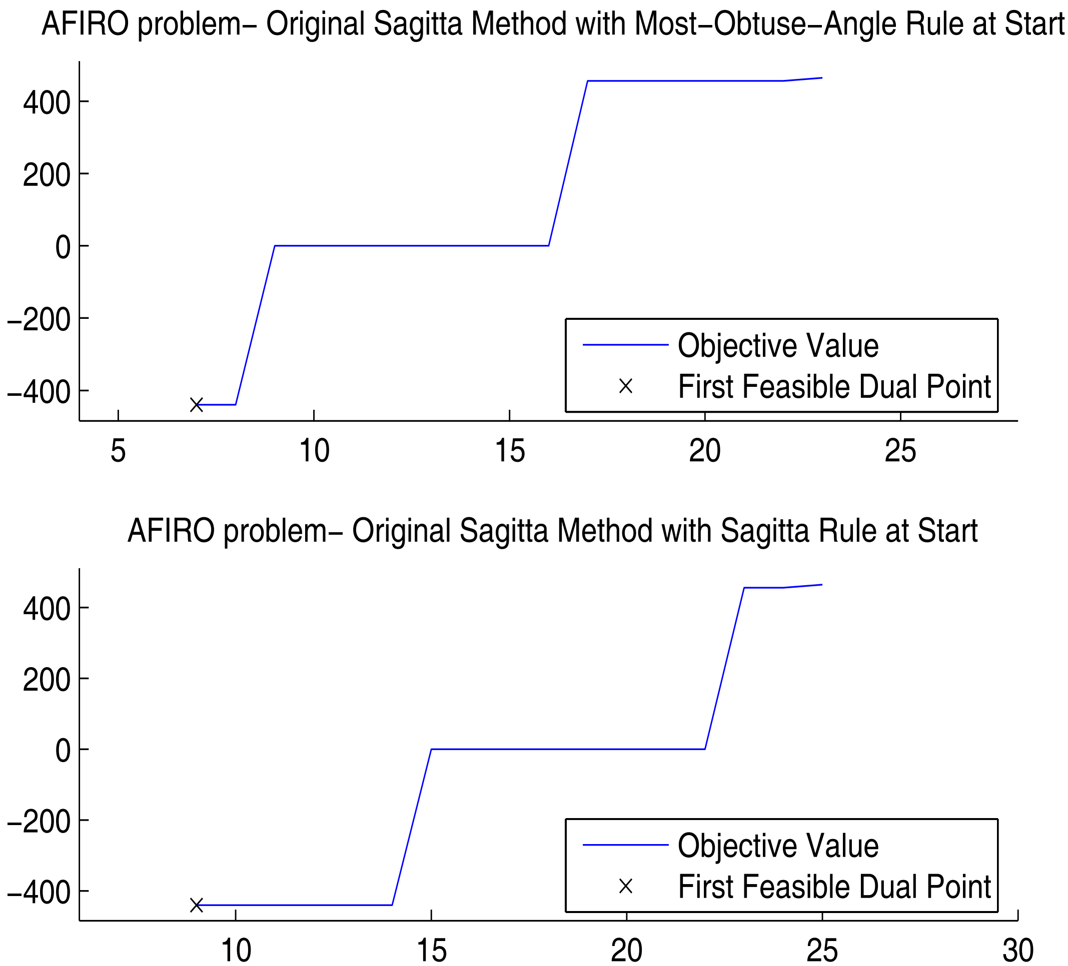


Fig. 1. *Objective of the problem AFIRO solved by using Sagitta Method.*

TABLE A.1. Computational results for the Sagitta Method when solving AFIRO *problem (#1, n=27, m=51)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	7	7	-4.40000000000000E+2	9	9	-4.40000000000000E+2
First feasible y	7	7	-4.40000000000000E+2	9	9	-4.40000000000000E+2
First square basis	–	–	–	–	–	–
First feasible x	23	20	4.64753142857142E+2	25	22	4.64753142857143E+2
Optimal solution	23	20	4.64753142857142E+2	25	22	4.64753142857143E+2
Restarts	0			0		
Time	0.03			0.03		

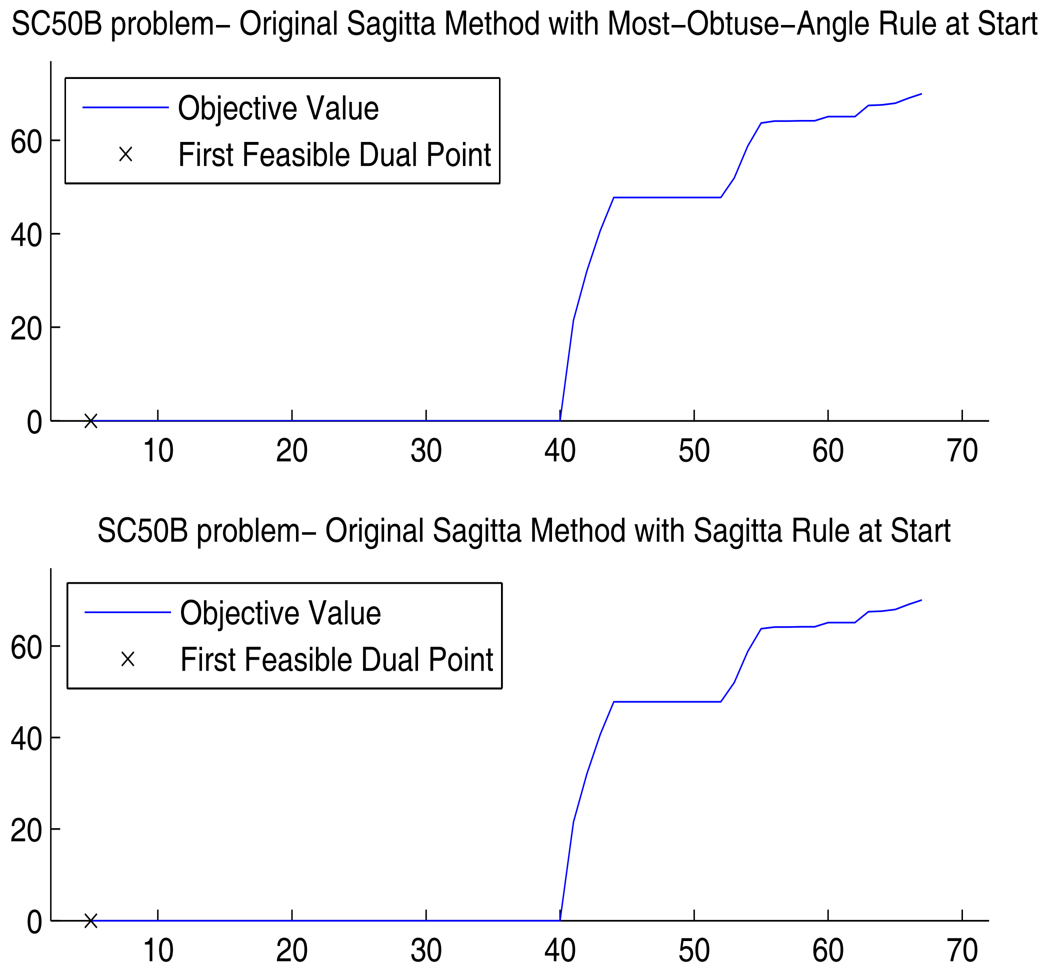
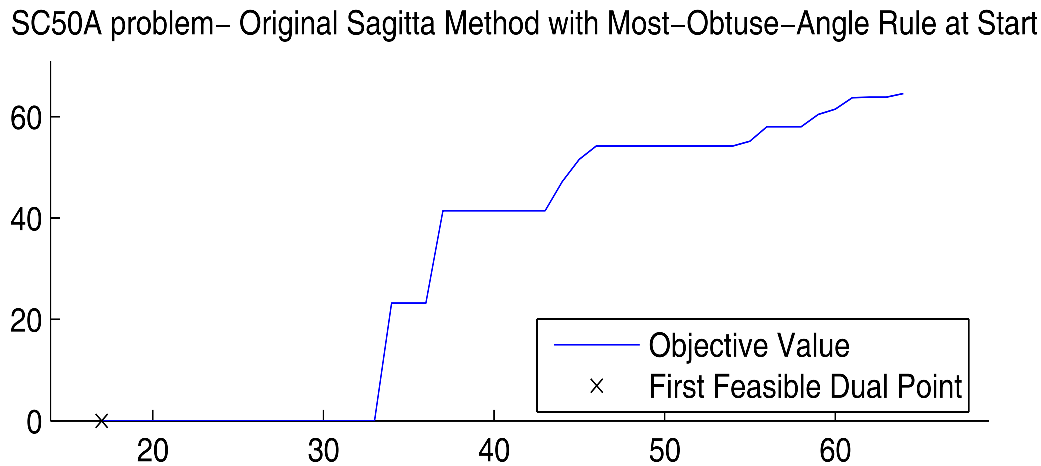


Fig. 2. *Objective of the problem SC50B solved by using Sagitta Method.*

TABLE A.2. Computational results for the Original Sagitta Method when solving SC50B *problem (#2, n=50, m=78)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	5	5	0.00000000000000E+0	5	5	0.00000000000000E+0
First feasible y	5	5	0.00000000000000E+0	5	5	0.00000000000000E+0
First square basis	–	–	–	–	–	–
First feasible x	67	48	7.00000000000000E+1	67	48	7.00000000000000E+1
Optimal solution	67	48	7.00000000000000E+1	67	48	7.00000000000000E+1
Restarts	0			0		
Time	0.08			0.06		



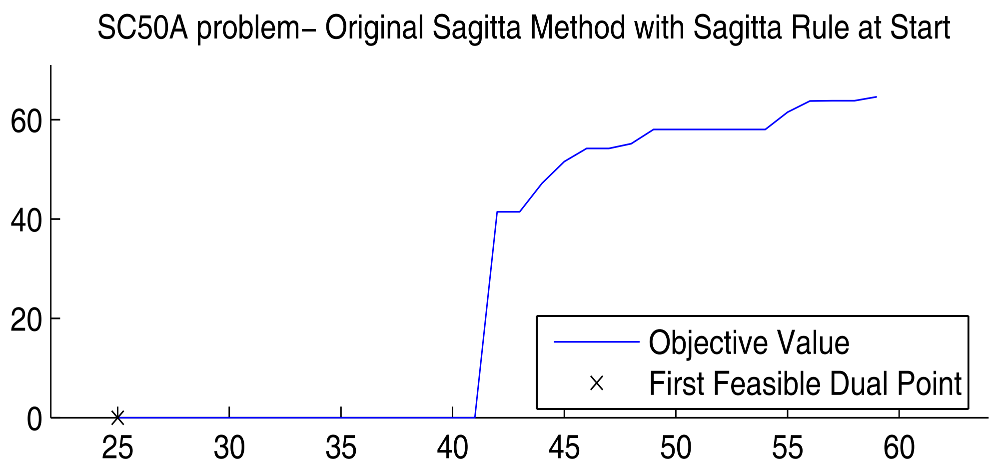


Fig. 3. *Objective of the problem SC50A solved by using Sagitta Method.*

TABLE A.3. Computational results for the Original Sagitta Method when solving SC50A *problem (#3, n=50, m=78)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	17	17	0.00000000000000E+0	25	25	0.00000000000000E+0
First feasible y	17	17	0.00000000000000E+0	25	25	0.00000000000000E+0
First square basis	–	–	–	–	–	–
First feasible x	64	49	6.45750770585645E+1	59	47	6.45750770585646E+1
Optimal solution	64	49	6.45750770585645E+1	59	47	6.45750770585646E+1
Restarts	0			0		
Time	0.08			0.06		

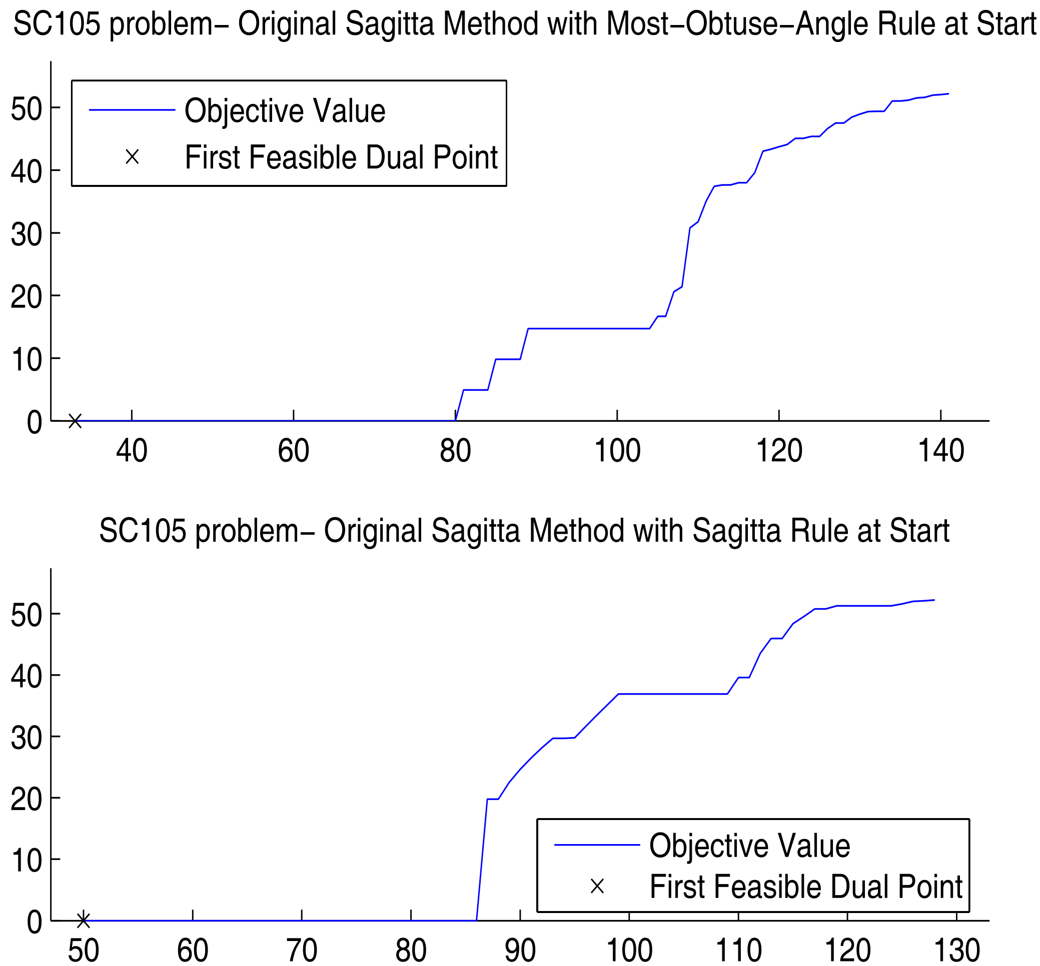


Fig. 4. *Objective of the problem SC105 solved by using Sagitta Method.*

TABLE A.4. Computational results for the Original Sagitta Method when solving SC105 *problem (#4, n=105, m=163)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	33	33	0.00000000000000E+0	50	50	0.00000000000000E+0
First feasible y	33	33	0.00000000000000E+0	50	50	0.00000000000000E+0
First square basis	–	–	–	–	–	–
First feasible x	141	104	5.22020612117073E+1	128	99	5.22020612117071E+1
Optimal solution	141	104	5.22020612117073E+1	128	99	5.22020612117071E+1
Restarts	0			0		
Time	0.30			0.22		

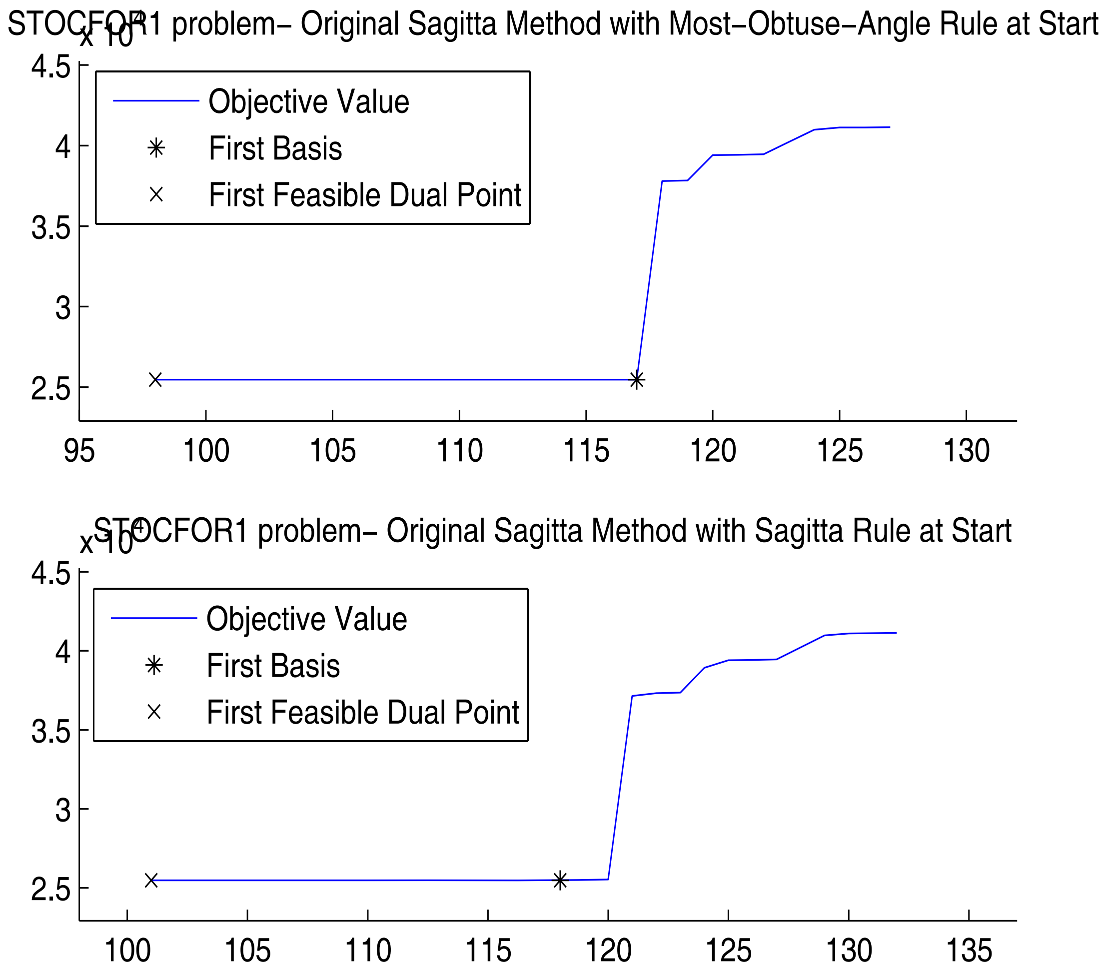


Fig. 5. *Objective of the problem STOCFOR1 solved by using Sagitta Method.*

TABLE A.5. Computational results for the Original Sagitta Method when solving STOCFOR1 *problem (#8, n=117, m=165)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	98	98	2.54686697284763E+4	101	101	2.54686697284763E+4
First feasible y	98	98	2.54686697284763E+4	101	101	2.54686697284763E+4
First square basis	117	117	2.54686697284763E+4	118	117	2.54686697284763E+4
First feasible x	127	117	4.11319762194316E+4	132	117	4.11319762194404E+4
Optimal solution	127	117	4.11319762194316E+4	132	117	4.11319762194404E+4
Restarts	0			0		
Time	0.16			0.17		

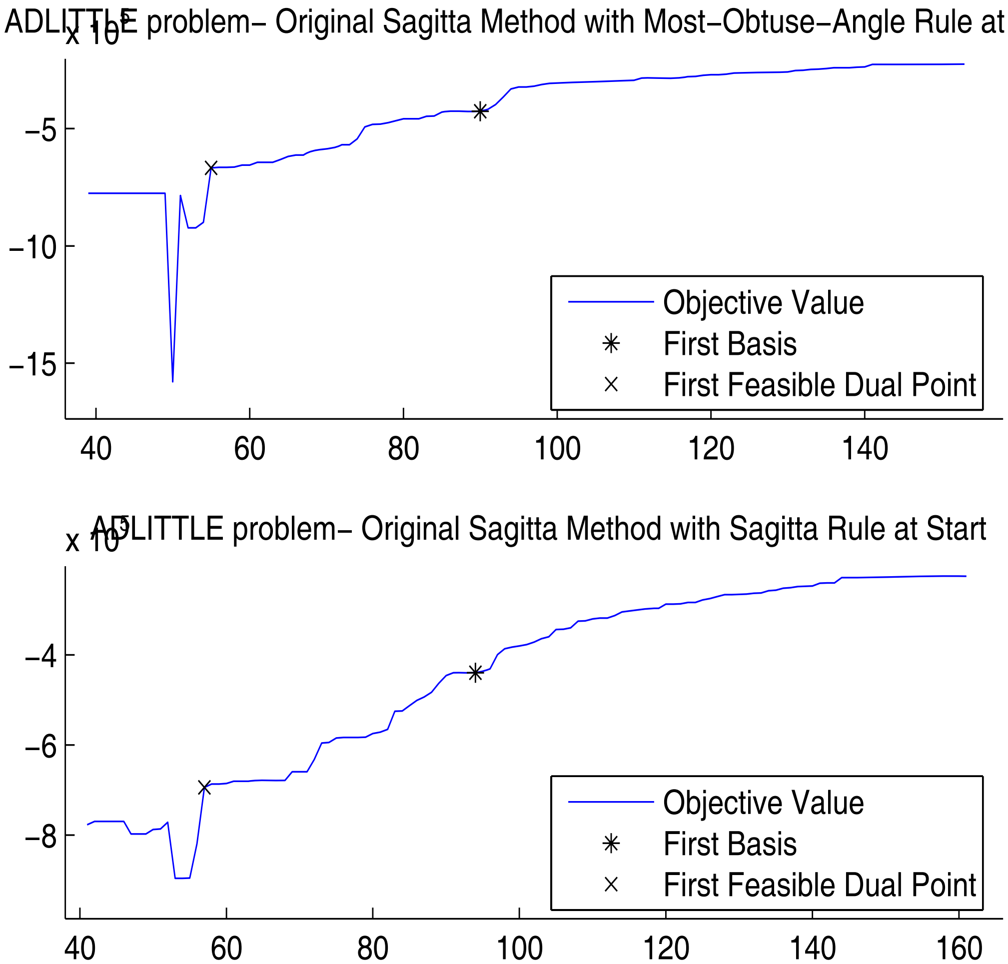


Fig. 6. *Objective of the problem ADLITTLE solved by using Sagitta Method.*

TABLE A.6. Computational results for the Original Sagitta Method when solving ADLITTLE *problem (#6, n=56, m=138)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	39	39	-7.76049131548757E+5	41	41	-7.77067884175367E+5
First feasible y	55	49	-6.66684176213086E+5	57	47	-6.94038753358815E+5
First square basis	90	56	-4.26104441100459E+5	94	56	-4.39859614774195E+5
First feasible x	153	56	-2.25494963162380E+5	161	56	-2.25494963162381E+5
Optimal solution	153	56	-2.25494963162380E+5	161	56	-2.25494963162381E+5
Restarts	0			0		
Time	0.31			0.30		

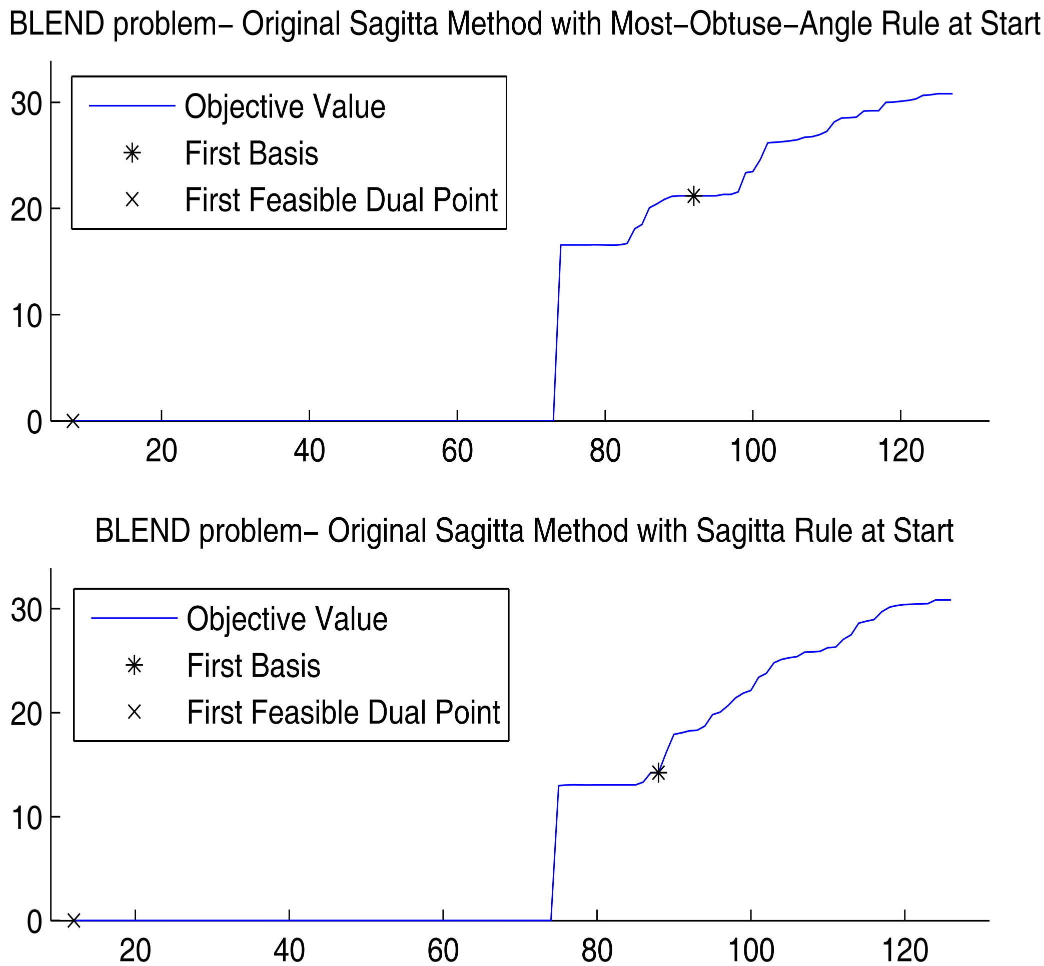
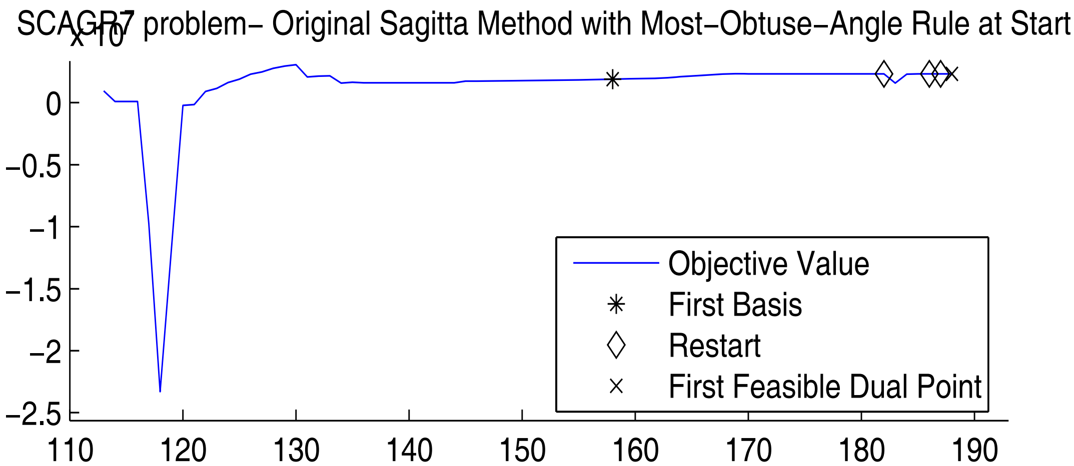


Fig. 7. *Objective of the problem BLEND solved by using Sagitta Method.*

TABLE A.7. Computational results for the Original Sagitta Method when solving BLEND *problem (#9, n=74, m=114)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	8	8	0.00000000000000E+0	12	12	3.55747926829032E-17
First feasible y	8	8	0.00000000000000E+0	12	12	3.55747926829032E-17
First square basis	92	74	2.12024452201228E+1	88	74	1.42170917832970E+1
First feasible x	127	74	3.08121498458327E+1	126	74	3.08121498458312E+1
Optimal solution	127	74	3.08121498458327E+1	126	74	3.08121498458312E+1
Restarts	0			0		
Time	0.25			0.23		



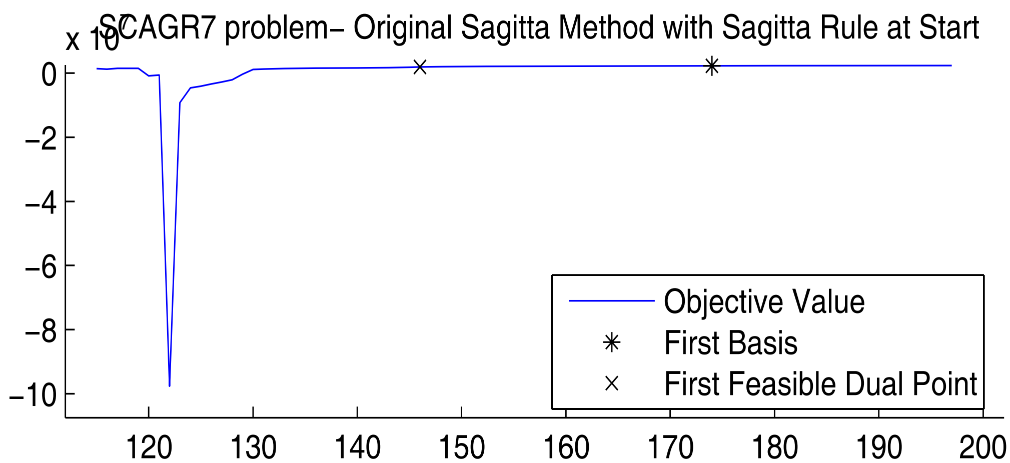


Fig. 8. *Objective of the problem SCAGR7 solved by using Sagitta Method.*

TABLE A.8. Computational results for the Original Sagitta Method when solving SCAGR7 *problem (#7, n=129, m=185)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	113	113	9.44922916266666E+5	115	115	1.43026947309867E+6
First feasible y	188	129	2.33138982433101E+6	146	117	1.95254933226732E+6
First square basis	158	129	1.89736985378474E+6	174	129	2.25298896756860E+6
First feasible x	182	129	2.33139671215523E+6	197	129	2.33138982433092E+6
Optimal solution	188	129	2.33138982433101E+6	197	129	2.33138982433092E+6
Restarts	3			0		
Time	0.61			0.59		

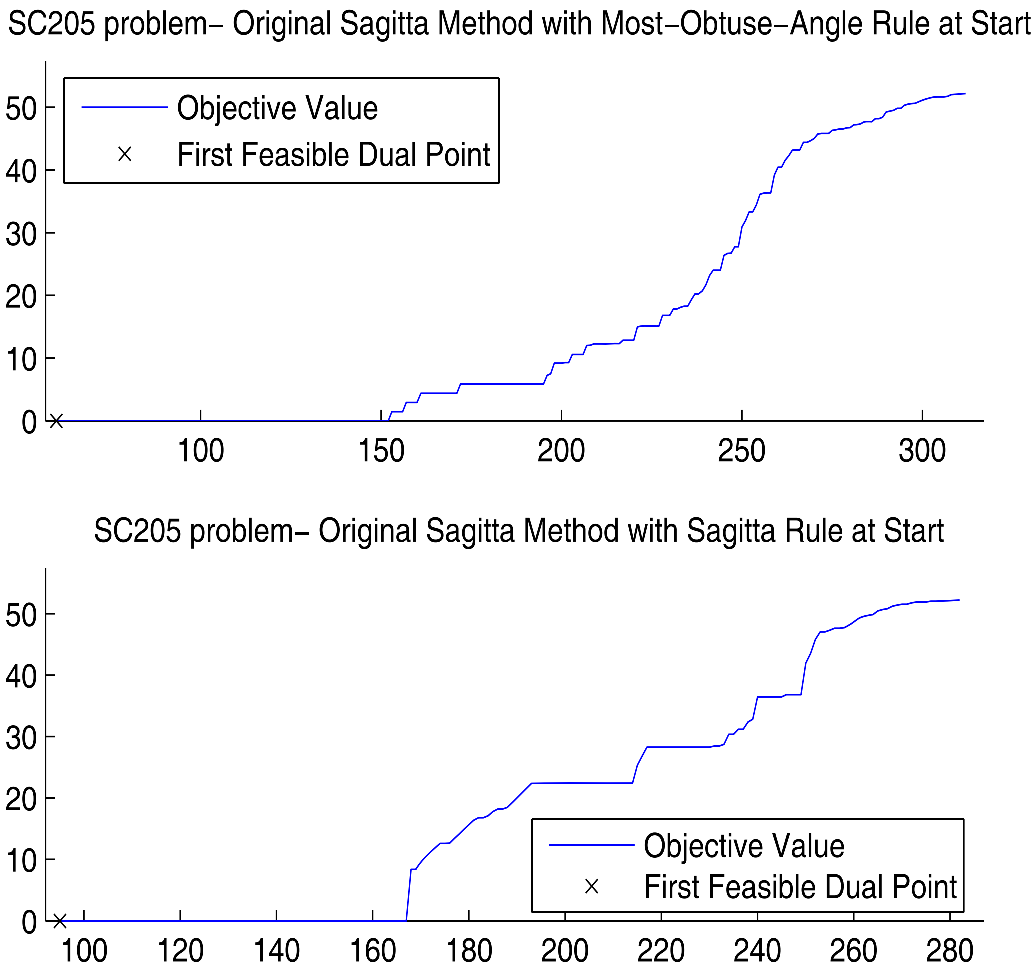


Fig. 9. *Objective of the problem SC205 solved by using Sagitta Method.*

TABLE A.9. Computational results for the Original Sagitta Method when solving SC205 *problem (#10, n=205, m=317)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	60	60	0.00000000000000E+0	95	95	0.00000000000000E+0
First feasible y	60	60	0.00000000000000E+0	95	95	0.00000000000000E+0
First square basis	–	–	–	–	–	–
First feasible x	312	203	5.22020612117081E+1	282	199	5.22020612117076E+1
Optimal solution	312	203	5.22020612117081E+1	282	199	5.22020612117076E+1
Restarts	0			0		
Time	2.03			1.80		

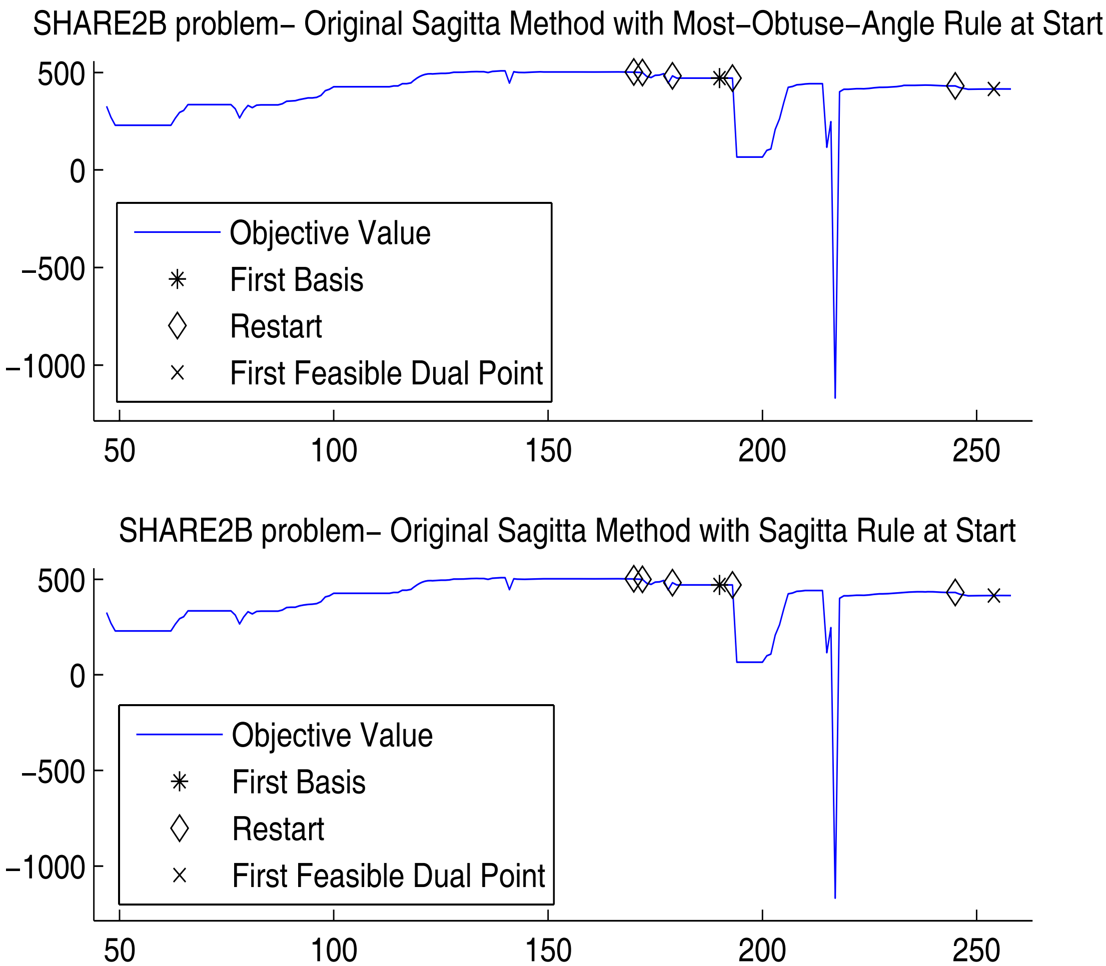


Fig. 10. *Objective of the problem SHARE2B solved by using Sagitta Method.*

TABLE A.10. Computational results for the Original Sagitta Method when solving SHARE2B *problem (#12, n=96, m=162)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	47	47	3.26616397459167E+2	47	47	3.26616397459167E+2
First feasible y	254	96	4.15347895178645E+2	254	96	4.15347895178645E+2
First square basis	190	96	4.70915753840341E+2	190	96	4.70915753840341E+2
First feasible x	170	86	5.02590239312383E+2	170	86	5.02590239312383E+2
Optimal solution	258	96	4.15732240741420E+2	258	96	4.15732240741420E+2
Restarts	5			5		
Time	0.63			0.64		

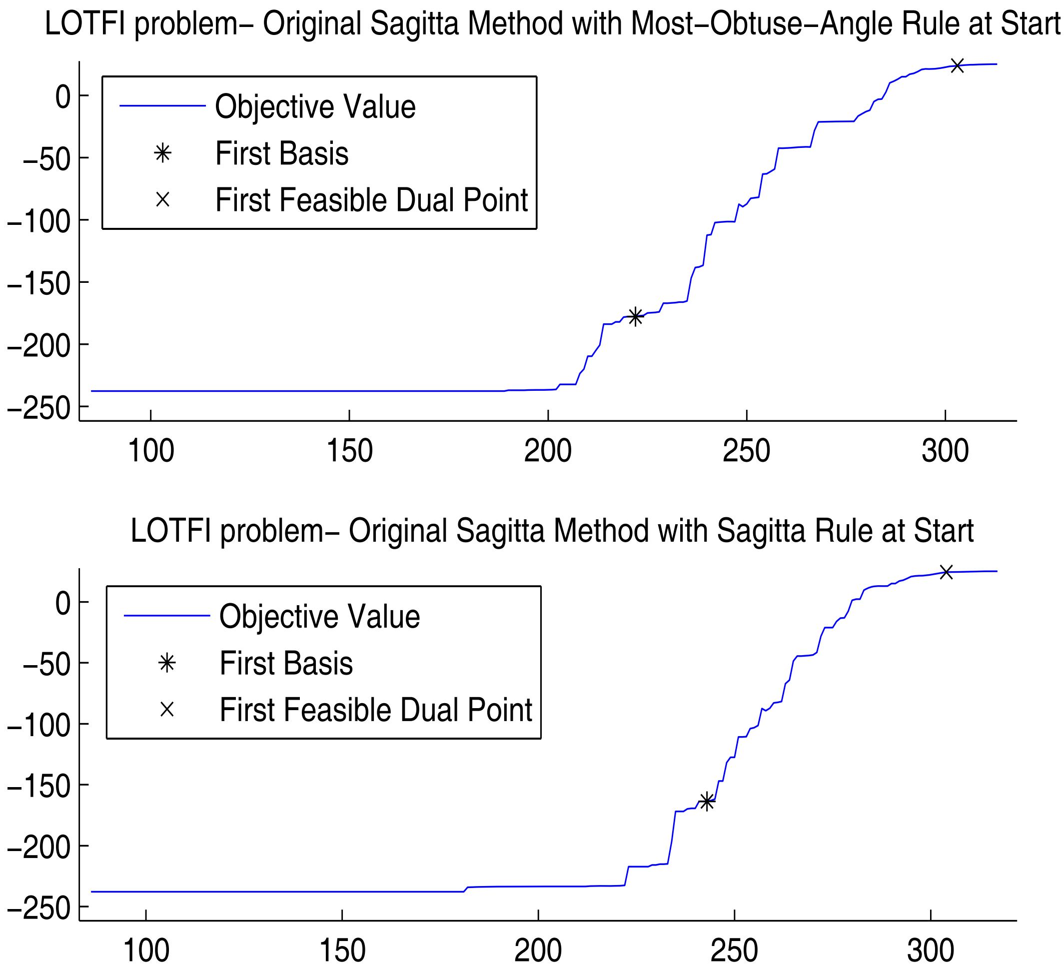


Fig. 11. *Objective of the problem LOTFI solved by using Sagitta Method.*

TABLE A.11. Computational results for the Original Sagitta Method when solving LOTFI *problem (#14, n=153, m=266)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	85	85	-2.37761499990000E+2	86	86	-2.37761499990000E+2
First feasible y	303	153	2.41748704660468E+1	304	153	2.43863704594371E+1
First square basis	222	153	-1.77817295941729E+2	243	153	-1.63719955570958E+2
First feasible x	313	153	2.52647060623677E+1	317	153	2.52647060567239E+1
Optimal solution	313	153	2.52647060623677E+1	317	153	2.52647060567239E+1
Restarts	0			0		
Time	1.78			1.80		

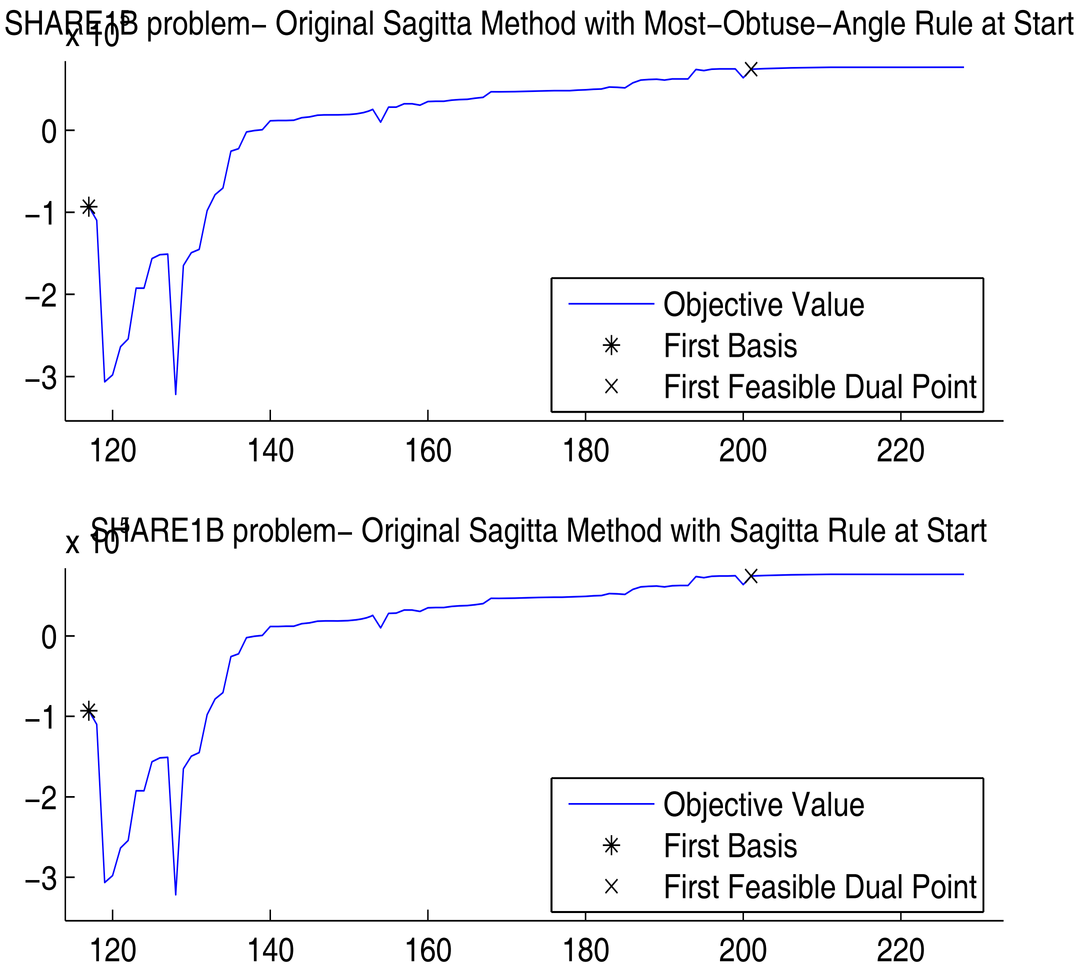


Fig. 12. *Objective of the problem SHARE1B solved by using Sagitta Method.*

TABLE A.12. Computational results for the Original Sagitta Method when solving SHARE1B *problem (#15, n=117, m=253)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	117	117	-9.31044263104192E+4	117	117	-9.31044263104192E+4
First feasible y	201	117	7.43291615207847E+4	201	117	7.43291615207847E+4
First square basis	117	117	-9.31044263104192E+4	117	117	-9.31044263104192E+4
First feasible x	228	117	7.65893185791879E+4	228	117	7.65893185791879E+4
Optimal solution	228	117	7.65893185791879E+4	228	117	7.65893185791879E+4
Restarts	0			0		
Time	0.80			0.83		

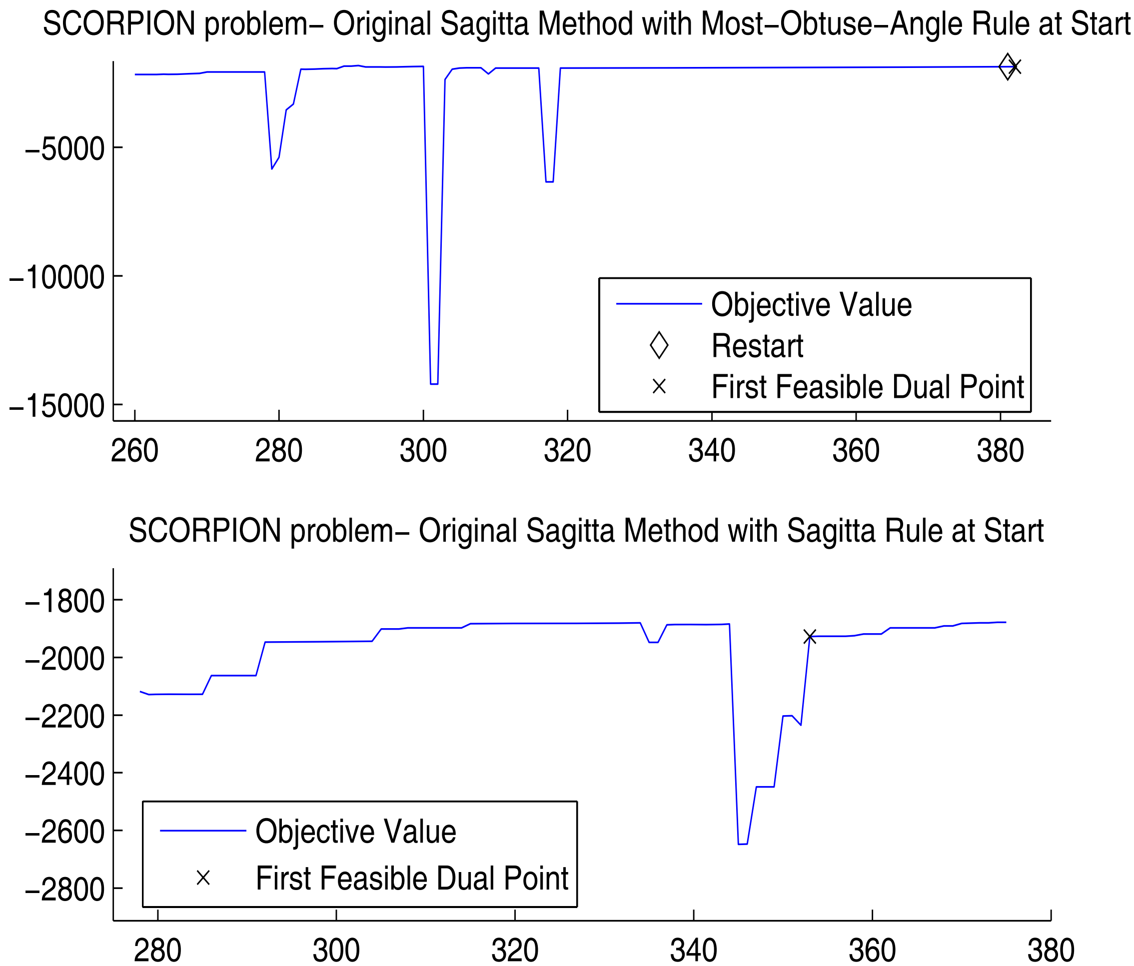


Fig. 13. *Objective of the problem SCORPION solved by using Sagitta Method.*

TABLE A.13. Computational results for the Original Sagitta Method when solving SCORPION *problem (#17, n=388, m=466)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	260	260	-2.17283406314009E+3	278	278	-2.11822973885512E+3
First feasible y	382	336	-1.87812482273811E+3	353	328	-1.92746413901251E+3
First square basis	–	–	–	–	–	–
First feasible x	381	336	-1.87507050455629E+3	375	339	-1.87812482273811E+3
Optimal solution	382	336	-1.87812482273811E+3	375	339	-1.87812482273811E+3
Restarts	1			0		
Time	4.84			4.72		

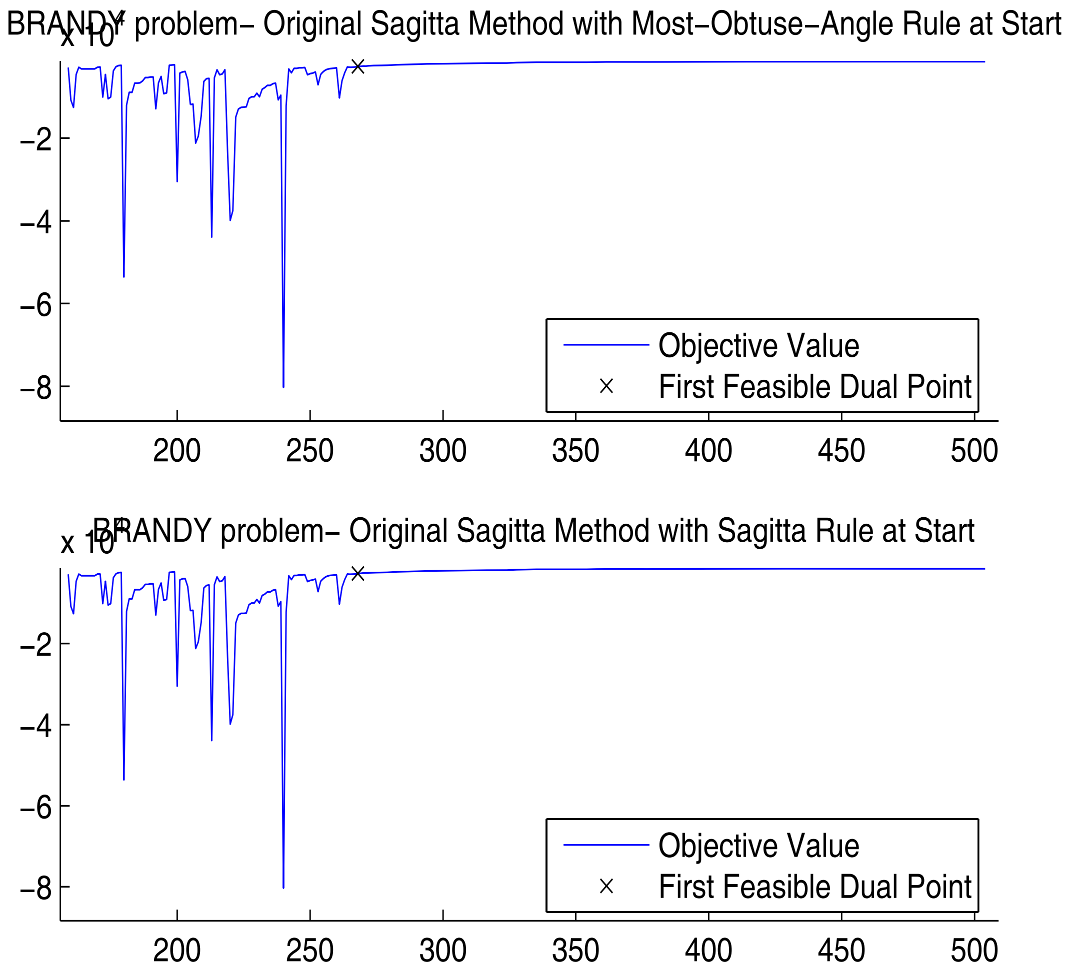


Fig. 14. *Objective of the problem BRANDY solved by using Sagitta Method.*

TABLE A.14. Computational results for the Original Sagitta Method when solving BRANDY *problem (#22, n=220, m=303)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	159	159	-2.92143186848942E+3	159	159	-2.92143186848942E+3
First feasible y	268	168	-2.68976446338926E+3	268	168	-2.68976446338926E+3
First square basis	–	–	–	–	–	–
First feasible x	504	170	-1.51850989648819E+3	504	170	-1.51850989648819E+3
Optimal solution	504	170	-1.51850989648819E+3	504	170	-1.51850989648819E+3
Restarts	0			0		
Time	3.67			3.25		

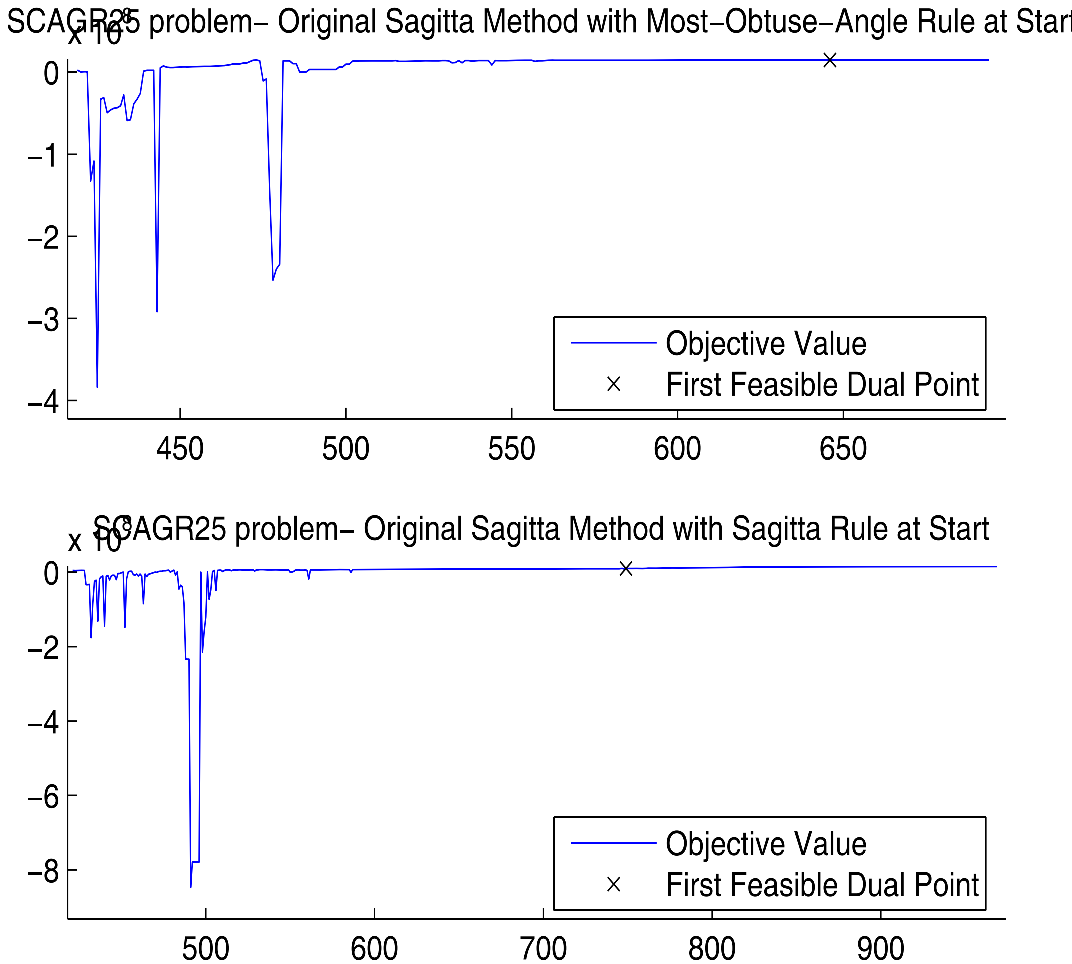


Fig. 15. *Objective of the problem SCAGR25 solved by using Sagitta Method.*

TABLE A.15. Computational results for the Original Sagitta Method when solving SCAGR25 *problem (#19, n=471, m=671)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	419	419	2.45890659952000E+6	421	421	4.87474326490666E+6
First feasible y	646	448	1.47126612296777E+7	749	424	9.63207728098069E+6
First square basis	–	–	–	–	–	–
First feasible x	694	448	1.47534330607685E+7	969	448	1.47534330607688E+7
Optimal solution	694	448	1.47534330607685E+7	969	448	1.47534330607688E+7
Restarts	0			0		
Time	23.03			36.25		

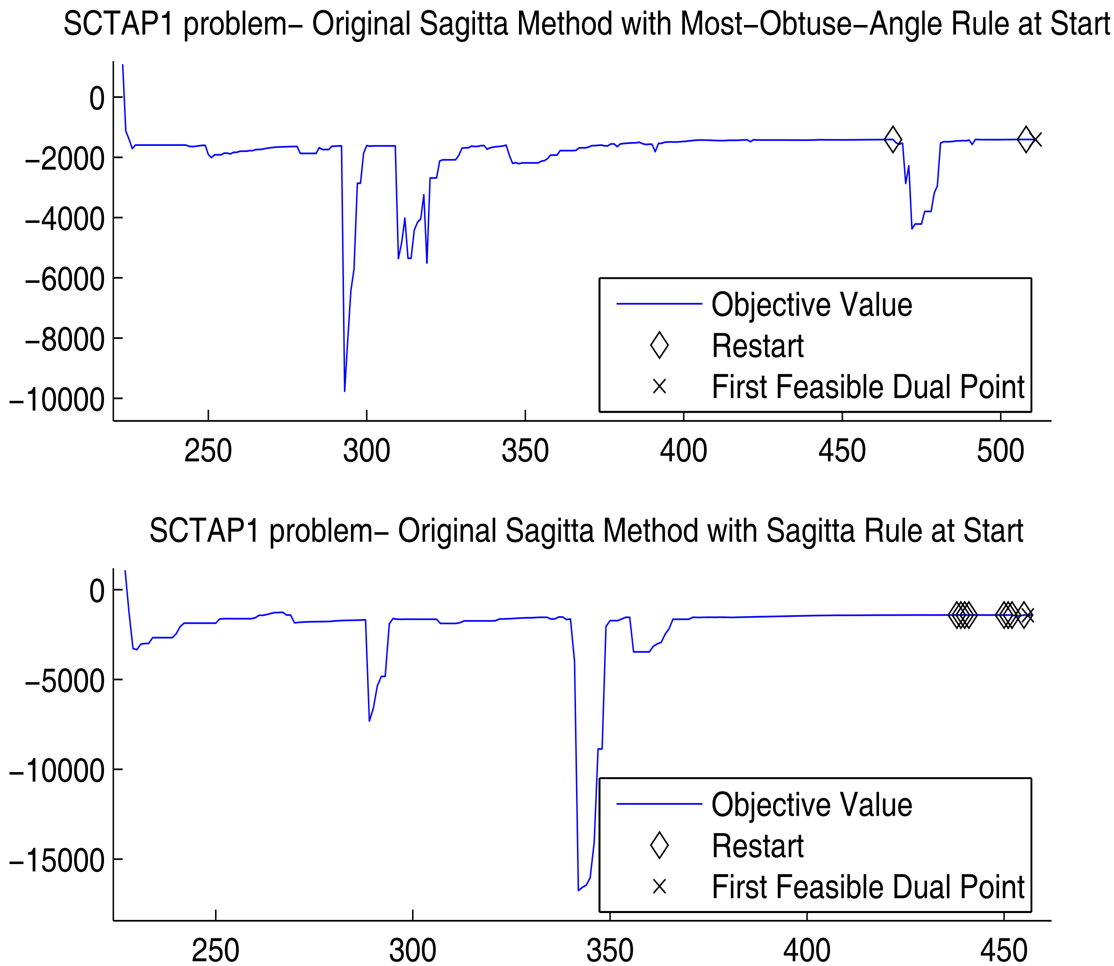


Fig. 16. *Objective of the problem SCTAP1 solved by using Sagitta Method.*

TABLE A.16. Computational results for the Original Sagitta Method when solving SCTAP1 *problem (#20, n=300, m=660)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	223	223	1.08894536289278E+3	227	227	1.09413552936798E+3
First feasible y	511	279	-1.41224999999999E+3	456	281	-1.42443749999998E+3
First square basis	–	–	–	–	–	–
First feasible x	466	279	-1.41224999999998E+3	438	281	-1.41224999999997E+3
Optimal solution	511	279	-1.41224999999999E+3	457	281	-1.41224999999997E+3
Restarts	2			8		
Time	9.05			7.75		

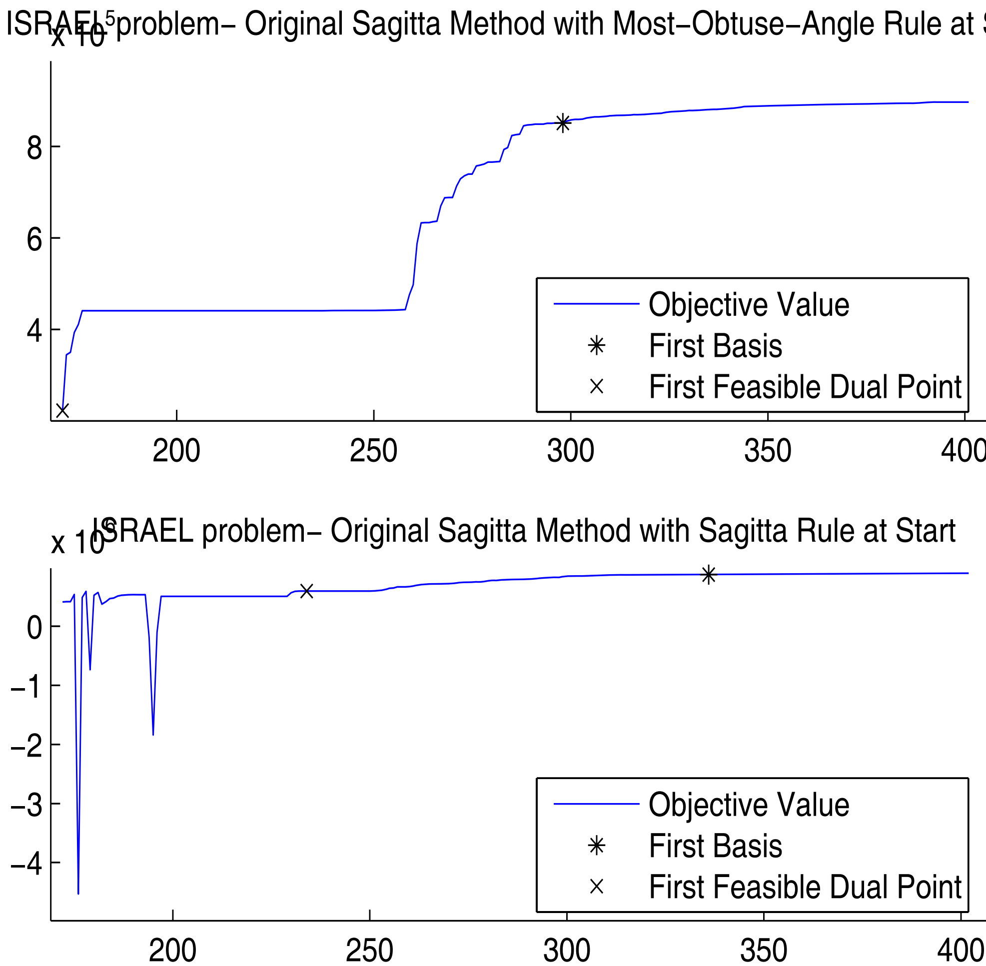


Fig. 17. *Objective of the problem ISRAEL solved by using Sagitta Method.*

TABLE A.17. Computational results for the Original Sagitta Method when solving ISRAEL *problem (#23, n=174, m=316)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	171	171	2.23216716865422E+5	172	172	4.13456017338819E+5
First feasible y	171	171	2.23216716865422E+5	234	172	5.96415605689331E+5
First square basis	298	174	8.50716980182927E+5	336	174	8.76466763965410E+5
First feasible x	401	174	8.96644821863053E+5	402	174	8.96644821863053E+5
Optimal solution	401	174	8.96644821863053E+5	402	174	8.96644821863053E+5
Restarts	0			0		
Time	2.83			2.69		

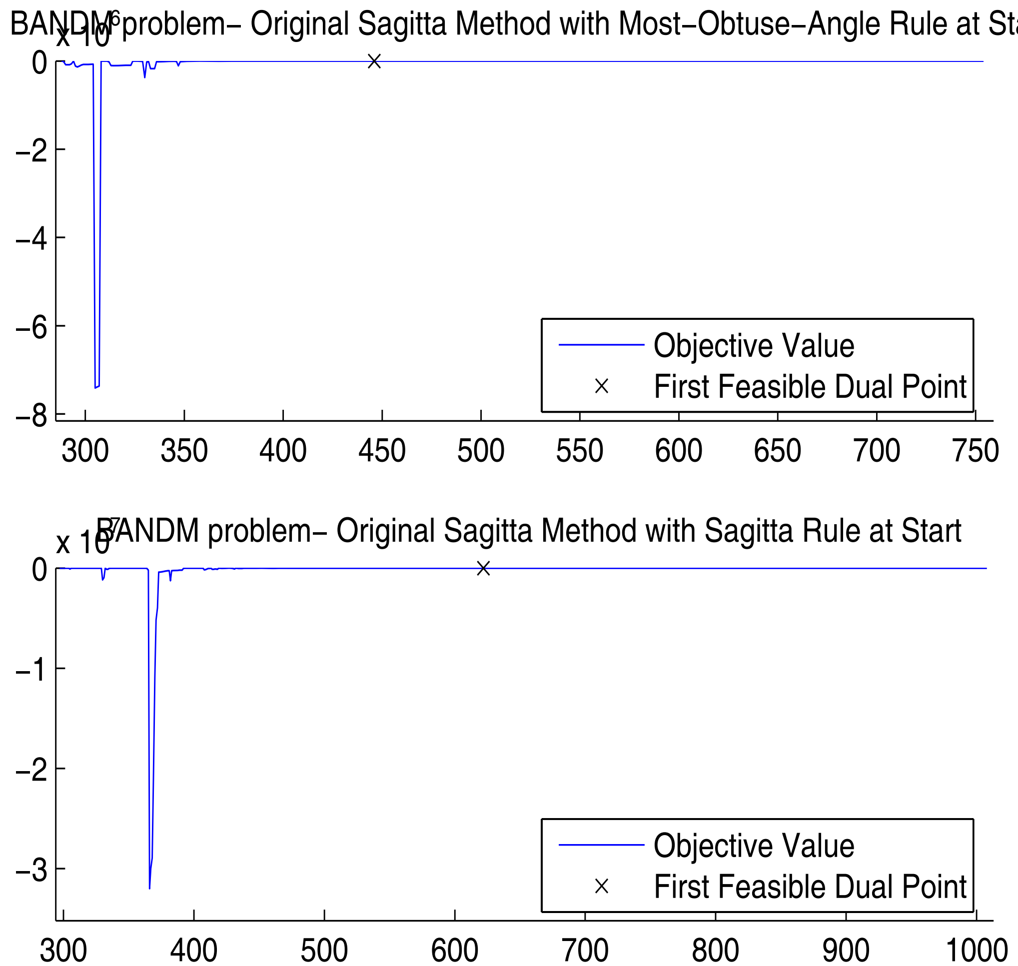


Fig. 18. *Objective of the problem BANDM solved by using Sagitta Method.*

TABLE A.18. Computational results for the Original Sagitta Method when solving BANDM *problem (#29, n=305, m=472)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	288	288	-1.24642632143687E+3	297	297	-1.09061934776363E+2
First feasible y	446	296	8.43951594911324E+1	622	303	6.05632473611761E+1
First square basis	–	–	–	–	–	–
First feasible x	754	304	1.58628018449912E+2	1008	304	1.58628018407146E+2
Optimal solution	754	304	1.58628018449912E+2	1008	304	1.58628018407146E+2
Restarts	0			0		
Time	13.42			19.19		

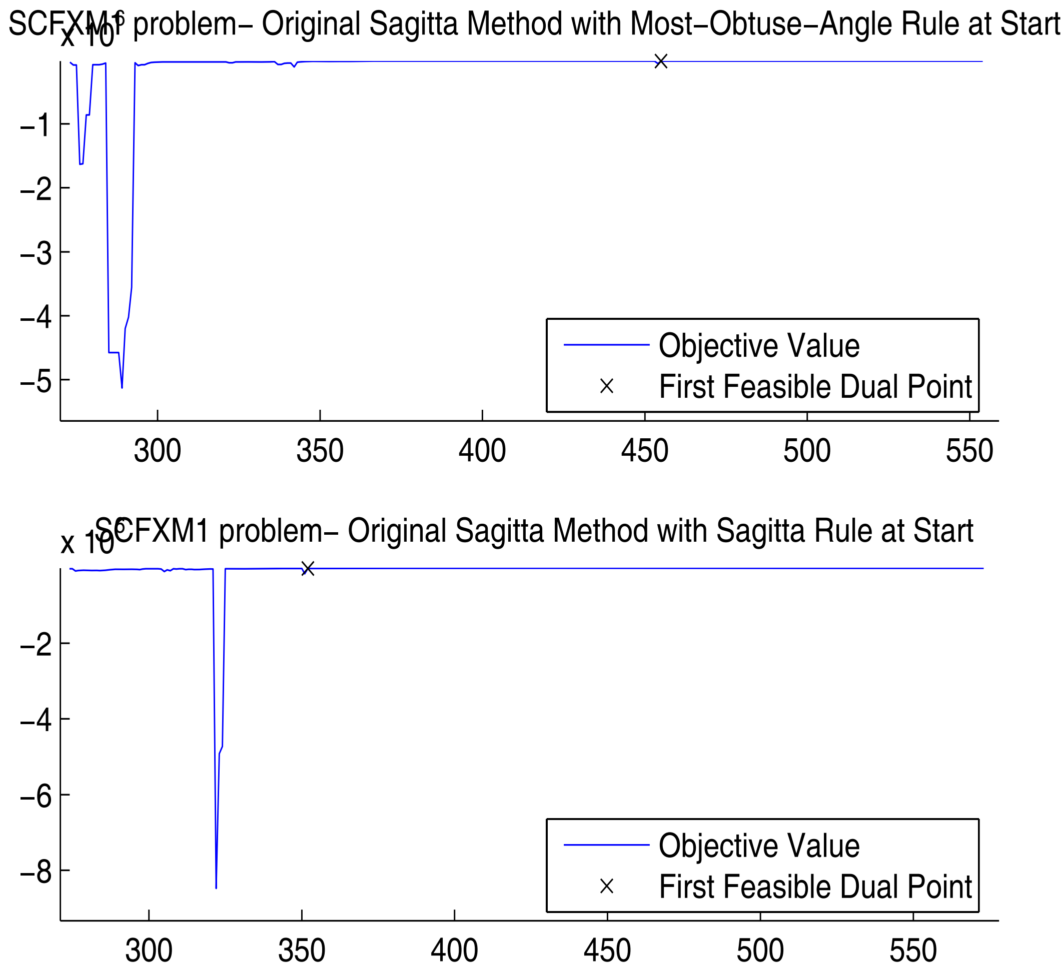


Fig. 19. *Objective of the problem SCFXM1 solved by using Sagitta Method.*

TABLE A.19. Computational results for the Original Sagitta Method when solving SCFXM1 *problem (#31, n=330, m=600)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	273	273	-3.82153236631949E+4	274	274	-3.20545988067005E+4
First feasible y	4 455	300	-1.89978792706094E+4	352	285	-2.69459962643029E+4
First square basis	–	–	–	–	–	–
First feasible x	554	320	-1.84167590283553E+4	573	319	-1.84167590283421E+4
Optimal solution	554	320	-1.84167590283553E+4	573	319	-1.84167590283421E+4
Restarts	0			0		
Time	10.50			11.22		

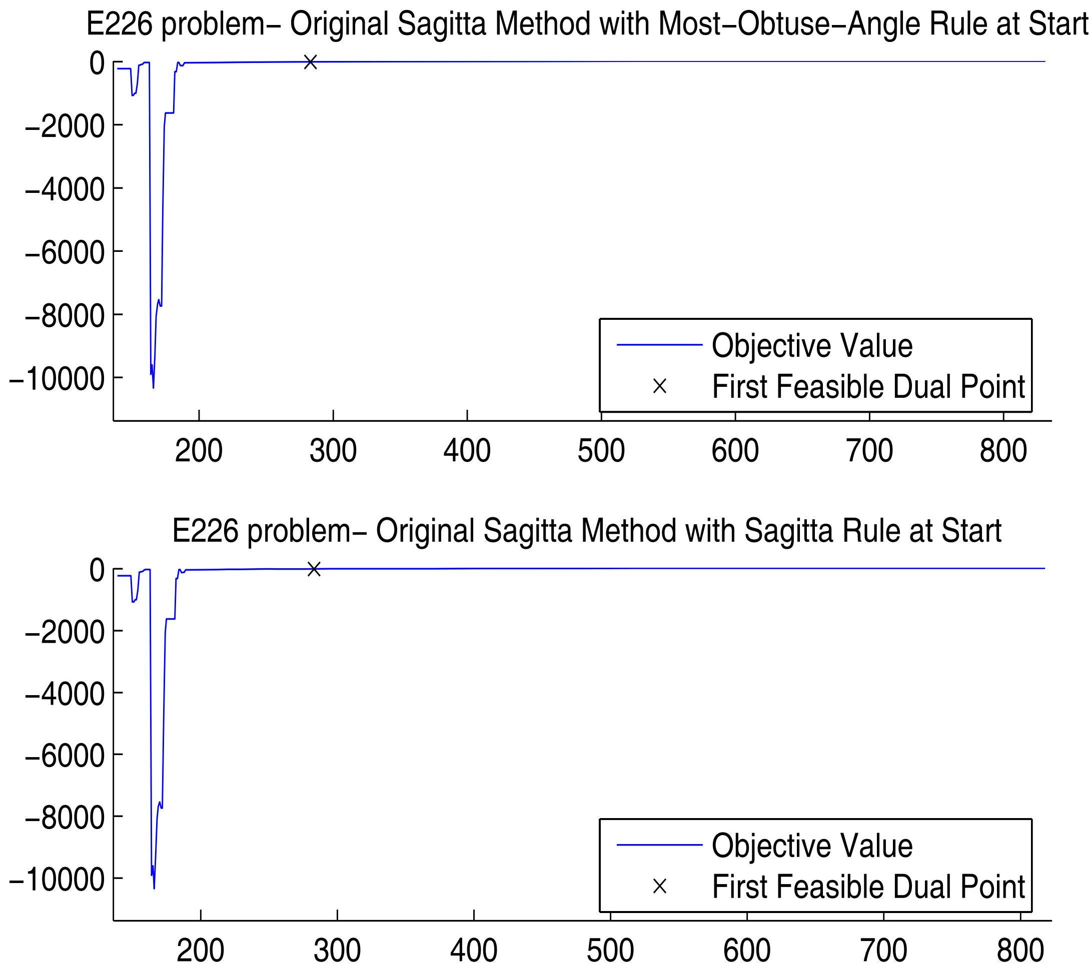
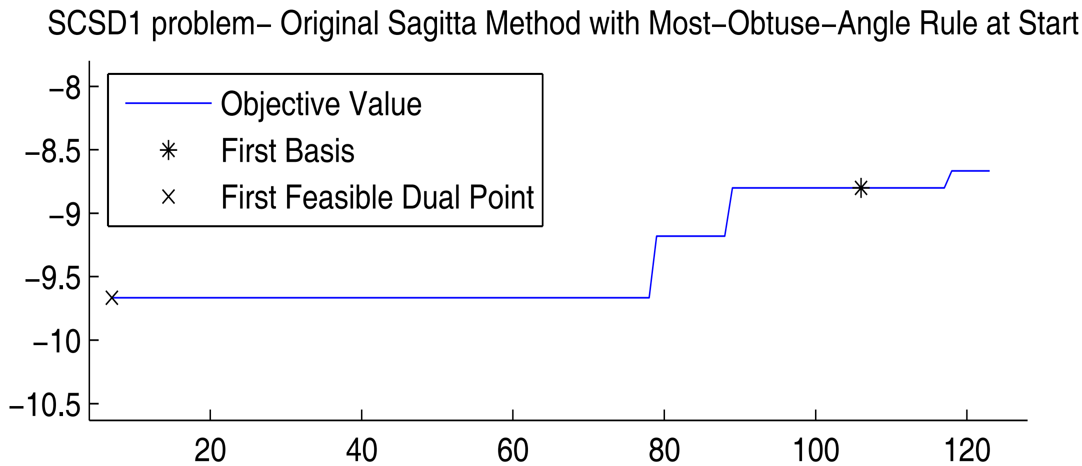


Fig. 20. *Objective of the problem E226 solved by using Sagitta Method.*

TABLE A.20. Computational results for the Original Sagitta Method when solving E226 *problem (#30, n=223, m=472)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	139	139	-2.24463304539531E+2	139	139	-2.24433565178187E+2
First feasible y	283	194	-6.79392418090953E+0	283	194	-6.70684024729339E+0
First square basis	–	–	–	–	–	–
First feasible x	831	213	1.87519290663021E+1	818	214	1.87519290662851E+1
Optimal solution	831	213	1.87519290663021E+1	818	214	1.87519290662851E+1
Restarts	0			0		
Time	9.02			8.70		



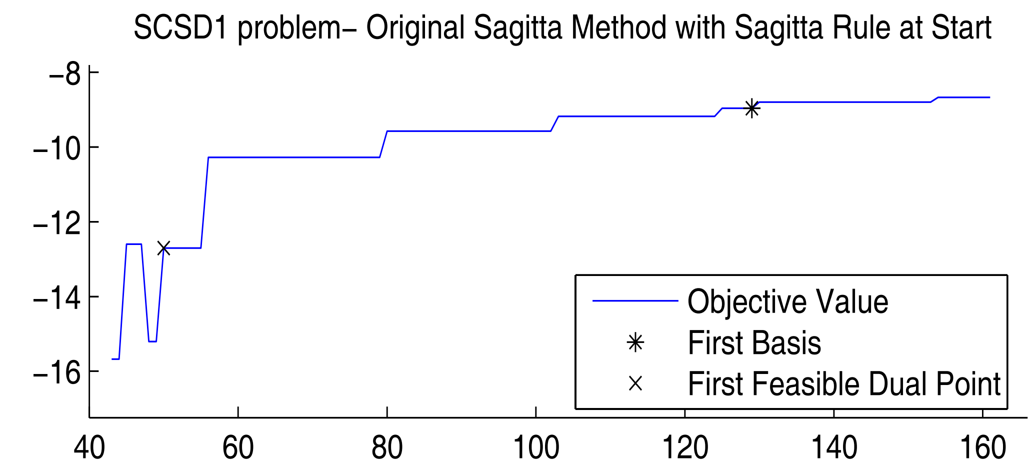


Fig. 21. *Objective of the problem SCSD1 solved by using Sagitta Method.*

TABLE A.21. Computational results for the Original Sagitta Method when solving SCSD1 *problem (#26, n=77, m=760)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	7	7	-9.66666671073451E+0	43	43	-1.56736112595916E+1
First feasible y	7	7	-9.66666671073451E+0	50	46	-1.27000000489486E+1
First square basis	106	77	-8.80000001349314E+0	129	77	-8.96000001703773E+0
First feasible x	123	77	-8.66666667433337E+0	161	77	-8.66666667433336E+0
Optimal solution	123	77	-8.66666667433337E+0	161	77	-8.66666667433336E+0
Restarts	0			0		
Time	0.52			0.72		

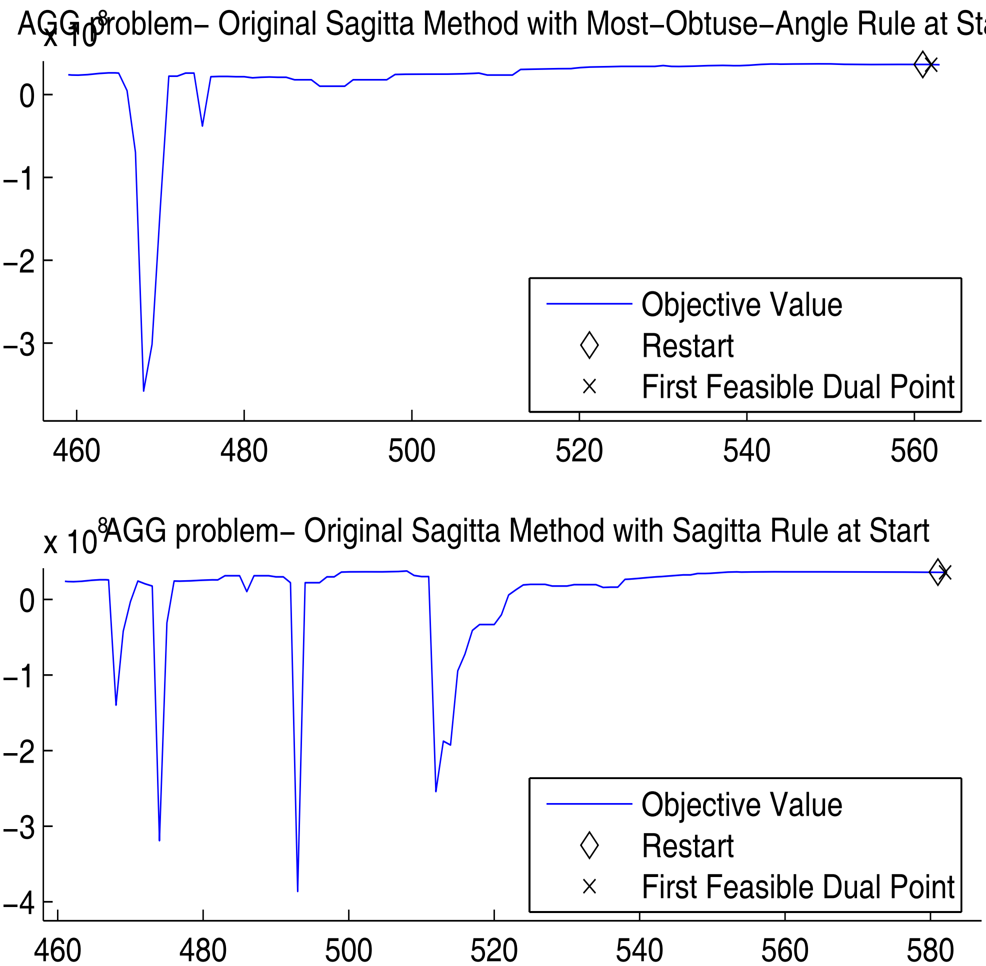
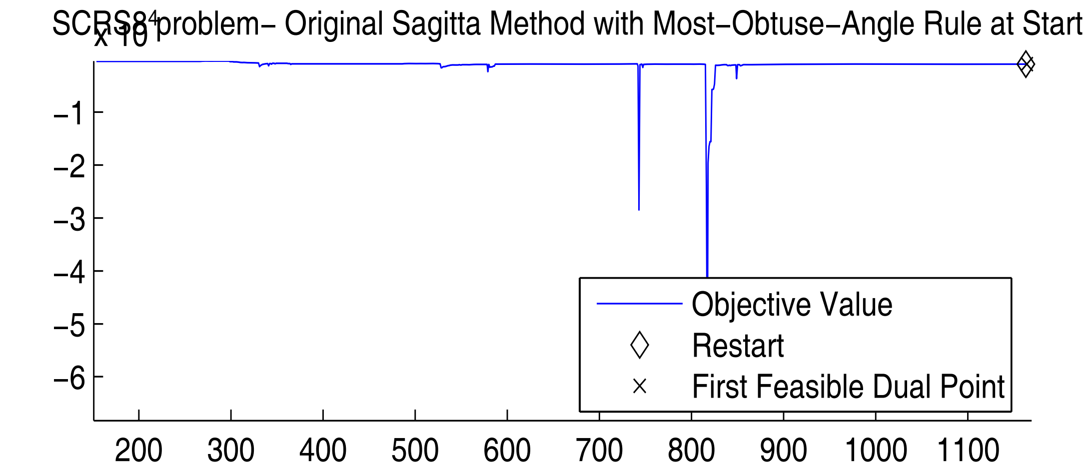


Fig. 22. *Objective of the problem AGG solved by using Sagitta Method.*

TABLE A.22. Computational results for the Original Sagitta Method when solving AGG *problem (#28, n=488, m=615)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	459	459	2.38048815442252E+7	461	461	2.41332005704515E+7
First feasible y	562	486	3.59917672865776E+7	582	486	3.59917672866520E+7
First square basis	–	–	–	–	–	–
First feasible x	561	486	3.63130844697354E+7	581	486	3.63130844695022E+7
Optimal solution	563	486	3.59917672865776E+7	582	486	3.59917672866520E+7
Restarts	1			1		
Time	14.47			15.64		



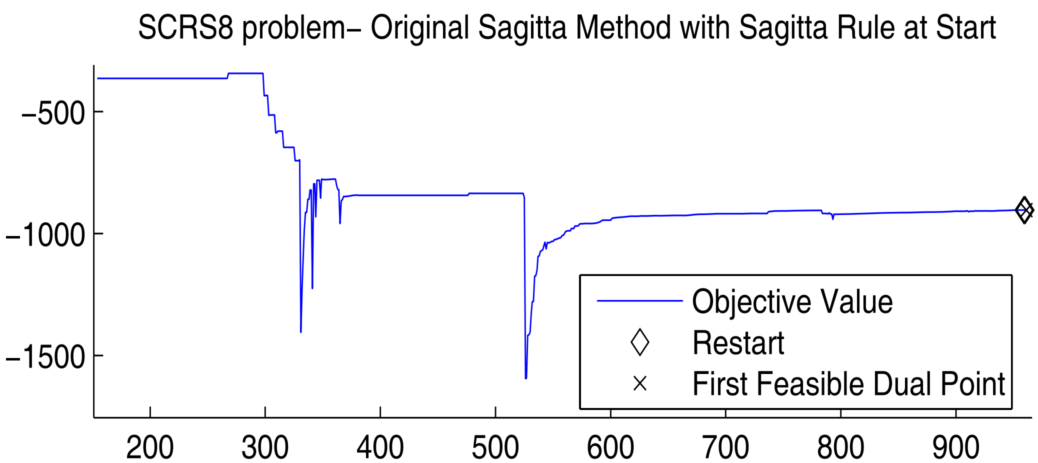


Fig. 23. *Objective of the problem SCRS8 solved by by using Sagitta Method.*

TABLE A.23. Computational results for the Original Sagitta Method when solving SCRS8 *problem (#34, n=490, m=1275)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	154	154	-3.65040485878916E+2	154	154	-3.65040485878916E+2
First feasible y	1164	479	-9.04296953800780E+2	961	478	-9.04296953800832E+2
First square basis	–	–	–	–	–	–
First feasible x	1163	479	-9.04296953800782E+2	959	478	-9.04296953800830E+2
Optimal solution	1164	479	-9.04296953800780E+2	961	478	-9.04296953800832E+2
Restarts	1			2		
Time	57.38			46.00		

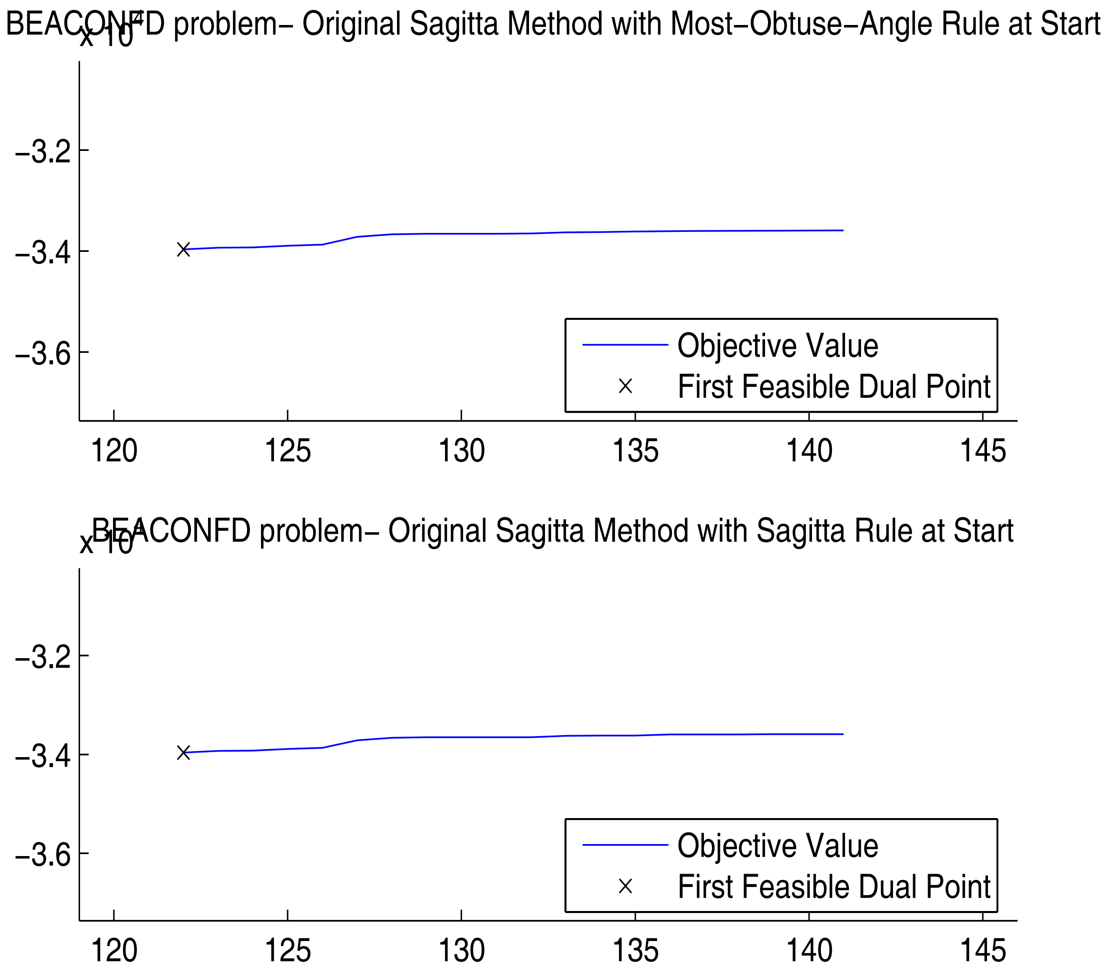


Fig. 24. *Objective of the problem BEACONFD solved by using Sagitta Method.*

TABLE A.24. Computational results for the Original Sagitta Method when solving BEACONFD *problem (#35, n=173, m=295)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	122	122	-3.39642730702000E+4	122	122	-3.39642730702000E+4
First feasible y	122	122	-3.39642730702000E+4	122	122	-3.39642730702000E+4
First square basis	–	–	–	–	–	–
First feasible x	141	122	-3.35924858072000E+4	141	122	-3.35924858072000E+4
Optimal solution	141	122	-3.35924858072000E+4	141	122	-3.35924858072000E+4
Restarts	0			0		
Time	0.39			0.36		

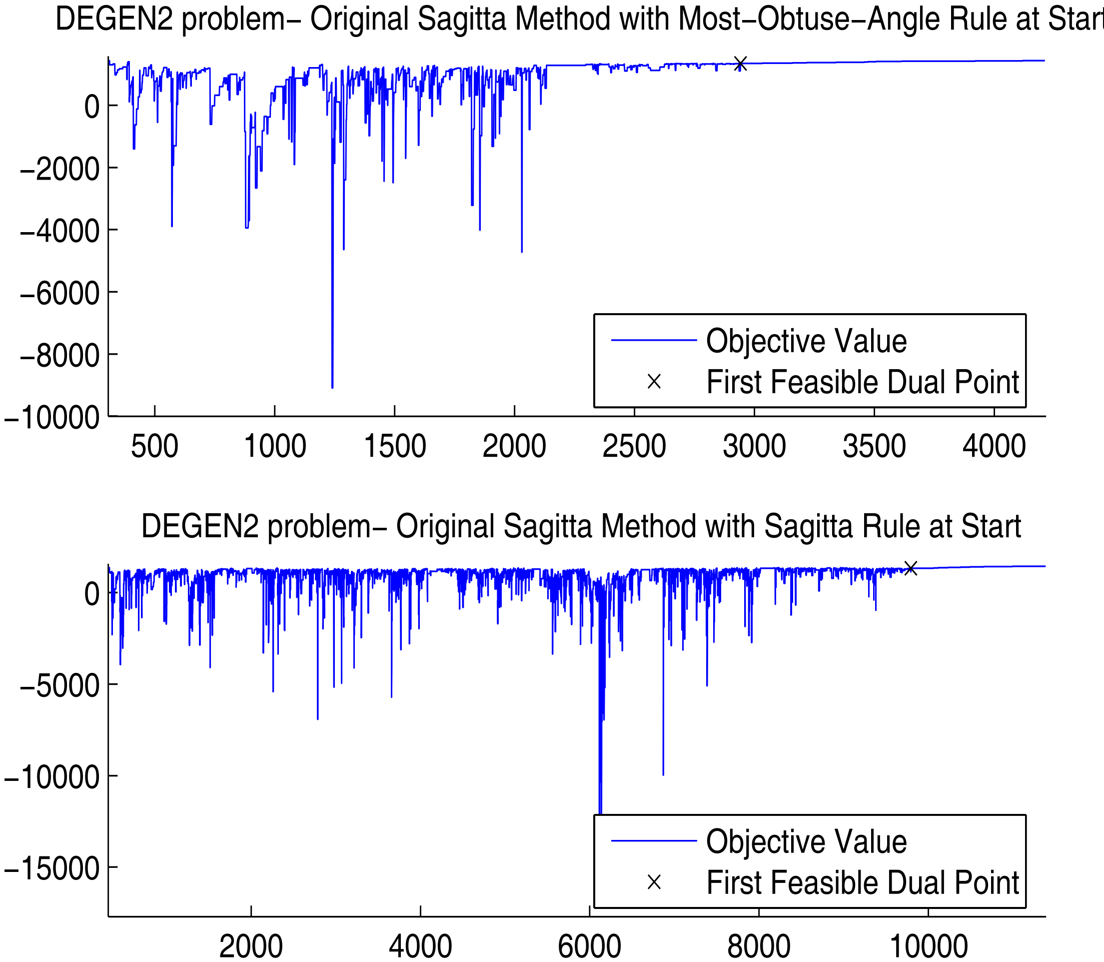


Fig. 25. *Objective of the problem DEGEN2 solved by using Sagitta Method.*

TABLE A.25. Computational results for the Original Sagitta Method when solving DEGEN2 *problem (#40, n=444, m=757)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	309	309	1.44072500000000E+3	313	313	1.36633000000000E+3
First feasible y	2943	440	1.34483333333336E+3	9794	442	1.33444999999989E+3
First square basis	–	–	–	–	–	–
First feasible x	4211	440	1.43517800000001E+3	11386	442	1.43517799999991E+3
Optimal solution	4211	440	1.43517800000001E+3	11386	442	1.43517799999991E+3
Restarts	0			0		
Time	245.08			702.64		

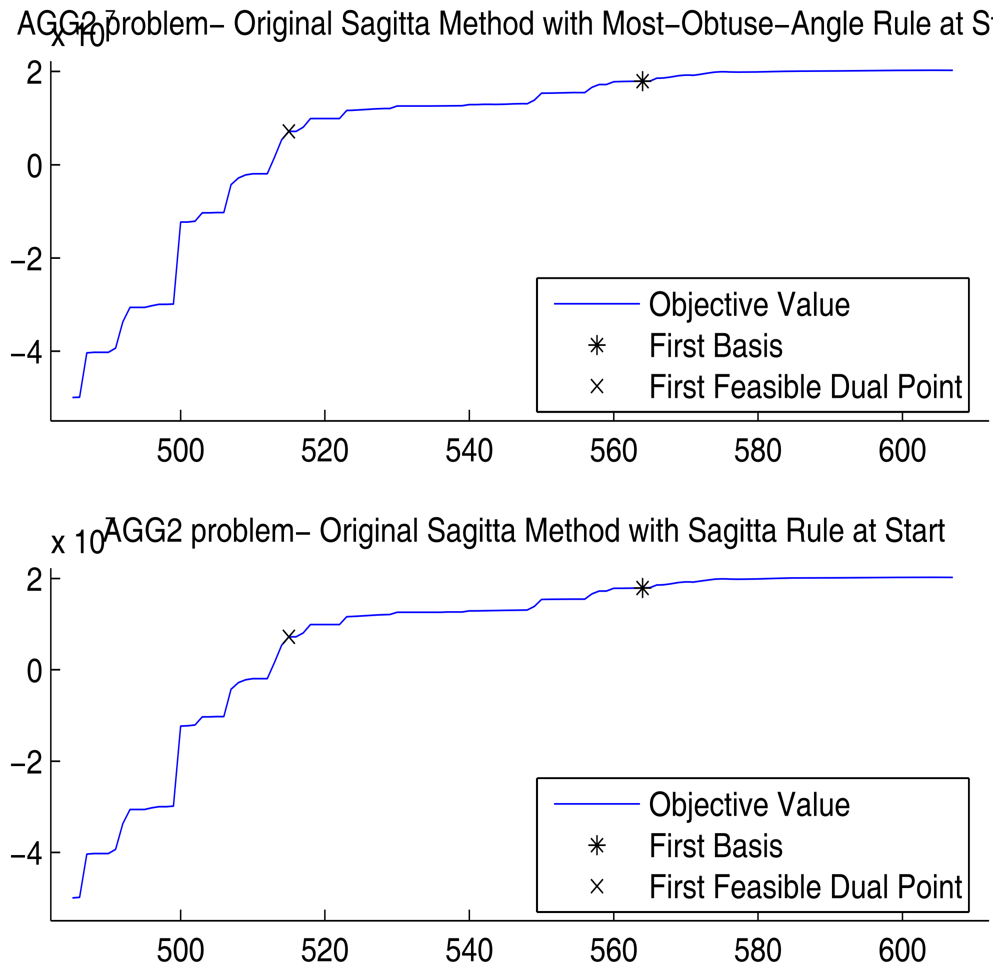


Fig. 26. *Objective of the problem AGG2 solved by using Sagitta Method.*

TABLE A.26. Computational results for the Original Sagitta Method when solving AGG2 *problem (#41, n=516, m=758)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	485	485	-4.99352395270087E+7	485	485	-4.99352395270087E+7
First feasible y	515	492	7.18059166916128E+6	515	492	7.18059166916128E+6
First square basis	564	516	1.79182965891465E+7	564	516	1.79182965891465E+7
First feasible x	607	516	2.02392523559771E+7	607	516	2.02392523559771E+7
Optimal solution	607	516	2.02392523559771E+7	607	516	2.02392523559771E+7
Restarts	0			0		
Time	21.02			20.98		

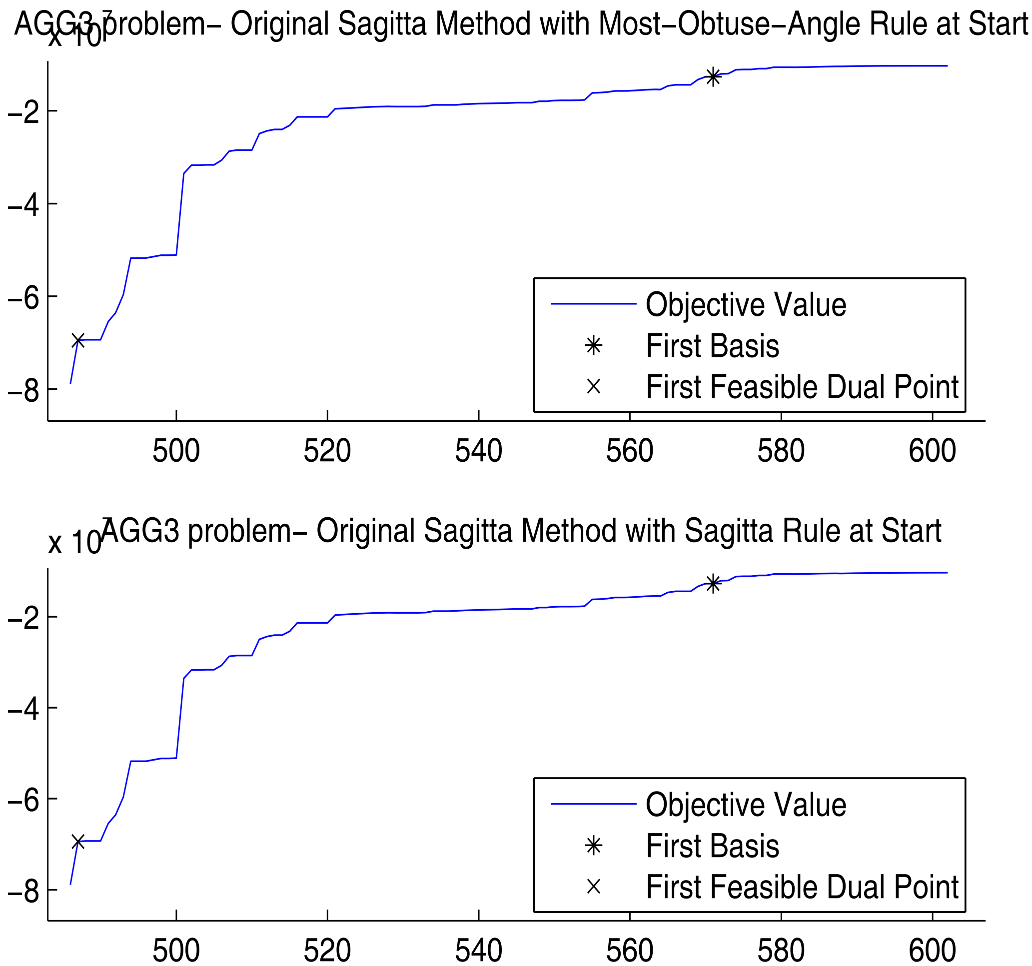


Fig. 27. *Objective of the problem AGG3 solved by using Sagitta Method.*

TABLE A.27. Computational results for the Original Sagitta Method when solving AGG3 *problem (#42, n=516, m=758)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	486	486	-7.89146265845531E+7	486	486	-7.89146265845531E+7
First feasible y	487	486	-6.94393521678370E+7	487	486	-6.94393521678370E+7
First square basis	571	516	-1.26801439626805E+7	571	516	-1.26801439626805E+7
First feasible x	602	516	-1.03121159350892E+7	602	516	-1.03121159350892E+7
Optimal solution	602	516	-1.03121159350892E+7	602	516	-1.03121159350892E+7
Restarts	0			0		
Time	20.33			20.38		

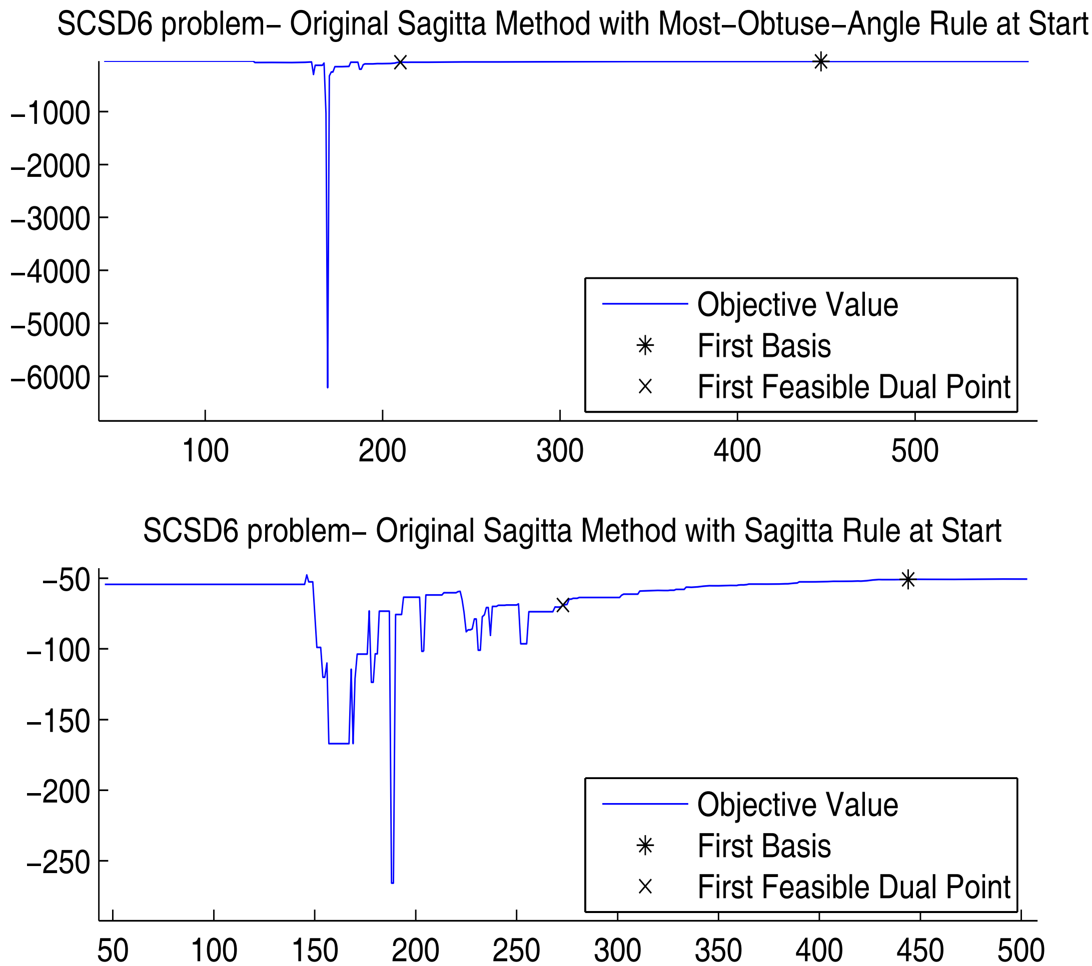
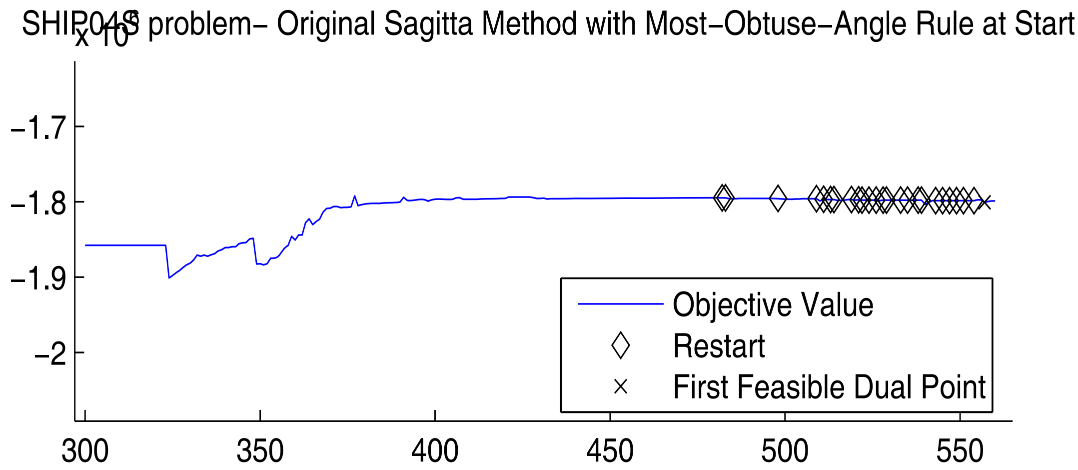


Fig. 28. *Objective of the problem SCSD6 solved by using Sagitta Method.*

TABLE A.28. Computational results for the Original Sagitta Method when solving SCSD6 *problem (#43, n=516, m=758)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	43	43	-4.90000000722709E+1	46	46	-5.42500000195656E+1
First feasible y	210	131	-6.93583147751484E+1	273	140	-6.89026580011426E+1
First square basis	447	147	-5.26650000653126E+1	444	147	-5.07500000810573E+1
First feasible x	564	147	-5.05000000771442E+1	503	147	-5.05000000776411E+1
Optimal solution	564	147	-5.05000000771442E+1	503	147	-5.05000000776411E+1
Restarts	0			0		
Time	7.86			6.86		



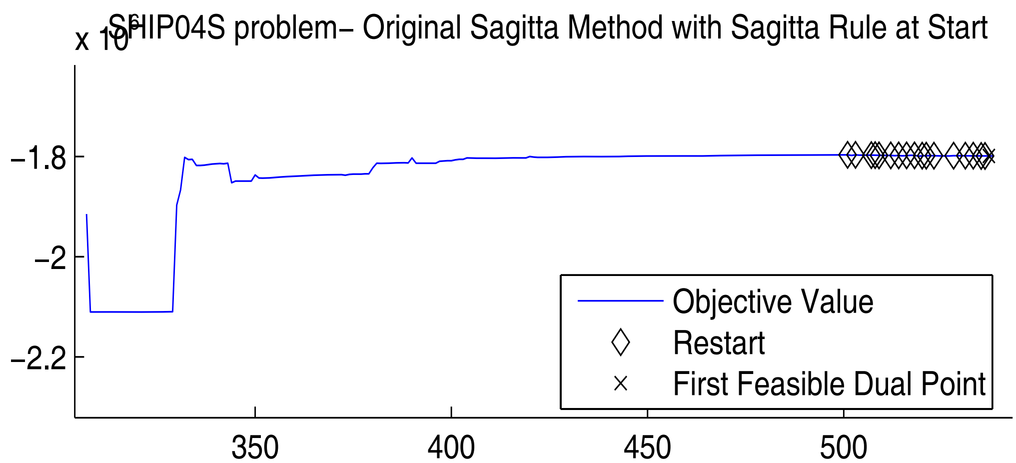


Fig. 29. *Objective of the problem SHIP04S solved by using Sagitta Method.*

TABLE A.29. Computational results for the Original Sagitta Method when solving SHIP04S *problem (#44, n=402, m=1506)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	300	300	-1.85761853959199E+6	307	307	-1.91457450260128E+6
First feasible y	554	325	-1.80055024260359E+6	537	327	-1.79878605195513E+6
First square basis	–	–	–	–	–	–
First feasible x	482	323	-1.79483272031952E+6	501	327	-1.79677347725534E+6
Optimal solution	557	325	-1.79871470044539E+6	538	327	-1.79871470044539E+6
Restarts	24			17		
Time	24.41			21.44		

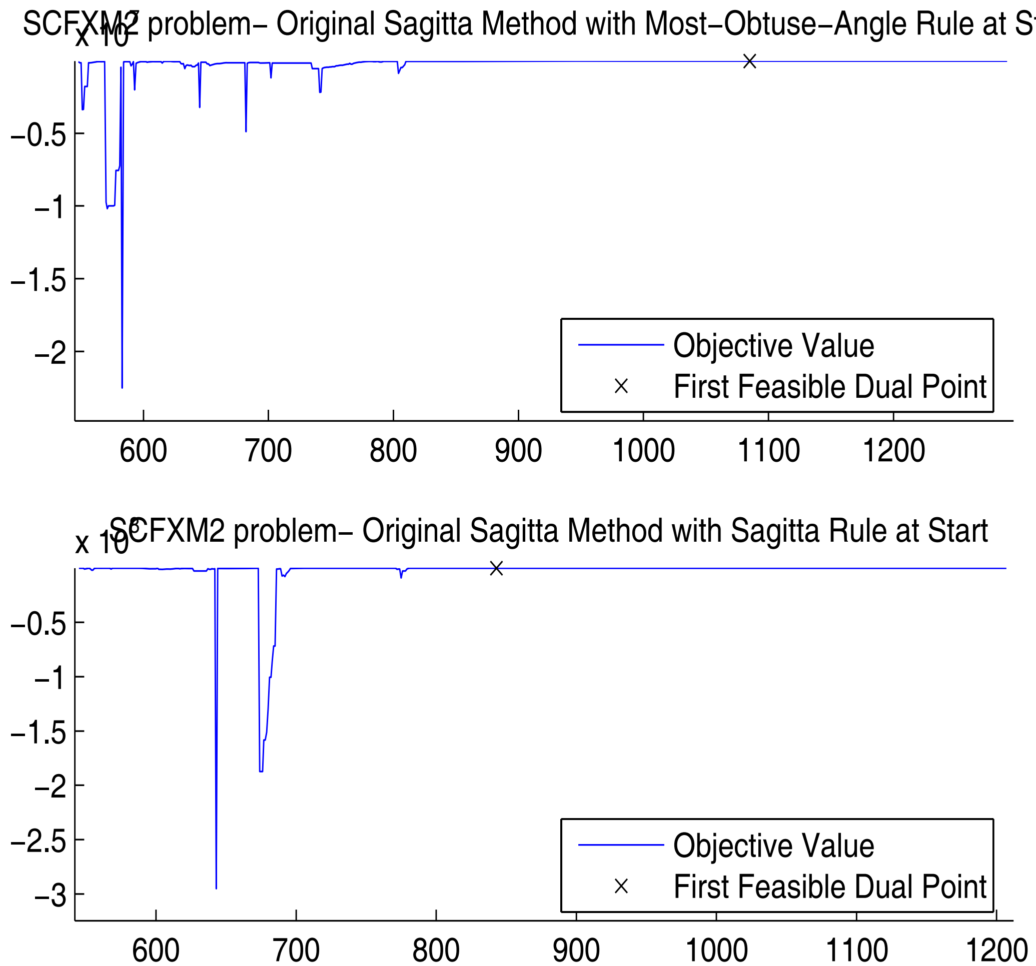


Fig. 30. *Objective of the problem SCFXM2 solved by using Sagitta Method.*

TABLE A.30. Computational results for the Original Sagitta Method when solving SCFXM2 *problem (#50, n=660, m=1200)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	*j*	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	548	548	-8.06800181458234E+4	545	545	-7.21650355030312E+4
First feasible y	1085	613	-3.80672801667306E+4	843	586	-4.53931152832170E+4
First square basis	–	–	–	–	–	–
First feasible x	1291	649	-3.66602615650319E+4	1207	647	-3.66602615650484E+4
Optimal solution	1291	649	-3.66602615650319E+4	1207	647	-3.66602615650484E+4
Restarts	0			0		
Time	99.47			90.88		

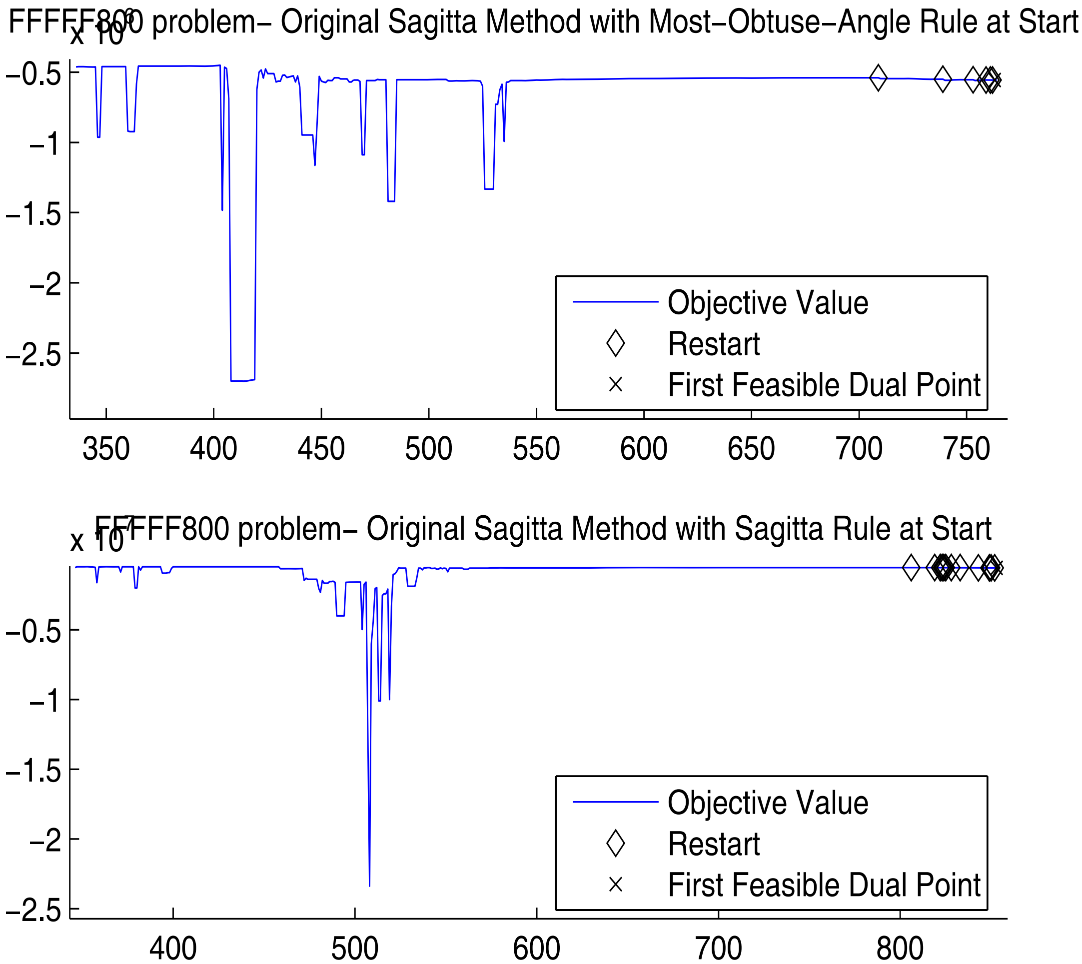


Fig. 31. *Objective of the problem FFFFF800 solved by using Sagitta Method.*

TABLE A.31. Computational results for the Original Sagitta Method when solving FFFFF800 *problem (#53, n=524, m=1028)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\vert\mathcal{A}_j\vert$	*Objective value*	j	$\vert\mathcal{A}_j\vert$	*Objective value*
First computed point	336	336	-4.62310312292295E+5	346	346	-5.43894509903694E+5
First feasible y	763	470	-5.55682032912169E+5	853	475	-5.55682032912253E+5
First square basis	–	–	–	–	–	–
First feasible x	709	450	-5.38535872057561E+5	806	469	-5.38907215735688E+5
Optimal solution	764	470	-5.55679564817521E+5	854	475	-5.55679564817608E+5
Restarts	6			12		
Time	31.92			36.69		

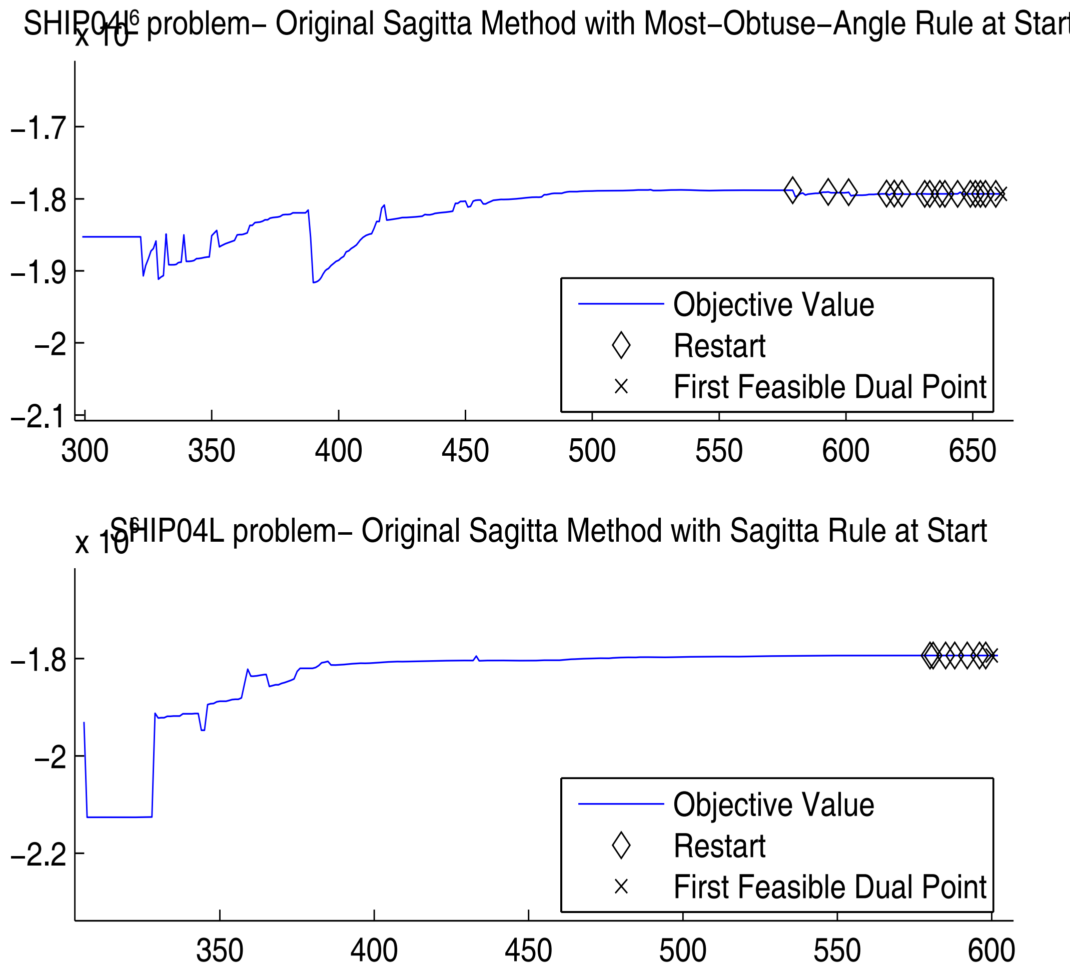


Fig. 32. *Objective of the problem SHIP04L solved by using Sagitta Method.*

TABLE A.32. Computational results for the Original Sagitta Method when solving SHIP04L *problem (#54, n=402, m=2166)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	299	299	-1.85266529842326E+6	306	306	-1.92965743392868E+6
First feasible y	661	323	-1.79332453797035E+6	600	327	-1.79334734385524E+6
First square basis	–	–	–	–	–	–
First feasible x	579	322	-1.78813945225358E+6	580	326	-1.79322642359446E+6
Optimal solution	661	323	-1.79332453797035E+6	602	327	-1.79332453797036E+6
Restarts	16			7		
Time	36.05			35.78		

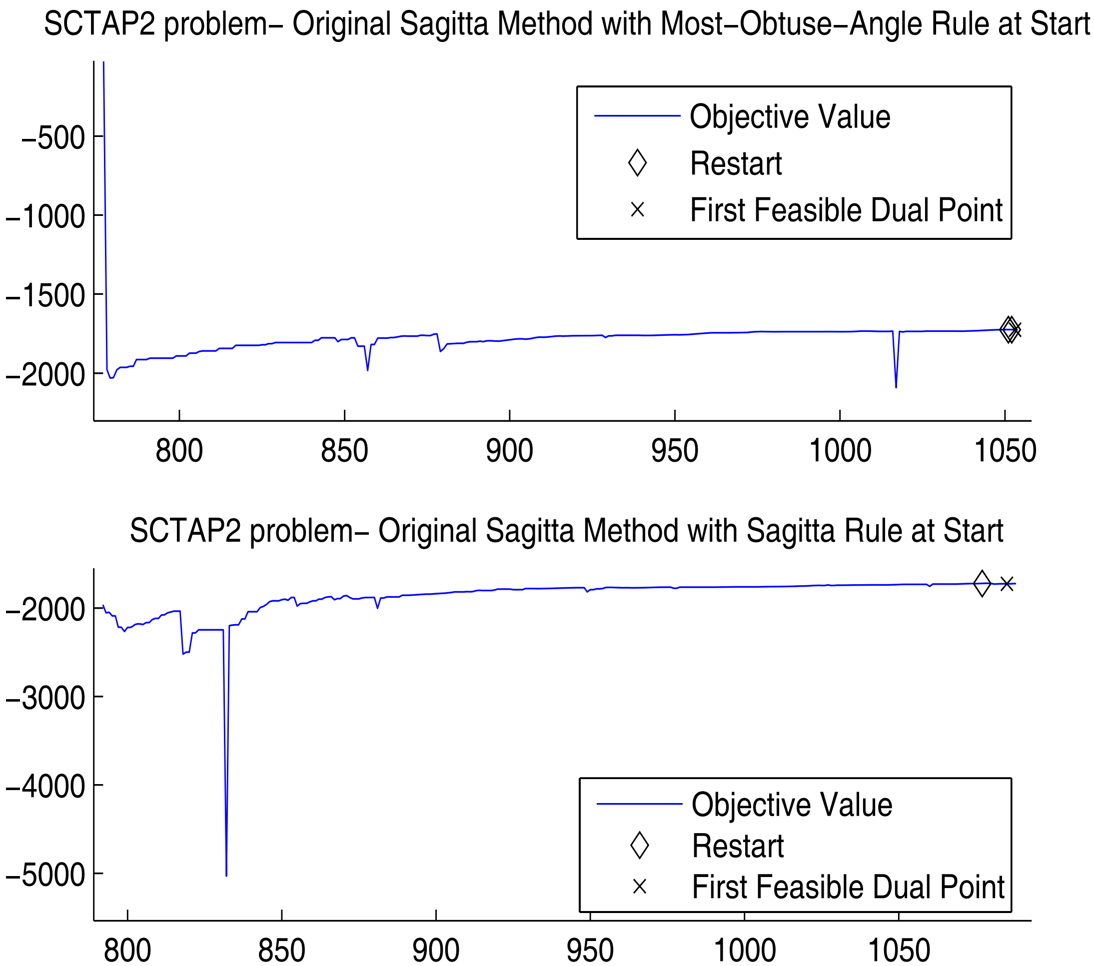


Fig. 33. *Objective of the problem SCTAP2 solved by using Sagitta Method.*

TABLE A.33. Computational results for the Original Sagitta Method when solving SCTAP2 *problem (#55, n=1090, m=2500)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	777	777	-2.65235040374149E+1	792	792	-1.96303730937751E+3
First feasible y	1053	869	-1.72480714285714E+3	1085	879	-1.72665967741935E+3
First square basis	–	–	–	–	–	–
First feasible x	1051	869	-1.72480714285714E+3	1077	876	-1.72337857142857E+3
Optimal solution	1053	869	-1.72480714285714E+3	1088	879	-1.72480714285714E+3
Restarts	2			1		
Time	167.61			173.89		

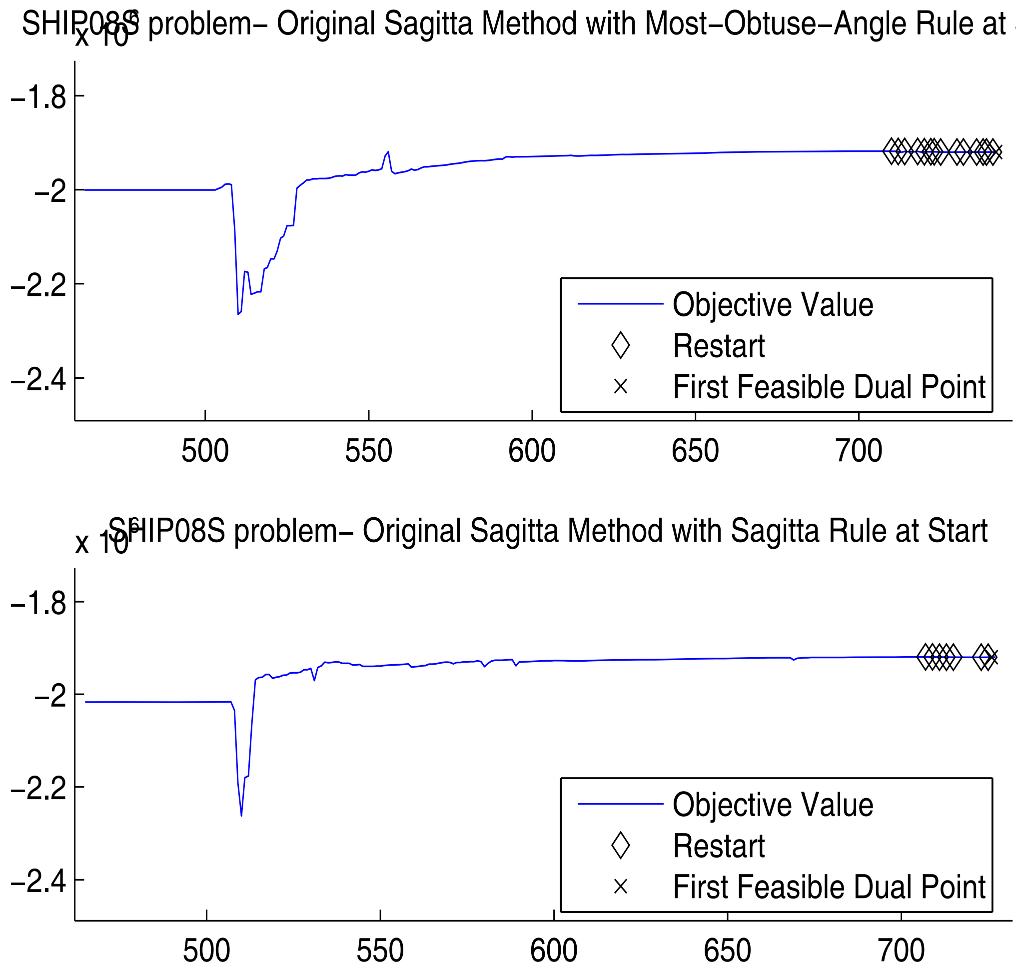


Fig. 34. *Objective of the problem SHIP08S solved by using Sagitta Method.*

TABLE A.34. Computational results for the Original Sagitta Method when solving SHIP08S *problem (#57, n=778, m=2735)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*	j	$\lvert\mathcal{A}_j\rvert$	*Objective value*
First computed point	463	463	-2.00049358446913E+6	465	465	-2.01664972229644E+6
First feasible y	742	503	-1.92009821053462E+6	726	506	-1.92009821053462E+6
First square basis	–	–	–	–	–	–
First feasible x	710	503	-1.91818310904259E+6	707	506	-1.91938829500898E+6
Optimal solution	742	503	-1.92009821053462E+6	727	506	-1.92009821053462E+6
Restarts	14			7		
Time	79.47			76.84		

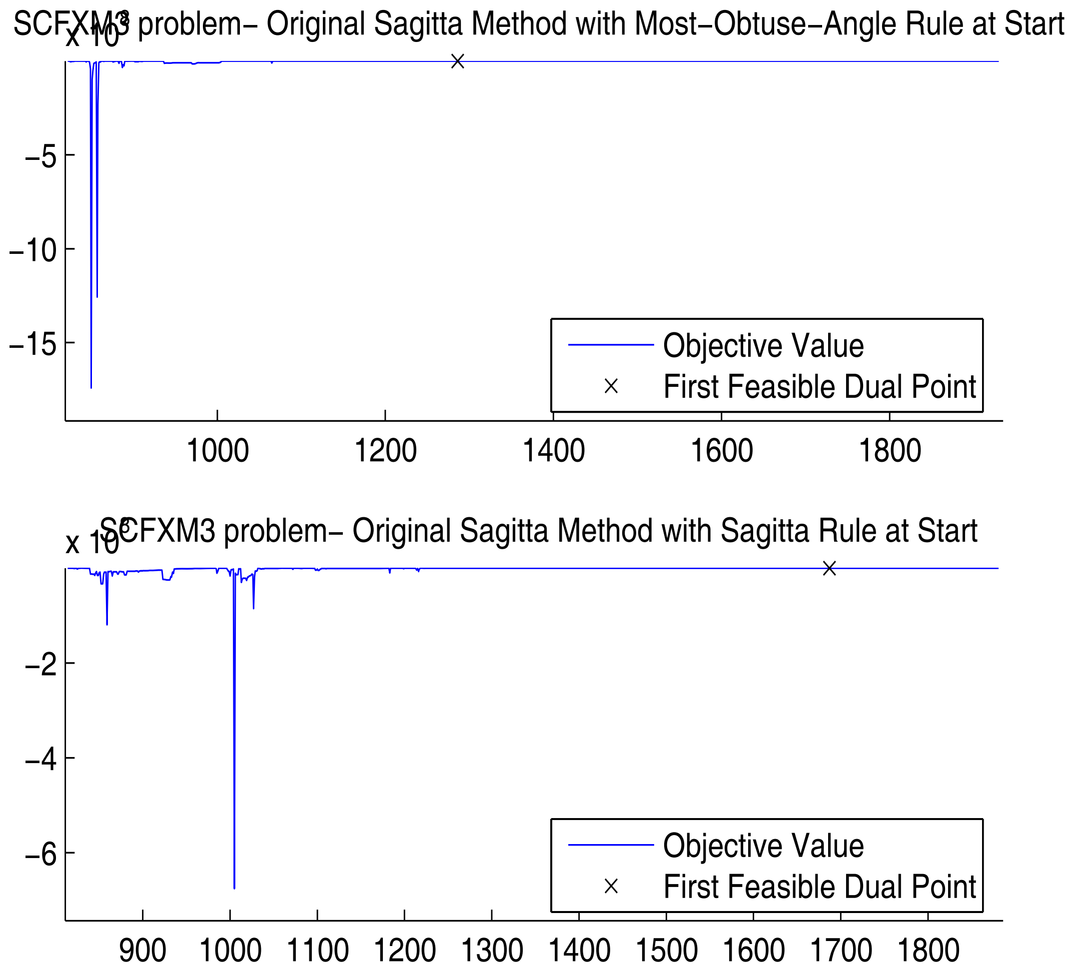


Fig. 35. *Objective of the problem SCFXM3 solved by using Sagitta Method.*

TABLE A.35. Computational results for the Original Sagitta Method when solving SCFXM3 *problem (#59, n=990, m=1800)*

Rule at Start:	*Most-Obtuse-Angle*			*Sagitta*		
	j	$\|\mathcal{A}_j\|$	*Objective value*	j	$\|\mathcal{A}_j\|$	*Objective value*
First computed point	822	822	-1.08299088954604E+5	814	814	-1.12145136427305E+5
First feasible y	1286	896	-8.71171626080006E+4	1687	929	-5.56797212171525E+4
First square basis	–	–	–	–	–	–
First feasible x	1930	974	-5.49012545486129E+4	1881	977	-5.49012545493280E+4
Optimal solution	1930	974	-5.49012545486129E+4	1881	977	-5.49012545493280E+4
Restarts	0			0		
Time	325.98			306.39		