TR-AC-0022

008

2次元プロットツール Ver.2（FD付）

新上 和正　山田 順一
北川 美宏 (ATR-I)

1998. 8.17

ATR環境適応通信研究所

目次

1. はじめに

２次元グラフプロットツールACRPlotToolを機能拡張し、Ver.2としてリリースしました。
今回拡張した主な機能は、カラー化、グラフの重ね描き、グラフ中へのラベル付与です。

前バージョン同様、種々のシミュレーションや実験から得られる数値データを、簡易なフォーマットで指定するだけで２次元グラフにプロットできます。またカラー化・グラフの重ね描きにより、複数のデータ系列をひとつのグラフ上で比較できるようになりました。グラフは、X-windowへ表示またはPS(Post Script),EPSファイルへ出力されますので、データの視覚化やドキュメント化にご活用ください。

2. 使用法

2.1. データファイルフォーマット

ACRPlotTool のデータファイルは、以下のフォーマットで記述されたASCII ファイルです。
なお、グラフ定義部の［線種］および［グラフ定義オプション］<参照：2.4.線種とグラフ定義オプション>は省略できます。

グラフ定義部	1行め	グラフタイトル
	2行め	グラフ定義数（G）　データ行数（R）　データ列数（C）
	3行め	第1グラフのx軸データ列　y軸データ列　［線種］［グラフ定義オプション...］
		第2グラフのx軸データ列　y軸データ列　［線種］［グラフ定義オプション...］
	…（G行分）	… … … …
		第Gグラフのx軸データ列　y軸データ列　［線種］［グラフ定義オプション...］
	3＋G行め	第1列データのタイトル
		第2列データのタイトル
	…（C行分）	…
		第C列データのタイトル
実データ部	3＋G＋C行め	第1列データ1　第2列データ1　第3列データ1　…　…　第C列データ1
		第1列データ2　第2列データ2　第3列データ2　…　…　第C列データ2
	…（R行分）	… … … … … …
		第1列データR　第2列データR　第3列データR　…　…　第C列データR

（注1）空行は無視されます。
（注2）「!」で始まる行は、コメント行となります。
（注3）「@@@@」で始まる行が実データ部にあると、それ以降のデータは無視されます。

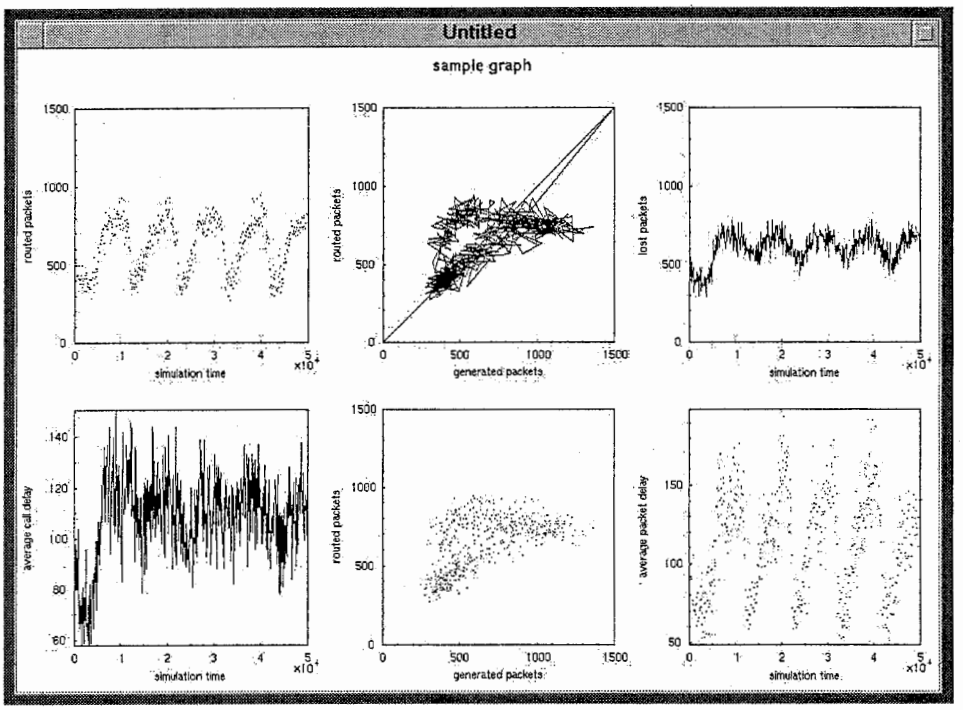
例：

```
sample graph
6   502   6
1  3  0
2  3  1
1  4  1
1  6  1
2  3  0
1  5  0
simulation time
generated packets
routed packets
lost packets
average packet delay
average call delay
100.0   547   362   339    67    66
200.0   598   544   499   100    94
300.0   434   521   545    92    98
400.0   468   437   494    91    94
500.0   375   433   385    92    80
600.0   387   421   458    75    80
700.0   504   439   441    77    81
800.0   486   500   478   106   104
  …  …  …  …  …  …  …
```

2.2. 実行

```
% plot -data data.1
```

この実行により、新しいウィンドウが開き次のようなグラフが表示されます。
開いたウィンドウは、（そのウィンドウにキーボードフォーカスがある状態で＝マウスカーソルをウィンドウ内に入れて）キーボードから'q'を入力すれば閉じられます。



また、コマンドラインオプション-ps でファイル名を指定すると、グラフをPostScript(PS)ファイルに出力できます。PSファイルは、以下のようにコマンド lpr を用いてPS対応プリンタへ出力することができます。

```
% plot -data data.1 -ps data.ps
% lpr -Pprinter_name data.ps
```

2.3. 実行時コマンドラインオプション

コマンド実行時に以下のオプションが使用できます。

- -data <ファイル名>
  読み込むデータファイル名を指定します。無指定の場合、既定値として fort.99 という名前のファイルが読み込まれます。

- -ps <ファイル名>
  出力する PostScript(PS)ファイル名を指定します。指定した場合、グラフはウィンドウ表示されません。

- -eps <ファイル名>
  出力する EPS ファイル名を指定します。指定した場合、グラフはウィンドウ表示されません。
  EPS ファイルは、EPS 形式に対応したアプリケーションソフトウェアで図として読み込むことができます。

- -landscape
  横置きでウィンドウ表示または PS または EPS ファイル出力されます(既定値)。

- -portrait
  縦置きでウィンドウ表示または PS または EPS ファイル出力されます。

- -aspectratio <比率>
  グラフの縦横比を実数で指定します。1 の場合、グラフが正方になります。

- -dotratio <比率>
  プロットする点の拡大縮小率を指定します。グラフ定義オプション dotsize の値とこの-dotratio の比率との積がプロットする点の大きさとなります。

- -head <ファイル名>
  グラフ定義部分のみのファイルを指定します。オプション-data と同時に指定した場合、-data で指定したデータファイルのグラフ定義部分は無視されます。

- -body <ファイル名>
  実データ部分のみのファイルを指定します。オプション-head と合わせて使用します。

2.4. 線種とグラフ定義オプション

データファイル中のグラフ定義部で以下の線種を指定できます。既定値は、0（点描）です。

線種	内容
0	点
1	実線
2	点線
3	1点鎖線
4	長点線
5	長1点鎖線

データファイル中のグラフ定義部で以下のグラフ定義オプションを使用できます。

オプション名	短縮形	内容	例	既定値
dotcolor	dc	色番号またはRGB値(#RRGGBB)でグラフの色を指定 0:black, 1:red, 2:blue, 3:green, 4:pink, 5:violet, 6:yellow, 7:orange, 8:brown, 9:spring-green	dotcolor=1 dotcolor=#00ffff	0
dotsize	ds	点の大きさ(線の太さ)を指定	dotsize=3	1
duplicate	d	直前のグラフへの重ね描きを指定	duplicate=1	-
label	l	ラベル用のデータ列を指定<参照：2.5.ラベルデータ列>	label=8	-
labeldistance	ld	ラベル文字列の中心と点との距離を指定	labeldistance=5	0
labelsize	ls	ラベル文字列の大きさを指定	labelsize=16	10

グラフ定義オプションは、空白文字で区切り、複数指定できます。同一オプションが、複数回指定された場合は、最後の指定が有効となります。

例1：

```
label=3 dotsize=2 dotcolor=1 duplicate=1
```

例2：（短縮形を使用）

```
l=3 ds=2 dc=1 d=1
```

2.5. ラベルデータ列

プロット点にラベル文字列を付けることができます。実データ列の並びにラベル文字列のデータ列を用意し、グラフ定義オプション label=**列番号** で指定します。また、数値データと区別するため、データ列タイトル部には「@label」と記述します。

例：

```
sample graph
6   502   7
1  3  0  label=7
2  3  1
1  4  1
1  6  1
2  3  0
1  5  0
simulation time
generated packets
routed packets
lost packets
average packet delay
average call delay
@label
100.0   547   362   339    67    66   min
200.0   598   544   499   100    94   @0
300.0   434   521   545    92    98   max
400.0   468   437   494    91    94   @0
500.0   375   433   385    92    80   @0
600.0   387   421   458    75    80   @0
700.0   504   439   441    77    81   @0
800.0   486   500   478   106   104   @0
  …   …   …   …   …   …   …
```

ラベルデータ列には、以下の予約語を使用できます。

予約語	内容
@0	ラベル無し
@x	点のｘ軸の値をラベルとする
@y	点のｙ軸の値をラベルとする
@000LABEL ～@359LABEL	ラベルLABEL の表示位置を指定する @に続く３桁の数字は角度、既定値 090

ラベルの表示位置は、次の手順で決定します<参照：2.4.線種とグラフ定義オプション>。

1. ラベルを付ける点Ａからグラフ定義オプションのlabeldistance分ｘ軸の正方向に離れた点Ｂを求める
2. 点Ａを中心として指定角度分だけ点Ｂを回転（左回りにプラス回転）させた点Ｃを求める
3. 点Ｃがラベル文字列の中心点となるようラベルを表示する

2.6. 稼働環境

HP-UX 10.xおよびSunOS 4.1.xでの稼働は確認済みです。ウィンドウ表示では、X-window(X11R6)環境を前提としています。なお、X-window環境があれば、他のプラットフォームでも稼働する可能性があります。

2.7. ファイル構成

本プロットツールは、以下のファイルで構成されています。

README
　プロットツール使用方法のドキュメント

bin/
　実行ファイルのディレクトリ

　plot_hp
　　HP用の実行ファイル

　plot_sun
　　SUN用の実行ファイル

sample/
　サンプルデータのディレクトリ

src/
　ソースプログラムのディレクトリ

　Imakefile
　　X環境上のMakefile作成パラメータファイル

　plot.c
　　プロットツールのメインモジュール

　draw.c
　　ウィンドウ表示／PostScript出力のモジュール

　graph.c
　　グラフ表示の汎用モジュール

　print.c
　　PostScript出力の汎用モジュール

　read.c
　　データファイル読み込みモジュール

　plot.h
　　プロットツールヘッダファイル

　graph.h
　　グラフ表示ヘッダファイル

tools/
　プロットツール関連ツールディレクトリ

2.8. 実行ファイルの作成

以下の手順により、各プラットフォーム用の実行ファイルを作成することができます。

1. 各ターゲットマシンへログインして、各プラットフォームの環境に合わせたMakefileを作成します。

```
% xmkmf
```

2. オブジェクトファイルをクリアします。

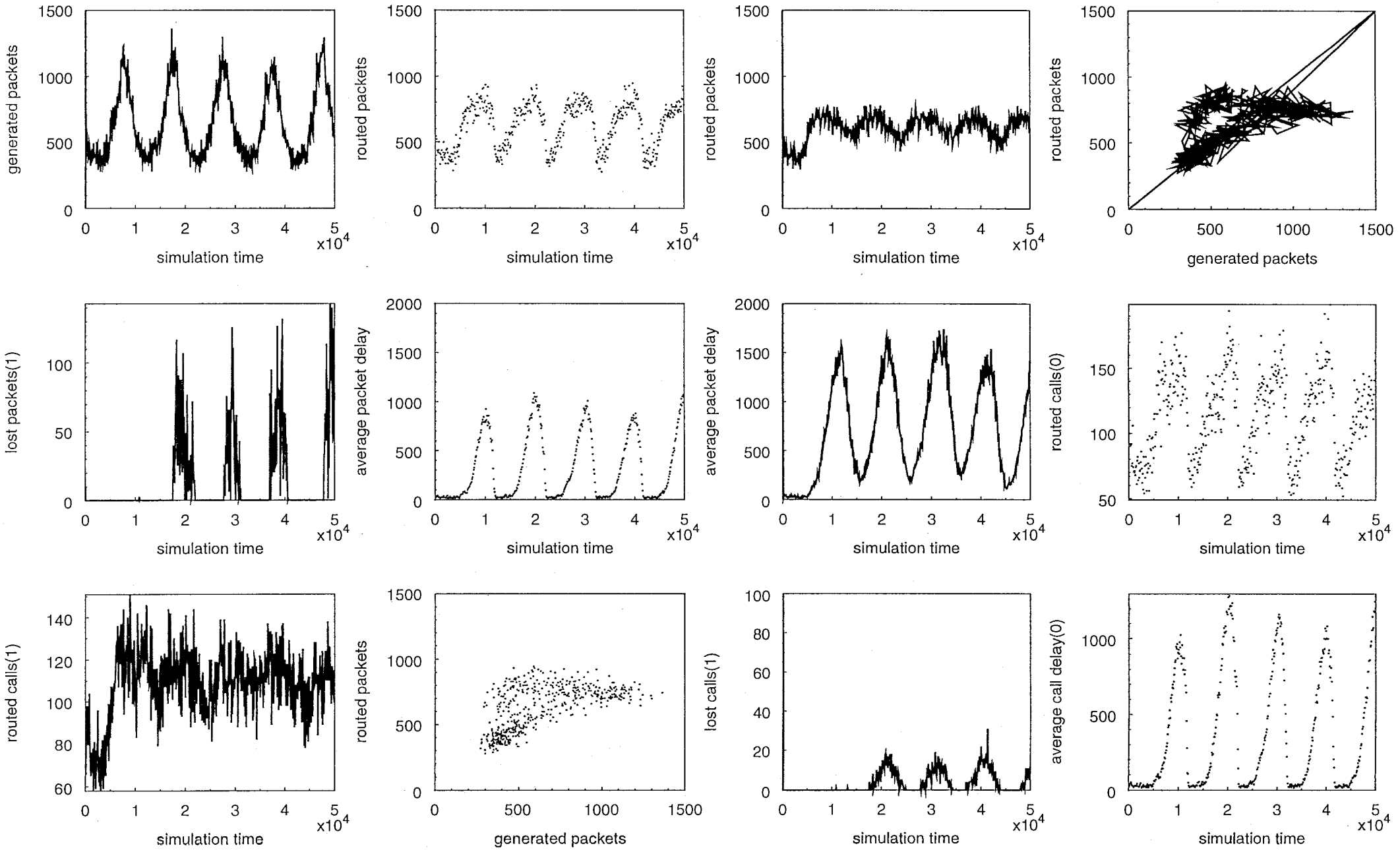
```
% make clean
```

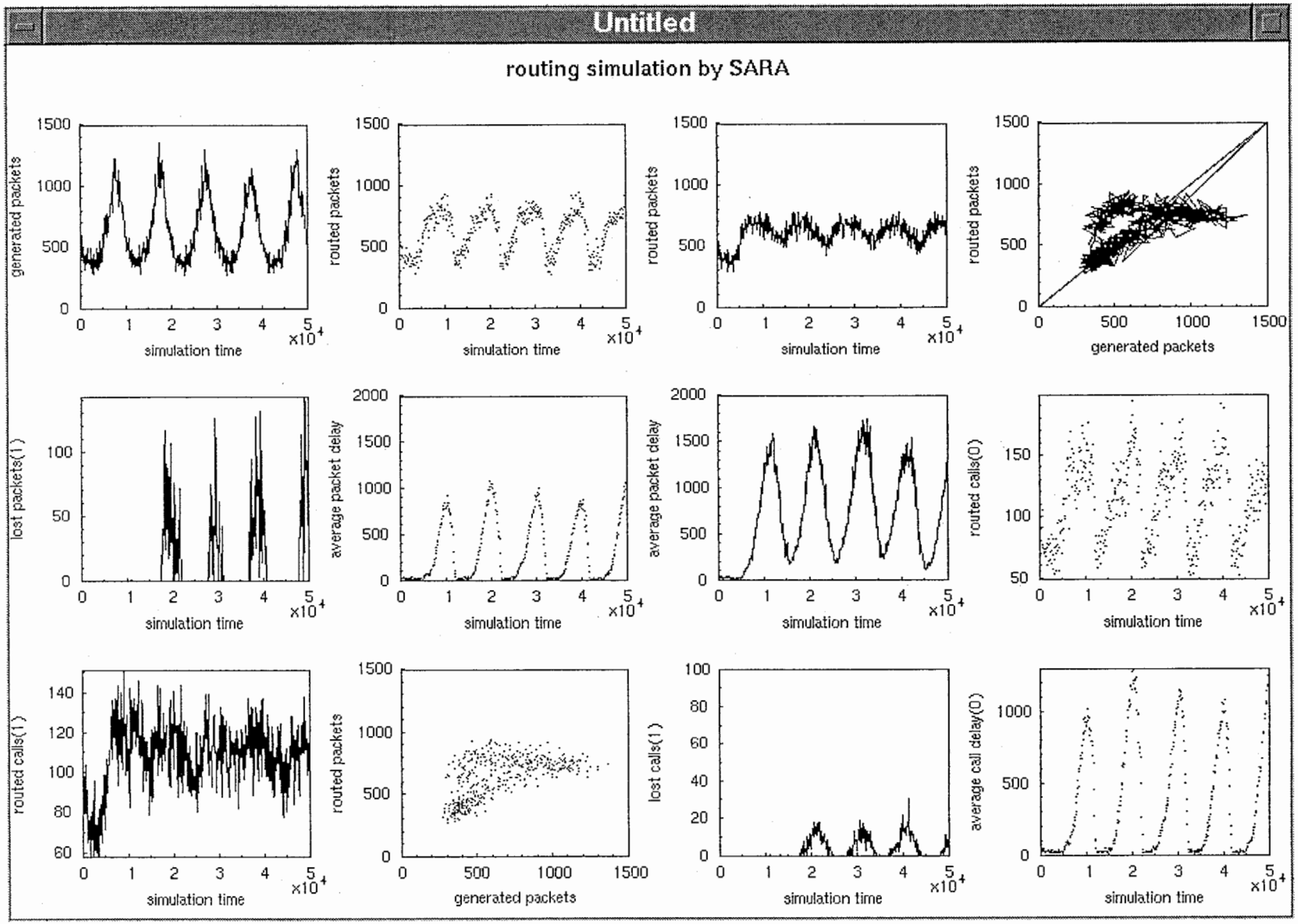
3. コンパイルして実行ファイルを作成します。

```
% make
```

A1 サンプルグラフおよびデータ

routing simulation by SARA




Untitled
routing simulation by SARA
generated packets
simulation time
routed packets
simulation time
routed packets
simulation time
routed packets
generated packets
lost packets(1)
simulation time
average packet delay
simulation time
average packet delay
simulation time
routed calls(0)
simulation time
routed calls(1)
simulation time
routed packets
generated packets
lost calls(1)
simulation time
average call delay(0)
simulation time
x10^4

routing simulation by SARA
12 502 14
1 2 1
1 3 0
1 4 1
2 3 1
1 6 1
1 7 0
1 8 1
1 9 0
1 10 1
2 3 0
1 12 1
1 13 0
simulation time
generated packets
routed packets
routed packets
lost packets(0)
lost packets(1)
average packet delay
average packet delay
routed calls(0)
routed calls(1)
lost calls(0)
lost calls(1)
average call delay(0)
average call delay(1)

0.0	0	0	0	0	0	0.0	0.0	67	66	0	0	0.0	0.0
0.0	1500	1500	1500	0	10	2000.0	2000.0	67	66	0	100	200.0	200.0
100.0	547	362	339	0	0	19.34	18.83	67	66	0	0	20.48	20.58
200.0	598	544	499	0	0	25.27	36.48	100	94	0	0	28.91	40.05
300.0	434	521	545	0	0	37.72	49.24	92	98	0	0	40.96	49.73
400.0	468	437	494	0	0	38.43	39.42	91	94	0	0	46.94	44.42
500.0	375	433	385	0	0	43.51	40.74	92	80	0	0	52.06	44.73
600.0	387	421	458	0	0	33.15	43.99	75	80	0	0	39.90	53.65
700.0	504	439	441	0	0	25.91	27.31	77	81	0	0	31.82	35.25
800.0	486	500	478	0	0	24.82	25.01	106	104	0	0	27.15	26.64
900.0	368	417	442	0	0	24.56	26.36	74	80	0	0	27.27	28.50
1000.0	323	333	323	0	0	16.08	16.25	71	69	0	0	19.25	17.43
1100.0	438	408	405	0	0	21.79	23.06	70	65	0	0	24.36	22.40
1200.0	374	385	399	0	0	22.55	27.03	71	76	0	0	26.46	33.07
1300.0	452	437	401	0	0	25.91	24.60	68	67	0	0	30.85	27.02
1400.0	368	382	383	0	0	24.24	36.13	74	70	0	0	27.04	36.13
1500.0	377	366	387	0	0	25.75	26.80	75	78	0	0	26.57	30.12
1600.0	492	420	410	0	0	23.44	27.08	61	58	0	0	29.94	31.68
1700.0	402	499	454	0	0	38.95	48.35	79	77	0	0	41.49	51.53
1800.0	394	401	434	0	0	24.38	40.34	82	81	0	0	25.95	39.62
1900.0	404	384	387	0	0	23.06	27.10	64	67	0	0	25.32	30.84
2000.0	403	415	425	0	0	21.42	27.71	77	77	0	0	24.72	28.19
2100.0	336	338	299	0	0	19.31	30.47	66	59	0	0	21.22	29.17
2200.0	501	441	478	0	0	19.02	28.44	71	77	0	0	25.07	36.16
2300.0	466	439	429	0	0	34.67	39.18	72	72	0	0	35.20	34.41
2400.0	480	450	465	0	0	39.77	44.14	78	78	0	0	45.55	48.16
2500.0	292	420	443	0	0	41.23	39.06	86	96	0	0	44.94	43.54
2600.0	331	338	326	-1.2	0	16.35	13.17	73	69	0	0	16.89	13.77
2700.0	418	370	384	0	0	20.08	20.64	62	62	0	0	22.55	22.31
2800.0	346	390	357	0	0	24.48	24.38	69	67	0	0	26.59	21.84
2900.0	447	361	370	0	0	20.38	22.68	57	60	0	0	20.69	26.13
3000.0	278	386	373	0	0	31.98	38.28	82	74	0	0	34.42	41.09
3100.0	401	352	360	0	0	18.78	22.20	54	58	0	0	20.73	24.90
3200.0	361	411	425	0	0	24.38	26.65	75	78	0	0	25.86	27.37
3300.0	390	366	367	0	0	19.34	19.77	66	66	0	0	23.98	22.85
3400.0	394	410	420	0	0	21.23	21.04	82	82	0	0	26.93	25.24
3500.0	325	350	341	0	0	15.49	13.98	76	75	0	0	17.23	15.75
3600.0	346	285	294	0	0	14.12	13.09	56	59	0	0	16.24	14.66
3700.0	501	408	394	0	0	24.11	21.21	70	69	0	0	30.14	24.00
3800.0	427	459	432	0	0	35.70	34.37	95	84	0	0	38.88	33.83
3900.0	318	424	439	0	0	23.70	37.49	77	84	0	0	26.46	38.00
4000.0	379	379	384	0	0	15.49	18.61	73	72	0	0	16.75	19.10
4100.0	494	332	384	0	0	21.64	18.63	56	65	0	0	23.21	23.39
4200.0	559	552	478	0	0	28.22	29.79	98	84	0	0	34.74	31.87
4300.0	343	420	472	0	0	34.81	37.21	75	89	0	0	35.41	38.64
4400.0	440	462	392	0	0	34.14	33.84	88	79	0	0	35.55	33.29
4500.0	501	430	455	0	0	23.91	33.60	87	87	0	0	29.91	35.81
4600.0	378	479	507	0	0	26.13	29.86	96	99	0	0	32.45	35.36
4700.0	392	390	391	0	0	21.12	24.47	70	72	0	0	21.44	24.99
4800.0	478	466	447	0	0	22.57	23.77	97	89	0	0	28.81	25.28
4900.0	545	478	491	0	0	21.74	23.35	89	99	0	0	24.84	27.51
5000.0	689	526	537	10	0	32.88	34.59	81	83	0	0	41.48	40.19
5100.0	559	614	656	0	0	49.34	47.98	111	107	0	0	61.94	52.22
5200.0	746	584	536	0	0	51.06	42.28	115	107	0	0	58.73	51.36
5300.0	507	589	555	0	0	55.61	51.12	90	98	0	0	55.68	53.77
5400.0	718	635	592	0	0	58.53	66.17	99	90	0	0	67.64	68.14
5500.0	764	642	567	0	0	58.76	62.65	101	104	0	0	68.67	68.89
5600.0	714	715	683	0	0	78.02	88.54	115	98	0	0	88.97	77.68
5700.0	712	683	625	0	0	78.45	93.96	137	109	0	0	92.28	93.66
5800.0	595	666	565	0	0	79.42	119.42	115	105	0	0	86.24	105.17
5900.0	780	730	621	0	0	86.52	111.42	137	108	0	0	104.12	110.33
6000.0	624	693	644	0	0	72.92	136.87	148	116	0	0	92.06	154.64
6100.0	724	743	716	0	0	68.94	138.79	130	127	0	0	76.46	167.45
6200.0	682	730	665	0	0	66.37	137.64	147	128	0	0	80.24	167.47
6300.0	822	575	643	0	0	66.47	160.27	118	119	0	0	81.21	179.86
6400.0	848	759	609	0	0	71.68	140.96	161	136	0	0	86.33	175.94
6500.0	758	642	654	0	0	98.70	157.46	90	125	0	0	118.80	217.06
6600.0	864	752	668	0	0	116.15	176.88	141	107	0	0	135.71	206.30
6700.0	863	842	697	0	0	121.95	168.22	170	129	0	0	142.21	180.30
6800.0	959	752	666	0	0	126.60	198.00	142	118	0	0	162.12	264.87
6900.0	962	732	760	0	0	132.47	216.02	148	130	0	0	141.03	223.57
7000.0	796	731	735	0	0	153.43	270.16	138	135	0	0	173.38	287.73
7100.0	1127	767	716	0	0	160.50	245.45	125	137	0	0	181.91	290.66
7200.0	1071	821	681	0	0	172.89	237.50	155	121	0	0	213.61	300.92
7300.0	1137	785	747	0	0	179.35	221.13	140	107	0	0	207.52	270.29
7400.0	1073	779	672	0	0	208.22	281.57	152	111	0	0	243.77	330.39
7500.0	1227	807	706	0	0	228.35	351.60	145	129	0	0	261.03	399.63
7600.0	1232	643	672	0	0	259.54	342.50	126	144	0	0	287.03	411.55
7700.0	1053	778	606	0	0	300.51	345.19	126	94	0	0	308.37	386.41
7800.0	1165	661	626	0	0	325.05	426.86	114	103	0	0	368.11	462.15
7900.0	1060	769	661	0	0	363.71	475.10	121	116	0	0	414.06	520.19
8000.0	1028	744	683	0	0	394.60	454.29	129	109	0	0	428.10	485.12
8100.0	1041	704	717	0	0	431.84	550.61	124	127	0	0	471.74	593.09
8200.0	1053	695	731	0	0	442.67	536.78	107	123	0	0	526.55	559.31
8300.0	833	800	741	0	0	504.68	592.53	136	127	0	0	475.85	699.96
8400.0	892	864	684	0	0	527.95	672.64	138	140	0	0	607.80	667.76
8500.0	1130	700	722	0	0	537.36	647.12	119	109	0	0	582.98	763.45
8600.0	950	799	637	0	0	608.65	722.87	156	102	0	0	628.84	861.09
8700.0	1003	699	649	0	0	582.39	676.21	153	120	0	0	677.79	753.43
8800.0	974	838	657	0	0	587.62	703.27	121	105	0	0	698.17	848.49
8900.0	788	834	740	0	0	619.40	763.73	154	151	0	0	673.04	860.36
9000.0	830	752	752	0	0	633.49	763.89	146	140	0	0	718.07	757.34
9100.0	806	828	688	0	0	683.92	868.63	131	119	0	0	763.73	985.99
9200.0	763	847	779	0	0	707.09	803.68	136	119	0	0	807.88	993.11
9300.0	748	827	718	0	0	768.57	830.53	168	125	0	0	847.55	975.47
9400.0	785	804	587	0	0	789.14	1007.86	132	99	0	0	858.84	1080.34
9500.0	855	903	632	0	0	814.91	923.91	174	102	0	0	946.44	998.03
9600.0	599	757	694	0	0	791.39	947.35	141	101	0	0	934.45	1052.41
9700.0	631	910	737	0	0	843.63	1023.37	164	129	0	0	964.21	1191.25
9800.0	622	734	658	0	0	792.24	1076.56	145	118	0	0	871.32	1179.80
9900.0	538	799	700	0	0	791.46	1093.95	129	107	0	0	909.03	1170.98
10000.0	593	813	660	0	0	801.55	1167.62	140	104	0	0	927.08	1345.99
10100.0	469	934	602	0	0	791.74	1271.92	144	97	0	0	902.71	1321.49
10200.0	482	724	557	0	0	924.59	1233.91	159	83	0	0	976.51	1388.73
10300.0	549	767	630	0	0	813.01	1168.51	160	113	0	0	941.67	1478.94
10400.0	549	738	684	0	0	855.19	1279.68	154	142	0	0	1024.72	1515.59
10500.0	443	849	767	0	0	757.75	1239.05	122	131	0	0	886.24	1595.07
10600.0	578	917	699	0	0	790.22	1272.33	154	124	0	0	836.97	1485.10
10700.0	463	790	631	0	1	857.49	1359.03	177	111	0	0	956.93	1431.99
10800.0	424	635	633	0	0	778.19	1453.43	125	110	0	0	874.95	1647.48
10900.0	385	754	629	0	1	725.49	1296.58	136	120	0	1	796.27	1515.63
11000.0	471	679	597	0	0	746.23	1317.43	130	112	0	0	798.26	1590.42
11100.0	461	793	765	0	0	623.03	1261.38	145	128	0	0	872.73	1484.11
11200.0	481	716	708	0	0	588.09	1458.81	150	137	0	0	766.61	1663.30
11300.0	377	775	739	0	0	599.51	1340.64	148	121	0	0	794.73	1601.61
11400.0	343	770	619	0	0	605.83	1477.31	134	131	0	0	691.49	1655.01
11500.0	382	748	676	0	0	500.89	1464.11	144	119	0	0	644.78	1717.54
11600.0	449	681	626	0	0	315.74	1359.33	159	114	0	0	456.07	1824.62
11700.0	375	616	694	0	0	207.78	1481.62	127	129	0	0	276.30	1676.26
11800.0	379	595	607	0	0	110.45	1542.25	116	134	0	0	111.36	1750.48
11900.0	338	466	559	0	0	78.11	1583.63	106	118	0	0	104.28	1789.39
12000.0	421	444	582	0	0	40.65	1559.91	73	126	0	0	48.33	1751.07
12100.0	405	408	665	-9	0	22.69	1543.46	84	137	0.08	0	29.47	1697.27
12200.0	349	388	620	0	0	22.50	1422.40	71	146	0	0	24.43	1765.33
12300.0	383	352	608	0	0	14.81	1400.14	71	142	0	0	18.24	1584.45
12400.0	374	365	602	0	0	17.91	1211.89	65	109	0	0	19.27	1544.79
12500.0	360	381	577	0	0	27.18	1149.18	65	119	0	0	24.24	1735.14
12600.0	421	436	594	0	0	23.36	1200.58	79	114	0	0	27.12	1585.35
12700.0	418	362	539	0	0	22.14	1255.48	59	108	0	0	22.98	1599.56
12800.0	341	349	583	0	0	22.21	1204.27	68	109	0	0	26.22	1533.57
12900.0	525	468	622	0	0	25.74	1070.19	68	118	0	0	29.77	1357.95
13000.0	358	432	595	0	0	26.99	1245.66	84	136	0	0	31.66	1419.91
13100.0	428	386	529	0	0	18.80	898.99	62	91	0	0	22.51	1336.76
13200.0	411	439	614	0	0	24.81	998.40	80	122	0	1	27.35	1320.31
13300.0	279	364	613	0	0	17.21	942.45	74	135	0	0	21.42	1276.87
13400.0	497	332	567	0	0	15.98	982.02	49	101	0	0	17.43	1146.59
13500.0	408	458	580	0	0	26.51	807.06	89	108	0	0	28.09	1144.19
13600.0	488	489	596	0	0	25.35	657.25	85	115	0	0	27.78	990.02

13700.0	443	445	596	0	0	28.18	677.80	85	102	0	0	32.88	872.46
13800.0	526	435	595	0	0	23.46	573.30	72	116	0	0	27.89	849.54
13900.0	466	477	520	0	0	42.28	523.03	70	100	0	0	49.31	695.65
14000.0	538	512	526	0	0	39.99	579.90	78	96	0	0	47.80	958.22
14100.0	452	472	586	0	0	49.32	614.60	76	116	0	0	53.96	831.51
14200.0	460	570	529	0	0	44.94	435.38	88	101	0	0	54.24	531.37
14300.0	395	434	625	0	0	24.47	502.15	81	116	0	0	30.34	666.90
14400.0	466	409	525	0	0	22.53	394.63	76	100	0	0	23.71	549.11
14500.0	489	518	506	0	0	25.26	253.49	100	79	0	0	31.86	329.29
14600.0	588	476	497	0	0	23.12	417.85	86	96	0	0	26.46	509.98
14700.0	527	492	545	0	0	37.31	361.88	89	89	0	0	40.70	354.43
14800.0	570	570	647	0	0	40.94	264.14	99	113	0	0	53.28	298.00
14900.0	451	555	519	0	0	48.25	336.41	113	118	0	0	53.53	439.09
15000.0	466	457	501	0	0	32.30	402.63	92	86	0	0	33.21	467.78
15100.0	513	472	585	0	0	32.90	333.61	91	122	0	0	36.25	341.27
15200.0	644	565	526	0	0	35.50	243.22	107	94	0	0	39.85	222.43
15300.0	530	616	703	0	0	39.36	216.74	109	121	0	0	49.20	242.64
15400.0	689	575	577	0	0	37.48	196.99	104	111	0	0	42.37	233.68
15500.0	575	553	554	0	0	49.28	227.85	98	88	0	0	49.73	286.08
15600.0	676	612	619	0	0	53.05	176.86	91	120	0	0	53.78	205.06
15700.0	780	654	672	0	0	63.27	193.47	129	107	0	0	82.12	180.49
15800.0	811	619	603	0	0	68.53	191.62	113	123	0	0	77.64	182.30
15900.0	679	752	658	0	0	86.42	199.56	131	107	0	0	103.56	217.85
16000.0	789	719	669	0	0	83.94	215.50	144	117	0	0	99.83	229.03
16100.0	694	612	593	0	0	85.61	244.35	99	121	0	0	85.28	317.97
16200.0	799	739	663	0	0	94.74	247.12	114	121	0	0	114.02	319.46
16300.0	936	657	657	0	0	121.57	215.62	120	104	0	0	131.61	230.91
16400.0	868	699	695	0	0	112.98	225.84	123	121	0	0	148.01	222.06
16500.0	1102	699	592	0	0	146.87	289.31	121	99	0	0	174.15	271.79
16600.0	847	701	773	0	0	150.85	269.77	119	144	0	0	154.14	309.18
16700.0	995	632	669	0	0	193.49	353.63	114	119	0	0	245.25	358.16
16800.0	976	757	774	0	0	186.48	369.36	102	104	0	0	230.00	421.92
16900.0	920	758	680	0	0	229.72	312.03	139	125	0	0	303.15	388.13
17000.0	1080	755	769	0	0	231.90	339.94	114	142	0	0	237.41	409.50
17100.0	1061	728	612	0	0	276.52	373.05	126	120	0	0	308.68	379.10
17200.0	1132	699	626	0	0	307.79	346.62	138	108	0	0	350.58	349.06
17300.0	1364	742	646	0	0	329.86	363.69	130	117	0	0	351.26	433.96
17400.0	1118	709	539	0	0	331.06	452.87	128	101	0	0	374.41	455.72
17500.0	997	711	678	0	0	367.68	412.27	101	107	0	0	412.94	483.36
17600.0	1051	714	661	0	40	409.01	482.55	115	120	0	4	463.84	594.33
17700.0	1196	823	660	0	35	409.79	503.68	145	130	0	0	499.76	544.30
17800.0	1078	709	784	0	13	461.30	570.38	130	120	0	0	491.54	578.20
17900.0	994	666	740	0	13	500.65	590.54	129	109	0	0	625.68	722.13
18000.0	1222	767	663	0	59	512.78	472.95	116	118	0	2	611.56	537.35
18100.0	1112	714	697	0	66	563.92	593.51	113	106	0	3	645.10	618.93
18200.0	1081	788	606	0	95	528.71	646.15	111	117	0	4	632.66	685.91
18300.0	1037	761	700	0	117	628.73	705.73	114	124	0	8	630.45	707.32
18400.0	1122	780	650	0	30	635.12	644.26	144	116	0	0	712.83	686.62
18500.0	918	792	680	0	25	644.57	674.95	128	117	0	1	713.29	712.97
18600.0	903	733	749	0	88	746.69	787.94	103	108	0	3	794.97	863.24
18700.0	904	814	723	0	91	705.16	800.78	101	98	0	4	824.54	938.65
18800.0	705	820	749	0	62	720.92	793.54	128	104	0	4	869.20	806.07
18900.0	739	790	741	0	18	816.29	843.88	146	115	0	1	966.06	900.86
19000.0	702	737	744	0	42	832.93	946.75	134	133	0	10	959.19	962.04
19100.0	662	722	617	0	88	859.31	1040.00	126	94	0	5	1013.73	1051.30
19200.0	679	728	624	0	75	861.29	1110.33	111	119	0	9	1070.06	1157.79
19300.0	690	908	693	0	39	834.39	988.84	149	93	0	5	1043.68	1097.37
19400.0	716	897	678	0	20	875.29	941.04	160	131	0	4	1038.21	1041.18
19500.0	676	855	753	0	28	991.76	1146.59	140	136	0	6	1082.38	1146.00
19600.0	571	855	650	0	107	1027.89	1220.16	155	113	0	10	1048.89	1126.26
19700.0	669	820	684	0	48	987.79	1257.57	153	116	0	12	1182.21	1364.85
19800.0	651	831	777	0	61	1087.28	1225.32	151	112	0	7	1174.93	1151.76
19900.0	508	808	691	0	0	1049.94	1271.36	177	125	0	4	1160.67	1239.00
20000.0	481	843	561	0	16	1048.49	1474.14	153	102	0	11	1299.20	1518.59
20100.0	609	840	733	0	49	1018.74	1177.42	161	141	0	8	1276.69	1448.32
20200.0	425	853	657	0	47	1001.89	1363.05	166	120	0	15	1199.08	1408.41
20300.0	594	933	629	0	0	1050.10	1443.91	194	121	0	11	1286.06	1587.21
20400.0	583	905	662	0	77	949.84	1233.20	182	127	0	14	1214.85	1337.49
20500.0	449	772	659	0	43	965.08	1396.49	156	118	0	11	1223.21	1483.77
20600.0	521	765	744	0	6	910.19	1300.12	171	124	0	10	1169.02	1443.56
20700.0	433	763	598	0	29	918.94	1499.40	165	103	0	17	1219.33	1408.80
20800.0	411	796	554	0	14	945.03	1501.37	160	97	0	18	1154.80	1597.62
20900.0	407	716	581	0	7	965.71	1637.82	163	124	0	12	1238.99	1755.59
21000.0	312	746	641	0	13	847.00	1663.22	172	107	0	14	1133.27	1922.15
21100.0	418	802	587	0	33	770.36	1529.73	168	88	0	16	1008.29	1563.34
21200.0	418	697	583	0	0	728.98	1644.46	142	125	0	15	987.58	1853.44
21300.0	438	652	631	0	1	753.54	1649.34	147	110	0	11	983.75	1504.09
21400.0	382	653	565	0	24	627.65	1433.30	120	101	0	17	788.76	1517.40
21500.0	438	629	587	0	72	532.24	1459.50	125	95	0	14	751.07	1493.73
21600.0	350	706	645	0	38	487.40	1530.00	145	124	0	15	718.13	1719.88
21700.0	387	634	689	0	22	467.63	1639.55	120	144	0	18	752.30	1818.13
21800.0	406	711	630	0	12	462.63	1400.90	154	96	0	9	686.21	1683.12
21900.0	444	660	558	0	15	306.42	1385.81	134	90	0	14	503.33	2007.10

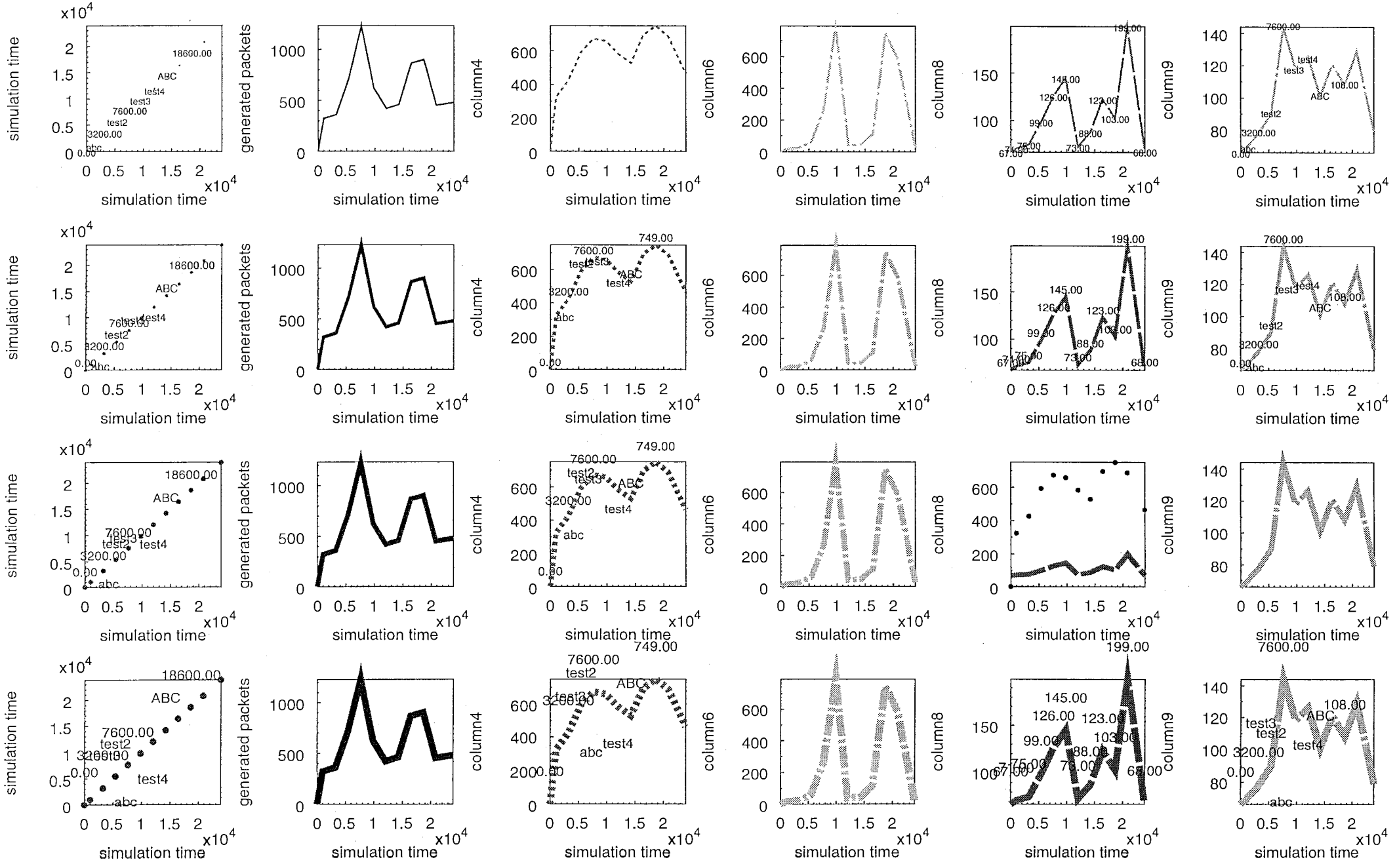
22000.0	392	667	660	0	0	165.35	1386.23	142	121	0	18	267.49	1593.61
22100.0	325	514	659	0	0	94.59	1409.78	106	112	0	6	141.35	1716.51
22200.0	345	435	659	0	0	33.27	1264.65	72	126	0	9	40.05	1491.44
22300.0	403	347	548	0	0	21.57	1359.66	59	99	0	9	22.91	1695.96
22400.0	349	367	590	0	0	19.83	1399.32	66	119	0	13	24.06	1705.72
22500.0	339	341	550	0	0	17.28	1221.08	67	106	0	7	19.40	1419.31
22600.0	367	365	576	0	0	17.70	1099.57	70	119	0	10	16.68	1622.44
22700.0	392	348	546	0	0	20.64	1271.85	65	112	0	7	24.74	1575.93
22800.0	373	364	597	0	0	27.16	1258.66	63	123	0	9	28.19	1539.35
22900.0	272	316	494	0	0	22.64	1206.50	70	94	0	12	23.93	1503.97
23000.0	429	342	515	0	0	23.52	1134.12	60	102	0	8	27.24	1478.75
23100.0	310	409	517	0	0	23.53	923.72	82	110	0	8	23.74	1151.96
23200.0	518	433	588	0	0	26.80	792.14	62	107	0	5	28.61	1186.77
23300.0	340	451	508	0	0	30.00	813.40	90	91	0	4	34.51	1076.17
23400.0	377	365	535	0	0	16.75	1033.44	73	112	0	9	19.06	1064.52
23500.0	448	386	580	0	0	15.29	747.02	78	114	0	8	17.99	782.34
23600.0	405	416	552	0	0	21.60	697.84	73	103	0	0	23.16	814.77
23700.0	385	426	580	0	0	24.13	676.09	74	90	0	1	27.96	789.67
23800.0	355	357	431	0	0	17.22	653.47	72	92	0	1	20.61	965.19
23900.0	438	372	516	0	0	16.64	654.95	70	106	0	2	18.47	566.39
24000.0	464	460	469	0	0	25.35	583.59	80	89	0	5	29.01	800.64
24100.0	347	407	517	0	0	25.24	630.63	74	99	0	5	27.86	730.63
24200.0	539	422	519	0	0	19.39	483.97	70	102	0	4	25.09	619.02
24300.0	568	529	586	0	0	33.78	421.32	77	92	0	3	39.17	458.13
24400.0	501	498	557	0	0	40.99	433.88	97	96	0	2	47.71	416.91
24500.0	463	466	493	0	0	41.48	367.99	89	92	0	1	48.61	460.20
24600.0	439	472	489	0	0	48.04	440.11	86	98	0	0	58.32	511.12
24700.0	634	549	539	0	0	33.60	367.67	100	87	0	0	44.42	418.99
24800.0	492	575	590	0	0	42.97	361.13	75	120	0	0	40.57	545.53
24900.0	602	503	587	0	0	40.82	325.29	105	111	0	1	49.91	416.06
25000.0	640	598	551	0	0	39.04	296.11	101	100	0	0	47.25	402.94
25100.0	472	531	590	0	0	46.33	272.15	98	109	0	0	56.94	354.81
25200.0	474	577	658	0	0	47.46	238.55	110	120	0	0	57.60	345.48
25300.0	602	461	512	0	0	33.01	228.89	69	81	0	0	36.06	241.79
25400.0	726	637	676	0	0	49.92	207.07	98	104	0	0	61.90	294.69
25500.0	645	630	581	0	0	57.55	200.96	98	98	0	0	70.20	252.84
25600.0	813	630	611	0	0	54.65	186.40	123	90	0	0	72.23	143.32
25700.0	635	549	600	0	0	80.12	242.41	102	106	0	0	82.92	230.71
25800.0	814	570	662	0	0	98.43	189.89	85	105	0	0	95.29	246.85
25900.0	760	656	682	0	0	107.71	194.22	117	119	0	0	134.97	247.72
26000.0	840	723	702	0	0	121.30	183.50	128	107	0	0	136.11	202.47
26100.0	861	724	593	0	0	133.46	207.67	104	95	0	0	149.44	254.82
26200.0	976	757	599	0	0	148.34	237.46	133	93	0	0	178.33	319.53
26300.0	827	780	638	0	0	156.80	241.89	126	119	0	0	174.44	280.20
26400.0	842	778	606	0	0	184.72	281.49	138	101	0	0	210.62	248.16
26500.0	842	741	632	0	0	175.82	320.22	153	117	0	0	195.29	385.68
26600.0	734	819	693	0	0	186.05	318.52	135	106	0	0	213.15	399.51
26700.0	984	780	624	0	0	209.25	345.88	130	115	0	0	230.23	389.71
26800.0	840	832	732	0	0	207.42	337.65	175	123	0	0	244.25	427.38
26900.0	1164	767	762	0	0	196.08	354.20	136	124	0	0	236.64	421.00
27000.0	885	777	720	0	0	211.08	366.11	139	139	0	0	245.36	377.70
27100.0	933	758	682	0	0	232.16	413.40	144	131	0	0	260.80	458.17
27200.0	969	714	684	0	0	240.88	451.42	126	117	0	0	265.50	476.47
27300.0	1116	734	688	0	0	259.70	397.77	138	127	0	0	301.87	424.82
27400.0	1048	787	674	0	0	287.35	444.54	142	121	0	0	343.23	556.68
27500.0	1300	697	568	0	0	284.06	461.54	122	111	0	0	327.97	446.79
27600.0	1091	709	590	0	0	317.58	486.05	125	99	0	0	349.63	623.12
27700.0	1117	768	678	0	0	357.59	472.17	133	113	0	0	362.37	524.11
27800.0	963	784	595	0	23	372.05	576.22	113	139	0	0	381.95	654.38
27900.0	1114	687	637	0	22	397.63	511.46	108	104	0	2	401.84	656.28
28000.0	1043	873	720	0	3	408.18	572.10	137	111	0	0	471.76	601.68
28100.0	1128	799	696	0	40	451.82	565.49	136	106	0	2	487.77	589.42
28200.0	979	725	708	0	26	491.10	546.08	124	110	0	2	556.11	681.80
28300.0	1063	820	715	0	76	512.81	670.59	114	107	0	5	558.54	645.38
28400.0	1103	692	692	0	61	552.18	718.67	100	93	0	1	617.18	804.66
28500.0	1002	735	677	0	32	602.09	783.89	120	100	0	1	703.43	811.05
28600.0	819	863	702	0	44	585.26	823.18	131	128	0	4	638.15	798.58
28700.0	974	729	687	0	6	637.47	737.94	142	120	0	0	734.78	786.26
28800.0	838	783	690	0	66	630.18	876.10	133	117	0	2	735.35	982.08
28900.0	830	830	655	0	20	708.16	817.10	132	110	0	0	754.22	921.31
29000.0	858	766	684	0	23	717.54	957.49	151	111	0	3	866.11	969.53
29100.0	762	785	687	0	0	700.37	898.11	151	129	0	1	784.81	1004.78
29200.0	639	776	600	0	72	760.52	939.65	133	90	0	11	927.19	1104.48
29300.0	803	811	655	0	69	774.93	1085.26	146	111	0	6	840.42	1012.67
29400.0	800	715	588	0	126	770.25	1231.62	123	94	0	13	885.23	1165.23
29500.0	718	857	728	0	66	745.95	987.52	145	104	0	6	906.01	1112.16
29600.0	759	809	625	0	111	804.64	1101.50	163	102	0	9	989.24	1236.98
29700.0	586	871	699	0	78	809.63	1005.04	153	110	0	8	981.36	1134.55
29800.0	464	829	654	0	36	932.53	1188.12	148	112	0	9	1023.64	1117.52
29900.0	621	811	753	0	16	915.61	1103.91	144	106	0	8	1055.48	1287.74
30000.0	595	787	679	0	8	883.07	1193.70	141	121	0	4	1036.37	1414.02
30100.0	608	841	550	0	0	905.57	1334.28	139	116	0	3	1071.23	1556.44
30200.0	469	794	740	0	56	955.56	1373.65	152	133	0	14	1100.79	1627.94

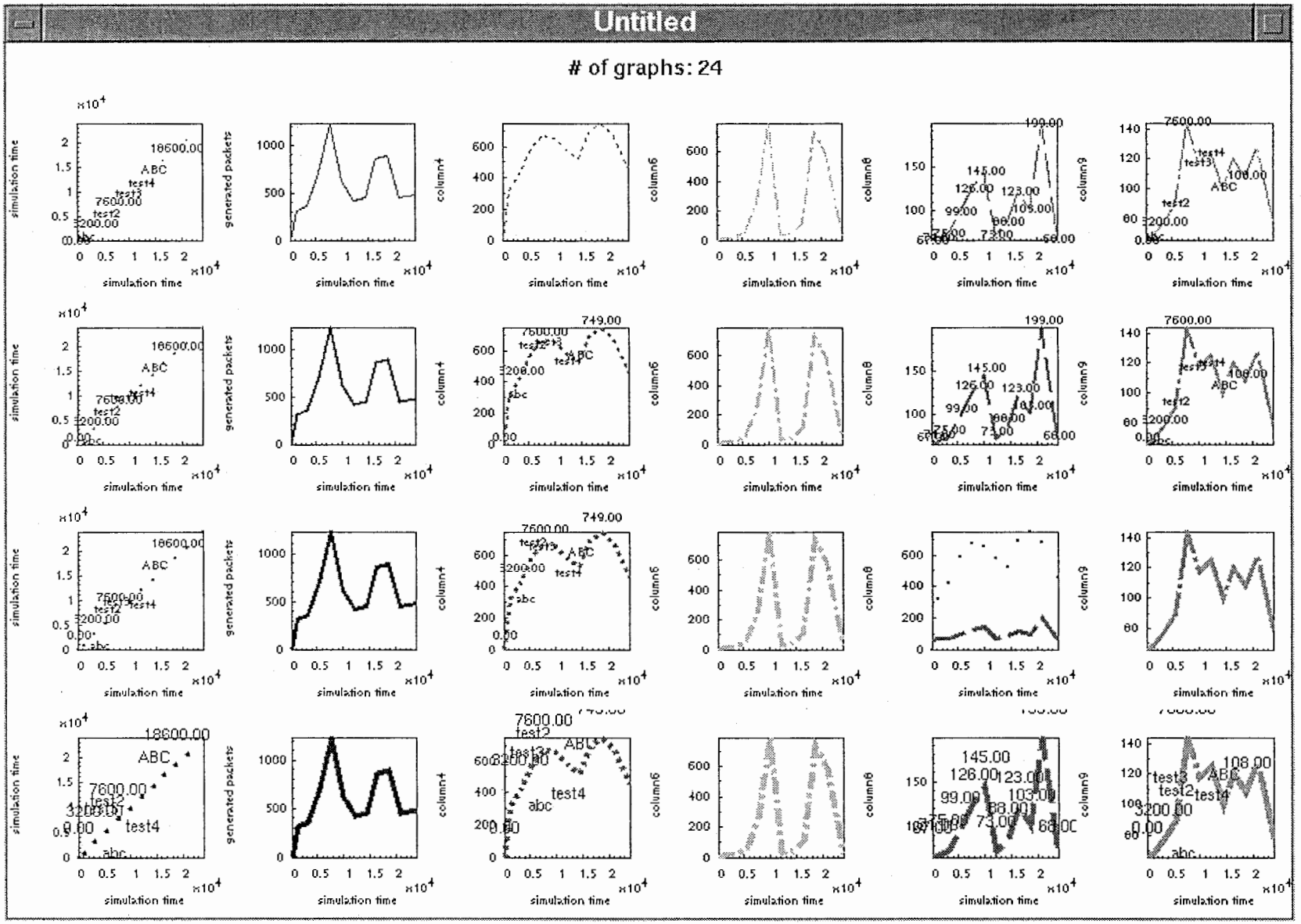
30300.0	588	743	645	0	62	928.98	1437.42	146	112	0	10	1144.61	1414.40
30400.0	472	845	711	0	26	957.92	1295.53	147	111	0	10	1160.80	1338.77
30500.0	490	767	635	0	18	950.20	1621.71	149	88	0	9	1131.40	1622.44
30600.0	419	830	690	0	11	1009.82	1526.06	176	109	0	10	1143.09	1779.00
30700.0	580	775	709	0	0	817.38	1396.80	136	129	0	8	1081.78	1675.56
30800.0	410	793	673	0	1	869.65	1490.72	162	113	0	19	1111.11	1625.67
30900.0	549	801	657	0	43	818.98	1400.29	164	121	0	12	1098.31	1838.20
31000.0	508	793	695	0	0	825.99	1498.91	143	128	0	8	998.46	1783.42
31100.0	525	745	631	0	9	792.34	1422.55	157	105	0	7	960.19	1602.56
31200.0	498	760	717	0	0	717.23	1659.88	139	124	0	12	970.24	1904.19
31300.0	350	811	704	0	0	697.52	1492.10	179	111	0	13	903.59	1794.25
31400.0	451	783	667	0	0	620.61	1605.41	168	120	0	17	798.09	1874.66
31500.0	379	728	599	0	0	530.71	1561.64	120	105	0	14	725.60	1648.51
31600.0	415	732	597	0	0	463.02	1726.80	150	126	0	16	563.37	1804.38
31700.0	492	650	566	0	0	396.24	1615.92	127	103	0	11	503.21	1612.83
31800.0	307	645	613	0	0	306.03	1688.58	130	98	0	3	385.52	1793.35
31900.0	438	599	625	0	0	262.76	1506.22	113	98	0	14	307.21	1717.05
32000.0	371	638	584	0	0	175.27	1473.91	145	93	0	12	219.61	1750.65
32100.0	366	536	620	0	0	77.78	1477.89	88	112	0	13	88.71	1927.95
32200.0	477	509	581	0	0	32.56	1370.22	92	88	0	8	43.88	1869.30
32300.0	276	354	613	0	0	20.54	1519.30	69	110	0	10	21.91	1778.20
32400.0	396	358	636	0	0	17.02	1459.38	76	123	0	9	18.63	1814.22
32500.0	418	343	594	0	0	15.50	1742.01	55	103	0	12	19.40	1935.42
32600.0	409	412	590	0	0	26.90	1592.39	66	121	0	12	31.98	1744.55
32700.0	423	445	596	0	0	31.79	1450.55	78	114	0	8	38.39	1895.17
32800.0	308	387	522	0	0	24.59	1603.30	82	110	0	13	27.46	1802.56
32900.0	373	298	532	0	0	16.98	1466.12	53	109	0	5	17.73	1660.58
33000.0	386	425	585	0	0	34.46	1342.79	60	116	0	5	38.14	1973.88
33100.0	299	301	521	0	0	24.29	1674.79	64	104	0	8	28.70	1896.19
33200.0	474	457	619	0	0	22.58	1117.33	61	104	0	6	26.18	1753.62
33300.0	313	372	611	0	0	23.61	1274.06	65	113	0	4	28.01	1617.12
33400.0	299	276	478	0	0	13.78	1382.56	54	117	0	4	14.28	1501.06
33500.0	451	378	540	0	0	16.76	1379.52	61	91	0	1	19.78	1648.15
33600.0	425	400	599	0	0	24.42	1036.96	68	93	0	5	26.56	1363.97
33700.0	480	481	605	0	0	33.83	943.25	75	97	0	1	35.71	1369.38
33800.0	477	516	624	0	0	29.67	1011.87	97	121	0	3	32.48	1528.42
33900.0	390	371	603	0	0	24.38	1062.29	75	120	0	2	23.58	1367.73
34000.0	375	427	633	0	0	28.15	885.90	79	120	0	3	30.81	1370.77
34100.0	443	350	540	0	0	18.23	824.50	60	105	0	0	22.26	1071.10
34200.0	378	396	532	0	0	30.03	908.51	68	117	0	1	30.96	1216.16
34300.0	435	460	473	0	0	35.55	707.98	71	82	0	1	38.21	1081.70
34400.0	428	445	489	0	0	33.86	802.09	88	108	0	0	38.49	1187.01
34500.0	396	446	527	0	0	27.82	698.24	93	105	0	0	33.81	1154.04
34600.0	438	382	543	0	0	21.17	564.65	66	112	0	0	23.42	905.67
34700.0	513	447	543	0	0	26.84	588.97	79	108	0	0	31.82	776.06
34800.0	534	563	564	0	0	30.72	437.92	93	112	0	0	34.40	758.29
34900.0	497	529	598	0	0	32.67	451.17	109	122	0	0	36.11	520.55
35000.0	503	482	552	0	0	31.12	412.43	83	102	0	0	38.36	508.25
35100.0	532	483	617	0	0	28.46	360.68	98	120	0	0	34.32	433.05
35200.0	668	578	622	0	0	37.33	306.84	111	116	0	0	41.89	381.19
35300.0	607	556	625	0	0	48.03	252.23	95	119	0	0	51.66	322.67
35400.0	584	536	555	0	0	61.86	379.51	97	110	0	0	72.15	519.11
35500.0	593	646	526	0	0	53.56	256.37	112	108	0	0	68.78	330.12
35600.0	705	553	569	0	0	60.98	306.50	95	97	0	0	64.63	233.77
35700.0	694	675	717	0	0	65.80	273.46	123	114	0	0	76.12	396.91
35800.0	785	609	570	0	0	74.49	289.22	113	103	0	0	98.23	382.40
35900.0	589	696	649	0	0	84.74	248.46	130	124	0	0	98.68	295.01
36000.0	836	711	665	0	0	81.91	232.75	146	114	0	0	100.13	210.93
36100.0	1029	637	667	0	0	95.62	265.96	106	119	0	0	118.77	382.56
36200.0	902	571	567	0	0	116.54	272.37	109	109	0	0	142.01	327.10
36300.0	852	649	604	0	0	126.65	300.12	85	121	0	0	132.36	349.05
36400.0	1016	767	657	0	0	168.78	287.86	126	112	0	0	174.14	390.24
36500.0	836	660	638	0	0	188.64	315.20	121	137	0	0	193.69	358.75
36600.0	841	695	649	0	0	203.19	388.06	129	133	0	0	222.67	452.58
36700.0	827	682	617	0	0	205.12	310.60	123	108	0	0	202.40	330.86
36800.0	942	715	713	0	0	233.68	325.25	130	122	0	0	248.47	331.90
36900.0	903	741	665	0	0	260.95	392.21	139	126	0	0	261.74	360.62
37000.0	936	724	721	0	78	262.17	420.32	134	133	0	4	291.09	414.28
37100.0	1073	730	714	0	0	311.43	384.55	130	133	0	0	348.41	404.54
37200.0	1033	680	640	0	0	321.65	398.00	137	126	0	0	327.33	437.09
37300.0	1074	733	672	0	95	351.92	478.25	110	89	0	4	393.67	465.19
37400.0	1048	756	630	0	17	378.02	498.63	131	132	0	0	409.80	460.05
37500.0	1062	707	642	0	24	402.29	492.38	112	112	0	1	436.87	514.83
37600.0	967	678	692	0	22	415.38	493.55	109	123	0	2	459.26	507.98
37700.0	1046	700	568	0	39	421.90	517.63	127	99	0	4	468.70	477.46
37800.0	1088	768	682	0	49	422.55	533.48	129	105	0	4	479.58	667.73
37900.0	1145	756	650	0	20	502.92	564.35	139	123	0	4	578.99	540.57
38000.0	930	812	624	0	67	516.30	602.29	128	112	0	4	628.27	664.88
38100.0	967	923	674	0	74	519.69	635.55	161	119	0	5	585.28	586.53
38200.0	1083	723	721	0	43	565.08	723.43	123	133	0	4	575.98	711.93
38300.0	916	901	754	0	26	573.32	709.58	156	127	0	2	670.11	681.04
38400.0	825	760	756	0	27	567.05	658.84	138	102	0	3	688.43	728.22
38500.0	878	708	664	0	127	621.85	703.44	141	126	0	6	694.72	671.17

38600.0	726	799	677	0	66	671.18	797.29	131	125	0	5	677.10	747.33
38700.0	749	760	652	0	74	715.50	854.16	150	96	0	7	847.99	945.49
38800.0	965	787	596	0	80	707.39	857.36	142	100	0	5	771.83	1031.71
38900.0	742	762	735	0	79	720.24	887.21	138	131	0	7	819.88	895.91
39000.0	770	927	745	0	45	749.22	973.21	170	117	0	4	873.51	889.67
39100.0	855	784	699	0	3	716.10	969.77	150	120	0	2	848.38	995.63
39200.0	704	782	658	0	43	767.27	879.79	132	98	0	7	875.32	1033.95
39300.0	701	796	647	0	99	813.06	1002.11	151	121	0	9	925.96	970.11
39400.0	663	800	625	0	103	835.93	1093.49	151	136	0	13	915.05	1202.89
39500.0	593	762	645	0	132	807.19	1106.49	124	112	0	15	903.40	1091.16
39600.0	615	752	779	0	72	837.44	1132.06	141	137	0	9	1000.35	1174.10
39700.0	589	947	709	0	15	806.90	1073.41	192	122	0	17	932.24	1065.13
39800.0	526	799	713	0	29	863.59	1194.29	152	113	0	13	1067.89	1194.18
39900.0	557	775	638	0	64	826.06	1120.90	157	111	0	13	996.20	1268.30
40000.0	533	875	659	0	36	789.45	1331.77	152	120	0	22	1051.50	1310.94
40100.0	558	895	610	0	63	828.54	1177.07	163	105	0	16	1001.51	1271.69
40200.0	521	750	566	0	48	880.40	1314.12	155	98	0	15	1081.17	1375.93
40300.0	503	829	689	0	14	834.14	1191.73	163	118	0	15	998.33	1281.28
40400.0	474	854	600	0	41	777.41	1375.28	188	115	0	14	999.31	1373.82
40500.0	482	855	621	0	15	757.13	1223.61	157	98	0	11	905.24	1178.36
40600.0	498	866	721	0	0	762.08	1158.10	162	105	0	16	924.06	1442.48
40700.0	485	823	648	0	1	663.50	1303.51	157	112	0	11	821.02	1350.21
40800.0	453	829	687	0	0	585.95	1270.46	199	129	0	15	746.20	1349.17
40900.0	407	602	598	0	0	712.09	1461.73	137	109	0	16	882.00	1443.66
41000.0	395	715	705	0	0	593.44	1401.22	135	123	0	18	809.51	1461.76
41100.0	464	694	610	0	0	535.14	1243.72	135	96	0	18	718.43	1415.17
41200.0	297	666	687	0	0	416.61	1316.68	150	111	0	13	592.96	1592.22
41300.0	411	668	646	0	0	386.33	1484.77	128	109	0	31	484.45	1504.77
41400.0	451	677	580	0	0	255.25	1390.25	131	96	0	11	287.26	1471.45
41500.0	291	634	625	0	0	158.63	1314.85	131	118	0	11	202.24	1604.35
41600.0	457	523	505	0	0	90.07	1236.45	114	92	0	13	147.69	1466.14
41700.0	379	474	574	0	0	36.92	1220.61	93	123	0	8	50.26	1410.92
41800.0	348	357	631	0	0	20.84	1339.73	68	116	0	13	22.03	1542.15
41900.0	370	424	541	0	0	22.76	1328.74	73	88	0	13	25.08	1394.72
42000.0	365	312	664	0	0	12.99	1311.95	60	103	0	12	16.26	1257.21
42100.0	336	373	487	0	0	19.81	1539.90	77	89	0	10	22.44	1605.96
42200.0	369	350	507	0	0	14.92	1384.72	72	108	0	12	17.27	1360.52
42300.0	350	305	590	0	0	17.47	1290.03	49	105	0	10	17.05	1659.58
42400.0	373	417	520	0	0	23.52	1219.87	85	102	0	11	25.15	1404.59
42500.0	436	363	599	0	0	22.02	1058.45	62	109	0	5	21.16	1477.30
42600.0	363	437	571	0	0	28.51	1090.66	80	101	0	5	31.43	1367.99
42700.0	298	304	591	0	0	26.85	1318.56	60	113	0	6	31.32	1573.03
42800.0	419	409	515	0	0	23.45	1087.52	72	105	0	9	27.15	1292.50
42900.0	416	356	542	0	0	22.90	1087.36	66	123	0	9	27.61	1399.37
43000.0	375	457	645	0	0	18.28	797.81	81	115	0	2	23.00	1142.21
43100.0	343	343	515	0	0	16.37	1014.50	74	121	0	3	17.13	1391.37
43200.0	350	337	474	0	0	16.27	932.86	58	97	0	1	18.79	1221.94
43300.0	347	330	501	0	0	13.49	855.02	80	113	0	6	14.62	1126.80
43400.0	486	350	433	0	0	21.34	690.28	52	90	0	1	22.36	929.67
43500.0	393	564	638	0	0	31.58	588.25	91	109	0	0	37.06	996.79
43600.0	441	354	530	0	0	18.46	674.33	69	125	0	1	19.80	833.87
43700.0	393	422	532	0	0	26.44	436.66	82	95	0	0	28.53	699.46
43800.0	409	391	492	0	0	25.04	525.70	68	109	0	2	31.77	747.92
43900.0	480	443	465	0	0	25.62	377.39	68	79	0	0	26.58	616.31
44000.0	408	496	548	0	0	29.51	276.93	87	110	0	0	33.18	394.45
44100.0	329	319	447	0	0	20.65	356.73	68	78	0	0	24.53	372.44
44200.0	418	412	447	0	0	27.97	375.44	68	84	0	0	32.33	541.69
44300.0	404	413	534	0	0	19.21	298.53	76	104	0	0	23.06	410.51
44400.0	371	366	491	0	0	15.34	285.72	76	108	0	0	18.10	497.29
44500.0	343	382	476	0	0	19.57	279.87	77	109	0	0	20.53	385.11
44600.0	482	410	549	0	0	18.13	256.40	77	100	0	0	19.20	307.32
44700.0	555	512	492	0	0	28.44	168.91	79	81	0	0	30.95	196.71
44800.0	527	476	548	0	0	34.02	200.36	89	103	0	0	42.68	311.65
44900.0	634	570	629	0	0	34.83	127.73	104	112	0	0	42.62	172.64
45000.0	555	539	514	0	0	47.26	125.05	88	88	0	0	53.04	193.74
45100.0	471	517	508	0	0	53.01	116.67	104	105	0	0	61.11	140.84
45200.0	726	518	499	0	0	43.28	112.99	93	92	0	0	49.45	128.98
45300.0	639	640	567	0	0	63.64	117.99	100	90	0	0	77.48	150.73
45400.0	677	669	622	0	0	67.14	133.68	113	98	0	0	81.96	145.23
45500.0	650	597	618	0	0	69.28	154.45	109	114	0	0	80.97	170.76
45600.0	554	579	532	0	0	84.37	157.52	99	93	0	0	89.75	195.83
45700.0	540	631	602	0	0	88.42	183.43	141	111	0	0	97.60	201.73
45800.0	891	621	690	0	0	64.51	151.18	101	116	0	0	90.41	170.10
45900.0	708	690	667	0	0	76.36	149.87	124	128	0	0	91.23	180.43
46000.0	773	667	596	0	0	84.83	154.56	107	108	0	0	97.21	203.81
46100.0	776	661	652	0	0	107.48	156.23	113	109	0	0	114.15	180.91
46200.0	887	706	694	0	0	107.67	184.13	134	128	0	0	126.61	201.46
46300.0	771	745	724	0	0	115.43	185.23	126	109	0	0	147.45	240.26
46400.0	1175	783	652	0	0	141.75	202.23	127	111	0	0	175.01	211.97
46500.0	1001	744	703	0	0	132.71	194.87	117	113	0	0	154.20	249.30
46600.0	1017	701	572	0	0	162.11	236.76	132	84	0	0	196.96	265.61
46700.0	994	691	682	0	0	173.11	233.27	96	108	0	0	190.10	276.90
46800.0	973	712	655	0	0	204.69	255.18	105	103	0	0	218.98	283.28

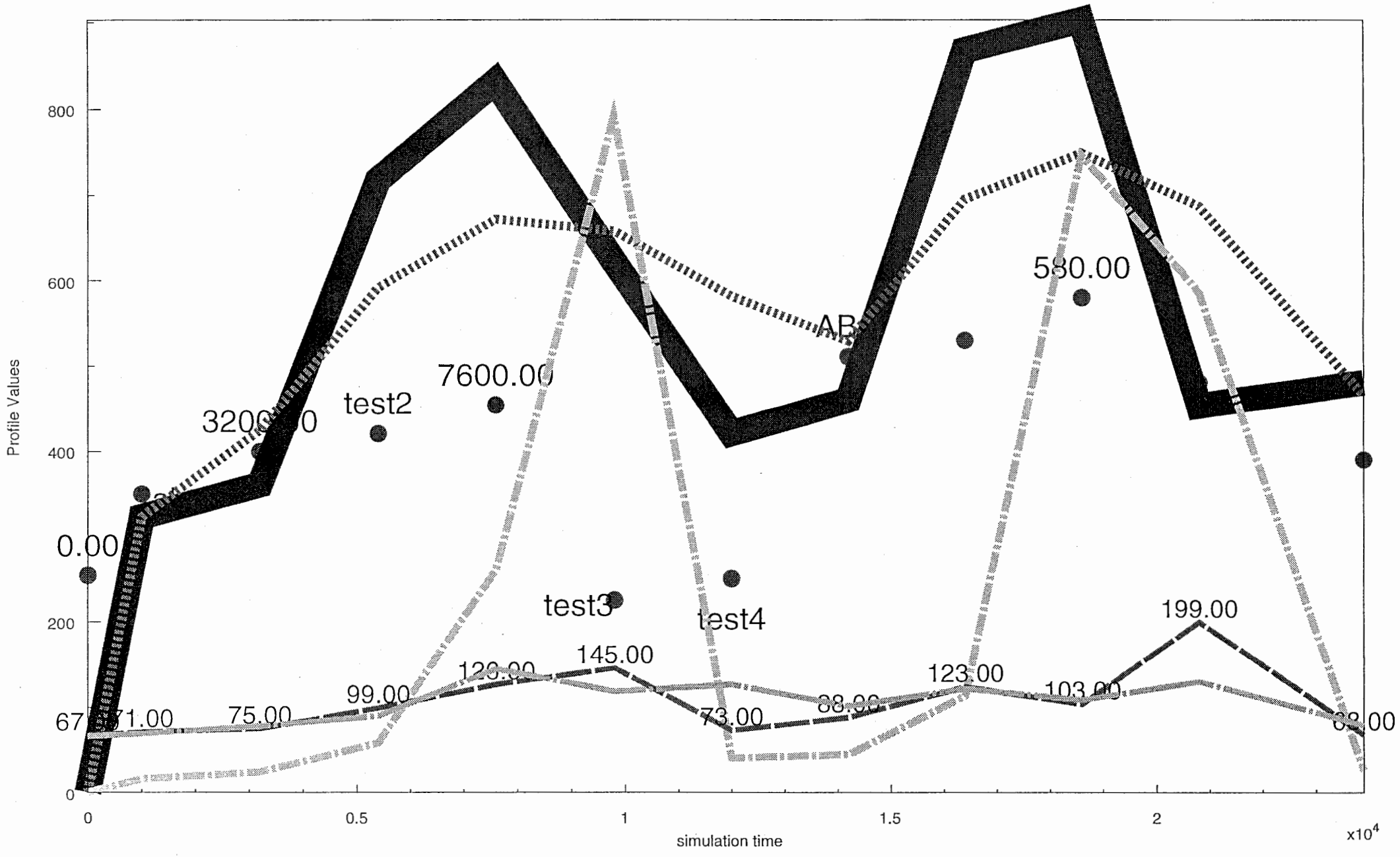
46900.0	1106	721	698	0	0	234.48	278.62	107	112	0	0	272.88	310.62
47000.0	990	698	688	0	0	260.85	314.06	117	122	0	0	293.14	311.50
47100.0	1135	739	619	0	0	284.43	284.83	121	106	0	0	303.31	329.38
47200.0	1173	687	644	0	0	327.70	358.09	104	106	0	0	371.47	379.12
47300.0	1171	713	604	0	0	374.12	419.57	129	107	0	0	404.26	433.81
47400.0	1161	738	659	0	0	382.46	432.38	105	92	0	0	420.28	456.23
47500.0	1176	782	776	0	0	400.51	407.42	146	124	0	0	477.46	441.02
47600.0	1245	725	674	0	0	450.77	460.44	134	89	0	0	511.07	484.95
47700.0	1179	766	604	0	0	477.21	489.52	138	113	0	0	543.12	532.51
47800.0	1231	689	717	0	0	481.06	495.15	119	94	0	0	542.67	529.04
47900.0	1295	719	631	0	31	525.22	553.67	98	100	0	0	613.81	633.34
48000.0	1137	767	721	0	36	499.07	590.78	113	119	0	1	578.63	566.89
48100.0	1054	707	638	0	63	597.52	625.78	105	124	0	2	614.77	697.95
48200.0	954	742	756	0	52	606.98	617.53	112	122	0	1	606.50	683.51
48300.0	855	765	678	0	43	622.45	703.47	123	114	0	2	676.36	753.79
48400.0	925	777	693	0	114	670.61	735.52	124	110	0	3	784.07	748.30
48500.0	897	767	701	0	64	728.72	740.79	130	114	0	2	785.05	735.54
48600.0	886	804	666	0	81	742.10	757.63	136	138	0	4	854.13	819.04
48700.0	952	737	707	0	1	787.52	863.93	144	127	0	0	857.28	862.51
48800.0	759	743	646	0	54	821.52	860.23	129	110	0	4	861.26	809.31
48900.0	770	727	656	0	23	820.96	973.01	118	112	0	2	947.39	931.80
49000.0	934	812	709	0	62	904.06	940.16	134	126	0	5	955.35	1041.07
49100.0	835	746	741	0	143	880.59	914.02	141	117	0	11	981.79	961.03
49200.0	691	801	689	0	87	920.41	1003.03	126	105	0	7	1057.10	1064.49
49300.0	753	768	656	0	109	974.12	1068.71	123	99	0	5	1145.66	999.28
49400.0	738	731	687	0	133	1031.13	1087.27	119	109	0	9	1059.22	1204.85
49500.0	766	770	695	0	140	962.03	1056.64	111	117	0	11	1071.82	1230.94
49600.0	540	829	707	0	48	1054.33	1076.71	130	111	0	6	1187.34	1048.04
49700.0	616	924	689	0	44	992.71	1098.58	141	115	0	9	1177.82	1242.04
49800.0	532	823	612	0	45	1070.58	1269.38	156	102	0	5	1249.25	1369.54
49900.0	644	896	571	0	125	1050.82	1256.83	162	100	0	7	1202.33	1356.84
50000.0	468	782	610	0	73	1167.65	1425.50	146	100	0	12	1281.76	1359.53

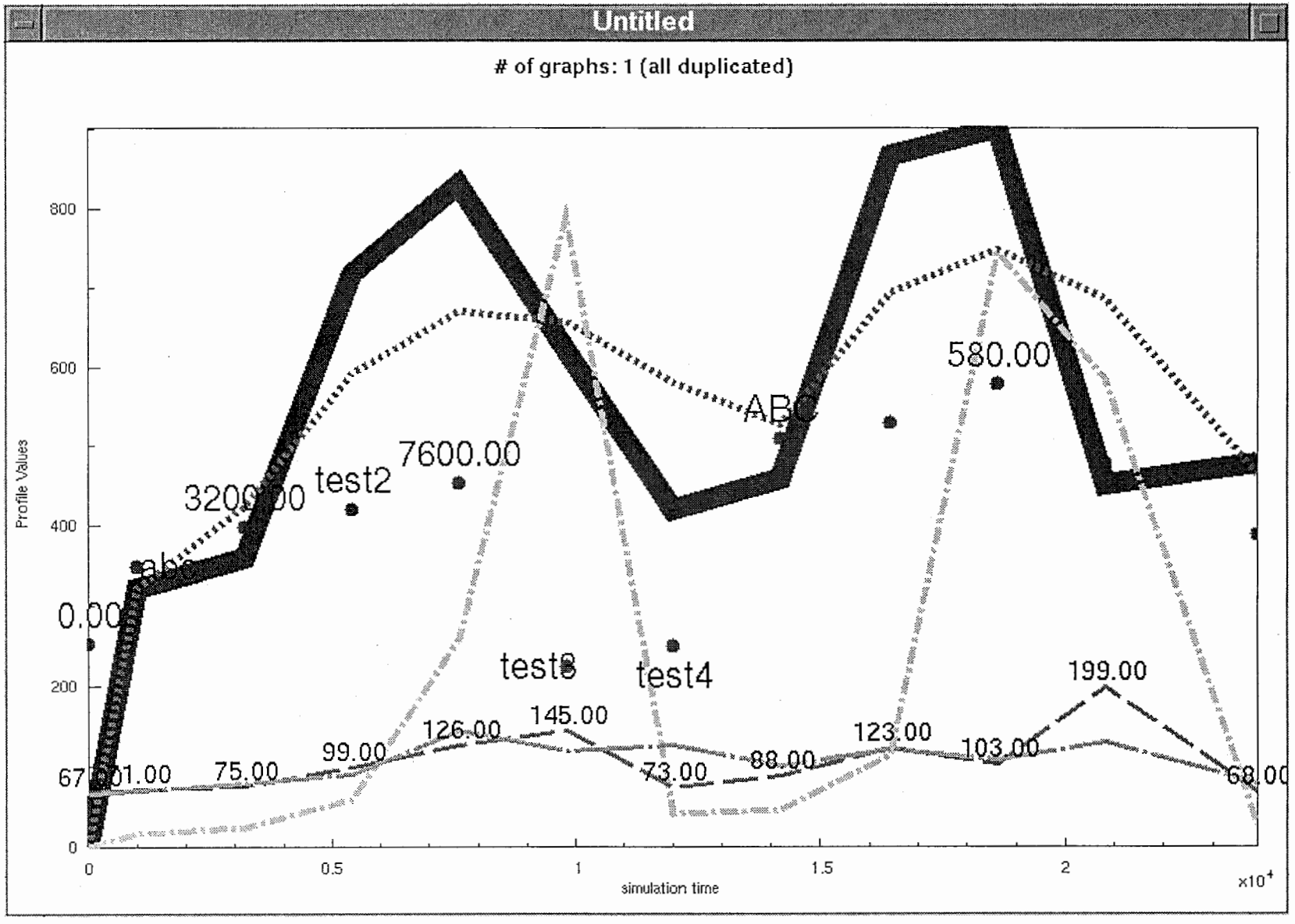
of graphs: 24


simulation time
generated packets
column4
column6
column8
column9
x10^4
18600.00
ABC
test4
test3
7600.00
test2
3200.00
abc
0.00
simulation time


Untitled
# of graphs: 24
simulation time
generated packets
column4
column6
column8
column9
18600.00
ABC
7600.00
test4
test3
test2
3200.00
0.00
abc
749.00
199.00
145.00
126.00
123.00
108.00
99.00
88.00
73.00
68.00
75.00

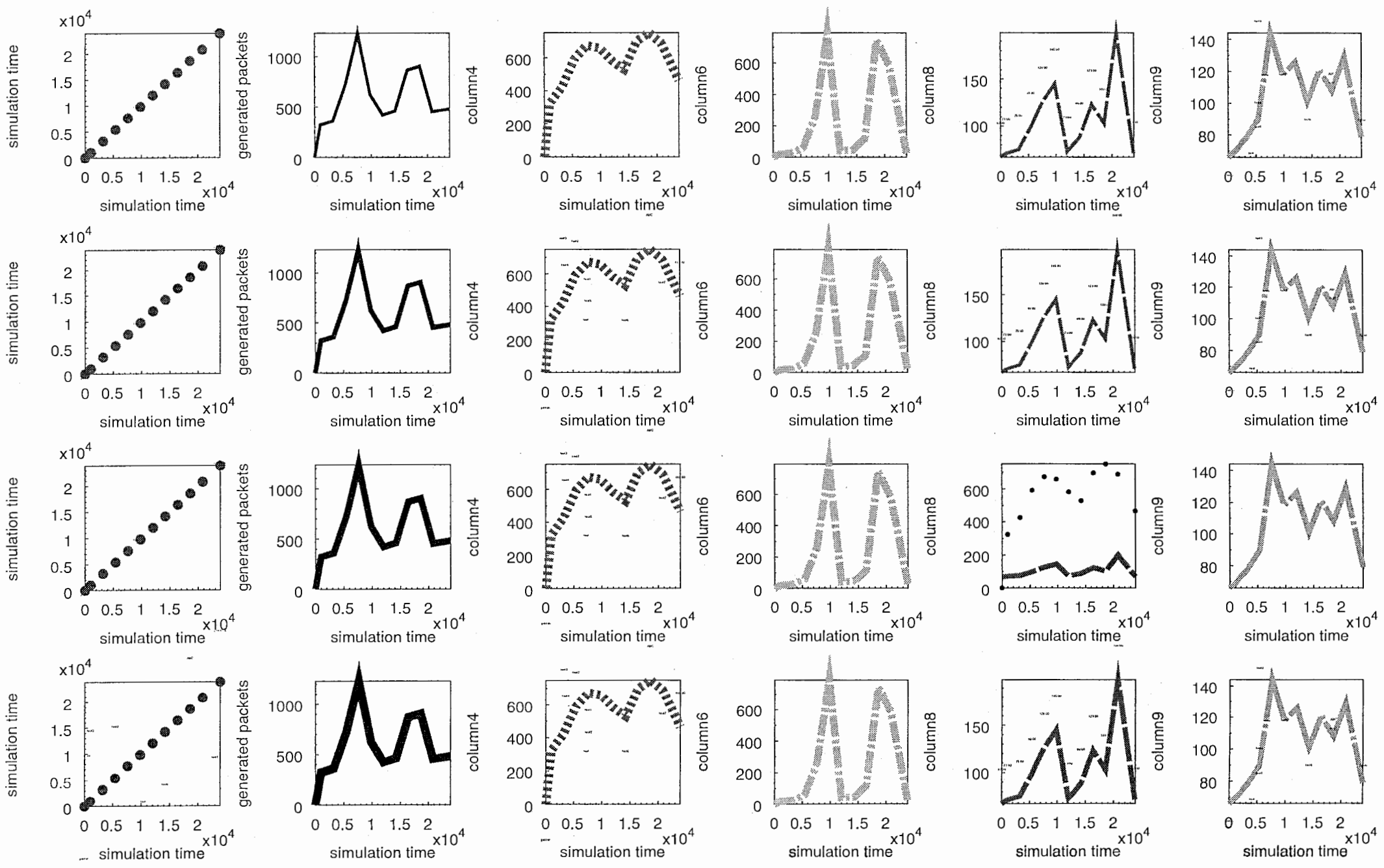
```
# of graphs: 24
25 12   11
1 1  0  dotcolor=#105523 dotsize=1 label=11 labelsize=6 labeldistance=0
1 2  1  dotcolor=2 dotsize=1 label=0 labelsize=6 labeldistance=0
1 4  2  dotcolor=8 dotsize=1 labeldistance=0
1 6  3  dotcolor=4 dotsize=1 label=0 labelsize=6 labeldistance=0
1 8  4  dotcolor=1 dotsize=1 label=8 labelsize=6 labeldistance=0
1 9  5  dotcolor=9 dotsize=1 label=11 labelsize=6 labeldistance=0
1 1  0  dotcolor=#105523 dotsize=2 label=11 labelsize=7 labeldistance=5
1 2  1  dotcolor=2 dotsize=2 label=0 labelsize=7 labeldistance=5
1 4  2  dotcolor=8 dotsize=2 label=11 labelsize=7 labeldistance=5
1 6  3  dotcolor=4 dotsize=2 label=0 labelsize=7 labeldistance=5
1 8  4  dotcolor=1 dotsize=2 label=8 labelsize=7 labeldistance=5
1 9  5  dotcolor=9 dotsize=2 label=11 labelsize=7 labeldistance=5
1 1  0  dotcolor=#105523 dotsize=3 label=11 labelsize=8 labeldistance=10
1 2  1  dotcolor=2 dotsize=3 label=0 labelsize=8 labeldistance=10
1 4  2  dotcolor=8 dotsize=3 label=11 labelsize=8 labeldistance=10
1 6  3  dotcolor=4 dotsize=3 label=0 labelsize=8 labeldistance=10
1 8  4  dotcolor=1 dotsize=3
1 4  0  duplicate=1 dotcolor=2 dotsize=3
1 9  5  dotcolor=9 dotsize=3
1 1  0  dotcolor=#105523 dotsize=4 label=11 labelsize=9 labeldistance=20
1 2  1  dotcolor=2 dotsize=4 label=0 labelsize=9 labeldistance=20
1 4  2  dotcolor=8 dotsize=4 label=11 labelsize=9 labeldistance=20
1 6  3  dotcolor=4 dotsize=4 label=0 labelsize=9 labeldistance=20
1 8  4  dotcolor=1 dotsize=4 label=8 labelsize=9 labeldistance=20
1 9  5  dotcolor=9 dotsize=4 label=11 labelsize=9 labeldistance=20
simulation time
generated packets
@label
column4
@label
column6
@label
column8
column9
@label
@label
     0.0     0     0     0     0     0.0     0.0     67    66     0.0   @x
  1000.0   323   333   323     0   16.08   16.25     71    69   17.43   @000abc
  3200.0   361   411   425     0   24.38   26.65     75    78   27.37   @x
  5400.0   718   635   592     0   58.53   66.17     99    90   68.14   @090test2
  7600.0  1232   643   672     0  259.54  342.50    126   144  411.55   @x
  9800.0   622   734   658     0  792.24 1076.56    145   118 1179.80   @180test3
 12000.0   421   444   582     0   40.65 1559.91     73   126 1751.07   @270test4
 14200.0   460   570   529     0   44.94  435.38     88   101  531.37   ABC
 16400.0   868   699   695     0  112.98  225.84    123   121  222.06   @0
 18600.0   903   733   749    88  746.69  787.94    103   108  863.24   @y
 20800.0   453   829   687     0  585.95 1270.46    199   129 1349.17   @0
 23900.0   480   443   465     0   25.62  377.39     68    79  616.31   @0
```

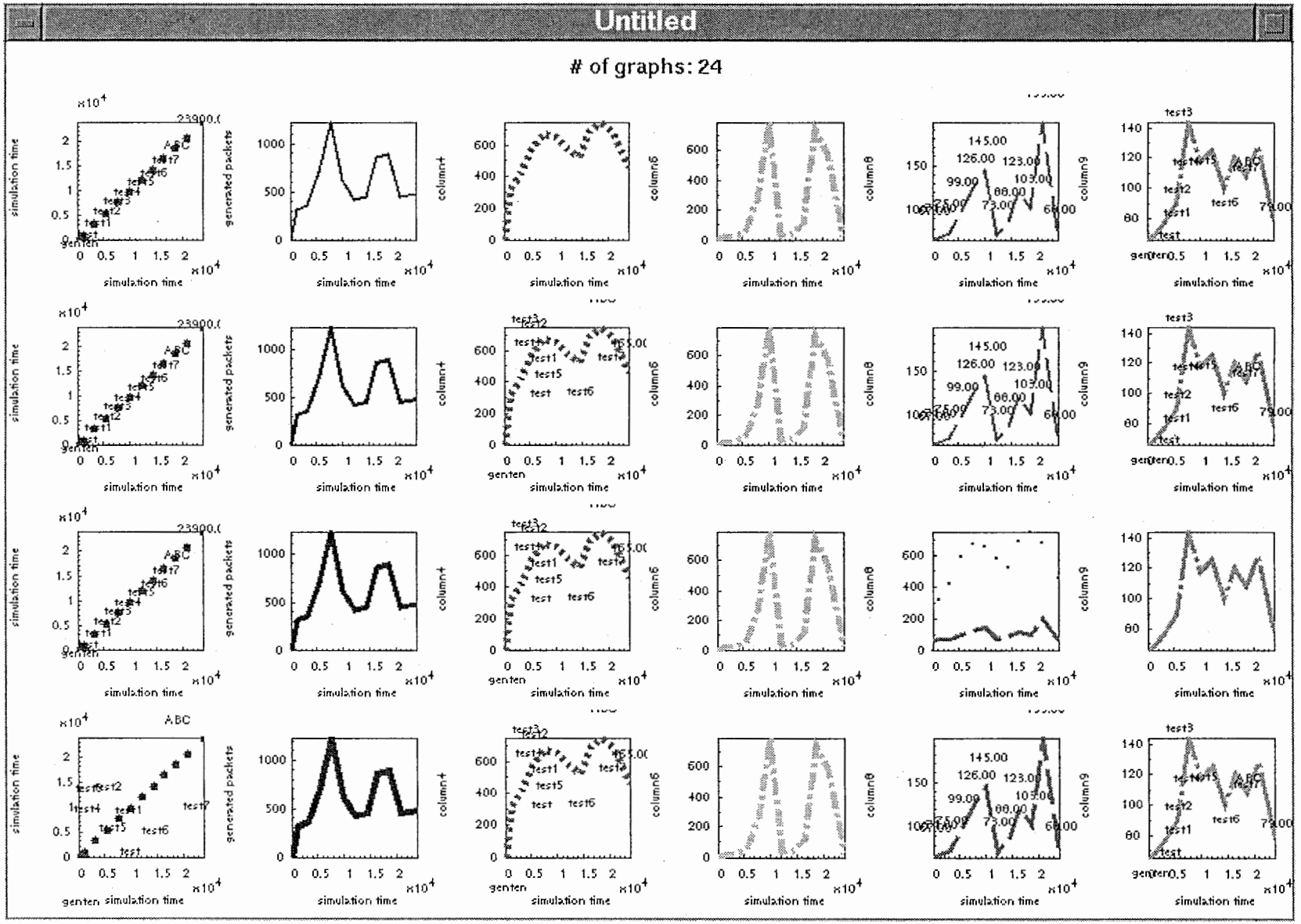

# of graphs: 1 (all duplicated)
Profile Values
simulation time
x10^4
0
200
400
600
800
0
0.5
1
1.5
2
0.00
test2
7600.00
test3
test4
580.00
AB
199.00
145.00
123.00
103.00
99.00
75.00
73.00


Untitled
# of graphs: 1 (all duplicated)
Profile Values
simulation time
x10^4
800
600
400
200
0
0
0.5
1
1.5
2
0.00
7600.00
test2
test3
test4
ABC
580.00
199.00
145.00
126.00
99.00
75.00
123.00
103.00
88.00
73.00
68.00

```
# of graphs: 1 (all duplicated)
6 12   12
1 12  0  dotcolor=#105523 dotsize=10 label=11 labelsize=20 labeldistance=20
1 2  1  duplicate=1 dotcolor=2 dotsize=15 label=0 labelsize=16 labeldistance=10
1 4  2  duplicate=1 dotcolor=8 dotsize=5 labeldistance=20
1 6  3  duplicate=1 dotcolor=4 dotsize=4 label=0 labelsize=16 labeldistance=10
1 8  4  duplicate=1 dotcolor=1 dotsize=3 label=8 labelsize=16 labeldistance=10
1 9  5  duplicate=1 dotcolor=9 dotsize=3 label=0 labelsize=20 labeldistance=10
simulation time
generated packets
@label
column4
@label
column6
@label
column8
column9
@label
@label
Profile Values
     0.0     0     0     0     0     0.0     0.0    67    66     0.0   @x             254
  1000.0   323   333   323     0   16.08   16.25    71    69   17.43   @000abc        350
  3200.0   361   411   425     0   24.38   26.65    75    78   27.37   @x             400
  5400.0   718   635   592     0   58.53   66.17    99    90   68.14   @090test2      421
  7600.0   832   643   672     0  259.54  342.50   126   144  411.55   @x             454
  9800.0   622   734   658     0  792.24 1076.56   145   118  879.80   @180test3      225
 12000.0   421   444   582     0   40.65  859.91    73   126  851.07   @270test4      250
 14200.0   460   570   529     0   44.94  435.38    88   101  531.37   ABC            511
 16400.0   868   699   695     0  112.98  225.84   123   121  222.06   @0             530
 18600.0   903   733   749    88  746.69  787.94   103   108  863.24   @y             580
 20800.0   453   829   687     0  585.95  970.46   199   129  849.17   @0             480
 23900.0   480   443   465     0   25.62  377.39    68    79  616.31   @0             390
```

of graphs: 24


simulation time
generated packets
column4
column6
column8
column9
x10^4
0 0.5 1 1.5 2
0 500 1000
0 200 400 600
0 100 150
80 100 120 140


Untitled
# of graphs: 24
simulation time
generated packets
column4
column6
column8
column9
genten
test
test1
test2
test3
test4
test5
test6
test7
ABC
145.00
126.00
123.00
105.00
99.00
88.00
73.00

```
# of graphs: 24
25 12   11
1 1  0  dotcolor=#105523 dotsize=6 label=11 labelsize=2 labeldistance=2
1 2  1  dotcolor=2 dotsize=2 label=0 labelsize=2 labeldistance=20
1 4  2  dotcolor=8 dotsize=5 labeldistance=20
1 6  3  dotcolor=4 dotsize=4 label=0 labelsize=2 labeldistance=20
1 8  4  dotcolor=1 dotsize=2 label=8 labelsize=2 labeldistance=20
1 9  5  dotcolor=9 dotsize=3 label=11 labelsize=2 labeldistance=10
1 1  0  dotcolor=#105523 dotsize=6 label=11 labelsize=2 labeldistance=2
1 2  1  dotcolor=2 dotsize=3 label=0 labelsize=2 labeldistance=20
1 4  2  dotcolor=8 dotsize=5 label=11 labelsize=2 labeldistance=20
1 6  3  dotcolor=4 dotsize=4 label=0 labelsize=2 labeldistance=20
1 8  4  dotcolor=1 dotsize=2 label=8 labelsize=2 labeldistance=20
1 9  5  dotcolor=9 dotsize=3 label=11 labelsize=2 labeldistance=10
1 1  0  dotcolor=#105523 dotsize=6 label=11 labelsize=2 labeldistance=2
1 2  1  dotcolor=2 dotsize=4 label=0 labelsize=2 labeldistance=20
1 4  2  dotcolor=8 dotsize=5 label=11 labelsize=2 labeldistance=20
1 6  3  dotcolor=4 dotsize=4 label=0 labelsize=2 labeldistance=20
1 8  4  dotcolor=1 dotsize=3
1 4  0  duplicate=1 dotcolor=2 dotsize=3
1 9  5  dotcolor=9 dotsize=3
1 1  0  dotcolor=#105523 dotsize=6 label=11 labelsize=2 labeldistance=30
1 2  1  dotcolor=2 dotsize=5 label=0 labelsize=2 labeldistance=20
1 4  2  dotcolor=8 dotsize=5 label=11 labelsize=2 labeldistance=20
1 6  3  dotcolor=4 dotsize=4 label=0 labelsize=2 labeldistance=20
1 8  4  dotcolor=1 dotsize=3 label=8 labelsize=2 labeldistance=20
1 9  5  dotcolor=9 dotsize=3 label=11 labelsize=2 labeldistance=10
simulation time
generated packets
@label
column4
@label
column6
@label
column8
column9
@label
@label
```

0.0	0	0	0	0	0.0	0.0	67	66	0.0	@270genten
1000.0	323	333	323	0	16.08	16.25	71	69	17.43	@000test
3200.0	361	411	425	0	24.38	26.65	75	78	27.37	@045test1
5400.0	718	635	592	0	58.53	66.17	99	90	68.14	@090test2
7600.0	1232	643	672	0	259.54	342.50	126	144	411.55	@135test3
9800.0	622	734	658	0	792.24	1076.56	145	118	1179.80	@180test4
12000.0	421	444	582	0	40.65	1559.91	73	126	1751.07	@225test5
14200.0	460	570	529	0	44.94	435.38	88	101	531.37	@270test6
16400.0	868	699	695	0	112.98	225.84	123	121	222.06	@315test7
18600.0	903	733	749	88	746.69	787.94	103	108	863.24	ABC
20800.0	453	829	687	0	585.95	1270.46	199	129	1349.17	@0
23900.0	480	443	465	0	25.62	377.39	68	79	616.31	@y

A2 プログラムソースリスト

```
/*
 * ACR PLOT TOOL Version 1.0
 *    (c) Copyright 1997
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * plot.c - プロットコマンド
 */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>

#include "plot.h"
#include "graph.h"

void main(int argc, char **argv)
{
  char filename[BUFSIZ+1];
  char *psfilename;             /* PSファイル名 */
  int  paperOrientation;        /* 紙の方向 */
  float aspectRatio;            /* 縦横比 */
  float dotSize;                /* 点の大きさ */
  int i;

  Display *display;
  Window mainWin;
  int screen;
  XSizeHints wmHint;
  int isDone;
  XEvent event;
  KeySym keySym;
  int key;
  char keyBuf[10];

  GC gc;

  DATA *data;

  float dotratio;
  char  headerfname[BUFSIZ+1];          /* ヘッダーファイル名 */
  char  bodyfname[BUFSIZ+1];            /* ヘッダーファイル名 */
  int   mode;                           /* POSTSCRIPT出力モード */

  /* コマンドライン引数の取得 */
  strcpy(filename, DEFAULT_FILENAME);
  psfilename = NULL;
  paperOrientation = DEFAULT_PAPER_ORIENTATION;
  aspectRatio = -1;
  dotSize = 1;
  dotratio = 1;
  strcpy( headerfname, "" );
  strcpy( bodyfname, "" );

/*fprintf(stderr, "%s", filename);*/
  for (i = 1;i < argc;i++) {
    if (!strcmp(argv[i], "-data")) {
      strcpy(filename, argv[++i]);
    }
    else if (!strcmp(argv[i], "-ps")) {
      mode = PS_ONLY_MODE;
      psfilename = argv[++i];
    }
    else if (!strcmp(argv[i], "-eps")) {
      mode = EPS_ONLY_MODE;
      psfilename = argv[++i];
    }
    else if (!strcmp(argv[i], "-landscape")) {
      paperOrientation = PAPER_LANDSCAPE;
    }
    else if (!strcmp(argv[i], "-portrait")) {
      paperOrientation = PAPER_PORTRAIT;
    }
    else if (!strcmp(argv[i], "-aspectratio")) {
      sscanf(argv[++i], "%g", &aspectRatio);
    }
    else if (!strcmp(argv[i], "-dotratio")){
      sscanf(argv[++i], "%g", &dotratio);
    }
    else if (!strcmp(argv[i], "-head")){
      strcpy(headerfname, argv[++i]);
    }
    else if (!strcmp(argv[i], "-body")){
      strcpy(bodyfname, argv[++i]);
    }
    else if (!strcmp(argv[i], "-help")) {
      usage();
      exit(0);
    }
    else {
      fprintf(stderr, "Unknown option '%s'.\n\n", argv[i]);
      usage();
      exit(1);
    }
  }

  /* データの読み込み */
  if ((data = ReadDataFile(filename, headerfname, bodyfname)) == NULL)
    exit(1);

  data->aspectRatio = aspectRatio;
  data->orientation = paperOrientation;
  data->dotratio = dotratio;
  if (psfilename)
    data->mode = mode;

  /* ディスプレイ接続 */
  if (data->mode == X_DRAW_MODE) {
    display = XOpenDisplay("");
    screen = DefaultScreen(display);
    data->display = display;
  }

  /* ウィンドウの生成 */
  CreatePlotWindows(data);

  if (data->mode == PS_ONLY_MODE || data->mode == EPS_ONLY_MODE) {
    SetPlotWindowSize(data);
    DrawGraph(data);
    PrintGraph(psfilename, data);
    exit(0);
  }

  isDone = 0;
  while (isDone == 0) {
    XNextEvent(display, &event);
    switch (event.type) {
    case Expose:
      if (event.xexpose.count == 0) {
#ifdef DEBUG
        fprintf(stderr, "OK Expose\n");
#endif
        RedrawGraph(data, event.xexpose.window);
      }
      break;

    case KeyPress:
```

```
      key = XLookupString((XKeyEvent*)&event,
                          keyBuf, 10, (KeySym*)&key, NULL);
      if (key == 1 && keyBuf[0] == 'q') {
        isDone = 1;
      }
      else if (key == 1 && keyBuf[0] == 'p') {
        if (data->mode == PS_ONLY_MODE)
          PrintGraph("tmp.ps", data);
        if (data->mode == EPS_ONLY_MODE)
          PrintGraph("tmp.eps", data);
      }
      break;

    case ConfigureNotify:
      if (event.xconfigure.window == data->mainWinId) {
#ifdef DEBUG
        fprintf(stderr, "OK ConfigureNotify(main)\n");
#endif
        /* ウィンドウの大きさの再設定 */
        data->mainWinWidth = event.xconfigure.width;
        data->mainWinHeight = event.xconfigure.height;
        SetPlotWindowSize(data);
        DrawGraph(data);
        if (psfilename) {
          PrintGraph(psfilename, data);
          isDone = 1;
        }
      }

      break;

    case DestroyNotify:
#ifdef DEBUG
      printf("OK Destroy\n");
#endif
      XDestroyWindow(data->display, data->mainWinId);
      XCloseDisplay(data->display);
      break;
    }
  }

  XDestroyWindow(data->display, data->mainWinId);
  XCloseDisplay(data->display);

  exit(0);
}

int usage()
{
  fprintf(stderr, "\nUsage: plot [options...]\n");
  fprintf(stderr, "  -data   <file>  data filename. default is 'fort.99'.\n");
  fprintf(stderr, "  -head   <file>  header filename. \n");
  fprintf(stderr, "  -body   <file>  body filename. setup with -header options.\n");
  fprintf(stderr, "  -ps     <file>  output PS filename.\n");
  fprintf(stderr, "  -eps    <file>  output EPS filename.\n");
  fprintf(stderr, "  -landscape    landscape.\n");
  fprintf(stderr, "  -portrait     portrait.\n");
  fprintf(stderr, "  -aspectratio  aspect ratio.\n");
  fprintf(stderr, "  -dotratio  dot size ratio. default is 1.\n" );
}
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *               Version 2.0
 *    (c) Copyright 1997,1998
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * draw.c - 描画関数
 */

#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "graph.h"
#include "plot.h"

/*#define DEBUG*/
/* グラフハンドル配列 */
int gGraphNo[MAX_GRAPH_COUNT];

int DrawGraph(DATA *data)
{
  int i;
  int colx, coly;
  float ratio;
  float size;
  float dotratio;
  int   collabel;
  int   j, col;
  Number *lfp;  /* ラベルデータがデータ列の時のポインタ */
  char   *lcp;  /* ラベルデータがラベル列の時のポインタ */

  if (data->mode == PS_ONLY_MODE || data->mode == EPS_ONLY_MODE)
    GrSetPSOnlyMode();

  /* グラフの初期化 */
  GrInit();

  for (i = 0;i < data->nGraphs;i++) {
    colx = data->colX[i];
    coly = data->colY[i];
    collabel = data->colLabel[i];
    lfp = NULL;
    lcp = NULL;

    /* グラフのオープン */
    gGraphNo[i] = GrOpen(data, i);

    /* 描画ウィンドウのサイズの設定 */
    GrSetWindowSize(gGraphNo[i],
                    data->winWidth[i], data->winHeight[i]);

    /* グラフのプロット */
    /* ラベルデータへのポインタを指定する */
    if( collabel > 0 ){
        if( data->colHeader[collabel-1][0] != '@' ){
            lfp = data->table + (collabel-1)*data->nRows;
        }else{
            col = 0;
            for( j = 0; j < collabel; j++ ){
                if( data->colHeader[j][0] == '@' ) col++;
            }
            lcp = data->table_label + (col-1)*data->nRows*MAX_LABELDATA_LEN;
        }
    }
    /* データへのポインタを渡す */
    GrPlot(gGraphNo[i],
           data->table + (colx-1)*data->nRows,
           data->nValidRows,
           data->table + (coly-1)*data->nRows,
           data->nValidRows,
           lfp,
           lcp);
    /* グラフのプロパティの設定 */
    ratio = data->aspectRatio;
    dotratio = data->dotratio;
    size = data->dotSize[i]*dotratio;
    GrSet(gGraphNo[i],
          GR_MAX_EXP, 3,
          GR_XLABEL, data->colHeader[colx-1],
          GR_YLABEL, data->colHeader[coly-1],
          GR_BOX, True,
          GR_LINE_STYLE,
            (data->lineType[i] == 0 ? GR_LINE_STYLE_DOT:
             (data->lineType[i] == 1 ? GR_LINE_STYLE_LINE:
              (data->lineType[i] == 2 ? GR_LINE_STYLE_DASH:
               (data->lineType[i] == 3 ? GR_LINE_STYLE_DOTTEDLINE:
                (data->lineType[i] == 4 ? GR_LINE_STYLE_DASH_LONG:
                 (data->lineType[i] == 5 ? GR_LINE_STYLE_DOTTEDLINE_LONG:
                  GR_LINE_STYLE_LINE))))))),
          GR_DOT_SIZE, &size,
          GR_DECORATION_COLOR, "",
          GR_GRAPH_TYPE, GR_TYPE_XY,
          GR_ASPECT_RATIO, &ratio,
          GR_DUPLICATE, data->duplicate[i],
          GR_DOTCOLOR, data->dot_color[i],
          GR_DOTLABEL_SIZE, data->dotlabel_fontsize[i],
          GR_DOTLABEL_DIST, data->dotlabel_dist[i],
          NULL);
/*    if (data->mode == X_DRAW_MODE)
      GrDraw(gGraphNo[i]);*/
  }

  /* タイトルウィンドウの描画 */
  if (data->mode == X_DRAW_MODE)
    grDrawTitleCenter(gGraphNo[0], data->titleWinId, data->title);

  if (data->mode == PS_ONLY_MODE || data->mode == EPS_ONLY_MODE)
    GrResetPSOnlyMode();

  return(0);
}

/*
 * グラフウィンドウの再描画
 */
int RedrawGraph(DATA *data, Window targetWin)
{
  int i;

  /* ターゲットのウィンドウのみを描画する */
  for (i = 0;i < data->nGraphs;i++) {
    if (GrGetWindow(gGraphNo[i]) == targetWin) {
      GrDraw(gGraphNo[i]);
      GrDrawLabel(gGraphNo[i]);
      return(0);
    }
  }

  /* タイトルウィンドウの描画 */
  grDrawTitleCenter(gGraphNo[0], data->titleWinId, data->title);

  return(0);
}

int PrintGraph(char *filename, DATA *data)
```

```
{
  int i;
  FILE *fp;
  float x, y, width, height;
  float paperWidth, paperHeight;
  float topMargin, bottomMargin, leftMargin, rightMargin;
  float scaleFont;
  float Bwidth, Bheight;

  /* PS出力ファイルのオープン */
  if ((fp = fopen(filename, "w")) == NULL) {
    fprintf(stderr, "Cannot open file '%s'.\n", filename);
    return(-1);
  }

  /* 各ページパラメータの設定(A4,by inch) */
  paperWidth   = 8.5;
  paperHeight  = 11.7;
  topMargin    = 0.4;
  bottomMargin = 0.4;
  leftMargin   = 0.7;
  rightMargin  = 0.4;
  scaleFont    = 0.75;                  /* フォントの全体的なスケーリング */

  /* ヘッダ出力 */
  if (data->mode == PS_ONLY_MODE) {
    fprintf(fp, "%%!PS-Adobe-1.0\n");
  }
  else {
    fprintf(fp, "%%!PS-Adobe-3.0 EPSF-3.0\n");

    Bwidth = (paperWidth * 72.0)/* + 1.0*/;
    Bheight = (paperHeight * 72.0)/* + 1.0*/;

    fprintf(fp, "%%%%BoundingBox: 5 5 %d %d \n", (int)Bwidth, (int)Bheight);
  }
  fprintf(fp, "%%%% DocumentFonts: Helvetica\n");
  fprintf(fp, "%%%% Title: %s\n", data->title);
  fprintf(fp, "%%%% Creator: \n");
  fprintf(fp, "%%%% CreationDate: \n");
  fprintf(fp, "%%%% For: \n");
  fprintf(fp, "%%%% Page: 1\n");
  fprintf(fp, "/inch { 72 mul } def\n");
  fprintf(fp, "/TopMargin    %g inch def\n", topMargin);
  fprintf(fp, "/BottomMargin %g inch def\n", bottomMargin);
  fprintf(fp, "/LeftMargin   %g inch def\n", leftMargin);
  fprintf(fp, "/RightMargin  %g inch def\n", rightMargin);
  fprintf(fp, "/PaperWidth   %g inch def\n", paperWidth);
  fprintf(fp, "/PaperHeight  %g inch def\n", paperHeight);
  fprintf(fp, "/TopLeftPos { LeftMargin PaperHeight TopMargin sub } def\n");
  fprintf(fp, "\n");
  fprintf(fp, "%% define font information\n");
  fprintf(fp, "/TheFontName /Helvetica def\n");
  fprintf(fp, "/TheFontSize  12 def\n");
  fprintf(fp, "/TheLineWidth 1 def                %% linewidth scaling factor\n");
  fprintf(fp, "/TheScaleX    1 def                %% x-axis scaling factor\n");
  fprintf(fp, "/TheScaleY    1 def                %% y-axis scaling factor\n");
  fprintf(fp, "/TheScaleFont 0.9 def              %% font scaling factor\n", scaleFont);
  fprintf(fp, "\n");
  fprintf(fp, "%% define procedures\n");
  fprintf(fp, "/dot1 {\n");
  /* ドット出力時に半径を指定できるようにパラメータ(r)を追加した。 */
  fprintf(fp, "%% called as: x y r dot1\n");
  fprintf(fp, "/r exch def\n");
  fprintf(fp, "TheScaleY mul /y exch def \n");
  fprintf(fp, "TheScaleX mul /x exch def\n");
  /* 指定された点の大きさの1/4を設定 */
  /* ドット出力時に半径を指定できるようにパラメータ(r)を追加した。 */
  fprintf(fp, "x y r 0 360 arc fill\n");
  fprintf(fp, "} def\n");
```

```
  fprintf(fp, "\n");
  fprintf(fp, "/line { \n");
  fprintf(fp, "%% called as: x1 y1 x2 y2 line\n");
  fprintf(fp, "TheScaleY mul /y2 exch def\n");
  fprintf(fp, "TheScaleX mul /x2 exch def\n");
  fprintf(fp, "TheScaleY mul /y1 exch def\n");
  fprintf(fp, "TheScaleX mul /x1 exch def\n");
  fprintf(fp, "x1 y1 moveto\n");
  fprintf(fp, "x2 y2 lineto\n");
  fprintf(fp, "stroke\n");
  fprintf(fp, "} def\n");
  fprintf(fp, "\n");
  fprintf(fp, "/rectangle { \n");
  fprintf(fp, "%% called as: x1 y1 x2 y2 rectangle\n");
  fprintf(fp, "TheScaleY mul /y2 exch def\n");
  fprintf(fp, "TheScaleX mul /x2 exch def\n");
  fprintf(fp, "TheScaleY mul /y1 exch def\n");
  fprintf(fp, "TheScaleX mul /x1 exch def\n");
  fprintf(fp, "x1 y1 moveto\n");
  fprintf(fp, "x1 y2 lineto\n");
  fprintf(fp, "x2 y2 lineto\n");
  fprintf(fp, "x2 y1 lineto\n");
  fprintf(fp, "closepath\n");
  fprintf(fp, "stroke\n");
  fprintf(fp, "} def\n");
  fprintf(fp, "\n");
  /* クリップ領域を設定する (現在はクリッピングしない) */
  /* fprintf(fp, "/setRectClip { \n"); */
  /* fprintf(fp, "%% called as: x y widthy height rectClip\n"); */
  /* fprintf(fp, "TheScaleY mul /height exch def\n"); */
  /* fprintf(fp, "TheScaleX mul /width exch def\n"); */
  /* fprintf(fp, "TheScaleY mul /y exch def\n"); */
  /* fprintf(fp, "TheScaleX mul /x exch def\n"); */
  /* fprintf(fp, "x 0.5 add y 0.5 add width 1 sub height 1 sub rectclip\n"); */
  /* fprintf(fp, "} def\n"); */
  /* fprintf(fp, "\n"); */

  /* 線分を出力する場合の開始指定マクロ */
  fprintf(fp, "/setStartLine { \n");
  fprintf(fp, "%% called as: x1 y1 setStartLine\n");
  fprintf(fp, "TheScaleY mul /y1 exch def\n");
  fprintf(fp, "TheScaleX mul /x1 exch def\n");
  /* 線分の接続タイプをsetlinejoinにて指定(0:miter,1:round,2:bevel) */
  fprintf(fp, "  0 setlinejoin\n");
  /* 線分の両端のキャップタイプをsetlinecapにて指定
                                        (0:butt,1:rount,2:projecting squre) */
  fprintf(fp, "  0 setlinecap\n");
  fprintf(fp, "x1 y1 moveto\n");
  fprintf(fp, "} def\n");
  fprintf(fp, "\n");

  /* 線分の出力ポイントを指定するマクロ */
  fprintf(fp, "/setLineTo { \n");
  fprintf(fp, "%% called as: x1 y1 setLineTo\n");
  fprintf(fp, "TheScaleY mul /y1 exch def\n");
  fprintf(fp, "TheScaleX mul /x1 exch def\n");
  fprintf(fp, "x1 y1 lineto\n");
  fprintf(fp, "} def\n");
  fprintf(fp, "\n");

  /* 線分の終了を指定するマクロ */
  fprintf(fp, "/setEndLine { \n");
  fprintf(fp, "%% called as: setEndLine\n");
  fprintf(fp, "stroke\n");
  fprintf(fp, "} def\n");
  fprintf(fp, "\n");

  /* テキスト出力オフセット指定対応版 */
  fprintf(fp, "/setText {\n");
  fprintf(fp, "%% called as: text x y xoffset yoffset setText\n");
```

```
fprintf(fp, "/yoffset exch def\n");
fprintf(fp, "/xoffset exch def\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
/* X座標にオフセットを加算する */
fprintf(fp, "0 sw sub 2 div x add xoffset add \n");
/* Y座標にオフセットを加算する */
fprintf(fp, "0 sh sub 2 div y add yoffset add \n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");

/* rightTextのオフセット指定対応版 */
fprintf(fp, "/rightText2 {\n");
/* オフセット指定対応により、パラメータを追加 */
fprintf(fp, "%% called as: text x y xoffset yoffset rightText2\n");
fprintf(fp, "/yoffset exch def\n");
fprintf(fp, "/xoffset exch def\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
/* X座標にオフセットを加算する */
fprintf(fp, "x sw sub xoffset add\n");
/* Y座標にオフセットを加算する */
fprintf(fp, "y sh 2 div sub yoffset add\n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");

/* leftTextのオフセット指定対応版 */
fprintf(fp, "/leftText2 {\n");
fprintf(fp, "%% called as: text x y xoffset yoffset leftText2\n");
fprintf(fp, "/yoffset exch def\n");
fprintf(fp, "/xoffset exch def\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
/* X座標にオフセットを加算する */
fprintf(fp, "x xoffset add\n");
/* Y座標にオフセットを加算する */
fprintf(fp, "y sh 2 div sub yoffset add\n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
fprintf(fp, "/centerText {\n");
fprintf(fp, "%% called as: text x y w h centerText\n");
fprintf(fp, "TheScaleY mul /height exch def\n");
fprintf(fp, "TheScaleX mul /width exch def\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
```

```
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
fprintf(fp, "width sw sub 2 div x add\n");
fprintf(fp, "height sh sub 2 div y add\n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
fprintf(fp, "/centerTextRot90 {\n");
fprintf(fp, "%% called as: text x y w h centerTextRot90\n");
fprintf(fp, "TheScaleX mul /height exch def\n");
fprintf(fp, "TheScaleY mul /width exch def\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
fprintf(fp, "height sh sub 2 div x add\n");
fprintf(fp, "width sw sub 2 div y add\n");
fprintf(fp, "moveto\n");
fprintf(fp, "90 rotate\n");
fprintf(fp, "show\n");
fprintf(fp, "-90 rotate\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
fprintf(fp, "/rightText {\n");
fprintf(fp, "%% called as: text x y rightText\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
fprintf(fp, "x sw sub\n");
fprintf(fp, "y sh 2 div sub\n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
fprintf(fp, "/leftText {\n");
fprintf(fp, "%% called as: text x y leftText\n");
fprintf(fp, "TheScaleY mul /y exch def\n");
fprintf(fp, "TheScaleX mul /x exch def\n");
fprintf(fp, "dup\n");
fprintf(fp, "stringwidth\n");
fprintf(fp, "pop\n");
fprintf(fp, "/sh TheFontSize TheScaleFont mul def\n");
fprintf(fp, "/sw exch def\n");
fprintf(fp, "x\n");
fprintf(fp, "y sh 2 div sub\n");
fprintf(fp, "moveto\n");
fprintf(fp, "show\n");
fprintf(fp, "stroke\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
fprintf(fp, "/setFontInfo {\n");
fprintf(fp, "%% called as: font-name font-size setFontInfo\n");
fprintf(fp, "/TheFontSize exch def\n");
fprintf(fp, "/TheFontName exch def\n");
fprintf(fp, "TheFontName findfont TheFontSize TheScaleFont mul scalefont setfont\n");
fprintf(fp, "} def\n");
fprintf(fp, "\n");
```

```
    fprintf(fp, "/setAxesScale {\n");
    fprintf(fp, "%% called as: x y scale-x scale-y setAxesScale\n");
    fprintf(fp, "/scaley exch def\n");
    fprintf(fp, "/scalex exch def\n");
    fprintf(fp, "TheScaleY mul /y exch def\n");
    fprintf(fp, "TheScaleX mul /x exch def\n");
    fprintf(fp, "gsave\n");
    fprintf(fp, "x y translate\n");
    fprintf(fp, "/TheScaleX TheScaleX scalex mul def\n");
    fprintf(fp, "/TheScaleY TheScaleY scaley mul def\n");
    fprintf(fp, "} def\n");
    fprintf(fp, "\n");
    fprintf(fp, "/restoreAxesScale {\n");
    fprintf(fp, "%% called as: x y scale-x scale-y restoreAxesScale\n");
    fprintf(fp, "/scaley exch def\n");
    fprintf(fp, "/scalex exch def\n");
    fprintf(fp, "TheScaleY mul /y exch def\n");
    fprintf(fp, "TheScaleX mul /x exch def\n");
    fprintf(fp, "grestore\n");
    fprintf(fp, "/TheScaleX TheScaleX scalex div def\n");
    fprintf(fp, "/TheScaleY TheScaleY scaley div def\n");
    fprintf(fp, "} def\n");
    fprintf(fp, "\n");
    fprintf(fp, "%% set initial/default values\n");
    fprintf(fp, "/Helvetica 12 setFontInfo\n");
    fprintf(fp, "0.5 setlinewidth\n");
    fprintf(fp, "\n");
    fprintf(fp, "%% set translation to whole paper\n");

  /*#ifdef PS_DEBUG*/
    /* 用紙全体の座標系の設定 */
    if (data->orientation == PAPER_LANDSCAPE) {
      fprintf(fp, "%% paper orientation is landscape.\n");
      fprintf(fp, "%.5g inch %.5g inch %.5g inch %.5g inch setAxesScale\n",
              paperWidth - rightMargin,
              bottomMargin,
              paperHeight - topMargin - bottomMargin,
              paperWidth - rightMargin - leftMargin);
      fprintf(fp, "90 rotate\n");
    }
    else {
      fprintf(fp, "%% paper orientation is portrait.\n");
      fprintf(fp, "%.5g inch %.5g inch %.5g inch %.5g inch setAxesScale\n",
              leftMargin,
              bottomMargin,
              paperWidth - rightMargin - leftMargin,
              paperHeight - topMargin - bottomMargin);
    }

    fprintf(fp, "\n");

    for (i = 0;i < data->nGraphs;i++) {
      /* 個々のグラフの相対位置、サイズを求める */
      x = (float)data->winX[i]/data->mainWinWidth;
      y = (float)(data->mainWinHeight-data->winY[i])/data->mainWinHeight;
      width = (float)data->winWidth[i]/data->mainWinWidth;
      height = (float)data->winHeight[i]/data->mainWinHeight;

      /* 個々のグラフ座標系の設定 */
      fprintf(fp, "%%%%\n");
      fprintf(fp, "%%%% Graph %d\n", i+1);
      fprintf(fp, "%%%%\n");
      fprintf(fp, "%.5g %.5g %.5g %.5g setAxesScale\n",
              x, y, width, height);
      fprintf(fp, "\n");

      /* グラフの描画出力 */
      GrPrint(gGraphNo[i], fp);

      /* 用紙全体の座標系へ戻す */
      fprintf(fp, "\n");
      fprintf(fp, "%.5g %.5g %.5g %.5g restoreAxesScale\n",
              x, y, width, height);
      fprintf(fp, "\n");
    }

    /* タイトルウィンドウの描画 */
    fprintf(fp, "%%%%\n");
    fprintf(fp, "%%%% Title\n");
    fprintf(fp, "%%%%\n");
    x = (float)data->titleWinX/data->mainWinWidth;
    y = (float)(data->mainWinHeight-data->titleWinY)/data->mainWinHeight;
    width = (float)data->titleWinWidth/data->mainWinWidth;
    height = -(float)data->titleWinHeight/data->mainWinHeight;
    grPSDrawTitleCenter(fp, gGraphNo[0], data->title,
                        &x, &y, &width, &height);

    /* 中心点の出力 */
    fprintf(fp, "%%%%\n");
    fprintf(fp, "%%%% Center mark\n");
    fprintf(fp, "%%%%\n");
    x = (float)data->titleWinX/data->mainWinWidth;
    y = (float)(data->mainWinHeight-data->titleWinY)/data->mainWinHeight;
    width = (float)data->titleWinWidth/data->mainWinWidth;
    height = -(float)data->titleWinHeight/data->mainWinHeight;
    grPSSetFont(fp, "Helvetica", NULL, NULL, 10);
    if (data->orientation == PAPER_LANDSCAPE) {
      fprintf(fp, "(%s) %.5g %.5g %.5g %.5g centerText\n",
              "+", 0.0, 1.0, 1.0, 0.08);
    }
    else {
      fprintf(fp, "(%s) %.5g %.5g %.5g %.5g centerText\n",
              "+", 0.0, 0.0, -0.08, 1.0);
    }

    /* Prologの出力 */
    fprintf(fp, "\n");
    fprintf(fp, "showpage\n");
    fclose(fp);

    return(0);
  }

  /*
   * プロットウィンドウの大きさの再設定
   */
  int SetPlotWindowSize(DATA *data)
  {
    int i;
    ulong wPitch, hPitch;
    int wCount, hCount;
    int offset;
    XWindowChanges attr;
    int win_num = -1;
  #ifdef DEBUG
    fprintf(stderr, "SetPlotWindowSize()\n");
  #endif

    offset = DEFAULT_TITLE_WINDOW_HEIGHT*data->mainWinHeight;

    /* 横方向表示の場合 */
    if (data->orientation == PAPER_LANDSCAPE) {
      switch (data->nWindows) {
      case 1:
        wCount = 1; hCount = 1; break;
      case 2:
        wCount = 2; hCount = 1; break;
      case 3:
        wCount = 3; hCount = 1; break;
      case 4:
```

```
      wCount = 2; hCount = 2; break;
    case 5:
    case 6:
      wCount = 3; hCount = 2; break;
    case 7:
    case 8:
      wCount = 4; hCount = 2; break;
    case 9:
      wCount = 3; hCount = 3; break;
    case 10:
    case 11:
    case 12:
      wCount = 4; hCount = 3; break;
    case 13:
    case 14:
    case 15:
      wCount = 5; hCount = 3; break;
    case 16:
      wCount = 4; hCount = 4; break;
    case 17:
    case 18:
    case 19:
    case 20:
      wCount = 5; hCount = 4; break;
    default:
      wCount = 6; hCount = 4; break;
    }
  }
  /* 縦方向表示の場合 */
  else {
    switch (data->nWindows) {
    case 1:
      wCount = 1; hCount = 1; break;
    case 2:
      wCount = 1; hCount = 2; break;
    case 3:
      wCount = 1; hCount = 3; break;
    case 4:
      wCount = 1; hCount = 4; break;
    case 5:
    case 6:
      wCount = 2; hCount = 3; break;
    case 7:
    case 8:
      wCount = 2; hCount = 4; break;
    case 9:
      wCount = 3; hCount = 3; break;
    case 10:
    case 11:
    case 12:
      wCount = 3; hCount = 4; break;
    case 13:
    case 14:
    case 15:
      wCount = 3; hCount = 5; break;
    case 16:
    case 17:
    case 18:
      wCount = 3; hCount = 6; break;
    case 19:
    case 20:
      wCount = 4; hCount = 5; break;
    default:
      wCount = 4; hCount = 6; break;
    }
  }

  wPitch = (ulong)data->mainWinWidth/wCount;
  hPitch = (ulong)(data->mainWinHeight-offset)/hCount;
```

```
  for (i = 0;i < data->nGraphs;i++) {
    if( data->duplicate[i] != DUPLICATE_ON ) win_num++;
    attr.x = (win_num % wCount) * wPitch;
    attr.y = offset + (int)(win_num / wCount) * hPitch;
    attr.width = wPitch;
    attr.height = hPitch;

    if (data->mode == X_DRAW_MODE && data->duplicate[i] == DUPLICATE_OFF )
      XConfigureWindow(data->display, data->winId[i],
                       CWX | CWY | CWWidth | CWHeight,
                       &attr);
    data->winX[i] = attr.x;
    data->winY[i] = attr.y;
    data->winWidth[i] = attr.width;
    data->winHeight[i] = attr.height;
  }

  /* タイトルウィンドウのリサイズ */
  attr.x = 0;
  attr.y = 0;
  attr.width = data->mainWinWidth;
  attr.height = offset;
  if (data->mode == X_DRAW_MODE)
    XConfigureWindow(data->display, data->titleWinId,
                     CWX | CWY | CWWidth | CWHeight,
                     &attr);
  data->titleWinX = attr.x;
  data->titleWinY = attr.y;
  data->titleWinWidth = attr.width;
  data->titleWinHeight = attr.height;

  return(0);
}

/*
 * ウィンドウの生成
 */
int CreatePlotWindows(DATA *data)
{
  ulong forePixel;
  ulong backPixel;
  ulong width, height;
  int screen;
  int i;
  int margin = 0;

  Window win_id[MAX_WINDOW_COUNT];
  int win_num = 0;

#ifdef DEBUG
  fprintf(stderr, "CreatePlotWindows()\n");
  margin = 1;
#endif
  /* ウィンドウのデフォルト値の設定 */
  if (data->orientation == PAPER_LANDSCAPE) {
    width = DEFAULT_MAIN_WINDOW_WIDTH;
    height = DEFAULT_MAIN_WINDOW_HEIGHT;
  }
  else {
    width = DEFAULT_MAIN_WINDOW_HEIGHT;
    height = DEFAULT_MAIN_WINDOW_WIDTH;
  }
  data->mainWinWidth = width;
  data->mainWinHeight = height;

  /* PS出力モードの場合、リターン */
  if (data->mode == PS_ONLY_MODE || data->mode == EPS_ONLY_MODE)
    return(0);

  screen = DefaultScreen(data->display);
```

```
    forePixel = WhitePixel(data->display, screen);
    backPixel = WhitePixel(data->display, screen);
    /* メインウィンドウの生成 */
    data->mainWinId = XCreateSimpleWindow(data->display,
                                DefaultRootWindow(data->display),
                                0, 0, width, height,
                                0, forePixel, backPixel);

    /* イベントの要請 */
    XSelectInput(data->display, data->mainWinId,
                 ExposureMask | KeyPressMask |
                 StructureNotifyMask | SubstructureNotifyMask);

    data->titleWinId = XCreateSimpleWindow(data->display,
                                           data->mainWinId,
                                           0, 0, width, height,
                                           margin, forePixel, backPixel);
    /* イベントの要請 */
    XSelectInput(data->display, data->titleWinId,
                 ExposureMask | KeyPressMask);
    XMapRaised(data->display, data->mainWinId);

    for( i = 0; i < MAX_WINDOW_COUNT; i++ ){
      win_id[i] = XCreateSimpleWindow(data->display,
                                           data->mainWinId,
                                           0, 0, 1, 1,
                                           margin, forePixel, backPixel);
      /* イベントの要請 */
      XSelectInput(data->display, win_id[i],
                   ExposureMask | KeyPressMask);
      XClearWindow(data->display, win_id[i]);

    }

    /* スタック順序の変更 */
    XRestackWindows(data->display, win_id, MAX_WINDOW_COUNT);

    /* 各グラフにウィンドウIDを割り当てる */
    data->winId[0] = win_id[0];

    for( i = 1; i < MAX_GRAPH_COUNT; i++ ){

      if( data->duplicate[i] != DUPLICATE_ON ){
          win_num++;
      }
      data->winId[i] = win_id[win_num];
    }
    /* プロットウィンドウのマップ */
    XMapSubwindows(data->display, data->mainWinId);
    return(0);
}
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *               Version 2.0
 *    (c) Copyright 1997, 1998
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * graph.c - グラフ描画関数
 */

#include <stdio.h>
#include <varargs.h>
#include <math.h>
#include <string.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>
#include "graph.h"
#include "plot.h"

/*****************************************************************************
 *        Name: GrInit
 *    Function: グラフ情報の初期化
 *    Argument: なし
 *      Return: なし
 * Description:
 *      グラフハンドルの初期化を行なう。
 *
 *****************************************************************************/
int GrInit(void)
{
    int i;
#ifdef DEBUG
    fprintf(stderr, "GrInit()\n");fflush(stderr);
#endif
    for (i = 0;i < GR_MAX_GRAPH;i++)
        gGraph[i].on = 0;
    return(0);
}

/*****************************************************************************
 *        Name: GrOpen
 *    Function: グラフのオープン
 *    Argument:
 *              Display *disp   I       ディスプレイ構造体
 *              Window  win     I       ウィンドウID
 *
 *      Return:
 *              >=0  割り当てたグラフハンドル
 *              < 0  失敗
 * Description:
 *      1 空いているグラフハンドルを割り当てる。
 *      2 割り当てたグラフ情報の初期化を行なう。
 *
 *****************************************************************************/
Graph GrOpen(DATA *data, int no)
{
    int         i;                      /* ループカウンタ */
    GrAttr      *p;                     /* カレントグラフ情報 */

   /* フォントサイズをグラフの個数が１２より大きいなら１／２にする。
      但しPS出力の時はそのまま。 */
    Display     *disp;
    Window      win;
    int         nwindows;
    int         mode;

    disp = data->display;
    win = data->winId[no];
    nwindows = data->nWindows;
    mode = data->mode;

#ifdef DEBUG
    fprintf(stderr, "GrOpen()\n");fflush(stderr);
#endif

    /* 空いているグラフ情報を捜す */
    for (i = 0;i < GR_MAX_GRAPH;i++) {
        if (gGraph[i].on == FALSE)
            break;
    }
    if (i >= GR_MAX_GRAPH)
        return(-1);

    p = gGraph+i;
    p->on               = TRUE;
    p->disp             = disp;
    p->win              = win;
    p->x                = GR_POS_DEFAULT_X;
    p->y                = GR_POS_DEFAULT_Y;
    p->width            = GR_POS_DEFAULT_WIDTH;
    p->height           = GR_POS_DEFAULT_HEIGHT;
    strcpy(p->line_style, GR_DEFAULT_LINE_STYLE);
    p->line_width       = GR_DEFAULT_LINE_WIDTH;
    p->dot_size         = 1.0;
    p->xdata            = NULL;
    p->xdatatype        = GR_FLOAT;
    p->xrange[0]        = 0;
    p->xrange[1]        = -1;
    p->ydata            = NULL;
    p->ydatatype        = GR_FLOAT;
    p->datacount        = 0;
    p->yrange[0]        = 0;
    p->yrange[1]        = -1;
    /* ラベルデータへのポインタ (数値(f)、または文字(c)) */
    p->labeldata_fp             = NULL;
    p->labeldata_cp             = NULL;
    /* ラベルデータ描画用フォント及びGC設定 */
    p->dotlabel_fontname        = strdup(GR_DEFAULT_FONTNAME);
    p->dotlabel_fontbold        = strdup(GR_DEFAULT_FONTBOLD);
    p->dotlabel_fontitalic      = strdup(GR_DEFAULT_FONTITALIC);
    p->dotlabel_fontsize        = GR_DEFAULT_FONTSIZE;
    if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
        p->fontsize         = GR_DEFAULT_FONTSIZE/2;
    }
    if (gPSOnlyMode == False &&
        grLoadFontByType(p->disp, p->dotlabel_fontname,
                         p->dotlabel_fontbold, p->dotlabel_fontsize,
                         p->dotlabel_fontitalic, &p->dotlabel_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
    }
    if (gPSOnlyMode == False) {
      p->dotlabel_gc            = XCreateGC(p->disp, p->win, 0, 0);
      p->dotlabel_color         = NULL;
      p->dotlabel_pixel         = BlackPixel(p->disp, 0);
      XSetFont(p->disp, p->dotlabel_gc, p->dotlabel_font->fid);
      XSetForeground(p->disp, p->dotlabel_gc, p->dotlabel_pixel);
    }
    else
      p->dotlabel_color         = NULL;
    /* 線、点描画用GC設定 (初期化) */
    if (gPSOnlyMode == False) {
      p->dot_gc                 = XCreateGC(p->disp, p->win, 0, 0);
      p->dot_color              = NULL;
      p->dot_pixel              = BlackPixel(p->disp, 0);
```

```
    XSetForeground(p->disp, p->dot_gc, p->dot_pixel);
  }
  else
    p->dot_color              = NULL;
  /* label distance */
  p->dotlabel_dist            = 0;
  /* duplicate flag */
  p->duplicate                = DUPLICATE_OFF;

  if (gPSOnlyMode == False) {
    p->foregc                 = XCreateGC(p->disp, p->win, 0, 0);
    p->backgc                 = XCreateGC(p->disp, p->win, 0, 0);
  }
  p->box                      = FALSE;
  p->aspect_ratio             = -1;
  p->xaxis                    = TRUE;
  p->yaxis                    = TRUE;

  p->fontname                 = strdup(GR_DEFAULT_FONTNAME);
  p->fontbold                 = strdup(GR_DEFAULT_FONTBOLD);
  p->fontitalic               = strdup(GR_DEFAULT_FONTITALIC);
  p->fontsize                 = GR_DEFAULT_FONTSIZE;
  if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
      p->fontsize               = GR_DEFAULT_FONTSIZE/2;
  }
  if (gPSOnlyMode == False &&
      grLoadFontByType(p->disp, p->fontname,
                       p->fontbold, p->fontsize, p->fontitalic,
                       &p->font)) {
      fprintf(stderr, "Error: Can't load font\n");
      return(-1);
  }
  p->xticks                   = NULL;
  p->num_xticks               = 0;
  p->yticks                   = NULL;
  p->num_yticks               = 0;
  p->xlabel                   = NULL;
  p->ylabel                   = NULL;
  p->xtick_marks              = NULL;
  p->num_xtick_marks          = 0;
  p->ytick_marks              = NULL;
  p->num_ytick_marks          = 0;
  p->xticks_minor             = NULL;
  p->num_xticks_minor         = 0;
  p->yticks_minor             = NULL;
  p->num_yticks_minor         = 0;
  p->xtick_scale              = NULL;
  p->ytick_scale              = NULL;
  p->decoration_color         = NULL;
  p->decoration_type          = 1;
  if (gPSOnlyMode == False)
    p->decogc                 = XCreateGC(p->disp, p->win, 0, 0);

  /* タイトルフォント */
  p->title_fontname           = strdup(GR_DEFAULT_TITLE_FONTNAME);
  p->title_fontbold           = strdup(GR_DEFAULT_TITLE_FONTBOLD);
  p->title_fontitalic         = strdup(GR_DEFAULT_TITLE_FONTITALIC);
  p->title_fontsize           = GR_DEFAULT_TITLE_FONTSIZE;
  if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
      p->title_fontsize         = GR_DEFAULT_TITLE_FONTSIZE;
  }
  if (gPSOnlyMode == False &&
      grLoadFontByType(p->disp,
                       p->title_fontname,
                       p->title_fontbold,
                       p->title_fontsize,
                       p->title_fontitalic,
                       &p->title_font)) {
      fprintf(stderr, "Error: Can't load font\n");
      return(-1);
```

```
  }
  if (gPSOnlyMode == False) {
    p->title_gc               = XCreateGC(p->disp, p->win, 0, 0);
    p->title_color            = NULL;
    p->title_pixel            = BlackPixel(p->disp, 0);
    XSetFont(p->disp, p->title_gc, p->title_font->fid);
    XSetForeground(p->disp, p->title_gc, p->title_pixel);
  }
  else
    p->title_color            = NULL;

  /* ラベルフォント */
  p->label_fontname           = strdup(GR_DEFAULT_LABEL_FONTNAME);
  p->label_fontbold           = strdup(GR_DEFAULT_LABEL_FONTBOLD);
  p->label_fontitalic         = strdup(GR_DEFAULT_LABEL_FONTITALIC);
  p->label_fontsize           = GR_DEFAULT_LABEL_FONTSIZE;
  if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
      p->label_fontsize         = GR_DEFAULT_LABEL_FONTSIZE/2;
  }
  if (gPSOnlyMode == False &&
      grLoadFontByType(p->disp,
                       p->label_fontname,
                       p->label_fontbold,
                       p->label_fontsize,
                       p->label_fontitalic,
                       &p->label_font)) {
      fprintf(stderr, "Error: Can't load font\n");
      return(-1);
  }
  if (gPSOnlyMode == False) {
    p->label_gc               = XCreateGC(p->disp, p->win, 0, 0);
    p->label_color            = NULL;
    p->label_pixel            = BlackPixel(p->disp, 0);
    XSetFont(p->disp, p->label_gc, p->label_font->fid);
    XSetForeground(p->disp, p->label_gc, p->label_pixel);
  }
  else
    p->label_color            = NULL;

  /* 目盛りフォント */
  p->tick_fontname            = strdup(GR_DEFAULT_TICK_FONTNAME);
  p->tick_fontbold            = strdup(GR_DEFAULT_TICK_FONTBOLD);
  p->tick_fontitalic          = strdup(GR_DEFAULT_TICK_FONTITALIC);
  p->tick_fontsize            = GR_DEFAULT_TICK_FONTSIZE;
  if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
      p->tick_fontsize          = GR_DEFAULT_TICK_FONTSIZE/2;
  }
  if (gPSOnlyMode == False &&
      grLoadFontByType(p->disp,
                       p->tick_fontname,
                       p->tick_fontbold,
                       p->tick_fontsize,
                       p->tick_fontitalic,
                       &p->tick_font)) {
      fprintf(stderr, "Error: Can't load font\n");
      return(-1);
  }
  if (gPSOnlyMode == False) {
    p->tick_gc                = XCreateGC(p->disp, p->win, 0, 0);
    p->tick_color             = NULL;
    p->tick_pixel             = BlackPixel(p->disp, 0);
    XSetFont(p->disp, p->tick_gc, p->tick_font->fid);
    XSetForeground(p->disp, p->tick_gc, p->tick_pixel);
  }
  else
    p->tick_color             = NULL;

  /* 目盛り(上付)フォント */
  p->ticksub_fontname = strdup(GR_DEFAULT_TICKSUB_FONTNAME);
  p->ticksub_fontbold = strdup(GR_DEFAULT_TICKSUB_FONTBOLD);
```

```
    p->ticksub_fontitalic = strdup(GR_DEFAULT_TICKSUB_FONTITALIC);
    p->ticksub_fontsize = GR_DEFAULT_TICKSUB_FONTSIZE;
    if ((nwindows > 12) && (mode == X_DRAW_MODE)) {
        p->ticksub_fontsize = GR_DEFAULT_TICKSUB_FONTSIZE/2;
    }
    if (gPSOnlyMode == False &&
        grLoadFontByType(p->disp,
                         p->ticksub_fontname,
                         p->ticksub_fontbold,
                         p->ticksub_fontsize,
                         p->ticksub_fontitalic,
                         &p->ticksub_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
    }
    if (gPSOnlyMode == False) {
      p->ticksub_gc       = XCreateGC(p->disp, p->win, 0, 0);
      p->ticksub_color    = NULL;
      p->ticksub_pixel    = BlackPixel(p->disp, 0);
      XSetFont(p->disp, p->ticksub_gc, p->ticksub_font->fid);
      XSetForeground(p->disp, p->ticksub_gc, p->ticksub_pixel);
    }
    else
      p->ticksub_color    = NULL;

    p->draw_mode        = GR_MODE_NORMAL;
    p->draw_size        = 0;
    p->draw_ptr         = 0;
    p->draw_buffer      = NULL;
    p->hold             = FALSE;
    p->max_exp          = GR_DEFAULT_MAX_EXP;
    p->graph_type       = GR_DEFAULT_GRAPH_TYPE;

    /* GC */
    if (gPSOnlyMode == False) {
      XSetForeground(p->disp, p->foregc, BlackPixel(p->disp, 0));
      XSetForeground(p->disp, p->backgc, WhitePixel(p->disp, 0));
      XSetFont(p->disp, p->foregc, p->font->fid);
    }

    return(i);
}

/*****************************************************************************
 *        Name: GrSet
 *    Function: グラフ情報の設定
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *              (ResourceID)    I       リソースID
 *              (ResourceValue) I       リソース値
 *      Return:
 *              0   成功
 *             -1   失敗
 * Description:
 *
 *****************************************************************************/
int GrSet(gr, va_alist)
Graph gr;
va_dcl
{
    GrAttr      *p;
    va_list     ap;
    int         prop;
    float       *xdata, *ydata;
    float       *xrange, *yrange;
    int         isfont = FALSE;
    int         istitle_font = FALSE;
    int         islabel_font = FALSE;
    int         istick_font = FALSE;
    int         isticksub_font = FALSE;
```

```
    int         is_draw_mode = FALSE;
    int         is_xrange = FALSE;
    int         is_yrange = FALSE;
    char        *str;

#ifdef DEBUG
    fprintf(stderr, "GrSet(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on) {
        return(-1);
    }

    p = gGraph+gr;
    va_start(ap);
    xdata = ydata = NULL;
    while (1) {
        switch (prop = va_arg(ap, int)) {
        case NULL:
            goto next;
            break;
        case GR_X:
            p->x = va_arg(ap, float);
            break;
        case GR_Y:
            p->y = va_arg(ap, float);
            break;
        case GR_WIDTH:
            p->width = va_arg(ap, float);
            break;
        case GR_HEIGHT:
            p->height = va_arg(ap, float);
            break;
        case GR_LINE_STYLE:
            strcpy(p->line_style, va_arg(ap, char *));
            break;
        case GR_LINE_WIDTH:
            p->line_width = va_arg(ap, float);
            break;
        case GR_DOT_SIZE:
            { float *f;
              f = va_arg(ap, float *);
              if( *f > 0 ){
                  p->dot_size = *f;
              }
            }
            break;
        case GR_XDATA:
            xdata = va_arg(ap, float *);
            break;
        case GR_XDATATYPE:
            p->xdatatype = va_arg(ap, int);
            break;
        case GR_YDATA:
            ydata = va_arg(ap, float *);
            break;
        case GR_YDATATYPE:
            p->ydatatype = va_arg(ap, int);
            break;
        case GR_DATACOUNT:
            p->datacount = va_arg(ap, int);
            break;
        case GR_TITLE:
            p->title = strdup(va_arg(ap, char *));
            break;
        case GR_WINDOW:
            p->win = va_arg(ap, Window);
            break;
        case GR_BOX:
            p->box = va_arg(ap, int);
            break;
```

```
case GR_ASPECT_RATIO:
    { float *f;
      f = va_arg(ap, float *);
      p->aspect_ratio = *f;
    }
    break;
case GR_XTICKS:
    p->xticks = va_arg(ap, float *);
    break;
case GR_NUM_XTICKS:
    p->num_xticks = va_arg(ap, int);
    break;
case GR_YTICKS:
    p->yticks = va_arg(ap, float *);
    break;
case GR_NUM_YTICKS:
    p->num_yticks = va_arg(ap, int);
    break;
case GR_XLABEL:
    p->xlabel = strdup(va_arg(ap, char *));
    break;
case GR_YLABEL:
    p->ylabel = strdup(va_arg(ap, char *));
    break;
case GR_DECORATION_COLOR:
    str = va_arg(ap, char *);
    if (str && *str != '\0') {
      p->decoration_color = strdup(str);
      if (gPSOnlyMode == False)
        grAllocHighLowColor(p->disp,
                            p->decoration_color,
                            p->decoration_pixel);
    }
    break;
case GR_XRANGE:
    xrange = va_arg(ap, float *);
    is_xrange = TRUE;
    break;
case GR_YRANGE:
    yrange = va_arg(ap, float *);
    is_yrange = TRUE;
    break;
case GR_FONTNAME:
    p->fontname = strdup(va_arg(ap, char *));
    isfont = TRUE;
    break;
case GR_FONTBOLD:
    p->fontbold = strdup(va_arg(ap, char *));
    isfont = TRUE;
    break;
case GR_FONTITALIC:
    p->fontitalic = strdup(va_arg(ap, char *));
    isfont = TRUE;
    break;
case GR_FONTSIZE:
    p->fontsize = va_arg(ap, int);
    isfont = TRUE;
    break;
case GR_TITLE_FONTNAME:
    p->title_fontname = strdup(va_arg(ap, char *));
    istitle_font = TRUE;
    break;
case GR_TITLE_FONTBOLD:
    p->title_fontbold = strdup(va_arg(ap, char *));
    istitle_font = TRUE;
    break;
case GR_TITLE_FONTITALIC:
    p->title_fontitalic = strdup(va_arg(ap, char *));
    istitle_font = TRUE;
    break;
```

```
case GR_TITLE_FONTSIZE:
    p->title_fontsize = va_arg(ap, int);
    istitle_font = TRUE;
    break;
case GR_TITLE_COLOR:
    {
        Colormap        cmap;
        char            *name;
        XColor          ci, cr;

        if (gPSOnlyMode == False)
          cmap = DefaultColormap(p->disp, 0);
        name = strdup(va_arg(ap, char *));
        if (p->title_color) {
            Free(p->title_color);
            if (gPSOnlyMode == False)
              XFreeColors(p->disp, cmap, &p->title_pixel, 1, 0);
        }
        if (!*name) {
            Free(name);
            p->title_color = NULL;
            if (gPSOnlyMode == False)
              p->title_pixel = BlackPixel(p->disp, 0);
            break;
        }
        p->title_color = name;
        if (gPSOnlyMode == False) {
          XAllocNamedColor(p->disp, cmap, p->title_color, &ci, &cr);
          p->title_pixel = cr.pixel;
          XSetForeground(p->disp, p->title_gc, p->title_pixel);
        }
    }
    break;
case GR_LABEL_FONTNAME:
    p->label_fontname = strdup(va_arg(ap, char *));
    islabel_font = TRUE;
    break;
case GR_LABEL_FONTBOLD:
    p->label_fontbold = strdup(va_arg(ap, char *));
    islabel_font = TRUE;
    break;
case GR_LABEL_FONTITALIC:
    p->label_fontitalic = strdup(va_arg(ap, char *));
    islabel_font = TRUE;
    break;
case GR_LABEL_FONTSIZE:
    p->label_fontsize = va_arg(ap, int);
    islabel_font = TRUE;
    break;
case GR_LABEL_COLOR:
    {
        Colormap        cmap;
        char            *name;
        XColor          ci, cr;

        if (gPSOnlyMode == False)
          cmap = DefaultColormap(p->disp, 0);
        name = strdup(va_arg(ap, char *));
        if (p->label_color) {
            Free(p->label_color);
            if (gPSOnlyMode == False)
              XFreeColors(p->disp, cmap, &p->label_pixel, 1, 0);
        }
        if (!*name) {
            Free(name);
            p->label_color = NULL;
            if (gPSOnlyMode == False)
              p->label_pixel = BlackPixel(p->disp, 0);
            break;
        }
```

```
            p->label_color = name;
            if (gPSOnlyMode == False) {
              XAllocNamedColor(p->disp, cmap, p->label_color, &ci, &cr);
              p->label_pixel = cr.pixel;
              XSetForeground(p->disp, p->label_gc, p->label_pixel);
            }
        }
        break;
    case GR_TICK_FONTNAME:
        p->tick_fontname = strdup(va_arg(ap, char *));
        istick_font = TRUE;
        break;
    case GR_TICK_FONTBOLD:
        p->tick_fontbold = strdup(va_arg(ap, char *));
        istick_font = TRUE;
        break;
    case GR_TICK_FONTITALIC:
        p->tick_fontitalic = strdup(va_arg(ap, char *));
        istick_font = TRUE;
        break;
    case GR_TICK_FONTSIZE:
        p->tick_fontsize = va_arg(ap, int);
        istick_font = TRUE;
        break;
    case GR_TICK_COLOR:
        {
            Colormap        cmap;
            char            *name;
            XColor          ci, cr;

            if (gPSOnlyMode == False)
              cmap = DefaultColormap(p->disp, 0);
            name = strdup(va_arg(ap, char *));
            if (p->tick_color) {
                Free(p->tick_color);
                if (gPSOnlyMode == False)
                  XFreeColors(p->disp, cmap, &p->tick_pixel, 1, 0);
            }
            if (!*name) {
                Free(name);
                p->tick_color = NULL;
                if (gPSOnlyMode == False)
                  p->tick_pixel = BlackPixel(p->disp, 0);
                break;
            }
            p->tick_color = name;
            if (gPSOnlyMode == False) {
              XAllocNamedColor(p->disp, cmap, p->tick_color, &ci, &cr);
              p->tick_pixel = cr.pixel;
              XSetForeground(p->disp, p->tick_gc, p->tick_pixel);
            }
        }
        break;
    case GR_TICKSUB_FONTNAME:
        p->ticksub_fontname = strdup(va_arg(ap, char *));
        isticksub_font = TRUE;
        break;
    case GR_TICKSUB_FONTBOLD:
        p->ticksub_fontbold = strdup(va_arg(ap, char *));
        isticksub_font = TRUE;
        break;
    case GR_TICKSUB_FONTITALIC:
        p->ticksub_fontitalic = strdup(va_arg(ap, char *));
        isticksub_font = TRUE;
        break;
    case GR_TICKSUB_FONTSIZE:
        p->ticksub_fontsize = va_arg(ap, int);
        isticksub_font = TRUE;
        break;
    case GR_TICKSUB_COLOR:
```

```
        {
            Colormap        cmap;
            char            *name;
            XColor          ci, cr;

            if (gPSOnlyMode == False)
              cmap = DefaultColormap(p->disp, 0);
            name = strdup(va_arg(ap, char *));
            if (p->ticksub_color) {
                Free(p->ticksub_color);
                if (gPSOnlyMode == False)
                  XFreeColors(p->disp, cmap, &p->ticksub_pixel, 1, 0);
            }
            if (!*name) {
                Free(name);
                p->ticksub_color = NULL;
                if (gPSOnlyMode == False)
                  p->ticksub_pixel = BlackPixel(p->disp, 0);
                break;
            }
            p->ticksub_color = name;
            if (gPSOnlyMode == False) {
              XAllocNamedColor(p->disp, cmap, p->ticksub_color, &ci, &cr);
              p->ticksub_pixel = cr.pixel;
              XSetForeground(p->disp, p->ticksub_gc, p->ticksub_pixel);
            }
        }
        break;
    case GR_DRAW_MODE:
        p->draw_mode = va_arg(ap, int);
        is_draw_mode = TRUE;
        break;
    case GR_DRAW_SIZE:
        {
            int     size;
            size = va_arg(ap, int);
            if (size < 0)
                break;
            if (p->draw_buffer)
                Free(p->draw_buffer);
            p->draw_buffer = (Number *)malloc(sizeof(Number)*size);
            p->draw_size = size;
            p->draw_ptr = 0;
            break;
        }
    case GR_HOLD:
        p->hold = va_arg(ap, int);
        break;
    case GR_MAX_EXP:
        p->max_exp = va_arg(ap, int);
        break;
    case GR_GRAPH_TYPE:
        p->graph_type = va_arg(ap, int);
        break;
    case GR_DUPLICATE:
        /* グラフの重ね合わせフラグの設定 */
        p->duplicate = va_arg( ap, int );
        break;
    case GR_DOTCOLOR:
        /* 点、線のカラー設定（デフォルトは黒）  */
        {
            Colormap        cmap;
            XColor          color;
            char            *colorname;
            char            *subcolname;

            colorname = strdup(va_arg(ap, char *));
            if (gPSOnlyMode == False)
              cmap = DefaultColormap(p->disp, 0);
            if( colorname[0] == '#' ){
```

```
                p->dot_color = strdup( colorname );
            }else if( (subcolname = strchr( colorname, ':' ) ) != NULL ){
                p->dot_color = strdup( subcolname );
                p->dot_color[0]='#';
            }else{
                p->dot_color = strdup( "#000000" );
            }
            if( gPSOnlyMode == False ){
                if( XParseColor( p->disp, cmap, p->dot_color, &color )
                   == 0 ){
                    printf( "XParseColor:can't get XColor..\n" );
                    color.red = 0;
                    color.green = 0;
                    color.blue = 0;
                }
                if( XAllocColor( p->disp, cmap, &color ) == 0 ){
                    printf( "XAllocColor:can't get color cell ..\n" );
                    p->dot_pixel = BlackPixel( p->disp, 0 );
                }else{
                    p->dot_pixel = color.pixel;
                }
                XSetForeground(p->disp, p->dot_gc, p->dot_pixel);
            }
        }
        break;
    case GR_DOTLABEL_SIZE:
        /* 点ラベルのフォントサイズの設定 */
        p->dotlabel_fontsize = va_arg(ap, int);
        if( gPSOnlyMode == False && p->dotlabel_fontsize > 0 ){
            if( p->dotlabel_font ){
                XFreeFont( p->disp, p->dotlabel_font );
            }
            if(grLoadFontByType( p->disp, p->dotlabel_fontname,
                                 p->dotlabel_fontbold,p->dotlabel_fontsize,
                                 p->dotlabel_fontitalic,
                                 &p->dotlabel_font )){
                fprintf( stderr, "Error:Can't load font\n" );
                return (-1);
            }
            XSetFont( p->disp, p->dotlabel_gc, p->dotlabel_font->fid );
        }
        break;
    case GR_DOTLABEL_DIST:
        /* ラベルをてんからどれだけはなして打つか設定 */
        {
            int dist;
            dist = va_arg(ap, int);
            if( dist > -1 ){
                p->dotlabel_dist = dist;
            }
        }
        break;
    default:
        break;
    }
}

next:
    va_end(ap);

    /* データ変更 */
    if (xdata) {
        int     len = sizeof(Number)*p->datacount;
        Free(p->xdata);
        p->xdata = (Number *)malloc(len);
        memcpy(p->xdata, xdata, len);
    }
    if (ydata) {
        int     len;
        int     rem;
        if (p->ydata)
            Free(p->ydata);
        switch (p->draw_mode) {
        case GR_MODE_NORMAL:
            len = sizeof(Number)*p->datacount;
            p->ydata = (Number *)malloc(len);
            memcpy(p->ydata, ydata, len);
            break;
        case GR_MODE_FLOW:
            rem = p->datacount - p->draw_size;
            if (rem > 0) {
                p->datacount -= rem;
                ydata += rem;
            }
            rem = p->datacount - (p->draw_size - p->draw_ptr);
            if (rem < 0) {
                len = sizeof(Number)*(p->draw_ptr+1-rem);
                memcpy(p->draw_buffer, p->draw_buffer+rem, len);
                p->draw_ptr += rem;
            }
            len = sizeof(Number)*p->datacount;
            memcpy(p->draw_buffer+p->draw_ptr, ydata, len);
            p->draw_ptr += p->datacount;
            break;
        }
        if (p->hold == FALSE) {
            grFreeAxisAttributes(gr);
        }
    }

    /* フォント変更 */
    if (gPSOnlyMode == False && isfont == TRUE) {
      if (p->font)
        XFreeFont(p->disp, p->font);
      if (grLoadFontByType(p->disp, p->fontname,
                           p->fontbold, p->fontsize, p->fontitalic,
                           &p->font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
      }
      XSetFont(p->disp, p->foregc, p->font->fid);
    }
    /* タイトルフォント変更 */
    if (gPSOnlyMode == False && istitle_font == TRUE) {
      XFreeFont(p->disp, p->title_font);
      if (grLoadFontByType(p->disp, p->title_fontname,
                           p->title_fontbold,
                           p->title_fontsize,
                           p->title_fontitalic,
                           &p->title_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
      }
      XSetFont(p->disp, p->title_gc, p->title_font->fid);
    }
    /* ラベルフォント変更 */
    if (gPSOnlyMode == False && islabel_font == TRUE) {
      if (p->label_font)
        XFreeFont(p->disp, p->label_font);
      if (grLoadFontByType(p->disp, p->label_fontname,
                           p->label_fontbold,
                           p->label_fontsize,
                           p->label_fontitalic,
                           &p->label_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
      }
      XSetFont(p->disp, p->label_gc, p->label_font->fid);
    }
    /* 目盛りフォント変更 */
```

```
    if (gPSOnlyMode == False && istick_font == TRUE) {
      if (p->tick_font)
        XFreeFont(p->disp, p->tick_font);
      if (grLoadFontByType(p->disp, p->tick_fontname,
                           p->tick_fontbold,
                           p->tick_fontsize,
                           p->tick_fontitalic,
                           &p->tick_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
      }
      XSetFont(p->disp, p->tick_gc, p->tick_font->fid);
    }
    /* 目盛り(上付)フォント変更 */
    if (gPSOnlyMode == False && isticksub_font == TRUE) {
      if (p->ticksub_font)
        XFreeFont(p->disp, p->ticksub_font);
      if (grLoadFontByType(p->disp, p->ticksub_fontname,
                           p->ticksub_fontbold,
                           p->ticksub_fontsize,
                           p->ticksub_fontitalic,
                           &p->ticksub_font)) {
        fprintf(stderr, "Error: Can't load font\n");
        return(-1);
      }
      XSetFont(p->disp, p->ticksub_gc, p->ticksub_font->fid);
    }

    /* 描画モード変更 */
    if (is_draw_mode == TRUE) {
    }

    if (is_xrange == TRUE) {
        p->xrange[0] = xrange[0];
        p->xrange[1] = xrange[1];
    }
    if (is_yrange == TRUE) {
        p->yrange[0] = yrange[0];
        p->yrange[1] = yrange[1];
    }

    return(0);
}

/*****************************************************************************
 *        Name: GrHold
 *    Function: グラフ情報の保持フラグのセット
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *      Return:
 *              0  成功
 *             -1  失敗
 * Description:
 *
 *****************************************************************************/
int GrHold(Graph gr, int on)
{
    GrAttr      *p;
#ifdef DEBUG
    fprintf(stderr, "GrHold(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph + gr;
    p->hold = on;
    return(0);
}

/*      8       16      24      32      40      48      56                      */
```

```
/*------+-------+-------+-------+-------+-------+-------+--------------------*/
/*****************************************************************************
 *        Name: GrPlot
 *    Function: グラフのプロット
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *      Return:
 *              0  成功
 *             -1  失敗
 * Description:
 *              設定されたリソース値にしたがって描画を行なう。
 *              実際にウィンドウに描画されるのは、次の GrDraw 関数で行なう。
 *
 *****************************************************************************/
int GrPlot(Graph gr, Number *x, int nx, Number *y, int ny, Number *lfp, char *lcp)
{
    GrAttr      *p;
    int         i;
#ifdef DEBUG
    fprintf(stderr, "GrPlot(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph + gr;

    Free(p->xdata);
    Free(p->ydata);
    /* yデータ */
    p->datacount = ny;
    if (y && ny > 0) {
        int     len;
        int     rem;

        switch (p->draw_mode) {
        case GR_MODE_NORMAL:
            len = sizeof(Number)*ny;
            p->ydata = (Number *)malloc(len);
            memcpy(p->ydata, y, len);
            break;

        case GR_MODE_FLOW:
            rem = p->datacount - p->draw_size;
            if (rem > 0) {
                p->datacount -= rem;
                y += rem;
            }
            rem = p->datacount - (p->draw_size - p->draw_ptr);
            if (rem < 0) {
                len = sizeof(Number)*(p->draw_ptr+1-rem);
                memcpy(p->draw_buffer, p->draw_buffer+rem, len);
                p->draw_ptr += rem;
            }
            len = sizeof(Number)*p->datacount;
            memcpy(p->draw_buffer+p->draw_ptr, y, len);
            p->draw_ptr += p->datacount;
            break;
        }
    }

    /* xデータ */
    if (x && nx > 0) {
        int     len = sizeof(Number)*nx;
        p->xdata = (Number *)malloc(len);
        memcpy(p->xdata, x, len);
    }
    else {
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  calc xdata\n");fflush(stderr);
#endif
```

```
        p->xdata = (Number *)malloc(sizeof(Number)*p->datacount);
        for (i = 0;i < p->datacount;i++)
            p->xdata[i] = i+1;
    }

    /* ドット毎のラベルデータへのポインタ */
    if ( lfp != NULL ){
        p->labeldata_fp = lfp;
    }
    if ( lcp != NULL ){
        p->labeldata_cp = lcp;
    }
    /* 保持フラグが立っていない場合、値の範囲をクリア */
    if (p->hold == FALSE) {
        grFreeAxisAttributes(gr);
        p->xrange[0] = p->yrange[0] = 0.0;
        p->xrange[1] = p->yrange[1] = -1.0;
    }

    return(0);
}

/*      8       16      24      32      40      48      56                  */
/*------+-------+-------+-------+-------+-------+-------+-------------------*/
/*****************************************************************************
 *        Name: GrDraw
 *    Function: グラフの描画
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *      Return:
 *              0  成功
 *             -1  失敗
 * Description:
 *              ウィンドウへグラフの描画を行なう。
 *
 *****************************************************************************/
int GrDraw(Graph gr)
{
    GrAttr      *p;
    int         i;
    XPoint      *ps;

#ifdef GRAPH_DEBUG
    fprintf(stderr, "GrDraw(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph + gr;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  hold: %d\n", p->hold);fflush(stderr);
#endif

    /* チェック */
    if (p->width <= 0 ||
        p->height <= 0 ||
        p->datacount < 0)
        return(-2);

    if (p->graph_type == GR_TYPE_METER) {
        return(grDrawMeter(gr));
    }

    /* 座標属性の計算 */
    if (p->hold == FALSE) {
        grCalcAxisAttributes(gr);
    }

    /* 描画領域の計算 */
    grCalcAxisOrigin(gr);
```

```
    /* duplicate がoff(0)の時だけ、グラフ描画領域をクリアする */
    if( p->duplicate == DUPLICATE_OFF ){
        grClearGraph(gr);
    }
    if (p->hold == FALSE) {

        /* 装飾の描画 */
        if (p->decoration_color) {
            grDrawDecoration(gr);
        }
        else{
            /* duplicate がoff(0)の時だけ、ウィンドウをクリアする */
            if( p->duplicate == DUPLICATE_OFF ){
                GrClear(gr);
            }
        }

        /* ボックス/座標軸の描画 */
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  draw box/axis\n");fflush(stderr);
#endif

        /* duplicateが off(0)の時だけ座標軸、メモリ文字列、
           X/Y軸ラベルを描画する */
        if( p->duplicate != DUPLICATE_ON ){
        if (p->box) {
            XDrawRectangle(p->disp, p->win, p->foregc,
                           p->Ox, p->Oy, p->Ow, p->Oh);
        }
        else {
            /* x座標軸 */
            XDrawLine(p->disp, p->win, p->foregc,
                      p->Ox-1, p->Oy-1 + p->Oh+2,
                      p->Ox-1 + p->Ow+2, p->Oy-1 + p->Oh+2);
            /* y座標軸 */
            XDrawLine(p->disp, p->win, p->foregc,
                      p->Ox-1, p->Oy-1 + p->Oh+2,
                      p->Ox-1, p->Oy-1);
            XDrawLine(p->disp, p->win, p->foregc,
                      p->Ox-1, p->Oy-1,
                      (int)(p->Ox-1 + p->Aw*GR_TICK_LENGTH/3),
                      p->Oy-1);
            XDrawLine(p->disp, p->win, p->foregc,
                      p->Ox-1 + p->Ow+2, p->Oy-1 + p->Oh+2,
                      p->Ox-1 + p->Ow+2,
                      (int)((p->Oy-1 + p->Oh+2)-p->Ah*GR_TICK_LENGTH/3));
        }
        /* x目盛り文字列の描画 */
        if (p->num_xtick_marks > 0) {
            int         i, x1, y1, x, y;
            float       rx, dx;

#ifdef GRAPH_DEBUG
            fprintf(stderr, "  draw xtick marks\n");fflush(stderr);
#endif
            rx = p->xrange[1] - p->xrange[0];

            if (rx == 0) dx = 0;
            else dx = p->Ow/rx;
            for (i = 0;i < p->num_xtick_marks;i++) {
                x1 = p->Ox + (p->xticks[i]-p->xrange[0])*dx;
                y1 = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN;
                grGetCenterPos(gr, p->xtick_marks[i],
                               x1, y1, 0, GR_POS_XTICKS_HEIGHT, &x, &y);
                XDrawString(p->disp, p->win, p->tick_gc,
                            x, y, p->xtick_marks[i],
                            strlen(p->xtick_marks[i]));
            }
        }
```

```
/* xスケールの描画 */
if (p->xtick_scale) {
  int  x, y, xx, yy;
  char str[20], *ptr;
  unsigned int width, height;
  x = p->Rx + p->Rw;
  y = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN
    + GR_POS_XTICKS_HEIGHT;

  strcpy(str, "x");
  strcat(str, p->xtick_scale);
  if ((ptr = strchr(str, '^')) != NULL)
    *ptr++ = '\0';
  grGetCenterPos(gr, str, x, y, -1, 0, &x, &y);
  XDrawString(p->disp, p->win, p->tick_gc,
              x, y, str, strlen(str));
  if (ptr) {
    grTextWidth(p->disp, p->win, p->tick_gc,
                str, strlen(str), &width, &height);
    XDrawString(p->disp, p->win, p->ticksub_gc,
                x + width, y - height/2, ptr, strlen(ptr));
  }
}

/* y目盛り文字列の描画 */
if (p->num_ytick_marks > 0) {
    int         i, x1, yl, x, y;
    float               ry, dy;
    ry = p->yrange[1] - p->yrange[0];
    if (ry == 0) dy = 0;
    else dy = p->Oh/ry;
    for (i = 0;i < p->num_ytick_marks;i++) {
        x1 = p->Ox - GR_POS_YTICKS_MARGIN;
        yl = p->Oy + p->Oh - (p->yticks[i]-p->yrange[0])*dy;
        grGetCenterPos(gr, p->ytick_marks[i], x1, yl, -1, 0, &x, &y);
        XDrawString(p->disp, p->win, p->tick_gc,
                    x, y, p->ytick_marks[i],
                    strlen(p->ytick_marks[i]));
    }
}

/* タイトル文字列の描画 */
if (p->title && *p->title) {
    int x, y;
    unsigned int        w, h;
    w = p->Aw;
    h = p->Ry;
    grGetCenterPos(gr, p->title, p->Ax, p->Ay, w, h, &x, &y);
    XDrawString(p->disp, p->win, p->title_gc,
                x, y, p->title, strlen(p->title));
}

/* xラベル文字列の描画 */
if (p->xlabel && *p->xlabel) {
    int x, y;
    unsigned int        w, h;

    w = p->Ow;
    h = (p->Ay + p->Ah) - (p->Ry + p->Rh);
    grGetCenterPos(gr, p->xlabel, p->Ox, p->Ry + p->Rh, w, h, &x, &y);
    XDrawString(p->disp, p->win, p->label_gc,
                x, y, p->xlabel, strlen(p->xlabel));
}

/* yラベル文字列の描画 */
if (p->ylabel && *p->ylabel) {
    int x, y;
    unsigned int        w, h;

    w = p->Rx - p->Ax;
```

```
            h = p->Oh;
/*          grGetCenterPos(gr, p->ylabel, p->Ax, p->Ay, h, w, &y, &x);
            grDrawStringDownToUp(p->disp, p->win, p->label_gc,
                                 x, p->Oy + p->Oh - (y - p->Ay),
                                 p->ylabel, strlen(p->ylabel));*/
            grGetCenterPos(gr, p->ylabel, 0, 0, h, w, &y, &x);
            grDrawStringDownToUp(p->disp, p->win, p->label_gc,
                                 p->Ax + w - x, p->Oy + p->Oh - y,
                                 p->ylabel, strlen(p->ylabel));
        }

        /* yスケールの描画 */
        if (p->ytick_scale) {
          int  x, y;
          char str[20], *ptr;
          unsigned int width, height;
          x = p->Ox;
          y = p->Ry;
          grGetCenterPos(gr, str, x, y, -1, 0, &x, &y);

          strcpy(str, "x");
          strcat(str, p->ytick_scale);
          if ((ptr = strchr(str, '^')) != NULL)
            *ptr++ = '\0';
          XDrawString(p->disp, p->win, p->tick_gc,
                      x, y, str, strlen(str));
          if (ptr) {
            grTextWidth(p->disp, p->win, p->tick_gc,
                        str, strlen(str), &width, &height);
            XDrawString(p->disp, p->win, p->ticksub_gc,
                        x + width, y - height/2, ptr, strlen(ptr));
          }
        }
    }

    }

    /* duplicateがoff(0)の時だけ、目盛を描画する */
    if( p->duplicate != DUPLICATE_ON ){
        /* x目盛りの描画 */
        if (p->xticks) {
            XSegment    seg[GR_TICK_MAX_ALL_TICKS];
            int         i, len;
            float       rx, dx;

#ifdef GRAPH_DEBUG
            fprintf(stderr, "  draw xticks\n");fflush(stderr);
#endif
            rx = p->xrange[1] - p->xrange[0];
            if (rx == 0) dx = 0;
            else dx = p->Ow/rx;

            /* Major目盛りの描画 */
            len = p->Ah*GR_TICK_LENGTH;
            if (len < 2) len = 2;
            for (i = 0;i < p->num_xticks;i++) {
                seg[i].x1 = seg[i].x2 = p->Ox + (p->xticks[i]-p->xrange[0])*dx;
                seg[i].y1 = p->Oy + p->Oh;
                seg[i].y2 = p->Oy + p->Oh - len;
            }
            XDrawSegments(p->disp, p->win, p->foregc, seg, p->num_xticks);

            /* Minor目盛りの描画 */
            len = p->Ah*GR_TICK_MINOR_LENGTH;
            if (len < 1) len = 1;
            if (p->num_xticks_minor > 0) {
                for (i = 0;i < p->num_xticks_minor;i++) {
                    seg[i].x1 = seg[i].x2 = p->Ox +
                        (p->xticks_minor[i]-p->xrange[0])*dx;
                    seg[i].y1 = p->Oy + p->Oh;
```

```
                    seg[i].y2 = p->Oy + p->Oh - len;
                }
                XDrawSegments(p->disp, p->win, p->foregc,
                              seg, p->num_xticks_minor);
            }

        }

        /* y目盛りの描画 */
        if (p->yticks) {
            XSegment    seg[GR_TICK_MAX_ALL_TICKS];
            int         i, len;
            float               ry, dy;
#ifdef GRAPH_DEBUG
            fprintf(stderr, "  draw yticks\n");fflush(stderr);
#endif

            ry = p->yrange[1] - p->yrange[0];
            if (ry == 0) dy = 0;
            else dy = p->Oh/ry;

            /* Major目盛りの描画 */
            len = p->Aw*GR_TICK_LENGTH;
            if (len < 2) len = 2;
            for (i = 0;i < p->num_yticks;i++) {
                seg[i].y1 = seg[i].y2 = p->Oy + p->Oh
                  - (p->yticks[i]-p->yrange[0])*dy;
                seg[i].x1 = p->Ox;
                seg[i].x2 = p->Ox + len;
            }
            XDrawSegments(p->disp, p->win, p->foregc, seg, p->num_yticks);

            /* Minor目盛りの描画 */
            len = p->Aw*GR_TICK_MINOR_LENGTH;
            if (len < 1) len = 1;
            if (p->num_yticks_minor > 0) {
              for (i = 0;i < p->num_yticks_minor;i++) {
                seg[i].y1 = seg[i].y2 = p->Oy + p->Oh
                  - (p->yticks_minor[i]-p->yrange[0])*dy;
                seg[i].x1 = p->Ox;
                seg[i].x2 = p->Ox + len;
              }
              XDrawSegments(p->disp, p->win, p->foregc, seg,
                            p->num_yticks_minor);
            }
        }
    }
    /* データの描画 */
    if (p->ydata) {
        XRectangle      rect[1];
        Number          x, y;
        GC              gc;
        int             line_width;
        /* データ描画用GC,線幅の設定 */
        gc = p->dot_gc;
        line_width = p->dot_size;
        /* データの相対座標から絶対座標への変換 */
        ps = (XPoint *)malloc(sizeof(XPoint)*(p->datacount));
        for (i = 0;i < p->datacount;i++) {
          x = (p->Ox + (p->xdata[i]-p->xrange[0])
                       /(p->xrange[1]-p->xrange[0])*p->Ow);
          y = (p->Oy + p->Oh - (p->ydata[i]-p->yrange[0])
                       /(p->yrange[1]-p->yrange[0])*p->Oh);
          ps[i].x = (int)(x+0.5);
          ps[i].y = (int)(y+0.5);
        }
#ifdef GRAPH_DEBUG_EX
        {
            int         i;
            fprintf(stderr, "  xrange = [%g %g], yrange = [%g %g]\n",
```

```
                    p->xrange[0], p->xrange[1],
                    p->yrange[0], p->yrange[1]);
            fprintf(stderr, "  data = [");
            for (i = 0;i < p->datacount;i++)
                fprintf(stderr, "(%g,%g;%d,%d)",
                        p->xdata[i], p->ydata[i],
                        ps[i].x, ps[i].y);
            fprintf(stderr, "]\n");
        }
#endif
        /* プロット領域をクリップする */
        rect[0].x = p->Ox-1;
        rect[0].y = p->Oy-1;
        rect[0].width = p->Ow+2;
        rect[0].height = p->Oh+2;
        XSetClipRectangles(p->disp, gc,
                           0, 0, rect, 1, Unsorted);
        /*
         * 線種による描きわけ
         */
        /* 直線 */
        if (!strcmp(p->line_style, GR_LINE_STYLE_LINE)) {
          XSetLineAttributes(p->disp, gc, line_width,
                             LineSolid, CapButt, JoinMiter);
          XDrawLines(p->disp, p->win, gc,
                     ps, p->datacount, 0);
          XSetLineAttributes(p->disp, gc, 0,
                             LineSolid, CapButt, JoinMiter);
        }
        /* 点 */
        else if (!strcmp(p->line_style, GR_LINE_STYLE_DOT)) {
          if (p->dot_size <= 1.0) {
            XDrawPoints(p->disp, p->win, gc,
                        ps, p->datacount, 0);
          }
          else {
            float d = p->dot_size/2;
            int i;
            for (i = 0;i < p->datacount;i++) {
              XFillArc(p->disp, p->win, gc,
                       (int)ps[i].x - d,
                       (int)ps[i].y - d,
                       (int)p->dot_size,
                       (int)p->dot_size,
                       0, 360*64);
            }
          }
        }
        /* 点線 */
        /* 点線をオプション指定に沿って描画する */
        else{
          XSetLineAttributes(p->disp, gc, line_width,
                             LineOnOffDash, CapButt, JoinMiter);

        /* 点線をオプション指定に沿って描画する */
          if(!strcmp(p->line_style, GR_LINE_STYLE_DASH)){
              XSetDashes(p->disp, gc, 0,
                     gr_dash_pattern_dash, GR_DASH_PATTERN_DASH_LEN);
          }
          else if(!strcmp(p->line_style, GR_LINE_STYLE_DOTTEDLINE)){
              XSetDashes(p->disp, gc, 0,
                     gr_dash_pattern_dottedline,
                         GR_DASH_PATTERN_DOTTEDLINE_LEN);
          }
          else if(!strcmp(p->line_style, GR_LINE_STYLE_DASH_LONG)){
              XSetDashes(p->disp, gc, 0,
                     gr_dash_pattern_dash_long, GR_DASH_PATTERN_DASH_LONG_LEN);
          }
          else if(!strcmp(p->line_style, GR_LINE_STYLE_DOTTEDLINE_LONG)){
              XSetDashes(p->disp, gc, 0,
```

```
                    gr_dash_pattern_dottedline_long,
                        GR_DASH_PATTERN_DOTTEDLINE_LEN);
          }
          XDrawLines(p->disp, p->win, gc,
                     ps, p->datacount, 0);
          XSetLineAttributes(p->disp, gc, 0,
                             LineSolid, CapButt, JoinMiter);
        }
        /* 各点にラベルを描く */
/*      if( p->labeldata_fp != NULL || p->labeldata_cp != NULL ){
            grDrawDotLabel( gr, ps );
        }*/
        XSetClipMask(p->disp, gc, (int)NULL);
        Free(ps);
    }

    XFlush(p->disp);
    /* 次のグラフが上書きに指定されているなら、それを呼び出す */
    p++;
    if( p->duplicate == DUPLICATE_ON ){
        GrDraw( gr+1 );
    }

    return(0);
}

int
grTextWidth(Display *display, Window window, GC gc,
            char string[], int length,
            unsigned int *textWidth, unsigned int *textHeight)
{
  Font font;
  XFontStruct *font_st;
  XCharStruct char_st;
  XGCValues gcval;
  int font_asc, font_dsc;
  int direction;

  /* 引数チェック */
  if (!string || !*string)
    return(-1);

  /* GC中のフォントの取得 */
  XGetGCValues(display, gc, GCFont | GCForeground | GCBackground, &gcval);

  /* フォント構造体の取得 */
  if ((font_st = XQueryFont(display, gcval.font)) == NULL)
    return(-1);
  XTextExtents(font_st, string, length,
               &direction, &font_asc, &font_dsc, &char_st);

  /* 文字幅の算出(by pixel) */
  *textWidth = XTextWidth(font_st, string, length);

  /* 文字高の算出(by pixel) */
  *textHeight = font_asc + font_dsc;
  return(0);
}

int
grDrawStringDownToUp(Display *display, Window window, GC gc,
                int x, int y, char string[], int length)
{
  Font font;
  XFontStruct *font_st;
  XCharStruct char_st;
  XGCValues gcval;
  int font_asc, font_dsc;
  int direction;
  unsigned int textWidth;
  unsigned int textHeight;
  XImage *src, *dst;
  Pixmap pix;
  int i, j;                         /* カウンタ */
  long pixel;
  unsigned int depth;
  int format;
  int bitmap_pad;

  /* 引数チェック */
  if (!string || !*string)
    return(-1);

  /* GC中のフォントの取得 */
  XGetGCValues(display, gc, GCFont | GCForeground | GCBackground, &gcval);

  /* フォント構造体の取得 */
  if ((font_st = XQueryFont(display, gcval.font)) == NULL)
    return(-1);
  XTextExtents(font_st, string, length,
               &direction, &font_asc, &font_dsc, &char_st);
  depth = DefaultDepth(display, DefaultScreen(display));

  /* 文字幅の算出(by pixel) */
  textWidth = XTextWidth(font_st, string, length);

  /* 文字高の算出(by pixel) */
  textHeight = font_asc + font_dsc;

  /* 文字列描画用のピクスマップ作成 */
  pix = XCreatePixmap(display, window, textWidth, textHeight, depth);
/*  XSetForeground(display, gc, gcval.background);*/
  XSetForeground(display, gc, WhitePixel(display, 0));
  XSetBackground(display, gc, gcval.foreground);
  XFillRectangle(display, pix, gc, 0, 0, textWidth, textHeight);
  XSetForeground(display, gc, gcval.foreground);
  XSetBackground(display, gc, gcval.background);

  /* 文字列の描画 */
  XDrawString(display, pix, gc, 0, font_asc, string, length);

  /* 文字列描画のピクスマップのコピー */
  format = ZPixmap;
  bitmap_pad = 8;
  src = XGetImage(display, pix, 0, 0, textWidth, textHeight,
                  1, format);
  if (src == NULL) return(-1);

  /* 回転先のイメージ構造体の作成 */
  dst = XCreateImage(display, DefaultVisual(display, DefaultScreen(display)),
                     depth, format, 0, NULL, textHeight, textWidth,
                     bitmap_pad, 0);
  if (dst == NULL) {
    XDestroyImage(src);
    return(-1);
  }
  dst->data = (unsigned char *)malloc(dst->bytes_per_line * textWidth*4);
  if (dst->data == NULL) {
    XDestroyImage(src);
    XFree(dst);
    return(-1);
  }
  /* 文字列の回転 */
  for (i = 0;i < textHeight;i++) {
    for (j = 0;j < textWidth;j++) {
      pixel = XGetPixel(src, j, i);
      XPutPixel(dst, i, textWidth - j - 1, pixel);
    }
  }
```

```
  /* ウィンドウへ描画 */
  XPutImage(display, window, gc, dst, 0, 0,
            x - font_asc, y - textWidth, textHeight, textWidth);

  XFreePixmap(display, pix);
  XDestroyImage(src);
  XFree(dst->data);
  return(0);
}

int
grDrawMeter(gr)
int     gr;
{
    GrAttr      *p;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grDrawMeter(%d)\n", gr);fflush(stderr);
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph+gr;
    /* チェック */
    if (p->width <= 0 ||
        p->height <= 0 ||
        p->datacount < 0)
        return(-2);

    /* 座標属性の計算 */
    if (p->hold == FALSE) {
        grCalcAxisAttributes(gr);
    }

    /* 描画領域の計算 */
    grCalcAxisOrigin(gr);
    /* duplicateがoff(0)の時だけ、グラフ描画領域をクリアする */
    if( p->duplicate == DUPLICATE_OFF ){
        grClearGraph(gr);
    }

    /* 装飾の描画 */
    if (p->hold == FALSE) {
        if (p->decoration_color) {
            grDrawDecoration(gr);
        }
        else {
            /* duplicateがoff(0)の時だけ、ウィンドウをクリアする */
            if( p->duplicate == DUPLICATE_OFF ){
                GrClear(gr);
            }
        }
    }

    /* ボックス/座標軸の描画 */
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  draw box/axis\n");fflush(stderr);
#endif
    /* y目盛り文字列の描画 */
    if (p->num_ytick_marks > 0) {
        int             i, x1, y1, x, y;
        float           ry, dy, ra, Oa;
        int             Ox, Oy, Oh, L;

        ry = p->yrange[1] - p->yrange[0];
        if (ry == 0) dy = 0;
        else dy = p->Oh/ry;

        Oa = atan((float)p->Oh/p->Ow*2);
        ra = M_PI - 2*Oa;
        Ox = p->Ox + p->Ow/2;
        Oy = p->Oy + p->Oh;
        Oh = p->Oh;

        for (i = 0;i < p->num_ytick_marks;i++) {
            dy = (p->yticks[i]-p->yrange[0])/ry;
            grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                              (float)Oh*GR_POS_METER_TICK_MARKS, &x1, &y1);
            grGetCenterPos(gr, p->ytick_marks[i], x1, y1, 0, 0, &x, &y);
            XDrawString(p->disp, p->win, p->tick_gc,
                        x, y, p->ytick_marks[i],
                        strlen(p->ytick_marks[i]));
        }
    }

    /* タイトル文字列の描画 */
    if (p->hold == FALSE) {
        if (p->title && *p->title) {
            int x, y;
            unsigned int        w, h;

            w = p->Aw;
            h = p->Ry;
            grGetCenterPos(gr, p->title, p->Ax, p->Ay, w, h, &x, &y);
            XDrawString(p->disp, p->win, p->title_gc,
                        x, y, p->title, strlen(p->title));
        }

        /* xラベル文字列の描画 */
        if (p->xlabel && *p->xlabel) {
        }
    }

    /* x目盛りの描画 */
    if (p->xticks) {
    }

    /* y目盛りの描画 */
    if (p->yticks) {
        XSegment        seg[GR_TICK_MAX_ALL_TICKS];
        int             i, len;
        float           ry, dy, ra, Oa;
        int             Ox, Oy, Oh, L, x, y;
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  draw yticks\n");fflush(stderr);
#endif

        ry = p->yrange[1] - p->yrange[0];
        if (ry == 0) dy = 0;
        else dy = p->Oh/ry;

        Oa = atan((float)p->Oh/p->Ow*2);
        ra = M_PI - 2*Oa;
        Ox = p->Ox + p->Ow/2;
        Oy = p->Oy + p->Oh;
        Oh = p->Oh;

        /* Major目盛りの描画 */
        len = p->Aw*GR_TICK_LENGTH;
        if (len < 2) len = 2;

        for (i = 0;i < p->num_yticks;i++) {
            dy = (p->yticks[i]-p->yrange[0])/ry;
            grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                              (float)Oh*GR_POS_METER_MAJOR_TICKS1, &x, &y);
            seg[i].x1 = x;
            seg[i].y1 = y;
            grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                              (float)Oh*GR_POS_METER_MAJOR_TICKS2, &x, &y);
            seg[i].x2 = x;
```

```
            seg[i].y2 = y;
        }
        for (i = 0;i < p->num_yticks;i++) {
            seg[i].x1 -= 1;
            seg[i].x2 -= 1;
        }
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[1]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, p->num_yticks);
        for (i = 0;i < p->num_yticks;i++) {
            seg[i].x1 += 2;
            seg[i].x2 += 2;
        }
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[2]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, p->num_yticks);
        for (i = 0;i < p->num_yticks;i++) {
            seg[i].x1 -= 1;
            seg[i].x2 -= 1;
        }
        XDrawSegments(p->disp, p->win, p->foregc, seg, p->num_yticks);


        /* Minor目盛りの描画 */
        len = p->Aw*GR_TICK_MINOR_LENGTH;
        if (len < 1) len = 1;
        for (i = 0;i < p->num_yticks_minor;i++) {
            dy = (p->yticks_minor[i]-p->yrange[0])/ry;
            grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                              (float)Oh*GR_POS_METER_MINOR_TICKS1, &x, &y);
            seg[i].x1 = x;
            seg[i].y1 = y;
            grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                              (float)Oh*GR_POS_METER_MINOR_TICKS2, &x, &y);
            seg[i].x2 = x;
            seg[i].y2 = y;
        }
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[0]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, p->num_yticks_minor);
        for (i = 0;i < p->num_yticks_minor;i++) {
            seg[i].x1 -= 1;
            seg[i].x2 -= 1;
        }
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[1]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, p->num_yticks_minor);
        for (i = 0;i < p->num_yticks_minor;i++) {
            seg[i].x1 += 2;
            seg[i].x2 += 2;
        }
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[2]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, p->num_yticks_minor);
    }

    /* データの描画 */
    if (p->ydata) {
        XSegment        seg[1];
        int             i, len;
        float           ry, dy, ra, Oa;
        int             Ox, Oy, Oh, L, x, y;
        float           yd;
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  draw ydata\n");fflush(stderr);
#endif
        /* データの平均を求める */
        yd = 0;
        for (i = 0;i < p->datacount;i++) {
            yd += p->ydata[i]/p->datacount;
        }

        ry = p->yrange[1] - p->yrange[0];
        if (ry == 0) dy = 0;
        else dy = p->Oh/ry;
        Oa = atan((float)p->Oh/p->Ow*2);
        ra = M_PI - 2*Oa;
        Ox = p->Ox + p->Ow/2;
        Oy = p->Oy + p->Oh;
        Oh = p->Oh;

        len = p->Aw*GR_TICK_LENGTH;
        if (len < 2) len = 2;

        dy = (yd-p->yrange[0])/ry;

        /* 範囲チェック */
        if (dy > 1.0)    dy = 1.1;
        else if (dy < 0) dy = -0.1;
        grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                          (float)Oh*GR_POS_METER_STING1, &x, &y);
        seg[0].x1 = x;
        seg[0].y1 = y;
        grGetPointByAngle(Ox, Oy, (float)Oa+ra*dy,
                          (float)Oh*GR_POS_METER_STING2, &x, &y);
        seg[0].x2 = x;
        seg[0].y2 = y;
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[0]);
        XDrawSegments(p->disp, p->win, p->foregc, seg, 1);

        seg[0].x1 -= 1;
        seg[0].x2 -= 1;
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[1]);
        XDrawSegments(p->disp, p->win, p->decogc, seg, 1);

        seg[0].x1 += 2;
        seg[0].x2 += 2;
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[2]);
        XDrawSegments(p->disp, p->win, p->foregc, seg, 1);
    }

    XFlush(p->disp);

    return(0);
}

int
grGetPointByAngle(x, y, a, h, xx, yy)
int     x, y;
float   a;
float   h;
int     *xx, *yy;
{
    float       w;

    w = h/tan(a);
    *xx = x - w;
    *yy = y - h;
    return(0);
}

int
grClearGraph(gr)
int     gr;
{
    GrAttr      *p;
    XPoint      ps[4];
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grClearGraph(%d)\n", gr);fflush(stderr);
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);
```

```
    p = gGraph+gr;
    ps[0].x = ps[3].x = p->Ox;
    ps[0].y = ps[1].y = p->Oy;
    ps[1].x = ps[2].x = p->Ox+p->Ow;
    ps[2].y = ps[3].y = p->Oy+p->Oh;

    XFillPolygon(p->disp, p->win, p->backgc,
                 ps, 4, Convex, CoordModeOrigin);
    return(0);
}

int
GrClear(gr)
int     gr;
{
    GrAttr      *p;
    XPoint      ps[4];
    unsigned int        width, height;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "GrClear(%d)\n", gr);fflush(stderr);
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph+gr;
    if (p->width <= 0 ||
        p->height <= 0)
        return(-2);

    grGetWindowSize(gr, &width, &height);
    ps[0].x = ps[3].x = 0;
    ps[0].y = ps[1].y = 0;
    ps[1].x = ps[2].x = width;
    ps[2].y = ps[3].y = height;
    XFillPolygon(p->disp, p->win, p->backgc,
                 ps, 4, Convex, CoordModeOrigin);
    return(0);
}

int
GrClose(gr)
int     gr;
{
    GrAttr      *p;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "GrClose(%d)\n", gr);fflush(stderr);
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || gGraph[gr].on)
        return(-1);

    p = gGraph+gr;
    p->on = 0;
    Free(p->xdata);
    Free(p->ydata);
    XFreeGC(p->disp, p->foregc);
    XFreeGC(p->disp, p->backgc);
    Free(p->fontname);
    Free(p->fontbold);
    Free(p->fontitalic);
    Free(p->title);
    Free(p->xticks);
    Free(p->yticks);
    Free(p->xlabel);
    Free(p->ylabel);
    if (p->xtick_marks) {
        int     i;
        for(i = 0;i < p->num_xtick_marks;i++)
            Free(p->xtick_marks[i]);
        Free(p->xtick_marks);
    }
    if (p->ytick_marks) {
        int     i;
        for(i = 0;i < p->num_ytick_marks;i++)
            Free(p->ytick_marks[i]);
        Free(p->ytick_marks);
    }
    Free(p->xticks_minor);
    Free(p->yticks_minor);
    Free(p->xtick_scale);
    Free(p->ytick_scale);
    Free(p->decoration_color);

    /* タイトルフォントの解放 */
    Free(p->title_fontname);
    Free(p->title_fontbold);
    Free(p->title_fontitalic);
    if (p->title_font)
        XFreeFont(p->disp, p->title_font);
    Free(p->title_color);
    if (p->title_gc)
        XFreeGC(p->disp, p->title_gc);

    /* ラベルフォントの解放 */
    Free(p->label_fontname);
    Free(p->label_fontbold);
    Free(p->label_fontitalic);
    if (p->label_font)
        XFreeFont(p->disp, p->label_font);
    Free(p->label_color);
    if (p->label_gc)
        XFreeGC(p->disp, p->label_gc);

    /* 目盛りフォントの解放 */
    Free(p->tick_fontname);
    Free(p->tick_fontbold);
    Free(p->tick_fontitalic);
    if (p->tick_font)
        XFreeFont(p->disp, p->tick_font);
    Free(p->tick_color);
    if (p->tick_gc)
        XFreeGC(p->disp, p->tick_gc);

    /* ラベルデータ描画用フォント、点、線描画用GCの解放 */
    Free(p->dotlabel_fontname);
    Free(p->dotlabel_fontbold);
    Free(p->dotlabel_fontitalic);
    if (p->dotlabel_font)
        XFreeFont(p->disp, p->dotlabel_font);
    Free(p->dotlabel_color);
    if (p->dotlabel_gc)
        XFreeGC(p->disp, p->dotlabel_gc);
    if (p->dot_gc)
        XFreeGC(p->disp, p->dot_gc);
    /* ラベルデータの解放 */
    XFree(p->labeldata_fp);
    XFree(p->labeldata_cp);
    return(0);
}

int
grDrawDecoration(gr)
int     gr;
{
    GrAttr      *p;
    XPoint      ps[6];
    int         pt[4];
    int         W, N;
```

```
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grDrawDecoration(%d)\n", gr);fflush(stderr);
#endif
    p = gGraph+gr;
    if (p->graph_type == GR_TYPE_METER) {
        W = 3;
        N = 3;

        /* 原色 */
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[0]);
        XFillRectangle(p->disp, p->win, p->decogc,
                       p->Ax, p->Ay, p->Aw, p->Ah);

        /* 内枠 */
        pt[0] = pt[1] = W;
        pt[2] = pt[3] = N;
        grDrawFrame(p->disp, p->win, p->decogc,
                    p->decoration_pixel[1],
                    p->decoration_pixel[2],
                    FALSE,
                    p->Rx, p->Ry, p->Rw, p->Rh, pt);

        /* 外枠 */
        N = 1;
        pt[0] = pt[1] = N;
        pt[2] = pt[3] = N;
        grDrawFrame(p->disp, p->win, p->decogc,
                    p->decoration_pixel[1],
                    p->decoration_pixel[2],
                    TRUE,
                    p->Ax, p->Ay, p->Aw, p->Ah, pt);

        XFillRectangle(p->disp, p->win, p->backgc,
                       p->Rx, p->Ry, p->Rw, p->Rh);
        return(0);
    }

    switch (p->decoration_type) {
    case 0:
        break;
    case 1:
        W = 4;
        N = 2;

        /* 原色 */
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[0]);
        XFillRectangle(p->disp, p->win, p->decogc,
                       p->Ax, p->Ay, p->Aw, p->Ah);

        /* 内枠 */
        pt[0] = pt[1] = W;
        pt[2] = pt[3] = N;
        grDrawFrame(p->disp, p->win, p->decogc,
                    p->decoration_pixel[1],
                    p->decoration_pixel[2],
                    FALSE,
                    p->Ox, p->Oy, p->Ow, p->Oh, pt);

        /* 外枠 */
        N = 1;
        pt[0] = pt[1] = N;
        pt[2] = pt[3] = N;
        grDrawFrame(p->disp, p->win, p->decogc,
                    p->decoration_pixel[1],
                    p->decoration_pixel[2],
                    TRUE,
                    p->Ax, p->Ay, p->Aw, p->Ah, pt);

        XFillRectangle(p->disp, p->win, p->backgc,
                       p->Ox, p->Oy, p->Ow, p->Oh);
        break;

    case 2:
        /* 原色 */
        XSetForeground(p->disp, p->decogc, p->decoration_pixel[0]);
        XFillRectangle(p->disp, p->win, p->decogc,
                       p->Ax, p->Ay, p->Aw, p->Ah);

        /* 内枠 */
        pt[0] = pt[1] = N;
        pt[2] = pt[3] = N;
        grDrawFrame(p->disp, p->win, p->decogc,
                    p->decoration_pixel[1],
                    p->decoration_pixel[2],
                    FALSE,
                    p->Rx, p->Ry, p->Rw, p->Rh, pt);

        XFillRectangle(p->disp, p->win, p->backgc,
                       p->Rx, p->Ry, p->Rw, p->Rh);
        break;
    }
    grClearGraph(gr);

    return(0);
}

int
grDrawFrame(disp, win, gc, high, low, outer, x, y, w, h, pt)
Display *disp;
Window  win;
GC      gc;
unsigned long   high, low;
int     outer;
int     x, y, w, h;
int     pt[4];
{
    XPoint      ps[6];

    if (outer == TRUE) {
        /* 暗色 */
        ps[0].x = x+w; ps[0].y = y+h;
        ps[1].x = 0; ps[1].y = -h;
        ps[2].x = -pt[2]; ps[2].y = pt[3];
        ps[3].x = 0; ps[3].y = h-pt[3]-pt[1];
        ps[4].x = -(w-pt[0]-pt[2]); ps[4].y = 0;
        ps[5].x = -pt[0]; ps[5].y = pt[1];
        XSetForeground(disp, gc, low);
        XFillPolygon(disp, win, gc,
                     ps, 6, Complex, CoordModePrevious);

        /* 明色 */
        ps[0].x = 0; ps[0].y = 0;
        ps[1].x = 0; ps[1].y = h;
        ps[2].x = pt[0]; ps[2].y = -pt[1];
        ps[3].x = 0; ps[3].y = -(h-pt[3]-pt[1]);
        ps[4].x = w-pt[0]-pt[2]; ps[4].y = 0;
        ps[5].x = pt[2]; ps[5].y = -pt[3];
        XSetForeground(disp, gc, high);
        XFillPolygon(disp, win, gc,
                     ps, 6, Complex, CoordModePrevious);
    }
    else {
        /* 暗色 */
        ps[0].x = x; ps[0].y = y;
        ps[1].x = 0; ps[1].y = h;
        ps[2].x = -pt[0]; ps[2].y = pt[1];
        ps[3].x = 0; ps[3].y = -(h+pt[1]+pt[3]);
        ps[4].x = w+pt[0]+pt[2]; ps[4].y = 0;
        ps[5].x = -pt[2]; ps[5].y = pt[3];
        XSetForeground(disp, gc, low);
```

```
        XFillPolygon(disp, win, gc,
                     ps, 6, Complex, CoordModePrevious);

        /* 明色 */
        ps[0].x = x+w; ps[0].y = y+h;
        ps[1].x = 0; ps[1].y = -h;
        ps[2].x = pt[2]; ps[2].y = -pt[3];
        ps[3].x = 0; ps[3].y = h+pt[1]+pt[3];
        ps[4].x = -(w+pt[0]+pt[2]); ps[4].y = 0;
        ps[5].x = pt[0]; ps[5].y = -pt[1];
        XSetForeground(disp, gc, high);
        XFillPolygon(disp, win, gc,
                     ps, 6, Complex, CoordModePrevious);
    }
    return(0);
}

int
grMakeTickMarks(ticks, num_ticks, marks, num_marks, tick_scale, max_exp)
float   *ticks;
int     num_ticks;
char    ***marks;
int     *num_marks;
char    **tick_scale;
int     max_exp;
{
    char        **strs;
    int         i, n;
    float       x, x1, x2, ex, exp_val;
    char        str[40];
    char        *fmt;
    char        *p;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grMakeTickMarks()\n");fflush(stderr);
#endif

    if (num_ticks <= 0) {
        return(0);
    }

    x1 = (ticks[0] > 0 ? ticks[0]: -ticks[0]);
    x2 = (ticks[num_ticks-1] > 0 ? ticks[num_ticks-1]: -ticks[num_ticks-1]);
    x = (x1 < x2 ? x2: x1);
    ex = 1;
    strs = (char **)malloc(sizeof(char *)*num_ticks);

    fmt = "%3.1e";
    sprintf(str, fmt, x);
    p = strrchr(str, 'e');
    if (p) {
        *p++ = '\0';
        n = atoi(p);
        if (max_exp < n || n < -max_exp) {
            ex = exp10(n);
            sprintf(str, "10^%d", n);
            *tick_scale = strdup(str);
        }
#ifdef GRAPH_DEBUG
            fprintf(stderr, "  x1=%g,x2=%g,x=%g,p=%s,n=%d, ex=%g, scale=%s\n",
                    x1, x2, x, p, n, ex, *tick_scale ? *tick_scale: "null");
#endif
    }

    fmt = "%g";
    for (i = 0;i < num_ticks;i++) {
        sprintf(str, fmt, ticks[i]/ex);
        strs[i] = strdup(str);
    }
    *num_marks = num_ticks;
    *marks = strs;
```

```
#ifdef GRAPH_DEBUG
    {
        int     i;
        fprintf(stderr, "  tick marks = [");
        for (i = 0;i < *num_marks;i++)
            fprintf(stderr, "\"%s\" ", (*marks)[i]);
        fprintf(stderr, "]\n");fflush(stderr);
    }
#endif
    return(0);
}

int
grMakeXtickMarks(gr)
int     gr;
{
    GrAttr      *p;
    int         i, x1, y1, x, y;
    float       rx, dx;
    char        str[40];
    char        *fmt;
    double      max_value;
#ifdef GRAPH_DEBUG
    frintf(stderr, "grMakeXtickMarks(%d)\n", gr);fflush(stderr);
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph+gr;

    /* x軸の目盛り文字列 */
    if (p->num_xticks > 0 && !p->xticks) {
        fprintf(stderr, "Error: fatal in grMakeXtickMarks(%d)\n",
                p->num_xticks);
        return(-1);
    }
    if (p->num_xticks > 0) {
      max_value = pow(10.0, p->max_exp);
        p->xtick_marks = (char **)malloc(sizeof(char *)*p->num_xticks);
/*      if (p->xrange[0] <= -10000.0 || p->xrange[1] >= 10000.0)*/
        if (p->xrange[0] <= -max_value || p->xrange[1] >= max_value)
            fmt = "%3.1e";
        else
            fmt = "%g";
        for (i = 0;i < p->num_xticks;i++) {
            sprintf(str, fmt, p->xticks[i]);
            p->xtick_marks[i] = strdup(str);
        }
    }
    p->num_xtick_marks = p->num_xticks;

#ifdef GRAPH_DEBUG
    {
        int     i;
        frintf(stderr, "  xtick_marks = [");
        for (i = 0;i < p->num_xtick_marks;i++)
            fprintf(stderr, "\"%s\" ", p->xtick_marks[i]);
        fprintf(stderr, "]\n");fflush(stderr);
    }
#endif
    return(0);
}

int
grMakeYtickMarks(gr)
int     gr;
{
    GrAttr      *p;
```

```
    int         i, x1, y1, x, y;
    float       rx, dx;
    char        str[40];
    char        *fmt;
    double      max_value;

#ifdef GRAPH_DEBUG
    fprintf(stderr, "grMakeYtickMarks(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph+gr;

    /* y軸の目盛り文字列 */
    if (p->num_yticks > 0 && !p->yticks) {
        fprintf(stderr, "Error: fatal in grMakeYtickMarks (%d)\n",
                p->num_yticks);
        return(-1);
    }
    if (p->num_yticks > 0) {
      max_value = pow(10.0, p->max_exp);
        p->ytick_marks = (char **)malloc(sizeof(char *)*p->num_yticks);
        if (p->yrange[0] <= -max_value || p->yrange[1] >= max_value)
            fmt = "%3.1e";
        else
            fmt = "%g";
        for (i = 0;i < p->num_yticks;i++) {
            sprintf(str, fmt, p->yticks[i]);
            p->ytick_marks[i] = strdup(str);
        }
    }
    p->num_ytick_marks = p->num_yticks;

    return(0);
}

int
grGetWindowSize(gr, width, height)
int             gr;
unsigned int    *width;
unsigned int    *height;
{
    Window              root;
    unsigned int        x, y, border, depth;

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    XGetGeometry(gGraph[gr].disp, gGraph[gr].win, &root,
                 &x, &y, width, height, &border, &depth);
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grGetWindowSize(%d,%d,%d)\n",
            gr, *width, *height);fflush(stderr);
#endif
    return(0);
}

int
grGetCenterPos(gr, str, x, y, w, h, xx, yy)
int     gr;
char    *str;
int     x, y;
int     w, h;
int     *xx, *yy;
{
    XCharStruct cs;
    int         dir, ascent, decent;
    int width, height;
```

```
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grGetCenterPos(%d,\"%s\",%d,%d,%d,%d,...)\n",
            gr, str, x, y, w, h);fflush(stderr);
#endif

    XTextExtents(gGraph[gr].font, str, strlen(str), &dir,
                 &ascent, &decent, &cs);
    width = cs.rbearing - cs.lbearing;
    height = cs.ascent + cs.descent;

    /* 右寄せ */
    if (w < 0)
        *xx = x - (width - cs.lbearing);
    else
        *xx = x + (w - width)/2 + cs.lbearing;
    *yy = y + (h - height)/2 + cs.ascent;
#ifdef GRAPH_DEBUG
    fprintf(stderr, " end of grGetCenterPos()\n");fflush(stderr);
#endif
    return(0);
}

int
grGetCenterPosTitle(gr, str, x, y, w, h, xx, yy)
int     gr;
char    *str;
int     x, y;
int     w, h;
int     *xx, *yy;
{
    XCharStruct cs;
    int         dir, ascent, decent;
    unsigned int        width, height;

#ifdef GRAPH_DEBUG
    fprintf(stderr, "grGetCenterPosByGC(%d,\"%s\",%d,%d,%d,%d,...)\n",
            gr, str, x, y, w, h);fflush(stderr);
#endif
    XTextExtents(gGraph[gr].title_font, str, strlen(str), &dir,
                 &ascent, &decent, &cs);
    width = cs.rbearing - cs.lbearing;
    height = cs.ascent + cs.descent;

    /* 右寄せ */
    if (w < 0)
        *xx = x - (width - cs.lbearing);
    else
        *xx = x + (w - width)/2 + cs.lbearing;
    *yy = y + (h - height)/2 + cs.ascent;
#ifdef GRAPH_DEBUG
    fprintf(stderr, " end of grGetCenterPos()\n");fflush(stderr);
#endif
    return(0);
}

int
grMakeNormalTicks(range, ticks, n, Minor, m)
float   range[2];
float   **ticks;
int     *n;
float   **Minor;
int     *m;
{
    float       w;
    float       tks[GR_TICK_MAX_ALL_TICKS];
    int         i, len;
    int         DIV, N, A, M;
    float       T;
    int         div, c;
    float       t;
```

```
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grMakeNormalTicks([%g %g],...)\n",
            range[0], range[1]);fflush(stderr);
#endif
    if (!range || range[0] >= range[1]) {
        *ticks = NULL;
        *n = 0;
        *Minor = NULL;
        *m = 0;
        return(-1);
    }

    /*
     * 値の範囲をGR_TICK_NUM_TICKSで割った商の上１桁の数Aと桁数Nを求める。
     *      A == 1 -> M = 1, c=2
     *      A == 2 -> M = 2, c=2
     *      A <= 5 -> M = 5, c=5
     *      A <= 9 -> M = 10,c=2
     * 上記のMとNより、目盛り間隔Tは
     *      T = M*10^(N-1)
     * で算出する。
     */
    w = (range[1]-range[0])/GR_TICK_NUM_TICKS;
    N = floor(log10(w));
/*    A = floor((double)(w/exp10((double)N)));*/
    A = ceil((double)(w/exp10((double)N)));
    if      (A == 1) M = 1, c = 2;
    else if (A == 2) M = 2, c = 2;
    else if (A <= 5) M = 5, c = 5;
    else             M = 10,c = 2;
    T = M*exp10((double)N);
    DIV = ceil(range[0]/T);

#ifdef GRAPH_DEBUG
    fprintf(stderr, "  w=%g,N=%d, A=%d, M=%d, T=%g, div=%d\n",
            w, N, A, M, T, DIV);
#endif
    for (i = 0;i < GR_TICK_MAX_TICKS;i++) {
        if (range[1] < T*(DIV+i))
            break;
        tks[i] = T*(DIV+i);
    }
    *n = i;
    len = sizeof(float)*(*n);
    if ((*ticks = (float *)malloc(len)) == NULL) {
      return(-1);
    }
    memcpy(*ticks, tks, len);

    /* Minor目盛りの算出 */
    t = T/c;
    div = ceil(range[0]/t);
    for (i = 0;i < GR_TICK_MAX_ALL_TICKS;i++) {
        if (range[1] < t*(div+i))
            break;
        tks[i] = t*(div+i);
    }
    *m = i;
    len = sizeof(float)*(*m);
    if ((*Minor = (float *)malloc(len)) == NULL) {
      return(-1);
    }
    memcpy(*Minor, tks, len);

    return(0);
}

int
grCalcAxisOrigin(gr)
int     gr;
{
    GrAttr      *p;
    static int          i = 0;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grCalcAxisOrigin(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    /* ウィンドウ領域の取得 */
    p = gGraph+gr;

    /*
     * PS出力モードのため以下の関数は使わない
     * grGetWindowSize(gr, &(p->Ww), &(p->Wh));
     */

    /* メータ型のグラフ */
    if (p->graph_type == GR_TYPE_METER) {
        /* 描画領域の算出 */
        p->Ax = 0;
        p->Ay = 0;
        p->Aw = p->Ww;
        p->Ah = p->Wh;

        /* グラフ領域の算出 */
        p->Rx = p->Aw*p->x;
        p->Ry = p->Ah*p->y;
        p->Rw = p->Aw*p->width;
        p->Rh = p->Ah*p->height;

        /* プロット領域の算出 */
        p->Ox = p->Rx+p->Aw*GR_POS_METER_WIDTH_MARGIN/2;
        p->Oy = p->Ry+p->Ah*GR_POS_METER_HEIGHT_MARGIN;
        p->Ow = p->Rw-p->Aw*GR_POS_METER_WIDTH_MARGIN;
        p->Oh = p->Rh-p->Ah*GR_POS_METER_HEIGHT_MARGIN;
    }

    /* その他通常のグラフ */
    else {
        /* 描画領域の算出 */
        p->Ax = p->x*p->Ww;
        p->Ay = p->y*p->Wh;
        p->Aw = p->width*p->Ww;
        p->Ah = p->height*p->Wh;

        /* グラフ領域の算出 */
        p->Rx = p->Ax + GR_POS_GRAPH_AREA_LEFT_MARGIN;
        p->Ry = p->Ay + GR_POS_GRAPH_AREA_TOP_MARGIN;
        p->Rw = p->Aw - (GR_POS_GRAPH_AREA_LEFT_MARGIN + GR_POS_GRAPH_AREA_RIGHT_MARGIN);
        p->Rh = p->Ah - (GR_POS_GRAPH_AREA_TOP_MARGIN + GR_POS_GRAPH_AREA_BOTTOM_MARGIN);

        /* プロット領域の算出 */
        p->Ox = p->Rx + GR_POS_PLOT_AREA_LEFT_MARGIN;
        p->Oy = p->Ry + GR_POS_PLOT_AREA_TOP_MARGIN;
        p->Ow = p->Rw - (GR_POS_PLOT_AREA_LEFT_MARGIN + GR_POS_PLOT_AREA_RIGHT_MARGIN);
        p->Oh = p->Rh - (GR_POS_PLOT_AREA_TOP_MARGIN + GR_POS_PLOT_AREA_BOTTOM_MARGIN);

        /* 縦横比が指定されていた場合 */
        if (p->aspect_ratio > 0) {
          float width, height, margin, pitch;

          width  = (float)p->Oh * p->aspect_ratio;
          height = (float)p->Ow / p->aspect_ratio;

          /* 幅を変更する場合 */
          if (width < p->Ow) {
            pitch = (float)p->Ow - width;
            margin = pitch/2.0;
```

```
            p->Rx += (int)margin;
            p->Rw -= (int)pitch;
            p->Ax += (int)margin;
            p->Aw -= (int)pitch;
            p->Ox += (int)margin;
            p->Ow -= (int)pitch;
          }
          /* 高さを変更する場合 */
          else if (height < p->Oh) {
            pitch = (float)p->Oh - height;
            margin = pitch/2.0;
            p->Ry += (int)margin;
            p->Rh -= (int)pitch;
            p->Ay += (int)margin;
            p->Ah -= (int)pitch;
            p->Oy += (int)margin;
            p->Oh -= (int)pitch;
          }
        }
    }

    /* グラフ領域の控除 */
/*    if (p->title) {
        p->Oy += (int)(p->Wh*GR_POS_TITLE_HEIGHT);
        p->Oh -= (int)(p->Wh*GR_POS_TITLE_HEIGHT);
    }
    if (p->xlabel) {
        p->Oh -= (int)(p->Wh*GR_POS_XLABEL_HEIGHT);
    }
    if (p->ylabel) {
        p->Ox += (int)(p->Ww*GR_POS_YLABEL_WIDTH);
        p->Ow -= (int)(p->Ww*GR_POS_YLABEL_WIDTH);
    }*/

    return(0);
}

grGetPixelValueByName(disp, name, color)
Display *disp;
char    *name;
unsigned long *color;
{
    Colormap    cmap = DefaultColormap(disp, 0);
    XColor      c0, c1;

    if (!disp || !name || !*name)
        return(-1);
    XAllocNamedColor(disp, cmap, name, &c1, &c0);
    *color = c1.pixel;
    return(0);
}

int
grAllocHighLowColor(disp, name, pixels)
Display         *disp;
char            *name;
unsigned long   *pixels;
{
    XColor      c;
    Colormap    cmap = DefaultColormap(disp, 0);
    float       h, s, v;
    float       r, g, b;
    float       R = 65535;
    unsigned long plane_mask[8];

#ifdef GRAPH_DEBUG
    fprintf(stderr, "grAllocHighLowColor(%x,%s,)\n", disp,name);fflush(stderr);
#endif
    /* 3つのカラーセルの割り当て */
    if (XAllocColorCells(disp, cmap, FALSE, plane_mask, 0,
                         pixels, 3) == 0) {
        fprintf(stderr, "Error: Can't allocate color cells\n");
        return(-1);
    }

    /* 原色のストア */
    if (XParseColor(disp, cmap, name, &c) == 0) {
        fprintf(stderr, "Error: Can't find color name \"%s\"\n", name);
        return(-2);
    }
    c.pixel = pixels[0];
    XStoreColor(disp, cmap, &c);
    rgb_to_hsv(c.red/R, c.green/R, c.blue/R, &h, &s, &v);
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  source RGB=[%d,%d,%d]\n",
            c.red, c.green, c.blue);
    fprintf(stderr, "  source hsv=[%g,%g,%g]\n",
            h, s, v);fflush(stderr);
#endif

    /* high色のストア */
    hsv_to_rgb(h, s, v+GR_COLOR_HIGH, &r, &g, &b);
    c.red = r*R;
    c.green = g*R;
    c.blue = b*R;
    c.flags = DoRed | DoGreen | DoBlue;
    c.pixel = pixels[1];
    XStoreColor(disp, cmap, &c);
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  high RGB=[%d,%d,%d]\n",
            c.red, c.green, c.blue);fflush(stderr);
#endif

    /* low色のストア */
    hsv_to_rgb(h, s, v+GR_COLOR_LOW, &r, &g, &b);
    c.red = r*R;
    c.green = g*R;
    c.blue = b*R;
    c.flags = DoRed | DoGreen | DoBlue;
    c.pixel = pixels[2];
    XStoreColor(disp, cmap, &c);
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  low RGB=[%d,%d,%d]\n",
            c.red, c.green, c.blue);fflush(stderr);
#endif

    return(0);
}

int
grLoadFontByType(disp, name, bold, size, italic, fs)
Display *disp;
char    *name;
char    *bold;
int     size;
char    *italic;
XFontStruct **fs;
{
    char        buf[BUFSIZ];
    char        it;
    static char *fns[] = GR_FONT_NAME_RANGE, **p;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "grLoadFontByType(,%s,%s,%d,%s,,)\n",
            name ? name: "NULL",
            bold ? bold: "NULL",
            size,
            italic ? italic: "NULL");fflush(stderr);
#endif

    /* フォント名のチェック */
```

```
    if (!name)
        name = GR_DEFAULT_FONTNAME;
    else {
        p = fns;
        while (*p) {
            if (!strcmp(name, *p))
                goto next;
            p++;
        }
        fprintf(stderr, "WARNING: Can't find font name \"%s\"\n", name);
        name = GR_DEFAULT_FONTNAME;
    }
 next:

    /* フォントタイプのチェック */
    if (!bold)
        bold = GR_DEFAULT_FONTBOLD;

    /* サイズのチェック */
    if      (size <=  8) size =  8;
    else if (size <= 10) size = 10;
    else if (size <= 12) size = 12;
    else if (size <= 14) size = 14;
    else if (size <= 18) size = 18;
    else if (size <= 48) size = 24;
    else                 size = GR_DEFAULT_FONTSIZE;

    if (italic) it = italic[0];
    else        it = 'r';

    /* フォント名作成 */
    sprintf(buf, GR_FONT_FORMAT, name, bold, it, size*10);
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  font = \"%s\"\n", buf);fflush(stderr);
#endif
    if (!(*fs = XLoadQueryFont(disp, buf))) {
        return(-1);
    }
    return(0);
}

rgb_to_hsv(red, green, blue, h, s, v)
float red, green, blue;         /* 0～1.0 で表されるrgbの値 */
float *h, *s, *v;               /* 変換後のhsvの値 */
{
        float max, min;
        float r, g, b;

#ifdef GRAPH_DEBUG
    fprintf(stderr, "  RGB=[%g,%g,%g]\n",
            red, green, blue);
#endif
        max = red; min = red;

        if (green>max) max = green;
        if (green<min) min = green;

        if (blue>max) max = blue;
        if (blue<min) min = blue;

        *v = max;

        if (max!=0.0 && max != min) *s = (max-min)/max;
        else {
            *s = 0.0;
            *h = 0.0;
            return;
        }

        r = (max - red)/(max - min);
        g = (max - green)/(max - min);
        b = (max - blue)/(max - min);

        if (red == max) *h = b - g;
        else if (green == max) *h = 2 + r - b;
        else if (blue == max) *h = 4 + g - r;

        *h *= 60.0;

        if (*h<0.0) *h += 360.0;
}

hsv_to_rgb(h, s, v, red, green, blue)
     float h, s, v;             /* hsvの値 */
     float *red, *green, *blue; /* 0～1.0 で表されるrgbの値 */
{
        float i, f, p, q, t;

        h /= 60.0;

        i = (float)((int)h);
        f = h - i;
        p = v*(1-s);
        q = v*(1-s*f);
        t = v*(1 - (s*(1-f)));

        switch ((int)i) {

            case 0:
                *red = v;
                *green = t;
                *blue = p;
                break;

            case 1:
                *red = q;
                *green = v;
                *blue = p;
                break;

            case 2:
                *red = p;
                *green = v;
                *blue = t;
                break;

            case 3:
                *red = p;
                *green = q;
                *blue = v;
                break;

            case 4:
                *red = t;
                *green = p;
                *blue = v;
                break;

            case 5:
                *red = v;
                *green = p;
                *blue = q;
                break;
        }
}

int
grCalcAxisAttributes(gr)
int     gr;
{
```

```
    GrAttr      *p;
    int         nextg = 0;
    int         num_g;          /* 重ね合わせるグラフ数 */
    Number      maxx, minx;
    Number      maxy, miny;
    int         i;
    int         gr0;            /* 元のポイント位置保存 */

#ifdef GRAPH_DEBUG
/*    fprintf(stderr, "grCalcAxisAttributes(%d)\n", gr);fflush(stderr);*/
#endif

    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on) {
        return(-1);
    }

    gr0 = gr;
    p = gGraph+gr;

    if( p->xdata && (p->xrange[1] < p->xrange[0] )){
        /* 重ねあわせのベースでないときはまず、ベースまで戻る */
        if( p->duplicate == DUPLICATE_ON ){
            gr--;
            p = gGraph+gr;
            while( p->duplicate != DUPLICATE_OFF ){
                gr--;
                p = gGraph+gr;
            }
        }
        /* ベースのグラフ */
        p = gGraph+gr;
        /* 重ねて描くグラフも含めてX座標の最大最小を求める */
        maxx = -GR_INF;
        minx = GR_INF;
        while(1){
            if (p->xdata && (p->xrange[1] < p->xrange[0])) {
                for (i = 0;i < p->datacount;i++) {
                    if (maxx < p->xdata[i])
                        maxx = p->xdata[i];
                    if (p->xdata[i] < minx)
                        minx = p->xdata[i];
                }
            }else if( p->xdata && (p->xrange[1] > p->xrange[0] )){
                maxx = p->xrange[1];
                minx = p->xrange[0];
            }
            nextg++;
            if( gr+nextg < GR_MAX_GRAPH  ){
                p = gGraph+gr+nextg;
            }else{
                break;
            }

            if( p->duplicate != DUPLICATE_ON ) break;
        }
        num_g = nextg-1;
        /* 最大最小値が同じ場合 -1から+1とする */
        if (minx == maxx) {
            minx -= 1;
            maxx += 1;
        }
        nextg = 0;
        p = gGraph+gr;
        while( nextg < num_g+1 ){
            p = gGraph+gr+nextg;

            p->xrange[0] = minx;
            p->xrange[1] = maxx;
            nextg++;
        }
    }

#ifdef GRAPH_DEBUG
    fprintf(stderr, "  xrange(%d) = [%g %g]\n", p->datacount,
            p->xrange[0], p->xrange[1]);
#endif

    nextg = 0;
    p = gGraph+gr0;

    if( p->ydata && (p->yrange[1] < p->yrange[0] ) ){
        /* ベースのグラフ */
        p = gGraph+gr;
        /* yの最大最小値を、重ねて描くグラフも含めて算出 */
        maxy = -GR_INF;
        miny = GR_INF;
        while( nextg < num_g+1 ){
            if (p->ydata && (p->yrange[1] < p->yrange[0])) {
                for (i = 0;i < p->datacount;i++) {
                    if (maxy < p->ydata[i])
                        maxy = p->ydata[i];
                    if (p->ydata[i] < miny)
                        miny = p->ydata[i];
                }
            }else if( p->ydata && (p->yrange[1] > p->yrange[0] )){
                maxy = p->yrange[1];
                miny = p->yrange[0];
            }
            nextg++;
            p = gGraph+gr+nextg;

        }
        /* 最大最小値が同じ場合 -1から+1とする */
        if (miny == maxy) {
            miny -= 1;
            maxy += 1;
        }
        nextg = 0;
        p = gGraph + gr;
        while( nextg < num_g+1 ){
            p = gGraph+gr+nextg;
            p->yrange[0] = miny;
            p->yrange[1] = maxy;
            nextg++;
        }
    }

#ifdef GRAPH_DEBUG
        fprintf(stderr, "  yrange=[%g %g]\n",
                p->yrange[0], p->yrange[1]);fflush(stderr);
#endif

    p = gGraph+gr0;
        /* x目盛りの計算 */
    if (p->num_xticks != -1) {
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  calc xticks\n");fflush(stderr);
#endif
        grMakeNormalTicks(p->xrange, &(p->xticks), &(p->num_xticks),
                          &(p->xticks_minor), &(p->num_xticks_minor));
        grMakeTickMarks(p->xticks, p->num_xticks,
                        &(p->xtick_marks), &(p->num_xtick_marks),
                        &(p->xtick_scale),
                        p->max_exp);
    }

        /* y目盛りの計算 */
    if (p->num_yticks != -1) {
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  calc yticks\n");fflush(stderr);
```

```
#endif
        grMakeNormalTicks(p->yrange, &(p->yticks), &(p->num_yticks),
                          &(p->yticks_minor), &(p->num_yticks_minor));
        grMakeTickMarks(p->yticks, p->num_yticks,
                        &(p->ytick_marks), &(p->num_ytick_marks),
                        &(p->ytick_scale),
                        p->max_exp);
    }

    return(0);
}

int
grFreeAxisAttributes(gr)
int     gr;
{
    GrAttr      *p;
#ifdef GRAPH_DEBUG
/*    fprintf(stderr, "grFreeAxisAttributes(%d)\n", gr);fflush(stderr);*/
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on) {
        return(-1);
    }

    p = gGraph+gr;
    /* x目盛りの開放 */
    Free(p->xticks);
    p->num_xticks = 0;

    /* y目盛りの開放 */
    Free(p->yticks);
    p->num_yticks = 0;

    /* x目盛り文字列の開放 */
    if (p->xtick_marks) {
        int     i;
        for(i = 0;i < p->num_xtick_marks;i++)
            Free(p->xtick_marks[i]);
        Free(p->xtick_marks);
    }
    p->num_xtick_marks = 0;

    /* y目盛り文字列の開放 */
    if (p->ytick_marks) {
        int     i;
        for(i = 0;i < p->num_ytick_marks;i++)
            Free(p->ytick_marks[i]);
        Free(p->ytick_marks);
    }
    p->num_ytick_marks = 0;

    /* x小目盛りの開放 */
    Free(p->xticks_minor);
    p->num_xticks_minor = 0;

    /* y小目盛りの開放 */
    Free(p->yticks_minor);
    p->num_yticks_minor = 0;

    /* x,y範囲の初期化 */
    p->xrange[0] = p->yrange[0] = 0;
    p->xrange[1] = p->yrange[1] = -1;

    return(0);
}

int
grDrawTitleCenter(int gr, Window win, char *title)
{
  GrAttr *p;
  int x, y;
  unsigned int width, height;
  Window       root;
  unsigned int border, depth;


  p = gGraph + gr;
  XGetGeometry(p->disp, win, &root,
               &x, &y, &width, &height, &border, &depth);
  grGetCenterPosTitle(gr, title,
                0, 0, width, height, &x, &y);
  XDrawString(p->disp, win, p->title_gc,
              x, y, title,
              strlen(title));
  return(0);
}

int grPSDrawTitleCenter(FILE *fp, int gr, char *title, float *x, float *y,
                        float *width, float *height)
{
  GrAttr *p;

  p = gGraph + gr;
  grPSSetFont(fp,
              p->title_fontname,
              p->title_fontbold,
              p->title_fontitalic,
              p->title_fontsize);
  fprintf(fp, "  (%s) %.5g %.5g %.5g %.5g centerText\n",
          title,
          *x, *y, *width, *height);

  return(0);
}

/* 各点のラベル描画関数 */
int grDrawDotLabel( Graph gr, XPoint *ps )
{
    GrAttr      *p;
    int         i;
    char        label[MAX_LABELDATA_LEN];
    char        baselabel[MAX_LABELDATA_LEN];
    int         dist, arg;
    int         xdist, ydist;
    int         dir, ascent, decent, width, height;
    int         wh0, wh1, ww0, ww1;
    int         x, y;
    XCharStruct cs;
    int         arg_flg;

    p = gGraph + gr;
    dist = p->dotlabel_dist;
    if( dist < 0 ) dist = 0;
    /* 文字列を描く限界 */
/*    ww0 = p->Ox - 1;
    ww1 = p->Ox - 1 + p->Ow + 2;
    wh0 = p->Oy - 1 + p->Oh + 2;
    wh1 = p->Oy - 1;*/
    ww0 = p->Ax + 1;
    ww1 = p->Ax + p->Aw - 1;
    wh0 = p->Ay + p->Ah - 1;
    wh1 = p->Ay + 1;

    for( i = 0; i < p->datacount; i++ ){
        memset( label, 0, sizeof( label ) );
        arg = 0;
        arg_flg = 0;
        /* ラベルがデータ列指定の時 */
        if( p->labeldata_fp != NULL ){
            sprintf( label, "%.2f", *(p->labeldata_fp+i) );
```

```
        }else if( p->labeldata_cp != NULL ){
        /* ラベルがラベル列指定の時 */
            strcpy( baselabel, p->labeldata_cp+i*MAX_LABELDATA_LEN );
            if( strcmp( baselabel, "@x" ) == 0 ){
                /* 予約語@x指定の時 (x座標値をラベルとする) */
                sprintf( label, "%.2f", p->xdata[i] );
            }else if( strcmp( baselabel, "@y" ) == 0 ){
                /* 予約語@y指定の時 (y座標値をラベルとする) */
                sprintf( label, "%.2f", p->ydata[i] );
            }else if( strcmp( baselabel, "@0" ) == 0 ){
                /* 予約語@0指定の時、ラベルなし */
                strcpy( label, "" );
            }else if( baselabel[0] == '@' ){
                /* 予約語@000LABELの時 */
                arg_flg = 1;
                sscanf( baselabel, "@%d%s", &arg, label );
                if( dist == 0 ) dist = 1;
                arg = arg%360;
                XTextExtents( p->dotlabel_font, label, strlen( label ),
                          &dir, &ascent, &decent, &cs );
                width = cs.rbearing - cs.lbearing;
                height = cs.ascent + cs.descent;
                grGetLabelPosition(dist, arg, width, height, &xdist, &ydist);
            }else{
                strcpy( label, baselabel );
            }
        }
        if( strlen( label ) > 0 ){
            if( arg_flg == 0 ){
                XTextExtents( p->dotlabel_font, label, strlen( label ),
                              &dir, &ascent, &decent, &cs );
                width = cs.rbearing - cs.lbearing;
                height = cs.ascent + cs.descent;
                grGetLabelPosition(dist, 90, width, height, &xdist, &ydist);
            }
            x = ps[i].x + xdist;
            y = ps[i].y + ydist;
#if 0
            if( y > wh0 ) y = wh0 - 3;
            if( y - height < wh1 ) y = wh1 + height + 3;
            if( x < ww0 ) x = ww0 + 3;
            if( x + width > ww1 ) x = ww1 - width - 3;
#endif
            XDrawString( p->disp, p->win, p->dotlabel_gc,
                         x, y, label, strlen(label));
        }
    }
    return (0);

}


/*****************************************************************************
 *        Name: GrDrawLabel
 *    Function: グラフの各点毎のラベルの描画
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *      Return:
 *              0   成功
 *             -1   失敗
 * Description:
 *             ウィンドウへグラフの各点毎のラベルの描画を行なう。
 *
 *****************************************************************************/
int GrDrawLabel(Graph gr)
{
    GrAttr      *p;
    int         i;
    XPoint      *ps;
    float       x, y;

    p = gGraph + gr;

    /* データの相対座標から絶対座標への変換 */
    ps = (XPoint *)malloc(sizeof(XPoint)*(p->datacount));
    for (i = 0;i < p->datacount;i++) {
        x = (p->0x + (p->xdata[i]-p->xrange[0])
             /(p->xrange[1]-p->xrange[0])*p->0w);
        y = (p->0y + p->0h - (p->ydata[i]-p->yrange[0])
             /(p->yrange[1]-p->yrange[0])*p->0h);
        ps[i].x = (int)(x+0.5);
        ps[i].y = (int)(y+0.5);
    }

    /* 各点にラベルを描く */
    if( p->labeldata_fp != NULL || p->labeldata_cp != NULL ){
        grDrawDotLabel( gr, ps );
    }

    Free(ps);

    /* 次のグラフが上書きに指定されているなら、それを呼び出す */
    p++;
    if( p->duplicate == DUPLICATE_ON ){
        GrDrawLabel( gr+1 );
    }

    return (0);
}

int
grGetLabelPosition(int dist, int arg, int width, int height,
                   int *xdist, int *ydist)
{
    if( arg == 0 ){
        *xdist = dist;
        *ydist = 0;
    }else if( 0 < arg && arg < 90 ){
        *xdist = (int)(dist*cos(arg/180.0*M_PI)+0.5);
        *ydist = -(int)(dist*sin(arg/180.0*M_PI)+0.5);
    }else if( arg == 90 ){
        *xdist = 0;
        *ydist = -dist;
    }else if( 90 < arg && arg < 180 ){
        *xdist = (int)(dist*cos(arg/180.0*M_PI)-0.5);
        *ydist = -(int)(dist*sin(arg/180.0*M_PI)+0.5);
    }else if( arg == 180 ){
        *xdist = -dist;
        *ydist = 0;
    }else if( 180 < arg && arg < 270 ){
        *xdist = (int)(dist*cos(arg/180.0*M_PI)-0.5);
        *ydist = -(int)(dist*sin(arg/180.0*M_PI)-0.5);
    }else if( arg == 270 ){
        *xdist = 0;
        *ydist = +dist;
    }else if( 270 < arg && arg < 360 ){
        *xdist = (int)(dist*cos(arg/180.0*M_PI)+0.5);
        *ydist = -(int)(dist*sin(arg/180.0*M_PI)-0.5);
    }

    *xdist -= width/2;
    *ydist += height/2;
}
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *               Version 2.0
 *    (c) Copyright 1997,1998
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * print.c -
 */

#include <stdio.h>
#include <varargs.h>
#include <math.h>
#include <string.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>

#include "plot.h"
#include "graph.h"
extern char *psGetRgbColor( char* );

/*      8       16      24      32      40      48      56                    */
/*------+-------+-------+-------+-------+-------+-------+--------------------*/
/*****************************************************************************
 *        Name: GrPrint
 *    Function: グラフのPS出力
 *    Argument:
 *              Graph gr        I       グラフハンドル
 *              FILE out        I       出力ファイルハンドル
 *      Return:
 *              0  成功
 *             -1  失敗
 * Description:
 *              指定されたファイルへPS形式でグラフの出力を行なう。
 *
 *****************************************************************************/
int GrPrint(Graph gr, FILE *out)
{
    GrAttr      *p;
    int         i;
    XPoint      *ps;

#ifdef GRAPH_DEBUG
    fprintf(stderr, "GrPrint(%d)\n", gr);fflush(stderr);
#endif
    if (gr < 0 || gr >= GR_MAX_GRAPH || !gGraph[gr].on)
        return(-1);

    p = gGraph + gr;
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  hold: %d\n", p->hold);fflush(stderr);
#endif

    /* チェック */
    if (p->width <= 0 ||
        p->height <= 0 ||
        p->datacount < 0)
        return(-2);

    if (p->graph_type == GR_TYPE_METER) {
      /* not implemented */
    }

    /* 座標属性の計算 */
    if (p->hold == FALSE) {
        grCalcAxisAttributes(gr);
    }

    /* 描画領域の計算 */
    grCalcAxisOrigin(gr);
/*    grClearGraph(gr);*/

    /* ボックス/座標軸の描画 */
#ifdef GRAPH_DEBUG
    fprintf(stderr, "  draw box/axis\n");fflush(stderr);
#endif
    if (p->box) {
#ifdef DRAW
      XDrawRectangle(p->disp, p->win, p->foregc,
                     p->Ox, p->Oy, p->Ow, p->Oh);
#else
      fprintf(out, "  %.5g %.5g %.5g %.5g rectangle\n",
              (float)p->Ox/p->Ww,
             -(float)p->Oy/p->Wh,
              (float)(p->Ox + p->Ow)/p->Ww,
             -(float)(p->Oy + p->Oh)/p->Wh);
#endif
    }
    else {
#ifdef DRAW
      /* x座標軸 */
      XDrawLine(p->disp, p->win, p->foregc,
                p->Ox-1, p->Oy-1 + p->Oh+2,
                p->Ox-1 + p->Ow+2, p->Oy-1 + p->Oh+2);
      /* y座標軸 */
      XDrawLine(p->disp, p->win, p->foregc,
                p->Ox-1, p->Oy-1 + p->Oh+2,
                p->Ox-1, p->Oy-1);

      XDrawLine(p->disp, p->win, p->foregc,
                p->Ox-1, p->Oy-1,
                (int)(p->Ox-1 + p->Aw*GR_TICK_LENGTH/3),
                p->Oy-1);
      XDrawLine(p->disp, p->win, p->foregc,
                p->Ox-1 + p->Ow+2, p->Oy-1 + p->Oh+2,
                p->Ox-1 + p->Ow+2,
                (int)((p->Oy-1 + p->Oh+2)-p->Ah*GR_TICK_LENGTH/3));
#else
      /* x座標軸 */
      fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
              (float)p->Ox/p->Ww, (float)-(p->Oy + p->Oh)/p->Wh,
              (float)(p->Ox + p->Ow)/p->Ww, (float)-(p->Oy + p->Oh)/p->Wh);

      /* y座標軸 */
      fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
              (float)(p->Ox)/p->Ww, (float)-(p->Oy + p->Oh)/p->Wh,
              (float)(p->Ox)/p->Ww, (float)-(p->Oy)/p->Wh);

      fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
              (float)p->Ox/p->Ww,
             -(float)p->Oy/p->Wh,
              (float)(p->Ox + p->Aw*GR_TICK_LENGTH/3)/p->Ww,
             -(float)p->Oy/p->Wh);
      fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
              (float)(p->Ox + p->Ow)/p->Ww,
             -(float)(p->Oy + p->Oh)/p->Wh,
              (float)(p->Ox + p->Ow)/p->Ww,
             -(float)((p->Oy + p->Oh)-p->Ah*GR_TICK_LENGTH/3)/p->Wh);
#endif
    }

    /* x目盛り文字列の描画 */
    if (p->num_xtick_marks > 0) {
      int               i, x1, y1, x, y;
      float     rx, dx;
```

```
#ifdef GRAPH_DEBUG
      fprintf(stderr, "  draw xtick marks\n");fflush(stderr);
#endif
#ifdef DRAW
      rx = p->xrange[1] - p->xrange[0];
      if (rx == 0) dx = 0;
      else dx = p->Ow/rx;

      for (i = 0;i < p->num_xtick_marks;i++) {
        x1 = p->Ox + (p->xticks[i]-p->xrange[0])*dx;
        y1 = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN;
        grGetCenterPos(gr, p->xtick_marks[i],
                       x1, y1, 0, GR_POS_XTICKS_HEIGHT, &x, &y);
        XDrawString(p->disp, p->win, p->tick_gc,
                    x, y, p->xtick_marks[i],
                    strlen(p->xtick_marks[i]));
      }
#else
      /* フォントの設定 */
      grPSSetFont(out,
                  p->fontname,
                  p->fontbold,
                  p->fontitalic,
                  p->fontsize);

      rx = p->xrange[1] - p->xrange[0];
      if (rx == 0) dx = 0;
      else dx = p->Ow/rx;
      for (i = 0;i < p->num_xtick_marks;i++) {
        x1 = p->Ox + (p->xticks[i]-p->xrange[0])*dx;
        y1 = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN;
        fprintf(out, "  (%s) %.5g %.5g %.5g %.5g centerText\n",
                p->xtick_marks[i],
                (float)x1/p->Ww,
               -(float)y1/p->Wh,
                (float)0,
               -(float)GR_POS_XTICKS_HEIGHT/p->Wh);
      }
#endif
    }

    /* xスケールの描画 */
    if (p->xtick_scale) {
      int  x, y;
      char str[20], *ptr;
      unsigned int width, height;

#ifdef DRAW
      x = p->Ox + p->Ow;
      y = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN
        + GR_POS_XTICKS_HEIGHT;
      grGetCenterPos(gr, str, x, y, 2, 2, &x, &y);

      strcpy(str, "x");
      strcat(str, p->xtick_scale);
      if ((ptr = strchr(str, '^')) != NULL)
        *ptr++ = '\0';
      XDrawString(p->disp, p->win, p->tick_gc,
                  x, y, str, strlen(str));
      if (ptr) {
        grTextWidth(p->disp, p->win, p->tick_gc,
                    str, strlen(str), &width, &height);
        XDrawString(p->disp, p->win, p->ticksub_gc,
                    x + width, y - height/2, ptr, strlen(ptr));
      }
#else
      x = p->Ox + p->Ow;
      y = p->Oy + p->Oh + p->Ah*GR_POS_XTICKS_MARGIN
        + GR_POS_XTICKS_HEIGHT;
```

```
      strcpy(str, "x");
      strcat(str, p->xtick_scale);
      if ((ptr = strchr(str, '^')) != NULL)
        *ptr++ = '\0';

      grPSSetFont(out,
                  p->tick_fontname,
                  p->tick_fontbold,
                  p->tick_fontitalic,
                  p->tick_fontsize);
      fprintf(out, "  (%s) %.5g %.5g rightText\n",
              str,
              (float)x/p->Ww,
             -(float)y/p->Wh);

      if (ptr) {
        grPSSetFont(out,
                    p->ticksub_fontname,
                    p->ticksub_fontbold,
                    p->ticksub_fontitalic,
                    p->ticksub_fontsize);
        fprintf(out, "  (%s) %.5g %.5g leftText\n",
                ptr,
                (float)x/p->Ww,
               -(float)(y-4)/p->Wh);
      }
#endif

    }

    /* y目盛り文字列の描画 */
    if (p->num_ytick_marks > 0) {
      int               i, x1, y1, x, y;
      float             ry, dy;

#ifdef DRAW
      ry = p->yrange[1] - p->yrange[0];
      if (ry == 0) dy = 0;
      else dy = p->Oh/ry;
      for (i = 0;i < p->num_ytick_marks;i++) {
        x1 = p->Ox - GR_POS_YTICKS_MARGIN;
        y1 = p->Oy + p->Oh - (p->yticks[i]-p->yrange[0])*dy;
        grGetCenterPos(gr, p->ytick_marks[i], x1, y1, -1, 0, &x, &y);
        XDrawString(p->disp, p->win, p->tick_gc,
                    x, y, p->ytick_marks[i],
                    strlen(p->ytick_marks[i]));
      }
#else
      /* フォントの設定 */
      grPSSetFont(out,
                  p->fontname,
                  p->fontbold,
                  p->fontitalic,
                  p->fontsize);

      ry = p->yrange[1] - p->yrange[0];
      if (ry == 0) dy = 0;
      else dy = p->Oh/ry;
      for (i = 0;i < p->num_ytick_marks;i++) {
        x1 = p->Ox - GR_POS_YTICKS_MARGIN;
        y1 = p->Oy + p->Oh - (p->yticks[i]-p->yrange[0])*dy;
        fprintf(out, "  (%s) %.5g %.5g rightText\n",
                p->ytick_marks[i],
                (float)x1/p->Ww,
               -(float)y1/p->Wh);
      }
#endif
    }
```

```
    /* タイトル文字列の描画 */
    if (p->title && *p->title) {
      int       x, y;
      unsigned int      w, h;

#ifdef DRAW
      w = p->Aw;
      h = p->Ry;
      grGetCenterPos(gr, p->title, p->Ax, p->Ay, w, h, &x, &y);
      XDrawString(p->disp, p->win, p->title_gc,
                  x, y, p->title, strlen(p->title));
#else
#endif
    }

    /* 重ね表示対応によりタイトル文字列やスケールは１番最初のグラフに合わせる
                        (一番最初のグラフのタイトルとグラフ枠のみを表示) */
    if( p->duplicate != DUPLICATE_ON ) {
      /* xラベル文字列の描画 */
      if (p->xlabel && *p->xlabel) {
        int                     x, y;
        unsigned int    w, h;

#ifdef DRAW
        w = p->Ow;
        h = p->Ah - (p->Ry + p->Rh);
        grGetCenterPos(gr, p->xlabel, p->Ox, p->Ry+p->Rh, w, h, &x, &y);
        XDrawString(p->disp, p->win, p->label_gc,
                    x, y, p->xlabel, strlen(p->xlabel));
#else
        w = p->Ow;
        h = (p->Ay + p->Ah) - (p->Ry + p->Rh);
        grPSSetFont(out,
                    p->label_fontname,
                    p->label_fontbold,
                    p->label_fontitalic,
                    p->label_fontsize);
        fprintf(out, "  (%s) %.5g %.5g %.5g %.5g centerText\n",
                p->xlabel,
                (float)p->Ox/p->Ww,
               -(float)(p->Ry + p->Rh)/p->Wh,
                (float)w/p->Ww,
               -(float)h/p->Wh);
#endif
      }

      /* yラベル文字列の描画 */
      if (p->ylabel && *p->ylabel) {
        int             x, y;
        unsigned int    w, h;

#ifdef DRAW
        w = p->Rx - p->Ax;
        h = p->Oh;
        grGetCenterPos(gr, p->ylabel, p->Ax, p->Ay, h, w, &y, &x);
        grDrawStringDownToUp(p->disp, p->win, p->label_gc,
                             x, p->Oy + p->Oh - (y - p->Ay),
                             p->ylabel, strlen(p->ylabel));
#else
        w = p->Oh;
        h = p->Rx - p->Ax;
        grPSSetFont(out,
                    p->label_fontname,
                    p->label_fontbold,
                    p->label_fontitalic,
                    p->label_fontsize);
        fprintf(out, "  (%s) %.5g %.5g %.5g %.5g centerTextRot90\n",
                p->ylabel,
                (float)p->Rx/p->Ww,
               -(float)(p->Oy + p->Oh)/p->Wh,
                (float)w/p->Wh,
               -(float)h/p->Ww);
#endif
      }

      /* yスケールの描画 */
      if (p->ytick_scale) {
        int  x, y;
        char str[20], *ptr;
        unsigned int width, height;

#ifdef DRAW
        x = p->Ox;
        y = p->Ry;
        grGetCenterPos(gr, str, x, y, -1, 0, &x, &y);

        strcpy(str, "x");
        strcat(str, p->ytick_scale);
        if ((ptr = strchr(str, '^')) != NULL)
                *ptr++ = '\0';
        XDrawString(p->disp, p->win, p->tick_gc,
                    x, y, str, strlen(str));
        if (ptr) {
          grTextWidth(p->disp, p->win, p->tick_gc,
                      str, strlen(str), &width, &height);
          XDrawString(p->disp, p->win, p->ticksub_gc,
                      x + width, y - height/2, ptr, strlen(ptr));
        }
#else
        x = p->Ox;
        y = p->Ry;
        strcpy(str, "x");
        strcat(str, p->ytick_scale);
        if ((ptr = strchr(str, '^')) != NULL)
                *ptr++ = '\0';
        grPSSetFont(out,
                    p->tick_fontname,
                    p->tick_fontbold,
                    p->tick_fontitalic,
                    p->tick_fontsize);
        fprintf(out, "  (%s) %.5g %.5g rightText\n",
                str,
                (float)x/p->Ww,
               -(float)y/p->Wh);
        if (ptr) {
          grPSSetFont(out,
                      p->ticksub_fontname,
                      p->ticksub_fontbold,
                      p->ticksub_fontitalic,
                      p->ticksub_fontsize);
          fprintf(out, "  (%s) %.5g %.5g leftText\n",
                  ptr,
                  (float)x/p->Ww,
                 -(float)(y-4)/p->Wh);
        }

#endif
      }

      /* x目盛りの描画 */
      if (p->xticks) {
#ifdef DRAW
        XSegment  seg[GR_TICK_MAX_ALL_TICKS];
#endif
        int     i, len;
        float   rx, dx;

#ifdef GRAPH_DEBUG
        fprintf(stderr, "  draw xticks\n");fflush(stderr);
#endif
```

```
#ifdef DRAW
        rx = p->xrange[1] - p->xrange[0];
        if (rx == 0) dx = 0;
        else dx = p->Ow/rx;

        /* Major目盛りの描画 */
        len = p->Ah*GR_TICK_LENGTH;
        if (len < 2) len = 2;
        for (i = 0;i < p->num_xticks;i++) {
          seg[i].x1 = seg[i].x2 = p->Ox + (p->xticks[i]-p->xrange[0])*dx;
          seg[i].y1 = p->Oy + p->Oh;
          seg[i].y2 = p->Oy + p->Oh - len;
        }
        XDrawSegments(p->disp, p->win, p->foregc, seg, p->num_xticks);

        /* Minor目盛りの描画 */
        len = p->Ah*GR_TICK_MINOR_LENGTH;
        if (len < 1) len = 1;
        if (p->num_xticks_minor > 0) {
          for (i = 0;i < p->num_xticks_minor;i++) {
            seg[i].x1 = seg[i].x2 = p->Ox +
                        (p->xticks_minor[i]-p->xrange[0])*dx;
            seg[i].y1 = p->Oy + p->Oh;
            seg[i].y2 = p->Oy + p->Oh - len;
          }
          XDrawSegments(p->disp, p->win, p->foregc,
                        seg, p->num_xticks_minor);
        }
#else
        rx = p->xrange[1] - p->xrange[0];
        if (rx == 0) dx = 0;
        else dx = p->Ow/rx;

        /* Major目盛りの描画 */
        len = p->Ah*GR_TICK_LENGTH;
        if (len < 2) len = 2;
        for (i = 0;i < p->num_xticks;i++) {
          fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
                  (float)(p->Ox + (p->xticks[i]-p->xrange[0])*dx)/p->Ww,
                 -(float)(p->Oy + p->Oh)/p->Wh,
                  (float)(p->Ox + (p->xticks[i]-p->xrange[0])*dx)/p->Ww,
                 -(float)(p->Oy + p->Oh - len)/p->Wh);
        }

        /* Minor目盛りの描画 */
        len = p->Ah*GR_TICK_MINOR_LENGTH;
        if (len < 1) len = 1;
        if (p->num_xticks_minor > 0) {
          for (i = 0;i < p->num_xticks_minor;i++) {
            fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
                    (float)(p->Ox + (p->xticks_minor[i]-p->xrange[0])*dx)/p->Ww,
                   -(float)(p->Oy + p->Oh)/p->Wh,
                    (float)(p->Ox + (p->xticks_minor[i]-p->xrange[0])*dx)/p->Ww,
                   -(float)(p->Oy + p->Oh - len)/p->Wh);
          }
        }
#endif

      }

      /* y目盛りの描画 */
      if (p->yticks) {
#ifdef DRAW
        XSegment        seg[GR_TICK_MAX_ALL_TICKS];
#endif
        int     i, len;
        float   ry, dy;
#ifdef GRAPH_DEBUG
        fprintf(stderr, "  draw yticks\n");fflush(stderr);
#endif
```

```
#ifdef DRAW
        ry = p->yrange[1] - p->yrange[0];
        if (ry == 0) dy = 0;
        else dy = p->Oh/ry;

        /* Major目盛りの描画 */
        len = p->Aw*GR_TICK_LENGTH;
        if (len < 2) len = 2;
        for (i = 0;i < p->num_yticks;i++) {
          seg[i].y1 = seg[i].y2 = p->Oy + p->Oh
                    - (p->yticks[i]-p->yrange[0])*dy;
          seg[i].x1 = p->Ox;
          seg[i].x2 = p->Ox + len;
        }
        XDrawSegments(p->disp, p->win, p->foregc, seg, p->num_yticks);

        /* Minor目盛りの描画 */
        len = p->Aw*GR_TICK_MINOR_LENGTH;
        if (len < 1) len = 1;
        for (i = 0;i < p->num_yticks_minor;i++) {
          seg[i].y1 = seg[i].y2 = p->Oy + p->Oh
                    - (p->yticks_minor[i]-p->yrange[0])*dy;
          seg[i].x1 = p->Ox;
          seg[i].x2 = p->Ox + len;
        }
        XDrawSegments(p->disp, p->win, p->foregc, seg,
                      p->num_yticks_minor);
#else
        ry = p->yrange[1] - p->yrange[0];
        if (ry == 0) dy = 0;
        else dy = p->Oh/ry;

        /* Major目盛りの描画 */
        len = p->Aw*GR_TICK_LENGTH;
        if (len < 2) len = 2;
        for (i = 0;i < p->num_yticks;i++) {
          fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
                  (float)(p->Ox)/p->Ww,
                 -(float)(p->Oy + p->Oh - (p->yticks[i]-p->yrange[0])*dy)/p->Wh,
                  (float)(p->Ox + len)/p->Ww,
                 -(float)(p->Oy + p->Oh - (p->yticks[i]-p->yrange[0])*dy)/p->Wh);
        }

        /* Minor目盛りの描画 */
        len = p->Aw*GR_TICK_MINOR_LENGTH;
        if (len < 1) len = 1;
        if (p->num_yticks_minor > 0) {
          for (i = 0;i < p->num_yticks_minor;i++) {
            fprintf(out, "  %.5g %.5g %.5g %.5g line\n",
                    (float)(p->Ox)/p->Ww,
                   -(float)(p->Oy + p->Oh
                            - (p->yticks_minor[i]-p->yrange[0])*dy)/p->Wh,
                    (float)(p->Ox + len)/p->Ww,
                   -(float)(p->Oy + p->Oh
                            - (p->yticks_minor[i]-p->yrange[0])*dy)/p->Wh);
          }
        }
#endif
      }
    }

    /* データの描画 */
    if (p->ydata) {
      XRectangle        rect[1];
      Number          x, y, oldx, oldy;

      /* データの相対座標から絶対座標への変換 */
      ps = (XPoint *)malloc(sizeof(XPoint)*(p->datacount));
#ifdef DRAW
```

```
      for (i = 0;i < p->datacount;i++) {
        x = (p->Ox + (p->xdata[i]-p->xrange[0])
             /(p->xrange[1]-p->xrange[0])*p->Ow);
        y = (p->Oy + p->Oh - (p->ydata[i]-p->yrange[0])
             /(p->yrange[1]-p->yrange[0])*p->Oh);
        ps[i].x = rint(x);
        ps[i].y = rint(y);
      }
#endif
#ifdef GRAPH_DEBUG_EX
      {
        int             i;
        fprintf(stderr, "  xrange = [%g %g], yrange = [%g %g]\n",
                p->xrange[0], p->xrange[1],
                p->yrange[0], p->yrange[1]);
        fprintf(stderr, "  data = [");
        for (i = 0;i < p->datacount;i++)
          fprintf(stderr, "(%g,%g;%d,%d)",
                  p->xdata[i], p->ydata[i],
                  ps[i].x, ps[i].y);
        fprintf(stderr, "]\n");
      }
#endif
      /* プロット領域をクリップする */
      rect[0].x = p->Ox-1;
      rect[0].y = p->Oy-1;
      rect[0].width = p->Ow+2;
      rect[0].height = p->Oh+2;
#ifdef DRAW
      XSetClipRectangles(p->disp, p->foregc,
                         0, 0, rect, 1, Unsorted);
#endif

      fprintf(out, "  %.5g %.5g %.5g %.5g setAxesScale\n",
              (float)p->Ox/p->Ww, -(float)(p->Oy + p->Oh)/p->Wh,
              (float)p->Ow/p->Ww, (float)p->Oh/p->Wh);

      /* 描画領域のクリッピングをおこなう (現在はクリッピングしない)  */
      /* fprintf(out, "  %d %d %d %d setRectClip\n", 0, 0, 1, 1 ); */

      /* 線分の色を指定する */
      fprintf(out, "  %s setrgbcolor\n",psGetRgbColor(p->dot_color));

      /* 線分のサイズを指定する */
      fprintf(out, "  %f setlinewidth\n",p->dot_size);
      /*
       * 線種による描きわけ
       */
      /* 直線 */
      if (!strcmp(p->line_style, GR_LINE_STYLE_LINE)) {
#ifdef DRAW
        XSetLineAttributes(p->disp, p->foregc, p->line_width,
                           LineSolid, CapButt, JoinMiter);
        XDrawLines(p->disp, p->win, p->foregc,
                   ps, p->datacount, 0);
        XSetLineAttributes(p->disp, p->foregc, 0,
                           LineSolid, CapButt, JoinMiter);
#endif

        oldx = (p->xdata[0]-p->xrange[0])
          /(p->xrange[1]-p->xrange[0]);
        oldy = (p->ydata[0]-p->yrange[0])
          /(p->yrange[1]-p->yrange[0]);
        /* 線分の開始点を指定する */
        fprintf(out, "  %.5g %.5g setStartLine\n", oldx, oldy);
        for (i = 1;i < p->datacount;i++) {
          x = (p->xdata[i]-p->xrange[0])
            /(p->xrange[1]-p->xrange[0]);
          y = (p->ydata[i]-p->yrange[0])
            /(p->yrange[1]-p->yrange[0]);
          /* 線分のポイントを指定する */
          fprintf(out, "  %.5g %.5g setLineTo\n", x, y);
        }
        /* 線分終了を指定する */
        fprintf(out, "  setEndLine\n");
      }
      /* 点 */
      else if (!strcmp(p->line_style, GR_LINE_STYLE_DOT)) {
#ifdef DRAW
        XDrawPoints(p->disp, p->win, p->foregc,
                    ps, p->datacount, 0);
#endif

        x = (p->xdata[0]-p->xrange[0])
          /(p->xrange[1]-p->xrange[0]);
        y = (p->ydata[0]-p->yrange[0])
          /(p->yrange[1]-p->yrange[0]);
        /* ドットサイズの半径を指定できるように対応 */
        fprintf(out, "  %.6g %.6g %.6g dot1\n", x, y, (float)p->dot_size/2.0);
        for (i = 1;i < p->datacount;i++) {
          x = (p->xdata[i]-p->xrange[0])
            /(p->xrange[1]-p->xrange[0]);
          y = (p->ydata[i]-p->yrange[0])
            /(p->yrange[1]-p->yrange[0]);
          /* ドットサイズの半径を指定できるように対応 */
          fprintf(out, "  %.5g %.5g %.5g dot1\n", x, y, (float)p->dot_size/2.0);
        }
      }
      else if (!strcmp(p->line_style,  GR_LINE_STYLE_DASH)) {
        /* 線幅可変対応(線幅をdot_sizeの等倍に可変させる場合はコメントをとる)*/
        /* if(p->line_width != 0.0 ) */
        /* fprintf(out, "[%f] 0 setdash\n",
                                        p->dot_size*PS_DASH_PATTERN_DASH_LEN); */
        /* else */
        /* ラインタイプに点線を指定([2]は２単位分の点線を描く --  --  --....) */
        fprintf(out, "[%f] 0 setdash\n",PS_DASH_PATTERN_DASH_LEN);

        oldx = (p->xdata[0]-p->xrange[0])
          /(p->xrange[1]-p->xrange[0]);
        oldy = (p->ydata[0]-p->yrange[0])
          /(p->yrange[1]-p->yrange[0]);
        /* 線分の開始点を指定する */
        fprintf(out, "  %.5g %.5g setStartLine\n", oldx, oldy);

        for (i = 1;i < p->datacount;i++) {
          x = (p->xdata[i]-p->xrange[0])
            /(p->xrange[1]-p->xrange[0]);
          y = (p->ydata[i]-p->yrange[0])
            /(p->yrange[1]-p->yrange[0]);
          /* 線分のポイントを指定する */
          fprintf(out, "  %.5g %.5g setLineTo\n", x, y);
        }

        /* 線分終了を指定する */
        fprintf(out, "  setEndLine\n");
      }
/* 点線 */
      else if (!strcmp(p->line_style, GR_LINE_STYLE_DOTTEDLINE)) {
#ifdef DRAW
        XSetLineAttributes(p->disp, p->foregc, p->line_width,
                           LineOnOffDash, CapButt, JoinMiter);
        XSetDashes(p->disp, p->foregc, 0,
                   gr_dash_pattern_dot, GR_DASH_PATTERN_DOT_LEN);
        XDrawLines(p->disp, p->win, p->foregc,
                   ps, p->datacount, 0);
        XSetLineAttributes(p->disp, p->foregc, 0,
                           LineSolid, CapButt, JoinMiter);
#endif
        /* 線幅可変対応(線幅をdot_sizeの等倍にする場合はコメントをとる) */
        /* if(p->line_width != 0.0 ) */
```

```
    /* fprintf(out, "[%f 2 2 2] 0 setdash\n",
                              p->dot_size*PS_DASH_PATTERN_DOTTEDLINE_LEN); */
    /* else */
    /* ラインタイプに１点鎖線の指定を行う
                              ([12 2 2 2]は１２単位と２単位の鎖線を描く
                              ---------- -- ---------- -- .....) */
    fprintf(out, "[%f 2 2 2] 0 setdash\n",PS_DASH_PATTERN_DOTTEDLINE_LEN );

    oldx = (p->xdata[0]-p->xrange[0])
      /(p->xrange[1]-p->xrange[0]);
    oldy = (p->ydata[0]-p->yrange[0])
      /(p->yrange[1]-p->yrange[0]);
    /* 線分の開始点を指定する */
    fprintf(out, "  %.5g %.5g setStartLine\n", oldx, oldy);

    for (i = 1;i < p->datacount;i++) {
      x = (p->xdata[i]-p->xrange[0])
        /(p->xrange[1]-p->xrange[0]);
      y = (p->ydata[i]-p->yrange[0])
        /(p->yrange[1]-p->yrange[0]);
      /* 線分のポイントを指定する */
      fprintf(out, "  %.5g %.5g setLineTo\n", x, y);
    }

    /* 線分終了を指定する */
    fprintf(out, "  setEndLine\n");
  }
  else if (!strcmp(p->line_style, GR_LINE_STYLE_DASH_LONG)) {
    /* 線幅可変対応(線幅をdot_sizeの等倍にする場合はコメントをとる) */
    /* if(p->line_width != 0.0 ) */
    /* fprintf(out, "[%f 2] 0 setdash\n",
                              p->dot_size*PS_DASH_PATTERN_DASH_LONG_LEN); */
    /* else */
    /* ラインタイプに長点線の指定を行う([20 2]は２０単位の長点線を描く
                              ---------- ---------- .....) */
    fprintf(out, "[%f 2] 0 setdash\n",PS_DASH_PATTERN_DASH_LONG_LEN);

    oldx = (p->xdata[0]-p->xrange[0])
      /(p->xrange[1]-p->xrange[0]);
    oldy = (p->ydata[0]-p->yrange[0])
      /(p->yrange[1]-p->yrange[0]);
    /* 線分の開始点を指定する */
    fprintf(out, "  %.5g %.5g setStartLine\n", oldx, oldy);

    for (i = 1;i < p->datacount;i++) {
      x = (p->xdata[i]-p->xrange[0])
        /(p->xrange[1]-p->xrange[0]);
      y = (p->ydata[i]-p->yrange[0])
        /(p->yrange[1]-p->yrange[0]);
      /* 線分のポイントを指定する */
      fprintf(out, "  %.5g %.5g setLineTo\n", x, y);
    }

    /* 線分終了を指定する */
    fprintf(out, "  setEndLine\n");
  }
  else if (!strcmp(p->line_style, GR_LINE_STYLE_DOTTEDLINE_LONG)) {
    /* 線幅可変対応(線幅をdot_sizeの等倍にする場合はコメントをとる) */
    /* if(p->line_width != 0.0 ) */
    /* fprintf(out, "[%f 2 2 2] 0 setdash\n",
                        p->dot_size*PS_DASH_PATTERN_DOTTEDLINE_LONG_LEN); */
    /* else */
    /* ラインタイプに長１点鎖線の指定を行う
                              ([48 2 2 2]は４８単位と２単位の鎖線を描く
                        ---------------- -- ---------------- -- .....) */
    fprintf(out, "[%f 2 2 2] 0 setdash\n",
                                      PS_DASH_PATTERN_DOTTEDLINE_LONG_LEN);

    oldx = (p->xdata[0]-p->xrange[0])
      /(p->xrange[1]-p->xrange[0]);
```

```
    oldy = (p->ydata[0]-p->yrange[0])
      /(p->yrange[1]-p->yrange[0]);
    /* 線分の開始点を指定する */
    fprintf(out, "  %.5g %.5g setStartLine\n", oldx, oldy);

    for (i = 1;i < p->datacount;i++) {
      x = (p->xdata[i]-p->xrange[0])
        /(p->xrange[1]-p->xrange[0]);
      y = (p->ydata[i]-p->yrange[0])
        /(p->yrange[1]-p->yrange[0]);
      /* 線分のポイントを指定する */
      fprintf(out, "  %.5g %.5g setLineTo\n", x, y);
    }

    /* 線分終了を指定する */
    fprintf(out, "  setEndLine\n");
  }

  /* 線分の幅を指定する */
  fprintf(out, "  %f setlinewidth\n",p->dot_size);

  /* 線分の色指定をデフォルトに戻す */
  fprintf(out, "  0 0 0 setrgbcolor\n");

  /* ラベル表示データを作成する */
  psDrawDotLabel( out, gr, ps );

  /* クリップ領域を解除する (現在はクリッピングしない)  */
  /* fprintf(out, "  initclip\n"); */

  fprintf(out, "  stroke\n");
  fprintf(out, "  %.5g %.5g %.5g %.5g restoreAxesScale\n",
          (float)p->Ox/p->Ww, -(float)(p->Oy + p->Oh)/p->Wh,
          (float)p->Ow/p->Ww, (float)p->Oh/p->Wh);
#ifdef DRAW
    XSetClipMask(p->disp, p->foregc, (int)NULL);
#endif
    Free(ps);
  }

  return(0);
}

int grPSSetFont(FILE *fp, char *fontName, char *fontBold,
                char *fontItalic, int fontSize)
{
  char name[40];
  char bold[40];
  char italic[40];

  if (!fontName)
    return(-1);

  strcpy(name, fontName);
  *name = (char)toupper(*name);
  if (fontBold && strcmp(fontBold, "medium")) {
    strcpy(bold, fontBold);
    *bold = (char)toupper(*bold);
    strcat(name, "-");
    strcat(name, bold);
  }
  else if (fontItalic && strcmp(fontItalic, "roman")) {
    strcpy(bold, fontItalic);
    *italic = (char)toupper(*italic);
    strcat(name, "-");
    strcat(name, italic);
  }
  fprintf(fp, "/%s %d setFontInfo\n", name, fontSize);
  return(0);
}
```

```
int PSDrawLines(Number xdata[], Number ydata[], int datacount,
                Number xrange[], Number yrange[])
{
  int i;
  Number dx, dy;

  dx = xrange[1] - xrange[0];
  dy = yrange[1] - yrange[0];

  for (i = 0;i < datacount;i++) {
    printf("  %lg %lg line\n",
           (xdata[i]-xrange[0])/dx,
           (ydata[i]-yrange[0])/dy);
  }
  return(0);
}

/* ＲＧＢカラー取得関数ＰＳ用 */
char *psGetRgbColor( char *aDotColor )
{
        int             i;
        int             Color;
        float           RColor;
        float           GColor;
        float           BColor;
        static char     Buff[64];
        char            DotColor[64];
#ifdef  DBG
        fprintf( stderr, "psGetRgbColor : Call !!! \n" );
        fprintf( stderr, "psGetRgbColor : aDotColor = \"%s\" \n", aDotColor );
#endif
        strcpy( Buff, "0 0 0" );

        if( aDotColor == NULL ) {
                return( Buff );
        }

        if( aDotColor[0] == '#' ) {
                /* Ｘの表示指定をＰＳの表示指定に変換する */
                Buff[0] = aDotColor[1];
                Buff[1] = aDotColor[2];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                RColor = (float)Color;
                RColor = RColor * ( 1.0 / 255.0 );

                Buff[0] = aDotColor[3];
                Buff[1] = aDotColor[4];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                GColor = (float)Color;
                GColor = GColor * ( 1.0 / 255.0 );

                Buff[0] = aDotColor[5];
                Buff[1] = aDotColor[6];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                BColor = (float)Color;
                BColor = BColor * ( 1.0 / 255.0 );
        }
        else {
                /* ｐｌｏｔ０～９指定をＲＧＢに変換する */
                for( i = 0; aDotColor[i] != '\0' && aDotColor[i] != ':'; i ++ );

                if( aDotColor[i] != ':' ) {
                        return( Buff );
                }
                /* strcpy(DotColor,&aDotColor[i+1]); */
                Buff[0] = aDotColor[i+1];
                Buff[1] = aDotColor[i+2];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                RColor = (float)Color;
                RColor = RColor * ( 1.0 / 255.0 );

                Buff[0] = aDotColor[i+3];
                Buff[1] = aDotColor[i+4];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                GColor = (float)Color;
                GColor = GColor * ( 1.0 / 255.0 );

                Buff[0] = aDotColor[i+5];
                Buff[1] = aDotColor[i+6];
                Buff[2] = '\0';
                sscanf( Buff, "%x", &Color );
                BColor = (float)Color;
                BColor = BColor * ( 1.0 / 255.0 );
        }

        sprintf( Buff, "%f %f %f", RColor, GColor, BColor );
#ifdef  DBG
        fprintf( stderr, "psGetRgbColor : Buff = \"%s\" \n", Buff );
        fprintf( stderr, "psGetRgbColor : Normal End ! \n\n" );
#endif

        return( Buff );
}


/* 各点のラベル描画関数 */
int psDrawDotLabel( FILE *fp, Graph gr, XPoint *ps )
{
        GrAttr  *p;
        int     i;
        char    label[MAX_LABELDATA_LEN];
        char    baselabel[MAX_LABELDATA_LEN];
        int     dist, arg;
        int     xdist, ydist;
        int     dir, ascent, decent, width, height,size;
        int     x, y;
        float   fx, fy;
        int     arg_flg;
        int     set_flag;

        p = gGraph + gr;
        dist = p->dotlabel_dist;
        if( dist < 0 ) dist = 0;

        /* ポイント数分ラベルがあれば表示する */
        for( i = 0; i < p->datacount; i++ ){
                memset( label, 0, sizeof( label ) );
                arg = 0;
                arg_flg = 0;

                if( p->labeldata_fp != NULL ){
                        sprintf( label, "%.2f", *(p->labeldata_fp+i) );
                }else if( p->labeldata_cp != NULL ){
                        strcpy( baselabel, p->labeldata_cp+i*MAX_LABELDATA_LEN );
                        if( strcmp( baselabel, "@x" ) == 0 ){
                                /* Ｘ座標データを表示ラベルとして使用する */
                                sprintf( label, "%.2f", p->xdata[i] );
                        }else if( strcmp( baselabel, "@y" ) == 0 ){
                                /* Ｙ座標データを表示ラベルとして使用する */
                                sprintf( label, "%.2f", p->ydata[i] );
                        }else if( strcmp( baselabel, "@0" ) == 0 ){
                                /* ラベル無しを指定 */
                                strcpy( label, "" );
```

```
                        }else if( baselabel[0] == '@' ){
                                /* ラベルにユーザーラベルと表示角度を指定 */
                                arg_flg = 1;
                                sscanf( baselabel, "@%d%s", &arg, label );
                                if( dist == 0 ) dist = 1;
                                arg = arg%360;

                                /* 各角度でのオフセットを算出する */
                                if( arg == 0 ){
                                        xdist = dist;
                                        ydist = 0;
                                }else if( 0 < arg && arg < 90 ){
                                        xdist = (int)( (float)dist*cos((float)arg/180.0*M_
PI) );
                                        ydist = (int)( (float)dist*sin((float)arg/180.0*M_
PI) );
                                }else if( arg == 90 ){
                                        xdist = 0;
                                        ydist = dist;
                                }else if( 90 < arg && arg < 180 ){
                                        xdist = (int)( (float)dist*cos((float)arg/180.0*M_
PI) );
                                        ydist = (int)( (float)dist*sin((float)arg/180.0*M_
PI) );
                                }else if( arg == 180 ){
                                        xdist = -dist;
                                        ydist = 0;
                                }else if( 180 < arg && arg < 270 ){
                                        xdist = (int)( (float)dist*cos((float)arg/180.0*M_
PI) );
                                        ydist = (int)( (float)dist*sin((float)arg/180.0*M_
PI) );
                                }else if( arg == 270 ){
                                        xdist = 0;
                                        ydist = -dist;
                                }else if( 270 < arg && arg < 360 ){
                                        xdist = (int)( (float)dist*cos((float)arg/180.0*M_
PI) );
                                        ydist = (int)( (float)dist*sin((float)arg/180.0*M_
PI) );
                                }
                        }else{
                                strcpy( label, baselabel );
                        }
                }

                /* ラベル表示の指定があればラベルを表示 */
                if( strlen( label ) > 0 ){
                        /* 角度指定がなされていない場合は上に表示 */
                        if( arg_flg == 0 ){
                                xdist = 0;
                                ydist = dist;
                        }

                        width = height = 0;
                        size = p->dotlabel_fontsize;
                        /* フォント情報をロード */
                        psLoadFontSize( label, p->dotlabel_fontname,
                                p->dotlabel_fontbold, p->dotlabel_fontitalic,
                                &size, &width, &height);

                        /* X座標を算出する */
                        fx = (p->xdata[i]-p->xrange[0]) /
                                                (p->xrange[1]-p->xrange[0]);
                        /* Y座標を算出する */
                        fy = (p->ydata[i]-p->yrange[0]) /
                                                (p->yrange[1]-p->yrange[0]);

                        /* フォント情報を設定する */
                        grPSSetFont(fp, p->dotlabel_fontname,
                                p->dotlabel_fontbold,
                                p->dotlabel_fontitalic,
                                p->dotlabel_fontsize);

                        /* 各角度に対したラベル表示位置を指定する */
                        if( arg == 0 && arg_flg == 0 ) {
                                /* ラベルの中心指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( arg == 0 ) {
                                /* ラベル左詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( 0 < arg && arg < 90 ) {
                                /* ラベル左詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( arg == 90 ) {
                                /* ラベルの中心指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( 90 < arg && arg < 180 ) {
                                /* ラベル右詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( arg == 180 ) {
                                /* ラベル右詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( 180 < arg && arg < 270 ) {
                                /* ラベル右詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( arg == 270 ) {
                                /* ラベルの中心指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        } else if( 270 < arg && arg < 360 ) {
                                /* ラベル左詰めを指定 */
                                fprintf(fp,
                                "(%s) %.5g %.5g %.5g %.5g setText\n",
                                label, fx, fy, (float)xdist, (float)ydist );
                        }
                }
        }

        return (0);
}

int
psLoadFontSize(label, name, bold, italic, size, width, height)
char    *label;
char    *name;
char    *bold;
char    *italic;
int     *size;
int     *width;
int     *height;
{
        char    buf[BUFSIZ];
        static char     *fns[] = GR_FONT_NAME_RANGE, **p;
```

```
        /* フォント名のチェック */
        if (!name)
                name = GR_DEFAULT_FONTNAME;
        else {
                p = fns;
                while (*p) {
                        if (!strcmp(name, *p))
                                goto next;
                        p++;
                }
                fprintf(stderr, "WARNING: Can't find font name \"%s\"\n", name);
                name = GR_DEFAULT_FONTNAME;
        }
next:
        /* フォントタイプのチェック */
        if (!bold)
                bold = GR_DEFAULT_FONTBOLD;

        /* サイズのチェック */
        if      (*size <=  8) *size =  8;
        else if (*size <= 10) *size = 10;
        else if (*size <= 12) *size = 12;
        else if (*size <= 14) *size = 14;
        else if (*size <= 18) *size = 18;
        else if (*size <= 48) *size = 24;
        else                  *size = GR_DEFAULT_FONTSIZE;

        *width = (*size) * strlen( label );
        *height = (*size);

        return(0);
}
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *               Version 2.0
 *    (c) Copyright 1997,1998
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * read.c - データファイル読み込み
 */

#include <stdio.h>

#include <string.h>
#define __gColorTable__
/*#define READDBG*/
#include "plot.h"
#include "graph.h"

static int gLineNo;                              /* データファイルの行番号 */

/*
 * データファイルの読み込み
 */
DATA *ReadDataFile(char *datafname, char *headerfname, char *bodyfname)
{
  FILE *fp;
  char *buf, *p;
  DATA *data;
  int i, j;
  double val;
  int   cnt_labeldata = 0;      /* ラベル列の数 */
  char  val_lbl[MAX_LABELDATA_LEN];
  char  filename[BUFSIZ+1];
#ifdef DEBUG
  fprintf(stderr, "ReadDataFile(%s)\n", filename);
#endif

  /* ボディーデータはヘッダとの組み合わせでのみ使用可能 */
  if( strlen( bodyfname ) > 0 && strlen( headerfname) == 0 ) {
      usage();
      return( NULL );
  }

  /* ヘッダファイル指定の時は、ヘッダファイルをオープンする */
  if( strlen( headerfname ) > 0 ) {
      strcpy( filename, headerfname );
  }else{
      strcpy( filename, datafname );
  }

  /* データファイルのオープン （ヘッダファイル指定の時はヘッダファイル
     をオープン:VERSION2 )*/
  if ((fp = fopen(filename, "r")) == NULL) {
    fprintf(stderr, "Cannot open file '%s'.\n", filename);
    return(NULL);
  }

  /* データの領域確保 */
  if ((data = (DATA *)malloc(sizeof(DATA))) == NULL) {
    fprintf(stderr, "Cannot allocate memory.\n");
    fclose(fp);
    return(NULL);
  }

  /* タイトル文字列の読み込み */
  if ((buf = ReadLine(fp)) == NULL) {
    fprintf(stderr, "%s:%d: unexpected end of file\n",
            filename, gLineNo);
    return(NULL);
  }

#ifdef DEBUG
  fprintf(stderr, "[%s]", buf);
#endif
/*  if (strncmp(buf, HEADER_TITLE, strlen(HEADER_TITLE))) {
    fprintf(stderr, "Not found title string.\n");
    return(NULL);
  }*/
/*  strcpy(data->title, buf+strlen(HEADER_TITLE));*/
  strcpy(data->title, buf);

  /* グラフの数、データの行数、データ列数の読み込み */
  if ((buf = ReadLine(fp)) == NULL) return(NULL);
  sscanf(buf, "%d %d %d",
         &(data->nGraphs), &(data->nRows), &(data->nCols));
#ifdef DEBUG
  fprintf(stderr, "nGraphs=%d,nRows=%d,nCols=%d\n",
          data->nGraphs, data->nRows, data->nCols);
#endif

  /* チェック */
  if (data->nGraphs <= 0 || data->nGraphs > MAX_GRAPH_COUNT) {
    fprintf(stderr, "%s:%d: number of graphs is out of range (%d)\n",
            filename, gLineNo, data->nGraphs);
    fclose(fp);
    return(NULL);
  }
  if (data->nCols <= 0) {
    fprintf(stderr, "%s:%d: number of columns is less than 0 (%d)\n",
            filename, gLineNo, data->nCols);
    fclose(fp);
    return(NULL);
  }
  if (data->nRows <= 0) {
    fprintf(stderr, "%s:%d: number of rows is less than 0 (%d)\n",
            filename, gLineNo, data->nRows);
    fclose(fp);
    return(NULL);
  }

  /* グラフのX/Y軸の列番号、線種の読み込み
     線種指定は以下の指定が可能。
       0    折線
       1    点
       2    点線
       3    1点鎖線
       4    長点線
       5    長1点鎖線

     オプション指定の読み込み
        オプション指定は省略可能。指定の方法は、<属性>=<値>とし、属性は複数設定で
     きる。同一属性が複数回指定された場合は、最後の指定が有効となる。属性の種類
     は以下の通り。
       ・label          ラベル列指定。データ列の列番号を指定する。ラベル列は必ずし
                        も列ラベルで@label指定した列でなくてもよい。
       ・dotsize        線幅、点の大きさの指定。デフォルト値は1。
       ・dotcolor       グラフの色の指定。指定の方法は、カラー番号(ヘッダーで定義
                        済みの10色程度の色番号)の他、"#RRGGBB"のようにRGB値での指
                        定も可能。
       ・labelsize      ラベル文字列の大きさの指定。
       ・duplicate      duplicate=1の場合、直前のグラフ上に描画する。
                        複数行に渡って指定可能。
       ・labeldist      ラベル文字列を描画する位置の指定。
                        どれだけデータ点と離すかを指定する。
         例) label=3 dotsize=2 dotcolor=1 duplicate=1
  */
  data->nWindows = 0;
```

```
  for (i = 0;i < data->nGraphs;i++) {
    if ((buf = ReadLine(fp)) == NULL) return(NULL);
    sscanf(buf, "%d %d %d",
           &(data->colX[i]), &(data->colY[i]), &(data->lineType[i]));

    /* チェック */
    if (data->colX[i] < 1 || data->colX[i] > data->nCols) {
      fprintf(stderr, "%s:%d: number of x column no. is out of range (%d)\n",
                      filename, gLineNo, data->colX[i]);
      fclose(fp);
      return(NULL);
    }
    if (data->colY[i] < 1 || data->colY[i] > data->nCols) {
      fprintf(stderr, "%s:%d: number of y column no. is out of range (%d)\n",
                      filename, gLineNo, data->colY[i]);
      fclose(fp);
      return(NULL);
    }

    /* グラフのX/Y軸の列番号、線種に続くオプション指定の数を求める */
    if (CheckGraphOption(data, i, buf) < 0) {
      fprintf(stderr, "%s:%d: graph option error\n",
              filename, gLineNo);
      fclose(fp);
      return(NULL);
    }

    /* ウィンドウ数のカウント */
    if (!data->duplicate[i]) {
      data->nWindows++;
    }
  }

  if (data->nWindows < 1 || data->nWindows > MAX_WINDOW_COUNT) {
    fprintf(stderr, "%s:%d: number of windows is out of range (%d)\n",
                    filename, gLineNo, data->nWindows);
    fclose(fp);
    return(NULL);
  }

  /* グラフのX/Y軸のラベルの取得 */
  for (i = 0;i < data->nCols;i++) {
    if ((buf = ReadLine(fp)) == NULL) {
      fprintf(stderr, "%s:%d: unexpected end of file\n",
              filename, gLineNo);
      return(NULL);
    }
    strncpy(data->colHeader[i], buf, MAX_HEADER_LEN);
    /* ラベル列の数を数える */
    if( strcmp( data->colHeader[i], "@label" ) == 0 ){
      cnt_labeldata++;
    }
#ifdef DEBUG
    fprintf(stderr, "label %d: %s\n", i, data->colHeader[i]);
#endif
  }

  /* 指定データ列がラベル列になってないかチェックする */
  for (i = 0; i < data->nGraphs; i++ ){
      if( strcmp( data->colHeader[data->colX[i]-1], "@label" ) == 0 ){
          fprintf( stderr, "order X column[%d] is Label Data column.\n",
                   data->colX[i] );
          fclose(fp);
          return (NULL);
      }
      if( strcmp( data->colHeader[data->colY[i]-1], "@label" ) == 0 ){
          fprintf( stderr, "order Y column[%d] is Label Data column.\n",
                   data->colY[i] );
          fclose(fp);
          return (NULL);
      }
  }

  /* データ領域の確保 */
  if ((data->table = (Number *)malloc(sizeof(Number)*(data->nCols*data->nRows))) == NULL)
{
    fprintf(stderr, "Cannot allocate memory.(%dx%d)\n",
            data->nCols, data->nRows);
    fclose(fp);
    return(NULL);
  }
  /* ラベルデータ領域の確保 */
  if( cnt_labeldata > 0 ){
      if( (data->table_label = (char *)malloc(
        sizeof(char)*cnt_labeldata*data->nRows*MAX_LABELDATA_LEN )) == NULL ){
          fprintf( stderr, "Cannot allocate label data memory.(%dx%d)\n",
                   cnt_labeldata, data->nRows );
          fclose(fp);
          return (NULL);
      }else{
          memset( data->table_label, 0, sizeof( data->table_label ) );
      }
  }
  /* ヘッダファイル指定の時はファイルを閉じ、データファイルをオープンする */
  if( strlen( headerfname ) > 0 ) {
      fclose(fp);
      if( strlen( bodyfname ) > 0 ) {
        strcpy( filename, bodyfname );
      }
      else {
        strcpy( filename, datafname );
      }
      gLineNo = 0;
      /* データファイルのオープン */
      if ((fp = fopen(filename, "r")) == NULL) {
          fprintf(stderr, "Cannot open file '%s'.\n", filename);
          return(NULL);
      }

      /* データファイル指定時のみヘッダを読み飛ばす */
      if( strlen( bodyfname ) == 0 ) {
        /* データファイルにヘッダが含まれているときは、そこを読み飛ばす */
        if( fpSetToData(fp, data) == -1 ) return (NULL);
      }
  }

  /* データの読み込み */
  for (i = 0;i < data->nRows;i++) {

    /* データを一行読み込む */
    if ((buf = ReadLine(fp)) == NULL) {
      fprintf(stderr, "%s:%d: Warning: valid rows less than previous.(%d<%d)\n",
              filename, gLineNo, i, data->nRows);
      break;
    }
    if (strncmp(buf, STOP_STRING, strlen(STOP_STRING)) == 0) {
      break;
    }

    p = buf;
    cnt_labeldata = 0;
    for (j = 0;j < data->nCols;j++) {
      if (*p == '\0') {
        fprintf(stderr, "%s:%d: valid columns less than previous.(%d<%d)\n",
                filename, gLineNo, j, data->nCols);
        fclose(fp);
        return(NULL);
      }
      if( strcmp( data->colHeader[j], "@label" ) != 0 ){
```

```
            sscanf( p, "%lf", &val );
            /* 列優先でデータを格納する */
            *(data->table + j*data->nRows + i) = (Number)val;
        }else{
            sscanf( p, "%s", val_lbl );
            strcpy( (data->table_label +
                     (cnt_labeldata*data->nRows+i)*MAX_LABELDATA_LEN),
                    val_lbl );
            cnt_labeldata++;
        }
#ifdef DEBUG
        fprintf(stderr, "%s(%lf),", p, val);
#endif

        /* 次の空白文字までよみ飛ばす */
        while (*p != ' ' && *p != '\t' && *p != '\n' && *p != '\0') p++;

        /* 次の空白以外の文字までよみ飛ばす */
        while (*p == ' ' || *p == '\t' || *p == '\n') p++;

      }
    }
    data->nValidRows = i;

    fclose(fp);
    return(data);
}

/*
 * ファイルから一行よみ出す
 *      空行をよみ飛ばすことに注意
 *      1カラム目が"!"で始まる行はコメント行と見なして読み飛ばす
 *                                                  (Version 2.0)
 */
char *ReadLine(FILE *fp)
{
    static char buf[MAX_BUFFER_LEN+1];
    char *p;

/*  gLineNo = 0;*/
    while (fgets(buf, MAX_BUFFER_LEN, fp)) {
      gLineNo++;
      p = buf;
      while (*p == ' ' || *p == '\t' || *p == '\n') p++;
      if ((*p != '\0') && (*p != '!')) {
        if (p[strlen(p)-1] == '\n')
          p[strlen(p)-1] = '\0';
        return(p);
      }
    }

    fclose(fp);
    return(NULL);
}

/*
        グラフ定義のオプション指定を設定する
*/
int
SetGraphOption(DATA *data, int n, char *opt)
{
    char  zoku[MAX_OPTION_LEN];
    char  *c;
    int   i;
    int   num;
    float wnum;

    if ((c = StringIndex(opt, '=')) != NULL) {
      i = strlen(opt)-strlen(c);
          strncpy(zoku, opt, i);
          zoku[i] = '\0';
          if (!strcmp(zoku, LABEL_OPTION) || !strcmp(zoku, LABEL_OPT_SHORT)) {
#ifdef READDBG
              printf("%s:%d\n", zoku, atoi(c+1));
#endif
              num = atoi(c+1);
              if( num < 0 || num > data->nCols ){
                  fprintf( stderr, "%s=%d is wrong\n", zoku, num );
                  data->colLabel[n] = -1;
              }else{
                  data->colLabel[n] = num;
              }
          } else if (!strcmp(zoku, DOT_SIZE_OPTION) ||
                     !strcmp(zoku, DOT_SIZE_OPT_SHORT)) {
              sscanf(c+1, "%f", &wnum);
#ifdef READDBG
              printf("%s:%f\n", zoku, wnum);
#endif
              if( wnum >= 0 ){
                  data->dotSize[n] = wnum;
              }else{
                  fprintf( stderr, "%s=%f is wrong\n", zoku, wnum );
              }
          } else if (!strcmp(zoku, DOT_COLOR_OPTION) ||
                     !strcmp(zoku, DOT_COLOR_OPT_SHORT)) {
              if ((c+1)[0] == '#') {
                  sprintf(data->dot_color[n], "%s", c+1);
              } else {
                  num = atoi(c+1);
                  if( num < 0 || num > MAX_COLOR_TABLE-1 ){
                      fprintf( stderr, "%s=%d is wrong\n", zoku, num );
                      sprintf( data->dot_color[n], "%s", gColorTable[0] );
                  }else{
                      sprintf(data->dot_color[n], "%s", gColorTable[num]);
                  }
              }
#ifdef READDBG
              printf( "dotcolor = %s\n", data->dot_color[n]);
#endif
          } else if (!strcmp(zoku, LABEL_SIZE_OPTION) ||
                     !strcmp(zoku, LABEL_SIZE_OPT_SHORT)) {
#ifdef READDBG
              printf("%s:%d\n", zoku, atoi(c+1));
#endif
              num = atoi(c+1);
              if( num > 0 ){
                  data->dotlabel_fontsize[n] = num;
              }else{
                  fprintf( stderr, "%s=%d is wrong\n", zoku, num );
              }
          } else if (!strcmp(zoku, DUPLICATE_OPTION) ||
                     !strcmp(zoku, DUPLICATE_OPT_SHORT)) {
#ifdef READDBG
              printf("%s:%d\n", zoku, atoi(c+1));
#endif
              num = atoi(c+1);
              if (num == 1 && n > 0 ) {
                  data->duplicate[n] = 1;
              }else if( num != 0 ){
                  fprintf( stderr, "%s=%d is wrong\n", zoku, num );
              }
          } else if (!strcmp(zoku, LABEL_DIST_OPTION) ||
                     !strcmp(zoku, LABEL_DIST_OPT_SHORT)) {
#ifdef READDBG
              printf("%s:%d\n", zoku, atoi(c+1));
#endif
              num = atoi(c+1);
              if( num >= 0 ){
                  data->dotlabel_dist[n] = num;
              }else{
```

```
                fprintf( stderr, "%s=%d is wrong\n", zoku, num );
            }
        }
        else {
          fprintf( stderr, "%s is invalid option\n", zoku );
          return(-1);
        }
          return(0);
    } else {
          fprintf( stderr, "%s is invalid option format\n", opt );
          return(-1);
    }

}

/*
        グラフ定義のグラフのX/Y軸の列番号、線種に続くオプション指定を調べる
*/
int
CheckGraphOption(DATA *data, int n, char *buf)
{
  int   cnt, char_cnt, total_cnt;
  int   cp;
  char  opt[MAX_OPTION_LEN];
  char  *p, *q;
  int   i;

  /* 初期値設定 */
  sprintf(data->dot_color[n], "%s", gColorTable[0]);
  data->duplicate[n]          = 0;
  data->dotSize[n]            = 1;
  data->colLabel[n]           = -1;
  data->dotlabel_fontsize[n]  = GR_DEFAULT_LABEL_FONTSIZE;
  data->dotlabel_dist[n]      = 0;
  cnt = 0;

  cp = 0;
  char_cnt = 0;
  total_cnt = 0;

  for (i = 0; i <= strlen(buf); i++) {

    /* 空白文字の場合 */
    if ((buf[i] == ' ') || (buf[i] == '\n') ||
        (buf[i] == '\t') || (buf[i] == '\0')) {

      /* 文字がある場合 */
      if (char_cnt > 0) {
        sscanf(&buf[cp], "%s", opt);
        if (total_cnt < 3 && isdigit(*opt)) {
          /* Skip reading */
        }
        else {
          SetGraphOption(data, n, opt);
          cnt++;
        }
        total_cnt++;
        cp = i;
        char_cnt = 0;
      }

      /* 文字がない場合 */
      else {
        /* Skip */
      }
    }

    /* 空白文字でない場合  */
    else {
      char_cnt++;
    }
  }

/*fprintf(stderr, "\n");*/
  return(cnt);
}

/*
        カラー定義文字列を別ける
*/
int
StringSeparate(char *s, char *color, int cn)
{
  int i;
  char *c;

  if (c = StringIndex(s, ':')) {
        switch (cn) {
        case 0:
          strncpy(color, s, strlen(s)-strlen(c));
          color[strlen(s)-strlen(c)] = '\0';
          break;
        default:
          strcpy(color, c+1);
          break;
        }
        return(0);
  } else {
        return(-1);
  }
}

char *
StringIndex(char *s, char c)
{
  int i;

  for (i = 0; i < strlen(s); i++) {
        if (s[i] == c) {
          return(&s[i]);
        }
  }
  return(NULL);
}

/* データファイルのヘッダを読み飛ばす */
int fpSetToData(FILE *fp, DATA *data)
{
    char *buf;
    int i;
    int graph, row, col;
    int rowcnt = 0;

    /* 2行目まですらす */
    for( i = 0; i < 2; i++ ){
        if ((buf = ReadLine(fp)) == NULL) return (-1);
        rowcnt++;
    }

    /* グラフ数、データ行数、データ列数を求める */
    sscanf( buf, "%d %d %d", &graph, &row, &col );

    /* ヘッダファイルのデータ行数、列数と同じかチェック */
    if( row != data->nRows || col != data->nCols ){
        printf( "file format is different. header file and data file\n" );
        return (-1);
    }

    /* グラフ指定行を読み飛ばす */
    for( i = 0; i < graph; i++ ){
```

```
        if ((buf = ReadLine(fp)) == NULL){
            return (-1);
        }
        rowcnt++;
    }

    /* グラフのX/Y軸ラベル行を読み飛ばす */
    for( i = 0; i < col; i++ ){
        if ((buf = ReadLine(fp)) == NULL){
            return (-1);
        }
        rowcnt++;
    }

    return (1);
}
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *             Version 2.0
 *    (c) Copyright 1997,1998
 *    ATR Adaptive Communications Research Laboratories
 *    All Rights Reserved
 */


/*
 * plot.h - プロット情報定義ファイル
 */
#ifndef __plot_h__
#define __plot_h__
#include <stdio.h>
#include <X11/Xlib.h>

#define Number                  float           /* 数値演算の精度 */

/* 用紙方向の定数 */
#define PAPER_LANDSCAPE         0               /* 横置き */
#define PAPER_PORTRAIT          1               /* 縦置き */

/* モード */
#define X_DRAW_MODE             0               /* X描画モード */
#define PS_ONLY_MODE            1               /* PS出力モード */
#define EPS_ONLY_MODE           2               /* EPS出力モード */

#define MAX_TITLE_LEN           256             /* タイトルの最大長さ */
#define MAX_HEADER_LEN          256             /* ヘッダの最大長さ */

#define MAX_GRAPH_COUNT         100             /* グラフの最大数 */
#define MAX_COLUMN_COUNT        100             /* 列の最大数 */
#define STOP_CHAR               '@'             /* 読み込み停止文字 */
#define STOP_STRING             "@@@@@"         /* 読み込み停止文字列 */
#define COMMENT_CHAR            '!'             /* コメント文字 */
#define MAX_BUFFER_LEN          4096            /* 行バッファの最大長さ */
#define HEADER_TITLE            "タイトル："
#define ulong                   unsigned long   /* 正長整数 */

/* デフォルト値の定義 */
#define DEFAULT_FILENAME        "fort.99"
#define DEFAULT_MAIN_WINDOW_WIDTH  (290*3)      /* 初期ウィンドウの幅 */
#define DEFAULT_MAIN_WINDOW_HEIGHT (200*3)      /* 初期ウィンドウの高さ */
#define DEFAULT_TITLE_WINDOW_HEIGHT 0.06        /* タイトルウィンドウの大きさ(相対値) */
#define DEFAULT_PAPER_ORIENTATION (PAPER_LANDSCAPE) /* 用紙の向き */

/* グラフウィンドウ数 */
#define MAX_WINDOW_COUNT        24              /* 重ね合わせも含めて、表示されるグラフの
数 */
/* オプション指定 */
#define MAX_OPTION_LEN          1024            /* オプション指定の最大長さ */
#define LABEL_OPTION            "label"         /* ラベル列指定 */
#define LABEL_OPT_SHORT         "l"             /* ラベル列指定（短縮形）*/
#define DOT_SIZE_OPTION         "dotsize"       /* 線幅、点の大きさの指定 */
#define DOT_SIZE_OPT_SHORT      "ds"            /* 線幅、点の大きさの指定（短縮形） */
#define DOT_COLOR_OPTION        "dotcolor"      /* グラフの色の指定 */
#define DOT_COLOR_OPT_SHORT     "dc"            /* グラフの色の指定（短縮形） */
#define LABEL_SIZE_OPTION       "labelsize"     /* ラベル文字列の大きさの指定 */
#define LABEL_SIZE_OPT_SHORT    "ls"            /* ラベル文字列の大きさの指定（短縮形） */
#define DUPLICATE_OPTION        "duplicate"     /* 直前のグラフ上への描画指定 */
#define DUPLICATE_OPT_SHORT     "d"             /* 直前のグラフ上への描画指定（短縮形） */
#define LABEL_DIST_OPTION       "labeldistance" /* ラベル文字列描画位置指定 */
#define LABEL_DIST_OPT_SHORT    "ld"            /* ラベル文字列描画位置指定（短縮形）*/
#define DUPLICATE_ON            1               /* 直前のグラフに描画 */
#define DUPLICATE_OFF           0               /* 直前のグラフに描画しない */
/* カラーテーブル */
#define MAX_COLOR_TABLE         10              /* カラーテーブルの数 */
#define MAX_COLOR_LEN           60              /* カラー定義の最大長さ */
#define MAX_LABELDATA_LEN       60              /* ラベルデータの文字列の大きさの最大 */
```

```
#ifdef __gColorTable__
char gColorTable[][MAX_COLOR_LEN] = {
  "black:000000",
  "red:FF0000",
  "blue:0000FF",
  "green:00FF00",
  "pink:FFB0C0",
  "violet:F080F0",
  "yellow:FFFF00",
  "orange:FFA000",
  "brown:A03030",
  "spring green:00FF80"
};
#else
extern char gColorTable[][MAX_COLOR_LEN];
#endif

/* プロット情報の定義 */
typedef struct _DATA {
  Display *display;                             /* ディスプレイ */
  char title[MAX_TITLE_LEN+1];                  /* タイトル */
  int nWindows;                                 /* ウィンドウの数 */
  int nGraphs;                                  /* グラフの数 */
  int nRows;                                    /* データ行数 */
  int nCols;                                    /* データ列数 */
  int nValidRows;                               /* 有効列数 */
  int colX[MAX_GRAPH_COUNT];                    /* X軸の列番号 */
  int colY[MAX_GRAPH_COUNT];                    /* Y軸の列番号 */
  int colLabel[MAX_GRAPH_COUNT];                /* ラベル列の列番号 */
  Number rangeX[MAX_GRAPH_COUNT][2];            /* X軸の範囲 */
  Number rangeY[MAX_GRAPH_COUNT][2];            /* Y軸の範囲 */
  int lineType[MAX_GRAPH_COUNT];                /* グラフの線種 */
  char colHeader[MAX_COLUMN_COUNT][MAX_HEADER_LEN+1];  /* グラフのヘッダ */
  Window winId[MAX_GRAPH_COUNT];                /* ウィンドウのID */
  int winX[MAX_GRAPH_COUNT];                    /* ウィンドウの左上のx座標 */
  int winY[MAX_GRAPH_COUNT];                    /* ウィンドウの左上のy座標 */
  unsigned int winWidth[MAX_GRAPH_COUNT];       /* 各グラフウィンドウの幅 */
  unsigned int winHeight[MAX_GRAPH_COUNT];      /* 各グラフウィンドウの高 */
  Number *table;                                /* データの配列 */
  char *table_label;                            /* ラベルデータの配列 */
  int   dotlabel_fontsize[MAX_GRAPH_COUNT];         /* ラベルデータの
                                                   文字列の大きさ
                                                   ( label_size オプション用) */
  int   dotlabel_dist[MAX_GRAPH_COUNT];         /* ラベルデータ位置
                                                   ( label_distanceオプション用 ) */
  char  dot_color[MAX_GRAPH_COUNT][MAX_COLOR_LEN];      /* グラフの色の指定
                                                   ( dot_color オプション用 ) */
  int   duplicate[MAX_GRAPH_COUNT];             /* duplicateオプション用 */


  float dotratio;                               /* 点の大きさ
                                                   ( コマンド引数 -dotratio用)  */
  Window mainWinId;                             /* メインウィンドウのID */
  unsigned int mainWinWidth;                    /* メインウィンドウの幅 */
  unsigned int mainWinHeight;                   /* メインウィンドウの高 */
  int orientation;                              /* 用紙／画面の方向 */
  Window titleWinId;                            /* タイトルウィンドウのID */
  int titleWinX;                                /* タイトルウィンドウのx座標 */
  int titleWinY;                                /* タイトルウィンドウのy座標 */
  unsigned int titleWinWidth;                   /* タイトルウィンドウの幅 */
  unsigned int titleWinHeight;                  /* タイトルウィンドウの高 */
  int mode;                                     /* 描画/出力モード */
  float aspectRatio;                            /* 縦横比 */
  float dotSize[MAX_GRAPH_COUNT];               /* 点の大きさ */
} DATA;

#if 0
/* プロット情報の初期化 */
#define INIT_DATA(d) \
    {   int i; \
```

```
        (d)->title[0] = '\0'; \
        (d)->nGraphs = 0; \
        (d)->nRows = 0; \
        (d)->nCols = 0; \
        (d)->nValidRows = 0; \
        for (i = 0;i < MAX_GRAPH_COUNT;i++) { \
          (d)->colX[i] = 0; \
          (d)->colY[i] = 0; \
          (d)->rangeX[i][0] = 0; \
          (d)->rangeX[i][1] = 1; \
          (d)->rangeY[i][0] = 0; \
          (d)->rangeY[i][1] = 1; \
          (d)->lineType[i] = 0; \
          (d)->winId[i] = 0; \
          (d)->winX[i] = 0; \
          (d)->winY[i] = 0; \
          (d)->winWidth[i] = 1; \
          (d)->winHeight[i] = 1; \
        } \
        for (i = 0;i < MAX_COLUMN_COUNT;i++) { \
          (d)->colHeader[i][0] = '\0'; \
        } \
        (d)->table = NULL; \
        (d)->mainWinId = 0; \
        (d)->mainWinWidth = 1; \
        (d)->mainWinHeight = 1; \
        (d)->orientation = DEFAULT_PAPER_ORIENTATION; \
        (d)->titleWinId = 0; \
        (d)->titleWinX = 1; \
        (d)->titleWinY = 1; \
        (d)->titleWinWidth = 1; \
        (d)->titleWinHeight = 1; \
        (d)->mode = X_DRAW_MODE; \
        (d)->aspectRatio = -1; \
        (d)->dotSize = 0.8; \
    }
#endif

DATA *ReadDataFile(char *, char *, char *);
char *ReadLine(FILE *);
int      StringSeparate(char *, char *, int);
int  SetGraphOption(DATA *, int, char *);
int  CheckGraphOption(DATA *, int, char *);
int  StringSeparate(char *, char *, int);
char *StringIndex(char *, char);
int fpSetToData(FILE *, DATA *);

#endif  /* end of __plot_h__ */
```

```
/*
 * ACR PLOT TOOL Version 1.0
 *     (c) Copyright 1997
 *     ATR Adaptive Communications Research Laboratories
 *     All Rights Reserved
 */


/*
 * graph.h - グラフ情報の定義ファイル
 */

#ifndef __graph_h__
#define __graph_h__

#define Number  float
#define Graph                   int             /* グラフハンドル */

#define GR_MAX_GRAPH            100
#define GR_INF                  1000000
#define GR_EPS                  1.0e-6
#define GR_DEFFONTNAME  "-adobe-helvetica-bold-r-normal--0-0-75-75-p-0-iso8859-1"

#define GR_INT                  1
#define GR_FLOAT                2
#define GR_CHAR                 3
#define GR_STRING               4
#define GR_ADDR                 5
#define GR_DOUBLE               6
#define GR_SHORT                7

#define GR_TYPE_XY              1
#define GR_TYPE_METER           2

#define GR_X                            1
#define GR_Y                            2
#define GR_WIDTH                        3
#define GR_HEIGHT                       4
#define GR_XDATA                        5
#define GR_XDATATYPE                    6
#define GR_YDATA                        7
#define GR_DATACOUNT                    8
#define GR_YDATATYPE                    9
#define GR_TITLE                        10
#define GR_WINDOW                       11
#define GR_BOX                          12
#define GR_XTICKS                       13
#define GR_NUM_XTICKS                   14
#define GR_YTICKS                       15
#define GR_NUM_YTICKS                   16
#define GR_XAXIS                        17
#define GR_YAXIS                        18
#define GR_XLABEL                       19
#define GR_YLABEL                       20
#define GR_DECORATION_COLOR             21
#define GR_XRANGE                       22
#define GR_YRANGE                       23
#define GR_FONTNAME                     24
#define GR_FONTBOLD                     25
#define GR_FONTITALIC                   26
#define GR_FONTSIZE                     27
#define GR_TITLE_FONTNAME               28
#define GR_TITLE_FONTBOLD               29
#define GR_TITLE_FONTITALIC             30
#define GR_TITLE_FONTSIZE               31
#define GR_TITLE_COLOR                  32
#define GR_LABEL_FONTNAME               33
#define GR_LABEL_FONTBOLD               34
#define GR_LABEL_FONTITALIC             35
#define GR_LABEL_FONTSIZE               36
#define GR_LABEL_COLOR                  37
#define GR_DRAW_MODE                    38
#define GR_DRAW_SIZE                    39
#define GR_HOLD                         40
#define GR_MAX_EXP                      41
#define GR_GRAPH_TYPE                   42
#define GR_DECORATION_TYPE              43
#define GR_LINE_STYLE                   44
#define GR_TICK_FONTNAME                45  /* 目盛りフォント名 */
#define GR_TICK_FONTBOLD                46  /* 目盛りフォント太さ */
#define GR_TICK_FONTITALIC              47  /* 目盛りフォント傾斜 */
#define GR_TICK_FONTSIZE                48  /* 目盛りフォントサイズ */
#define GR_TICK_COLOR                   49  /* 目盛りフォント色 */
#define GR_TICKSUB_FONTNAME             50  /* 目盛りフォント名 */
#define GR_TICKSUB_FONTBOLD             51  /* 目盛りフォント太さ */
#define GR_TICKSUB_FONTITALIC           52  /* 目盛りフォント傾斜 */
#define GR_TICKSUB_FONTSIZE             53  /* 目盛りフォントサイズ */
#define GR_TICKSUB_COLOR                54  /* 目盛りフォント色 */
#define GR_LINE_WIDTH                   55
#define GR_ASPECT_RATIO                 56
#define GR_DOT_SIZE                     57
#define GR_DUPLICATE                    58
#define GR_DOTCOLOR                     59
#define GR_DOTLABEL_SIZE                60
#define GR_DOTLABEL_DIST                61

#define GR_LINE_STYLE_LINE              "-"
#define GR_LINE_STYLE_DOT               "."
#define GR_LINE_STYLE_DOTTEDLINE        "-."
#define GR_LINE_STYLE_DASH              "--"
#define GR_LINE_STYLE_ROUND             "o"
#define GR_LINE_STYLE_X                 "x"
#define GR_LINE_STYLE_DOTTEDLINE_LONG   "-.-."
#define GR_LINE_STYLE_DASH_LONG         "----"
/*
 * グラフ情報宣言
 */
typedef struct {
    int         on;                     /* 有効フラグ */
    Display     *disp;                  /* ディスプレイ構造体 */
    Window      win;                    /* Xウィンドウ ID */
    float       x;                      /* 左上相対座標 */
    float       y;                      /* 左上相対座標 */
    float       width;                  /* 相対幅 */
    float       height;                 /* 相対高さ */
    char        line_style[3];          /* 線種 */
    float       line_width;             /* 線幅 */
    float       dot_size;
    Number      *xdata;                 /* xデータ */
    int         xdatatype;              /*  */
    Number      xrange[2];              /* xデータ範囲 */
    Number      *ydata;                 /* yデータ配列 */
    int         ydatatype;              /*  */
    int         datacount;              /* データ数 */
    float       yrange[2];              /* yデータ範囲 */
    char        *labeldata_cp;          /* ラベルデータ配列
                                           (ラベル列をラベルとするとき) */
    float       *labeldata_fp;          /* ラベルデータ配列
                                           (データ列をラベルとするとき) */
    char        *dotlabel_fontname;     /* ラベルデータフォント名 */
    char        *dotlabel_fontbold;     /* ラベルデータフォント太さ */
    char        *dotlabel_fontitalic;   /* ラベルデータフォント傾斜 */
    int         dotlabel_fontsize;      /* ラベルデータフォントサイズ */
    XFontStruct *dotlabel_font;         /* ラベルデータフォント構造体 */
    GC          dotlabel_gc;            /* ラベルデータフォント GC */
    char        *dotlabel_color;        /* ラベルデータの色 */
    unsigned long dotlabel_pixel;       /* ラベルデータのピクセル値 */
    GC          dot_gc;                 /* 線、点描画用GC */
    char        *dot_color;             /* 線、点の色 */
    unsigned long dot_pixel;            /* 線、点のピクセル値 */
```

```
    int         dotlabel_dist;              /* ラベルデータ位置 */
    int         duplicate;                  /* グラフを重ねるか示すフラグ */
    char        *title;                     /* タイトル文字列 */
    int         box;                        /* ボックスフラグ */
    float       aspect_ratio;               /* 縦横比 */
    GC          foregc;                     /* 前景GC */
    GC          backgc;                     /* 後景GC */
    char        *fontname;                  /* フォント名 */
    char        *fontbold;                  /* フォント太さ */
    char        *fontitalic;                /* フォント傾き */
    int         fontsize;                   /* フォントサイズ */
    float       *xticks;                    /* x目盛り配列 */
    int         num_xticks;                 /* x目盛り配列数 */
    float       *yticks;                    /* y目盛り配列 */
    int         num_yticks;                 /* y目盛り配列数 */
    int         xaxis;                      /*  */
    int         yaxis;                      /*  */
    char        *xlabel;                    /* xラベル */
    char        *ylabel;                    /* yラベル */
    char        **xtick_marks;              /* x目盛り文字列 */
    int         num_xtick_marks;            /* x目盛り文字列数 */
    char        **ytick_marks;              /* y目盛り文字列 */
    int         num_ytick_marks;            /* y目盛り文字列数 */
    float       *xticks_minor;              /* x Minor目盛り配列 */
    int         num_xticks_minor;           /* x Minor目盛り配列数 */
    float       *yticks_minor;              /* y Minor目盛り配列 */
    int         num_yticks_minor;           /* y Minor目盛り配列数 */
    char        *xtick_scale;               /* x スケール文字列 */
    char        *ytick_scale;               /* y スケール文字列 */
    int         decoration_type;            /* 装飾タイプ */
    char        *decoration_color;          /* 装飾色 */
    unsigned long decoration_pixel[3];      /* 装飾ピクセル値 */
    char        *title_fontname;            /* タイトルフォント名 */
    char        *title_fontbold;            /* タイトルフォント太さ */
    char        *title_fontitalic;          /* タイトルフォント傾斜 */
    int         title_fontsize;             /* タイトルフォントサイズ */
    XFontStruct *title_font;                /* タイトルフォント構造体 */
    char        *title_color;               /* タイトルフォント色 */
    unsigned long title_pixel;              /* タイトルフォントピクセル値 */
    GC          title_gc;                   /* タイトルフォントGC */
    char        *label_fontname;            /* ラベルフォント名 */
    char        *label_fontbold;            /* ラベルフォント太さ */
    char        *label_fontitalic;          /* ラベルフォント傾斜 */
    int         label_fontsize;             /* ラベルフォントサイズ */
    XFontStruct *label_font;                /* ラベルフォント構造体 */
    char        *label_color;               /* ラベルフォント色 */
    unsigned long label_pixel;              /* ラベルフォントピクセル値 */
    GC          label_gc;                   /* ラベルフォントGC */
    char        *tick_fontname;             /* 目盛りフォント名 */
    char        *tick_fontbold;             /* 目盛りフォント太さ */
    char        *tick_fontitalic;           /* 目盛りフォント傾斜 */
    int         tick_fontsize;              /* 目盛りフォントサイズ */
    XFontStruct *tick_font;                 /* 目盛りフォント構造体 */
    char        *tick_color;                /* 目盛りフォント色 */
    unsigned long tick_pixel;               /* 目盛りフォントピクセル値 */
    GC          tick_gc;                    /* 目盛りフォントGC */
    char        *ticksub_fontname;          /* 目盛り(上付)フォント名 */
    char        *ticksub_fontbold;          /* 目盛り(上付)フォント太さ */
    char        *ticksub_fontitalic;        /* 目盛り(上付)フォント傾斜 */
    int         ticksub_fontsize;           /* 目盛り(上付)フォントサイズ */
    XFontStruct *ticksub_font;              /* 目盛り(上付)フォント構造体 */
    char        *ticksub_color;             /* 目盛り(上付)フォント色 */
    unsigned long ticksub_pixel;            /* 目盛り(上付)フォントピクセル値 */
    GC          ticksub_gc;                 /* 目盛り(上付)フォントGC */
    int         draw_mode;                  /* 描画モード */
    int         draw_size;                  /* 描画サイズ */
    float       *draw_buffer;               /* 描画バッファ */
    int         draw_ptr;                   /* 描画ポインタ */
    int         hold;                       /* 設定値の保持フラグ */
    int         max_exp;                    /* 目盛りのスケール値 */
    int         graph_type;                 /* グラフの型 */
    /* private */
    int         Ww;                         /* ウィンドウ幅 */
    int         Wh;                         /* ウィンドウ高 */
    int         Ax;                         /* 描画領域x座標 */
    int         Ay;                         /* 描画領域y座標 */
    int         Aw;                         /* 描画領域幅 */
    int         Ah;                         /* 描画領域高 */
    int         Rx;                         /* グラフ領域x座標 */
    int         Ry;                         /* グラフ領域y座標 */
    int         Rw;                         /* グラフ領域幅 */
    int         Rh;                         /* グラフ領域高 */
    int         Ox;                         /* プロット領域x座標 */
    int         Oy;                         /* プロット領域y座標 */
    int         Ow;                         /* プロット領域幅 */
    int         Oh;                         /* プロット領域高 */
    GC          decogc;
    XFontStruct *font;
} GrAttr;

#define GR_MODE_NORMAL                  1
#define GR_MODE_ADD                     2
#define GR_MODE_FLOW                    3

#define GR_FONT_FORMAT "-*-%s-%s-%c-normal--*-%d-*-*-*-*-*-*"
#define GR_FONT_NAME_RANGE { "helvetica", "times", "lucida", "courier", "symbol", NULL}

#define GR_DEFAULT_FONTNAME             "helvetica"
#define GR_DEFAULT_FONTBOLD             "medium"
#define GR_DEFAULT_FONTITALIC           "roman"
#define GR_DEFAULT_FONTSIZE             10
#define GR_DEFAULT_LABEL_FONTNAME       GR_DEFAULT_FONTNAME
#define GR_DEFAULT_LABEL_FONTBOLD       GR_DEFAULT_FONTBOLD
#define GR_DEFAULT_LABEL_FONTITALIC     GR_DEFAULT_FONTITALIC
#define GR_DEFAULT_LABEL_FONTSIZE       10
#define GR_DEFAULT_TITLE_FONTNAME       GR_DEFAULT_FONTNAME
#define GR_DEFAULT_TITLE_FONTBOLD       "bold"
#define GR_DEFAULT_TITLE_FONTITALIC     GR_DEFAULT_FONTITALIC
#define GR_DEFAULT_TITLE_FONTSIZE       14
#define GR_DEFAULT_TICK_FONTNAME        GR_DEFAULT_FONTNAME
#define GR_DEFAULT_TICK_FONTBOLD        "medium"
#define GR_DEFAULT_TICK_FONTITALIC      "roman"
#define GR_DEFAULT_TICK_FONTSIZE        10
#define GR_DEFAULT_TICKSUB_FONTNAME     GR_DEFAULT_FONTNAME
#define GR_DEFAULT_TICKSUB_FONTBOLD     "medium"
#define GR_DEFAULT_TICKSUB_FONTITALIC   "roman"
#define GR_DEFAULT_TICKSUB_FONTSIZE     8
#define GR_DEFAULT_DRAW_MODE            GR_MODE_NORMAL
#define GR_DEFAULT_MAX_EXP              4
#define GR_DEFAULT_LINE_STYLE           "-"
#define GR_DEFAULT_LINE_WIDTH           0

#define GR_DEFAULT_GRAPH_TYPE           GR_TYPE_XY

#define GR_DASH_PATTERN_DASH_LEN         2
static char gr_dash_pattern_dash[] = {3, 3};
#define GR_DASH_PATTERN_DOTTEDLINE_LEN     4
static char gr_dash_pattern_dottedline[] = {12, 3, 3, 3};
#define GR_DASH_PATTERN_DASH_LONG_LEN    2
static char gr_dash_pattern_dash_long[] = {24, 6};
#define GR_DASH_PATTERN_DOTTEDLINE_LONG_LEN     2
static char gr_dash_pattern_dottedline_long[] = {48, 3, 3, 3};
/* ＰＳでのグラフの各線の長さを定義 */
#define PS_DASH_PATTERN_DASH_LEN            2.0
#define PS_DASH_PATTERN_DOTTEDLINE_LEN      12.0
#define PS_DASH_PATTERN_DASH_LONG_LEN       20.0
#define PS_DASH_PATTERN_DOTTEDLINE_LONG_LEN 12.0*4.0
#ifdef FOR_SGI
#ifndef EXTRN
#define EXTRN extern
```

```
#endif
EXTRN GrAttr    gGraph[GR_MAX_GRAPH];           /* グラフ情報 */
EXTRN int gPSOnlyMode;                         /* PS出力モード */
#else
GrAttr  gGraph[GR_MAX_GRAPH];           /* グラフ情報 */
int gPSOnlyMode;                        /* PS出力モード */
#endif

#define GrGetWindow(gr) (gGraph[(gr)].win)
#define GrSetWindowSize(gr, w, h) (gGraph[(gr)].Ww = (w), gGraph[(gr)].Wh = (h))
#define GrSetPSOnlyMode() (gPSOnlyMode = True)
#define GrResetPSOnlyMode() (gPSOnlyMode = False)

/*
 * プライベート定数の定義
 */
/* 目盛り定数定義 */
#define GR_TICK_NUM_TICKS               5       /* Major目盛りの数 */
#define GR_TICK_MAX_TICKS               20      /* Major目盛りの最大数 */
#define GR_TICK_MAX_TICKS_MINOR         5       /* Minor目盛りの最大数 */
#define GR_TICK_MAX_ALL_TICKS           ((GR_TICK_MAX_TICKS)*(GR_TICK_MAX_TICKS_MINOR)) /*
 目盛りの最大数 */
#define GR_TICK_LENGTH                  0.01    /* Major目盛りの相対長さ */
#define GR_TICK_MINOR_LENGTH            0.005   /* Minor目盛りの相対長さ */

#define GR_POS_DEFAULT_X                0.01    /* ウィンドウの左上の相対位置 */
#define GR_POS_DEFAULT_Y                0.01    /* ウィンドウの左上の相対位置 */
#define GR_POS_DEFAULT_WIDTH            0.98    /* ウィンドウの左上の相対幅 */
#define GR_POS_DEFAULT_HEIGHT           0.98    /* ウィンドウの左上の相対高さ */
#define GR_POS_LABEL_HEIGHT             19      /* ラベル領域の高さ(pixel) */
#define GR_POS_XTICKS_HEIGHT            20      /* x目盛りの高さ(pixel) */
#define GR_POS_YTICKS_WIDTH             28      /* y目盛りの幅(pixel) */
#define GR_POS_XTICKS_MARGIN            0.01    /* x目盛りの相対マージン */
#define GR_POS_YTICKS_MARGIN            8       /* x目盛りのマージン */
#define GR_POS_TITLE_HEIGHT             5       /* タイトル領域の高さ */
#define GR_POS_GRAPH_AREA_LEFT_MARGIN   (GR_POS_LABEL_HEIGHT) /*  */
#define GR_POS_GRAPH_AREA_RIGHT_MARGIN  5       /*  */
#define GR_POS_GRAPH_AREA_TOP_MARGIN    8       /*(GR_POS_LABEL_HEIGHT)*/
#define GR_POS_GRAPH_AREA_BOTTOM_MARGIN (GR_POS_LABEL_HEIGHT) /*  */
#define GR_POS_PLOT_AREA_LEFT_MARGIN    (GR_POS_YTICKS_WIDTH) /*  */
#define GR_POS_PLOT_AREA_RIGHT_MARGIN   5       /*  */
#define GR_POS_PLOT_AREA_TOP_MARGIN     10      /*  */
#define GR_POS_PLOT_AREA_BOTTOM_MARGIN  (GR_POS_XTICKS_HEIGHT) /*  */
#define GR_POS_METER_TICK_MARKS         0.90    /*  */
#define GR_POS_METER_WIDTH_MARGIN       0.10    /*  */
#define GR_POS_METER_HEIGHT_MARGIN      0.05    /*  */
#define GR_POS_METER_MAJOR_TICKS1       0.80    /*  */
#define GR_POS_METER_MAJOR_TICKS2       0.70    /*  */
#define GR_POS_METER_MINOR_TICKS1       0.78    /*  */
#define GR_POS_METER_MINOR_TICKS2       0.72    /*  */
#define GR_POS_METER_STING1             0.65    /*  */
#define GR_POS_METER_STING2             0.00    /*  */
#define GR_COLOR_LOW                    (-0.25) /*  */
#define GR_COLOR_HIGH                   (0.15)  /*  */

#define Free(a)                 {(a) ? ((void)free(a),a=NULL): NULL)

#ifndef exp10
#define exp10(a)  pow(10, (a))
#endif

/*
 * 関数の宣言
 */

#endif /* end of __graph_h__ */
```

A.3 関連ツール

1. 重ね描きデータ列識別用プログラム plotl

グラフを重ね描きした際のデータ列識別用に各データ列の最大値付近にデータ列名をラベルとして表示するプラグラムです。 プロットツールのラベル表示機能を利用しています。

使用法は、プロットツールとまったく同じです。

実処理は、以下の順序で実行されます。

1. plotl <引数> を実行
2. サブプログラム legend を呼び出し、データ列識別用ラベルを含む仮データファイル legend.$$を作成
3. <引数> -data legend.$$をプロットツールの引数とし、プロットツールを実行
4. 仮データファイル legend.$$を削除

plotl は c シェルスクリプト、legend はプロットツールのデータ読み込みルーチンを流用し作成した実行モジュールです。

plotl は、プロットツール中の tools ディレクトリにあります。

tools/
　プロットツール関連ツールディレクトリ

legend/
　重ね描きデータ列識別用プログラム plotl ディレクトリ

plotl
　HP 用の実行ファイル

legend
　HP 用の実行ファイル

plotl_sun
　SUN 用の実行ファイル

legend_sun
　SUN 用の実行ファイル