麻酔・集中治療とテクノロジー 2008

●編集

防衛医科大学校麻酔学講座教授
風間富栄

京都府立医科大学大学院医学研究科
麻酔科学准教授
橋本　悟

京都府立医科大学大学院医学研究科
麻酔科学教授
田中義文

simulation
intranet
software
database
network

序　文

京都府立医科大学の田中義文先生から巻頭の原稿依頼の催促をされ、今更ながら田中先生のテクノロジー学会に対する思い入れの深さを実感させられた。

私は、浜松医科大学名誉教授の池田先生を事務局長として、この研究会が発足した当時から、事務局が現在の京都府立医科大学に移るまで、実務的な一切を行ってきたため、当時の立場上運営について意見を言える立場ではなかったが、この学会を発足するにあたり、いろいろな議論があったことを知る機会があった。本会は発足以来、緩やか成長をしめした時期もあったが、ここ10年は成長ではなく、衰退している感じを否めない。その原因の大きな一つに、発足当時からモニタの分野がこの学会から外されたことである。これは今でも残念でならない。外されたといっても、歴代学会長の中にはその重要性を認識してモニタの分野にテクノロジーの光をあてた会もあった事を記憶している。しかし、学会の柱ではないことから、その試みは継承されず、最近はコンピューター関係の、それもかなり専門性の高い演題が多く、新入会員の伸びを阻害した大きな原因になっていると考える。私が本会の理事になってから、何度かモニタの分野を取り込んだ学会にすべきだと主張してきたが、今もって変わらぬままである。

事務局の運営、学会の維持は極めて大きな忍耐強いエネルギーを必要とする。その原動力、見返りはなんであろうか。やはり、若手医師に対する学会を通した教育に他ならないと思う。この会は、決して趣味を同じくする仲良しグループの会であってはならないと思っている。本会に対する思い入れが大きい田中先生をもってしても、この会の本質を変えることは難しいのであろうか。何故もっとオープンな会にならないのであろうか？コンピューターやIT機器が充実してきた今日、誰でもがモニタや医療機器に工夫ができるはずである。若手医師の更なる奮起と活発な意見、発表を本学会に投げかけてもらいたいものである。

第２５回日本麻酔・集中治療テクノロジー学会
会長 風間 富栄 (防衛医科大学校　麻酔科 教授)

– MEMO –

目　次

第 25 回日本麻酔・集中治療テクノロジー学会論文集

特別講演：テクノロジーからみた血液ガス測定の歴史

諏訪邦夫

2007 年秋に所沢市で行われた日本麻酔・集中治療テクノロジー学会総会で，会長の風間富栄先生の要請で行った講演の記録です．なお，前夜祭で行った『クイズで遊ぶ血液ガス …… 正史以外の問題を』の資料も一部採用しました．機会を与えて下さった風間先生と長い文章の掲載を許可してくださった田中義文先生に感謝します．

実際の講演では,「時代を遡るスタイルで提示」するやり方を採用しました．理由は新しいところをしっかりと話したいと考え，現代の技術と無関係の部分に詳しくなるのを防ぐ目的でしたので，本稿でもそれを採用します．

現代の血液ガス装置

まず，現代の血液ガス装置を概観します．装置はすべてコンピュータ制御で全自動コントロールになっており，医師や検査技師は装置を「使用するだけ」で，ケアする場面はほとんどありません．血液サンプルを注入して蓋を閉じて，あとは数値を打出されるのを待つだけです．

血液ガス装置は，P_{O_2} P_{CO_2} pH などの直接測定するパラメーターの他に,［HCO_3^-］その他の各種演算パラメーターがありますが，これも現在ではすべてコンピュータが自動的に行ってくれます．測定値の記録は当たり前で，病院の各種検査記録と直結している機器も少なくないでしょう．

「血液ガス測定」というと化学電極を用いた測定をさすのがふつうですが，他に分光光度計を用いてヘモグロビン濃度や酸素飽和度などを測定する機器があります．本来は別系統の装置ですが，最近ではこの装置も組み込まれるようになりました．この装置は，一酸化炭素中毒という比較的頻度の高い傷害の検出に威力を発揮する他に，呼吸と循環の動態の分析にも使われます．

帝京短期大学

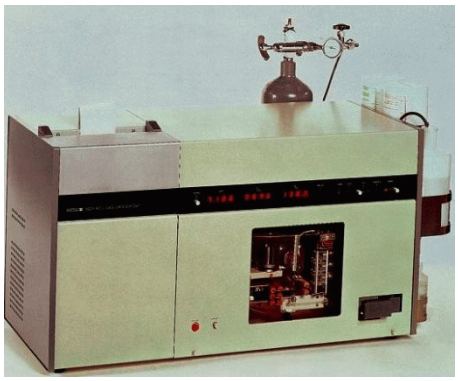


図 1 ラジオメーターの全自動血液ガス測定装置
まったく新しい方式でしたが，便利なだけでなくてトラブルなく動作した点もたいしたものと感心しました．

図 2 インテル 4004MPU 開発者 嶋　正利氏

一方，極端に小型な装置も開発され，患者のすぐ近くでの測定にも使われています．技術的には完成に近づいていると言えますが，消耗部品の費用が高価で「医療費削減」の要求のきつい時代に普及の制限因子となっています．

同じことが，経皮測定の問題に関しても言えます．血液ガス値のうちで特に P_{CO_2} は動脈血と相関の高い数値が経皮で得られ信頼度も高いもので，また乳幼児や新生児はもちろん成人で

も睡眠時呼吸障害などとの関係で用途も少なくないのですが，較正の問題と価格がネックになっています．この装置は，通常の血液ガス測定装置と異なって，患者1人に1台必要な「モニター機器」で廉価は必要条件なのに，その点が普及を妨げています．

ところで，血液ガス測定装置が全自動になったのは，1973年のラジオメーター社の発売が最初でした(図1)．この時，自動化に使用された装置はその少し前に発売されたインテル社のMPU 4004で，嶋正利氏の発明したものです(図2)．インテルと嶋氏の業績はもちろんですが，それを早速医療機器に組み込んでしかも十分に機能を発揮させた血液ガス装置製作会社の先見性に驚かされます．

他の会社がすぐに追従したのはもちろんで，間もなく血液ガス装置はすべてすべて自動化され，この機器とパラメーターは「臨床研究」用から，純粋の臨床検査のパラメーターになりました．因みに私は1976年に『血液ガスの臨床』という書籍を出版し，けっこう面倒なことも書いてあるのに多数の読者を得ましたが，一つの理由が全自動化によってこの測定が急速に普及したことにあったとは，ずっと後になって気づきました．

嶋正利氏は，日本麻酔・集中治療テクノロジー学会でも講演を伺いました．

1970年頃：アナログ機の完成

次に，血液ガス分析装置が手動機器として完成して手術室や集中治療室，それに一般検査室へ普及していく過程を概観します．

全自動機種の開発される直前の1970年頃，血液ガス測定装置は手動のアナログ機器として完成します(図3)．小さな改良が加えられて使いやすくなり，使用者側も知識や認識が増しました．それでも，当時の装置は電極系と電気系は分離していて，電気系はまだアナログでディスプレイだけがディジタル化しています．プリンターはオプションであり，高価だったので使わ

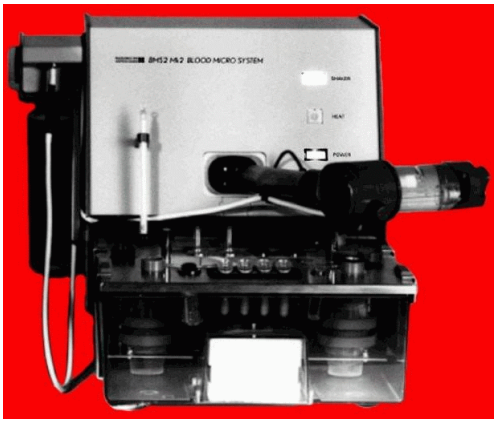


図3 全自動化される直前のアナログ型血液ガス装置 おそらくアナログ，手動式最後のモデルです．

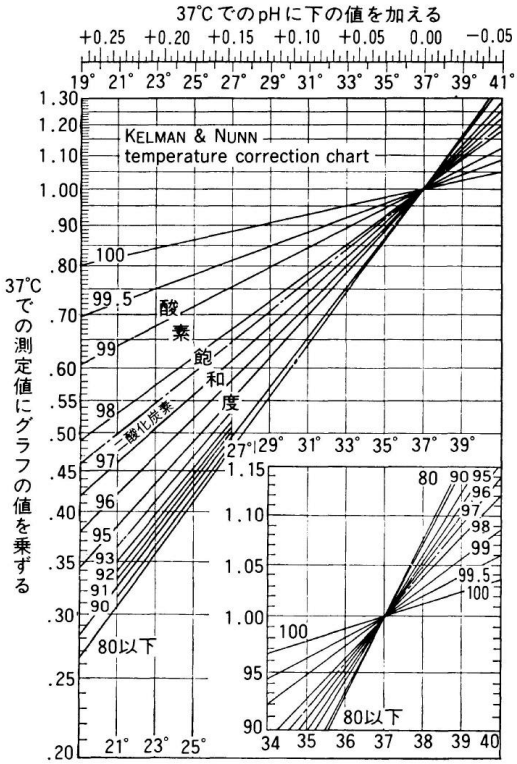


図4 ケルマンとナンのノモグラムの一部

ない場合も多かったようです．[HCO_3^-]のような二次パラメーターや，測定温と体温との差の温度補正などの計算はすべて手作業で，検査室には電卓・計算尺・ノモグラム・各種グラフなどが散乱していました(図4, 5)．

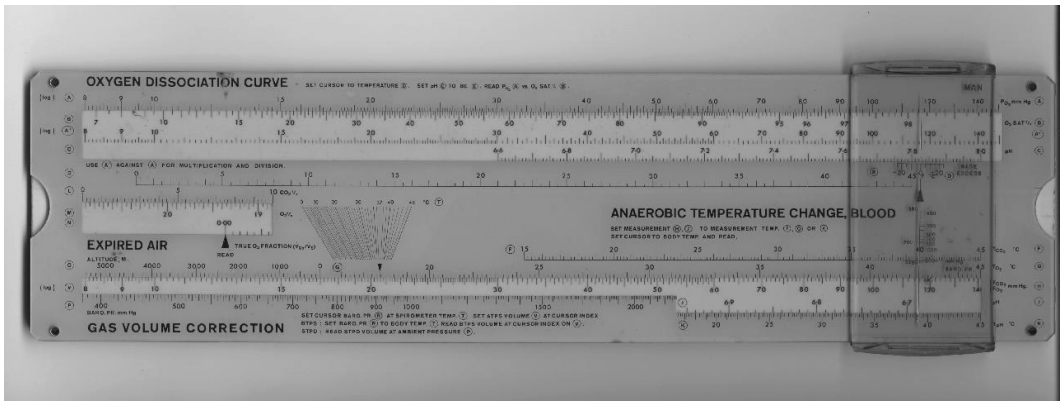


図 5 セブリングハウスの計算尺．たしか 200 ドルくらいでした．

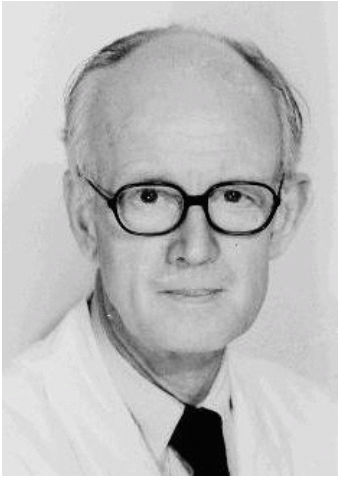


図 6 ベンディクセン先生

生涯の恩師です．2005 年に亡くなられました．

図 7 セブリングハウス先生

ずいぶん勉強させて貰いました．その後も，学会・研究会でよく会って楽しませていただきました．

図 8 レイヴァー先生

ボストン時代の恩師の一人です．ハーバードから出身のスイスのバーゼル大学主任になりました．1984 年に来日直前に 59 歳で亡くなりました．

私は，1969 年から 72 年までの 3 年間，UCSD(カリフォルニア大学サンディエゴ校) の病院で血液ガス測定室の管理を担当しました．これは手術室と外部との接点にあり，当初は手術室と集中治療室の測定だけを担当していましたが，やがて病院全体の血液ガス測定を引き受けることになりました．麻酔科主任のベンディクセン先生の推薦によるものです (図 6)． 当時のサンディエゴはまだ田舎で，会社のサービスを依頼するにはロスアンジェルスから技師を呼ぶ必要があり，結果的に「何でも自分で解決する」ように学習する要因になったでしょう．サンフランシスコとサンディエゴは 600 キロも離れていますが，セブリングハウス先生とは同じカリフォルニア大学同士の直通電話でつながっており，また先生は知っていることは何でも教えてくれ，質問にも答えてくださる方でずいぶん助けて頂きました．今ならメールでしょうが，手紙だったら埒が明かなかったことと推測します (図 7).

JOURNAL OF APPLIED PHYSIOLOGY
Vol. 32, No. 6, June 1972. *Printed in U.S.A.*

Pulmonary gas exchange in a tidally ventilated single alveolus model

KUNIO SUWA AND HENRIK H. BENDIXEN
Anesthesia Laboratory, University of California, San Diego, School of Medicine, La Jolla, California 92037

SUWA, KUNIO, AND HENRIK H. BENDIXEN. *Pulmonary gas exchange in a tidally ventilated single alveolus model.* J. Appl. Physiol. 32(6): 834–841. 1972.—The effects of cyclic fluctuations of alveolar gas composition, secondary to the tidal nature of ventilation, on the alveolar-arterial P_{O_2} difference ((A-a)DO_2)) and arterial-alveolar P_{CO_2} difference ((a-A)DCO_2)) were studied in a single alveolus model; the pulmonary blood flow also was varied with the respiratory cycle. The alveolar gas composition was calculated by numerically solving the instantaneous Fick's principle (expressed as two differential equations), while mean arterial and alveolar values were calculated as ventilation- and blood flow-weighted mean values. With blood flow steady, there were small (A-a)DO_2 and (a-A)DCO_2. The fluctuation of blood flow increased the difference markedly. When blood flow was decreasing toward end inspiration, the differences were positive; when the blood flow was increasing toward end inspiration, they became negative. The differences were influenced by a change in lung volume, tidal volume, and anatomical dead space. In the tidal ventilation, the mean alveolar gas may appear on a gas exchange ratio line lower than the R value. The alveolar air equation may overestimate $P_{A_{O_2}}$.

alveolar gas; tidal ventilation; lung model; alveolar-arterial P_{O_2}; dead space; mean alveolar gas

not the cyclic change in alveolar P_{O_2} and P_{CO_2} during one respiratory cycle influences the overall gas exchange through the lungs; the other is to what extent a cyclic change in blood flow, concurrent with tidal ventilation, influences this gas exchange process. A preliminary study, using a simpler model, revealed that tidal ventilation and fluctuating pulmonary blood flow may indeed cause arterial-to-alveolar P_{CO_2} difference, which may be positive or negative (27). In this study, the model has been elaborated further. A study, similar to this except focusing differently on the ventilatory flow pattern, was reported recently by Nye (19).

METHOD

The model. Assume the lung to consist of only one alveolus, connected to a dead space (V_D), through which gas flows with a square front. Once the gas enters the alveolus, it mixes instantaneously and completely with the alveolar gas, with which blood leaving the alveolus is in partial pressure equilibrium at any instant.

It was assumed that V_D contains the final part of the alveolar expirate from the previous cycle, and that it is mixed completely within itself just before the onset of the next inspiration.

図 9 1972 年に発表した，肺胞の往復換気モデル解析の論文

酸素と二酸化炭素の相互作用の解析などに骨を折りましたが，「二酸化炭素だけでなくて，酸素にも影響があるはずだから加えるように」と示唆して下さった査読者に感謝しています．

1963～66 年までの 3 年を過ごした MGH のレイヴァー先生も血液ガスに詳しく，やはりときどき電話で質問しましたが，こちらは血液ガス一般よりも彼が開発した酸素解離曲線の分析装置(電極対を用いる方法) に関係するものが多かったでしょう (図 8).

装置と直接は関係ありませんが，サンディエゴで忘れられないのはヘモグロビンの化学の勉強で，これには血液ガスや酸素解離曲線分析の理由もありますが，もう一つはモデル解析です．二酸化炭素の肺ガス交換モデルの解析をボストンと東京で行い，それを発展させた論文を書いて J Appl Physiol 誌に投稿したところ，「いい論文だが二酸化炭素だけではもったいない．酸素にも影響があるはずだからそれも含めるように」というコメントで，追加を要求されました．ところが，酸素を加えようとするとヘモグロビンに対して酸素と二酸化炭素が作用する問題を加えねばなりません．酸素と二酸化炭素が，ヘモグロビンに競合的に働くいわゆるボーア効果とホールデン効果です．それを数式化してプログラムに組むことに骨が折れましたが，しかし結果的には貴重な学習機会でした．これがなかったら，私の血液ガスに対する認識はずっと浅い

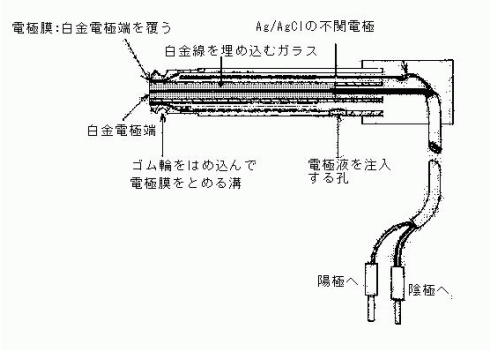


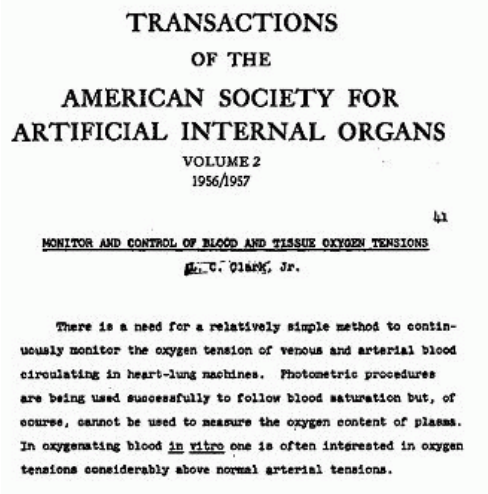
図 10 酸素電極
すでに使用のはじまっていた酸素測定用の白金電極端に膜を貼って，分極をふせぎ，同時に拡散面をしっかり定めることによって，較正が可能にになった．電極端の P_{O_2} はほぼゼロで，血液サンプルから電極端に酸素分子が動き，その「流れ」に応じて電流が流れる．

図 11 ルラン クラーク氏
心臓手術時の人工心肺の運転を担当していた．人工肺改良用に電極を開発した．自分では業績の偉大さをしばらく認識していなかったらしい．

レベルで終わったであろうと推測して感謝しています (図 9).

1955 年頃：P_{O_2} & P_{CO_2} 電極の創始

次に，酸素と二酸化炭素の電極の開発の問題を説明します．この二つの電極はほぼ同時に開発



TRANSACTIONS
OF THE
AMERICAN SOCIETY FOR
ARTIFICIAL INTERNAL ORGANS
VOLUME 2
1956/1957

41

MONITOR AND CONTROL OF BLOOD AND TISSUE OXYGEN TENSIONS

L. C. Clark, Jr.

There is a need for a relatively simple method to continuously monitor the oxygen tension of venous and arterial blood circulating in heart-lung machines. Photometric procedures are being used successfully to follow blood saturation but, of course, cannot be used to measure the oxygen content of plasma. In oxygenating blood in vitro one is often interested in oxygen tensions considerably above normal arterial tensions.

図 12 クラーク電極を記述する論文
掲載は人工臓器の雑誌で，入手は必ずしも容易ではない．上半分は表紙で，下半分は論文冒頭部分．ご覧のとおり，タイプライター印刷で中身の文章も，口頭発表の語調が残っている．図は 3 葉付属．

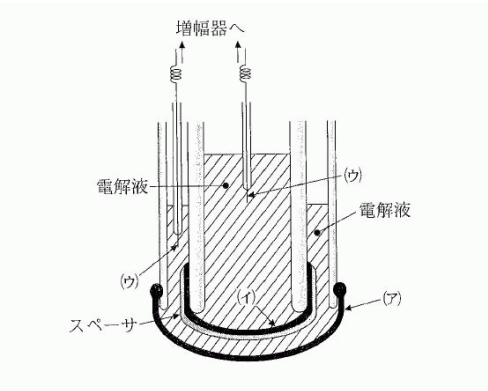


図 13 二酸化炭素電極の原理図
電極液に炭酸脱水酵素がないので，初期の製品は速度が極端に遅かった．血液を溶血させて，炭酸脱水酵素を加えれば高速にはなったが，すぐ失活して実用性はなかった．

されました．単純な偶然の要素と，プラスティック産業の開発と医療面の要求とが組み合わさっていると解釈します．

P_{O_2} 電極がおおやけになったのは 1956 年で，もちろんクラークによるもので，彼の書いたところによると当時手術室で人工心肺の開発を担当していて，人工肺の性能を把握する必要があっ

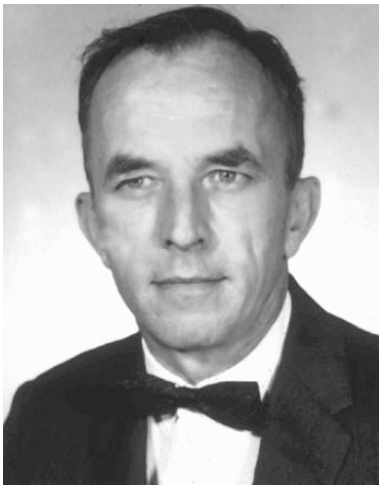


図 14 ストウ氏の写真
二酸化炭素電極の原理の開発と試作したが，学会発表の抄録のみで，論文は大学紀要に発表したので入手は不能.「ストウ/セブリングハウス電極」とも呼ぶ人も少なくないようです.

て電極を開発したと述べています (図 10–12). 1954 年には，膜をかぶせたプロトタイプをすでに作って使っていたようで，その経緯はずっと後の 1981 年になって CCM 誌 (Crit Care Med 1981. 9: 690–692) に詳しく述べています．一方，P_{CO_2} 電極を開発したストウ氏は「病院にポリオで人工呼吸を受けている患者が何人もいたのでそれをチェックする目的で開発した」と述べています (図 13, 14)．つまり，二つの電極はいずれも医療の現場での使用を狙って作られました.

ここにセブリングハウスが登場して重要な役割を果たします．クラークの発表は人工心肺の雑誌に発表され，ストウのそれは少し後に自分のイリノイ大学の紀要に発表されており，特に後者は未だに私は入手できないでいるくらい入手が困難な雑誌の論文です.

ストウが二酸化炭素電極を発表した時も，クラークが酸素電極を発表した時も，セブリングハウスがそれを聴いていて，すぐに重要性に気づきました．議論が好きで話も上手で，もちろん研究をどんどん進めて論文を沢山書く方ですから，酸素電極と二酸化炭素電極の情報は彼の活動を通じて世に広まりました．1958 年には，セブリングハウスは pH 電極も組み合わせて現在の「血液ガス分析装置」のプロトタイプを発表しました (図 15)．これが装置の商品化に大きな役割を果たしたでしょう．ラジオメーター社は，すでに販売していた分析装置 (pH と下に述べるアストラップ法による P_{CO_2} 分析) に，さっそく酸素電極を加えました．1959 年には IL 社 (Instrumentation Laboratory) が設立されて，すぐ翌年の 1960 年には最初の製品を販売しています．この会社は，現在まで継続して血液ガス分析装置を製作販売しています.

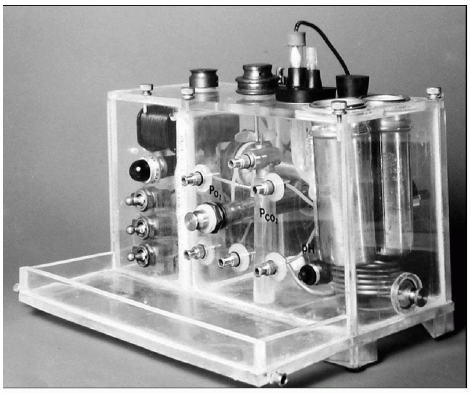


図 15 血液ガス分析装置
セブリングハウスの試作した，3 本の電極を組み込んだ「血液ガス分析装置」．後の装置のプロトタイプとなった．ワシントンのスミソニアン博物館に展示されている.

二酸化炭素電極の発明とほぼ同じ頃，アストラップ氏が「P_{CO_2} 電極なしでの P_{CO_2} 測定」に成功します．二酸化炭素電極が，pH 電極の出力を電極膜によって二酸化炭素のみを分離して P_{CO_2} 測定を行うのに対して，アストラップ氏の方法は血液の P_{CO_2} を意図的に変更して二酸化炭素平衡曲線を描き，それによって生の血液の pH から血液の P_{CO_2} を推測するので，両者は原理的によく似ています.

この方法は P_{CO_2} 測定としてはいわば間接法ともいうべきものですが，機器として完成度が高かったので測定値の信頼度が高く，1960 年代に入ってもずっと使用されました（図 16–19）.

なお，酸素電極と二酸化炭素電極は，見掛け

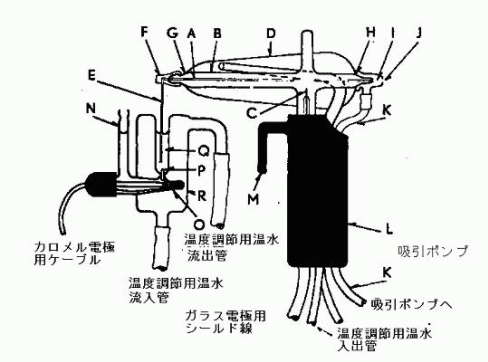


図 16 アストラップ法

初期の血液 pH 測定装置で，ピストル型 pH 電極とマイクロトノメータの組み合わせで，直接測定は pH だけで CO2 電極なしで，P_{CO_2} 測定を可能にしました．これがアストラップ法です．

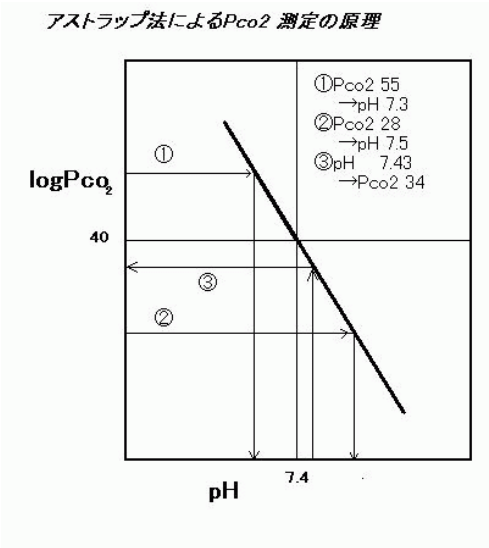


図 17 アストラップ法の測定原理

アストラップ法の原理を示す図です．血液の二酸化炭素平衡曲線は，pH-logP_{CO_2} 座標軸上で直線になります．それで，既知濃度の二酸化炭素で平衡 (トノメトリ) して pH を測定すると，二酸化炭素平衡直線が定まり，血液サンプルの pH を測定すると P_{CO_2} も算出できます．この原理は，CO2 電極のそれと酷似しています．動作の安定性などの点で，1965 年頃までは，CO2 電極より優位にあったかも知れません．

は似ていますが，動作はとても違うことを指摘します．酸素電極は「酸素分子を拡散させてそれ

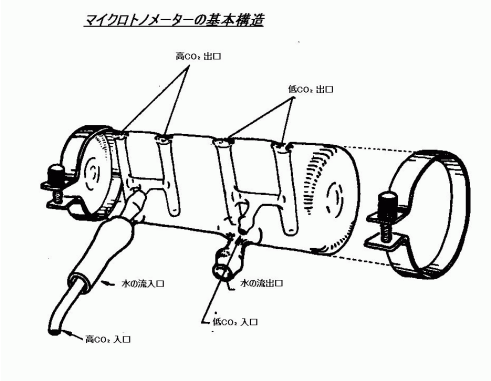


図 18 アストラップ法のマイクロトノメータ

アストラップ法に使用されたマイクロトノメータの図．微量の血液サンプルを二つの P_{CO_2} で平衡させ，pH を測定します．平衡室は各２つずつ備わっています．

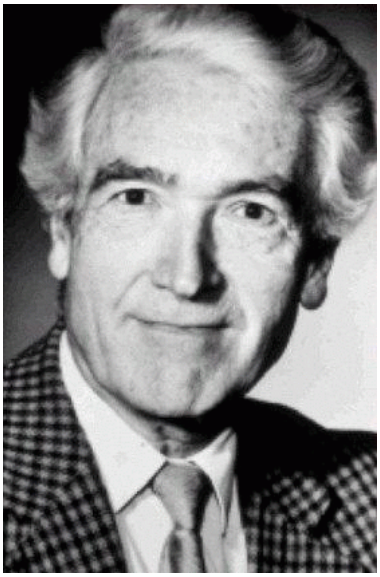


図 19 アストラップ氏

血液 pH 測定法を確立し，酸塩基平衡の進展に寄与しました．元来は純粋の生化学者でしたが，1950 年前後のポリオの流行がこの領域に踏み込むきっかけだった由．

に依存する電流を測定する」ので，膜/電極液/電極端の P_{O_2} が全部異なります.「酸素が流れて，電流が流れる」のですから，電極は原理的に酸素を消費します．いわば，電流を測定して「酸素の流れ」に換算する装置です．

一方，二酸化炭素電極は膜/電極液/電極端の

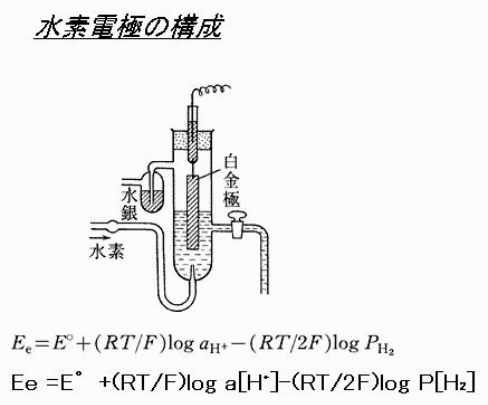


図 20 ネルンストが 1900 年に開発した水素電極
pH の測定が初めて可能になった．出力が高くて，ガルバノメーターでも十分に測れたが，ガス状水素ガスを使うので，血液ガス測定は困難でした．

P_{CO_2} が全部等しく，その P_{CO_2} に応じて pH が変化して，その pH を測定して P_{CO_2} に換算します．ただし，二酸化炭素の水和反応の速度が遅いので，初期の装置は測定に時間がかかり，この点もアストラップの装置が歓迎された理由の一つでした．

1962 年の秋に，東京大学医学部麻酔学教室には上記 IL 社の製品が導入されています．私は医師の 1 年生で使う機会はありませんでしたが，日本で血液ガス分析装置を導入した非常に早期のものです．もしかすると一号機だったかも知れません．教室を主宰されていた山村秀夫先生の先見性にも感心します．

pH 測定と塩橋

次に pH 測定の歴史を簡単に考察します．電気化学の歴史では，オストワルド・ファントホフ・ネルンストなどが重要な役割を果たしていますが，そのネルンストが 1900 年に水素電極を開発して pH の測定が初めて可能になりました (図 20)．この電極は出力が高くて，当時の貧弱な測定器 (ガルバノメーター) でも十分に測れましたが，一方で何しろ水素ガスを使うので，血液ガス測定は困難でした．たとえば，1904 年に発表された有名な「ボーア効果」の論文ではハッセルバルフが血液の pH の測定を担当していますが，血液と水素電極とを何度も平衡させた後に測定したという記録が残っています．

図 21 pH ガラス電極を開発したハーバー
このすぐ後の 1913 年ころ,「空中窒素の固定」という大きな業績を挙げて，1919 年のノーベル化学賞を受けていますが，ユダヤ人ゆえにその後は不遇な立場におかれました．

1909 年に，血液ガスの歴史で重要な事件が三つ起こっています．一つは，ヘンダーソンによる二酸化炭素と重炭酸イオンの平衡の記述（ヘンダーソン／ハッセルバルフ式のヘンダーソン式の部分），二つ目はソーレンセンによる pH の提案，三つ目が現在も使われる pH 測定用ガラス電極の開発です（図 21–22）．

ガラス電極を開発したのは，空中窒素の固定で名高いあのハーバーで，空中窒素の固定の仕事に数年先行しています．現在も残っている論文をみるとしっかりとした記述ですが，しかしガラス電極には血液のそれを測定する可能性はあるものの，水素電極とちがって得られる電圧は微弱です．1909 年当時は三極真空管がようやく発明された時点で，アンプリファイアの性能は貧弱だったので，折角のガラス電極も人体を対象としての実用性は乏しい状態でした．

ガラス電極が実用に供されるのは，第一次大戦後の不況の最中の 1930 年頃です．当時，カリ

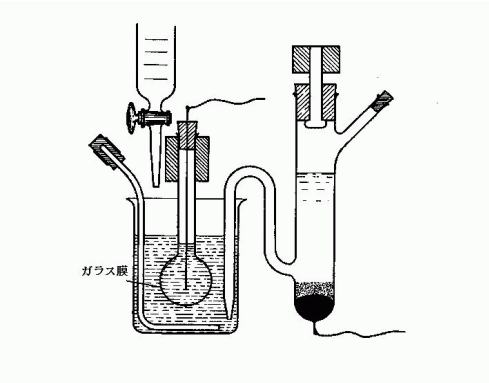


図 22 ハーバーのガラス電極の基本構造
ガラスが［H^+］に感じる性質を利用しており，さらに塩橋が描かれています．この装置は出力が低く，実用になるのは真空管アンプが普及する 1930 年以降でした．

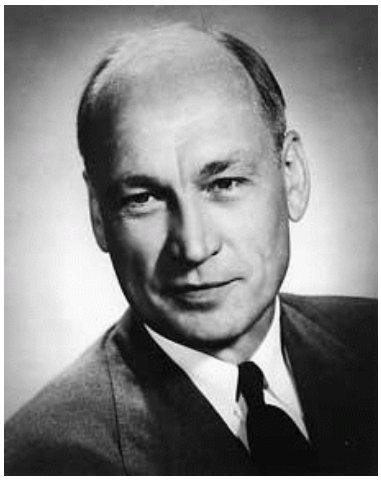
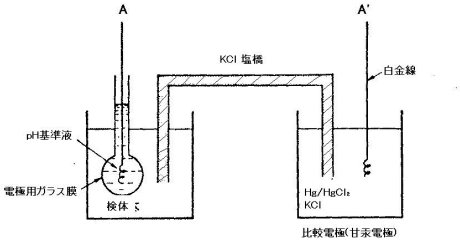


図 23 pH ガラス電極を実用化したベックマン
装置が優秀で，自らビジネスマンに転進してベックマン社を設立した．この会社は後に一時血液ガス分析装置も販売した．大変な長寿を保ち，没したのは 21 世紀になってから．血液 pH 測定法を確立し，酸塩基平衡の進展に寄与しました．元来は純粋の生化学者でしたが，1950 年前後のポリオの流行がこの領域に踏み込むきっかけだった由．

フォルニア工科大学の教授だったベックマンが，このガラス電極に当時ようやく実用化された真空管アンプを組み合わせて「pH 測定器」として販売しました．この装置は各方面で広く使われ，各種の pH 測定を容易にする役割を果たします．これをきっかけに，ベックマン氏は大学をやめてベックマン社を設立してビジネスマンとなって会社経営に力を注ぎ，自身も多額の寄付を寄せると同時に，大学の理事となってお金集めにも活躍しました．実に 104 歳の長寿を保ち，亡くなったのは 2004 年と比較的最近のことです．(図 23)



pH 電極の基本構造

図 24 「塩橋」とは何か
血液の pH 測定には「塩橋」が必要で，これを確立したのはデンマークのブジェルムで，1950 年頃のことです．K^+ と Cl^- はイオンサイズがほぼ等しく，拡散が等しいという性質を利用しています．

ベックマンの pH メーターでは,「血液の pH を直接測定」するのは困難でした．血液がガスを含む点，たんぱく質濃度が高い点，他のイオンもいろいろとあるなど複雑な故です．1950 年頃になって，ブジェルムが現在も使う「塩橋」を開発して，ようやく血液の pH の直接測定が可能となりました．カリウムイオンと塩素イオンのイオンサイズが等しくて拡散性がごく近く，接触電位がキャンセルしやすい故です．質量数の比較ではカリウムイオンが大きいのに，イオンのサイズはずっと近くなります．カリウムイオンは最外郭の電子を失って小さく，逆に塩素イオンは最外郭の欠損電子を一つ満たして大きくなる故です (図 24).

P_{O_2} 測定の最初の試みからポーラログラフィーまで

最後に，クラーク以前の酸素電極の歴史を扱います．歴史の上では，ダニールという人が 1897 年に酸素電極の原理を記述しています (図 25)．ダニールはネルンストの教室で活動していて，発

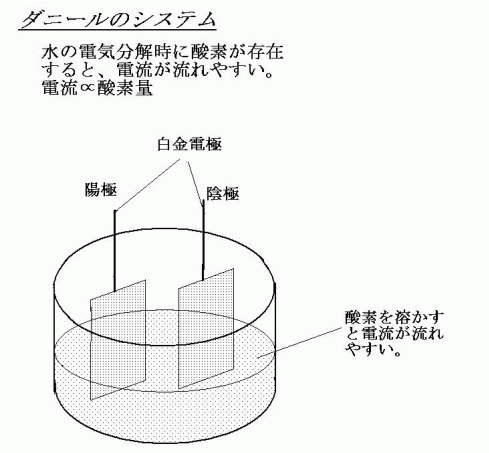


図 25 ダニールの酸素電極
ダニールの電極の模式図を諏訪が描いたものです．原論文は入手できませんでした．

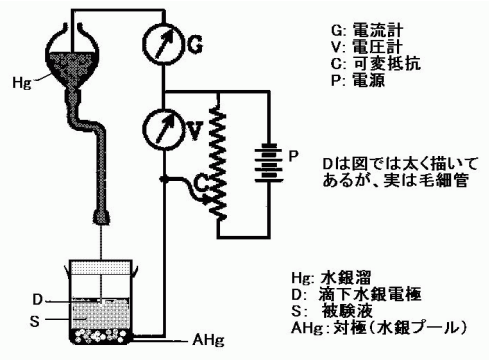


図 26 ポーラログラフィーの模式図
可変抵抗器をずらして抵抗値を変えることによって印加電圧を変えて，それに対応する電流を測定するのが本法の原理です．で，1950 年頃のことです．K^+ と Cl^- はイオンサイズがほぼ等しく，拡散が等しいという性質を利用しています．

見したようです．ただし，当時の技術ではこの微弱な電流を測定すること自体が困難で，まして生体サンプルをあつかうなどは不可能だったようで，実用的な測定の記録は見当たりません．

酸素の電気的測定が次に登場するのは，ヘイロフスキーのポーラログラフィーで，1922 年に「水銀滴下」という方法を用いて，各種化学物質の電圧と電流の関係を詳細に分析しました．水銀滴下は，これによって電極面を常に新しく保ち，分極を避ける目的でした．ちなみに滴下速度

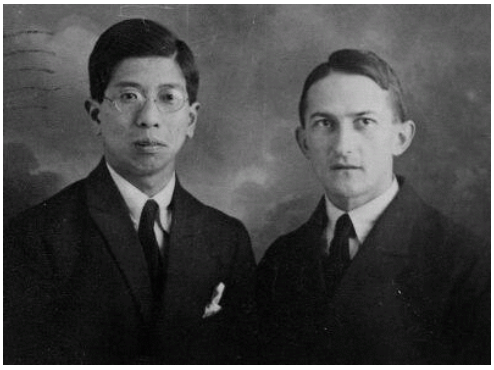


図 27 ヘイロフスキーと志方益三氏
ヘイロフスキーは手動で電圧を変えて電流を読み取っていたものを，志方氏が自動化して連続的に読み取れるように変更し，図を描く速度が上昇しこの方法の有用性が向上しました．で，1950 年頃のことです．K+ と Cl- はイオンサイズがほぼ等しく，拡散が等しいという性質を利用しています．

は，条件によりますが通常は数秒に 1 回だそうです． ところで，この研究には日本から留学していた志方益三氏 (のちの名古屋大学教授) が協力しており，ヘイロフスキーが後にノーベル化学賞を受けたときも，その受賞講演でしっかりと謝辞を述べています．

ヘイロフスキーの研究とダニールの電極などから，さっそくハダカの酸素電極がつくられて生体の P_{O_2} 測定も試みられるようになりましたが，血液での測定が可能になったのはクラークが電極に膜を貼って分極を防止し，同時に電極をしっかりと較正可能にしたことによります．

意外に永いカプノメトリーの歴史

最後に，血液ガスに近縁のカプノメトリーの歴史を少しだけ扱います.「カプノメトリー」は,「カプノ」が「炭素・煙」などをあらわし「二酸化炭素を測る」意味ですが,「気相の二酸化炭素を測る」ことに使います．以前は，質量分析（マススペクトロメトリー）で測定した例もありますが，それは麻酔薬の測定も同時に行おうとした故です．現在では，二酸化炭素も吸入麻酔薬も赤外線吸収で測定でき，質量分析による手法はカプノメトリーとしては実質的に消滅しました．

赤外線吸収による二酸化炭素分析は歴史が古く，1952 年には医学応用の報告があります．日本でも，1962 年頃には浅山健先生らが麻酔のモニターとして使用して報告しているものの，当時の装置は大型高価で不安定なだけでなくて，使用の度ごとに既知濃度の二酸化炭素を流して較正する必要があり，長く「臨床研究機器」に留まりました．

二酸化炭素が赤外線を吸収する原理と測定

二酸化炭素が赤外線を吸収するのが,「温暖化ガス」としてさわがれる理由です．二酸化炭素分子は，炭素 C が中央にあり酸素 O が両端に配置した直線構造をしています．こういう構造の物質は，C を中心に２つのバネで O が直線配置されている分子が，その縦軸方向に振動する要素と，分子軸の周りに各原子が回転するやや小さな要素があります．この振動や回転のエネルギーは連続ではなくて，とびとびのエネルギー準位をとるのできっちり決まります．その際の振動系の固有振動数は 10^{13}/秒程度で，光速をこの値で割り算すると赤外線の領域（800nm–1 μm）の値になります．このスペクトルは，分子毎に異なるので，それを詳細に調べて現在のように吸入麻酔薬も同じ原理で測定できるようになりました．

もっとも，呼気の二酸化炭素濃度の場合は濃度が 5%と高値で，スペクトルがきれいに細い線にならず”Pressure broadening” あるいは”Collision broadening”という現象が起こります．分子一個の反応で済まずに，隣り合った分子がぶつかって「本来の細い吸収スペクトルでなくて，にじんで広いぼんやりした帯になる」ということです．この作用は，装置のデザインで邪魔になる一方で，実測の気圧（たとえば一気圧）と測定点の気圧（陰圧で吸引するので必ず一気圧より低くなります）との間の較正に使えて具合がよいということです．この説明は，日本光電の山森伸二氏（2009 年現在，荻野記念研究所長）からしばらく以前に伺いました．

カプノメーターの広範囲の採用

1980 年以降，赤外線吸収を利用したカプノメーターが，麻酔での換気モニターとしてひろく使われえるようになり現在では「事実上の標準装置」として確立しました．機器側からみると装置が改良されて小型になり，その上に較正が容易ないし不要になり価格が低下し，さらに吸入麻酔薬も測定できるようになるなど改良が進みました．

二酸化炭素の測定で呼吸を確認するのは，1980 年以降にアメリカで確立普及しました．この話を私がはじめて聞いたのは 1980 年頃で，そのときは「呼吸を確認するだけのために，高価な赤外線測定器を使うなんて」と思いました．ところが，1984 年にニューヨークのコロンビア大学でこの装置の完全に装備された手術室で１ヵ月間働いた経験から,「どうやら自分が誤っていた」と感じました．呼吸の確認は重大で，装置なしには容易でないことも多い故です．それに，機器が普及すれば価格はどんどん下がります．

この手法がアメリカで普及したのにはあの国の事情もあります．患者が太って気管挿管がむずかしく，呼吸による胸の動きも見えにくく，聴診も困難で，特殊な装置を使用しないと挿管成功が確認できません．医学や医療の進歩に国民が不健康なほうが望ましいというのは，もちろん問題の立て方が逆ですが一面の事実で，他にもたとえば冠状動脈疾患（狭心症や心筋梗塞）の，冠状動脈バイパス手術が開発されたのはそれがアメリカで多いからで，患者の少ない日本では考えられません．

余談ですが，コロンビア大学の付属病院に赤外線による二酸化炭素分析器が導入されたのは，こんないきさつです．大学のスポンサーであるお金持ちの奥様が，手術の際に麻酔のあやまちで重体になりました．生命はとりとめたものの，後遺症が残りました．原因は気管挿管のあやまちで，気管内チューブが正規の位置に挿入されておらず，しばらく呼吸ができなかったのに，患

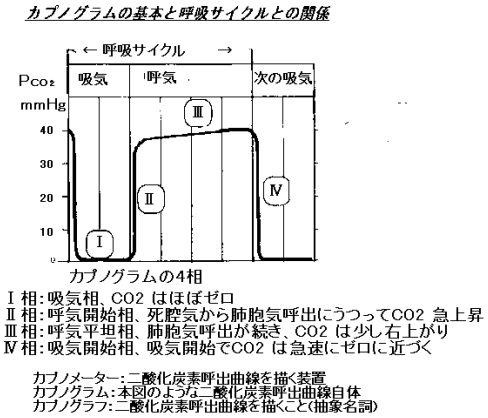


図 28 カプノグラフの曲線の意味を示す図
図の下に「カプノメーター」,「カプノグラム」,「カプノグラフ」等の意味を書きました.

者が太っていて気づかれませんでした．状況が落着いてから，このお金持ちが「こうした事故を確実に防ぐことはできないのか．妻のみじめな状況の再発を防ぐ装置はないのか」と申し出て，これに対応して話し合っているうちに,「それでは百万ドルほど寄付しよう」ということになったそうです．事故そのものはコロンビア大学で起こったことではないのだそうですが．当時の麻酔科の主任教授だったベンディクセン先生からうかがった話です.

カプノメトリーの使用は,「呼吸が止まっていない」「人工呼吸器と患者がつながっている」「気管チューブが気管に入っている」などの確認に使うほかに，その形や数値の分析で「人工呼吸器のリズムと自発呼吸が合致していない」「患者の呼気のパターンが異常」「患者の代謝が異常」などの検出の情報も与えてくれます.

血液ガスと関係する偉人たち

番外として，血液ガスに関係する偉人たちでここまでに述べなかった人たちを少し紹介します．ボーアといえば「ボーア効果」のほかにも，呼吸死腔のボーアの式や肺毛細管での P_{O_2} の上昇を検討するボーア積分などの業績があります.

おまけに，この人の子息があの大物理学者で原子模型の提唱者のニールス ボーアで，孫のボーアもやはり物理学者で，この二人は別個にノーベル物理学賞を受けています (図 29).

図 29 ボーア一家の 3 人
子供と孫はノーベル賞受賞．左から，生理学者 C. ボーア，あとの二人は物理学者，N. ボーアと A. ボーア

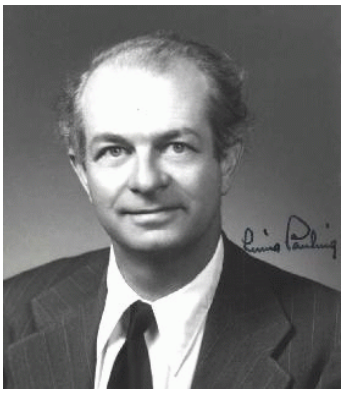


図 30 ライナス・ポーリング
ポーリングは業績が多彩ですが，酸素の常磁性を利用した測定器の開発，鎌状貧血の分析でグロビン分子のアミノ酸配列の異常示唆して「分子病」という概念提出の二つが重要でしょう．麻酔科医としては，吸入麻酔薬の作用機序仮説も忘れられません.

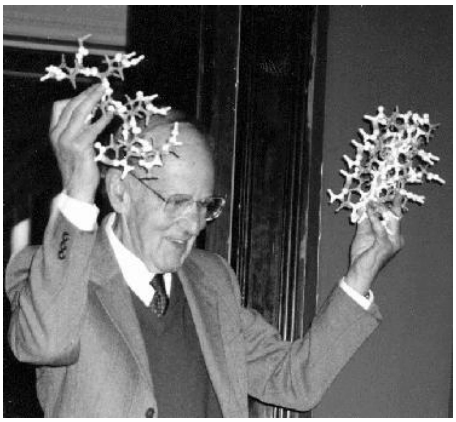


図 31 マックス・ペルーズ
ペルーズは「ヘモグロビンの構造解析」で，現在の基礎概念を確立しました.

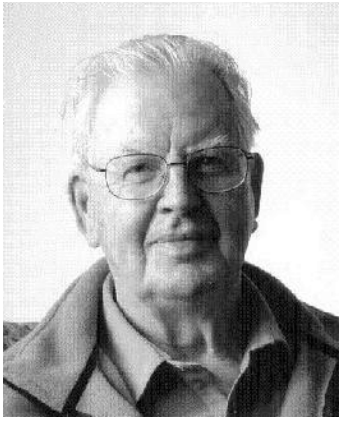


図 32 ポントピダン氏
ポントピダン氏はMGHのICUを長期にわたって指導．現在も元気．祖父が，デンマークの有名な詩人で1910年代にノーベル文学賞を受けている由．

ポーリングは業績が多彩ですが，血液ガスに関係した研究としては，ポーリング式酸素濃度計は酸素の常磁性を利用したもので，現在も一部の装置に使われています．もう一つは，鎌状貧血の分析でグロビン分子のアミノ酸配列の異常を示唆して,「分子病」という概念を提出しました．吸入麻酔薬の作用機序仮説も忘れられません (図 30).

ペルーズのノーベル賞は「たんぱく質のX線解析」となっていますが，実は「ヘモグロビンの構造解析」で，他のたん白の解析はしていません．しかし，現在みるあの構造の基礎を築きました (図 31).

ついでにもう一人，ノーベル賞と縁のある人を． MGH で ICU を確立したポントピダン氏は，1922 年生まれでさすがに引退されていますが，2009 年初頭現在もお元気です．この方の祖父が，デンマークの有名な詩人で 1910 年代にノーベル文学賞を受けているとは，ずっと後になって知りました．ちなみに「ポントピダン」は「橋のある村」を意味するラテン語 (ポンスが，脳の「橋」を意味するのはご存知ですね）だそうです.

ABSTRACT

History of blood gas instrument, especially its technological aspect.

Kunio Suwa, M.D.

Technological developments of blood gas instrument were briefly reviewed, both in individual electrode and the measuring system in general.

Teikyo Junior College

調節呼吸，自発呼吸共に動作する外部増設気道内圧アラーム

田中 義文

はじめに

数年前より全身麻酔器のベンチレーターについて，気道内圧アラーム装着の義務化が指導され，それに伴って従来から別製品として市販されていた気道内圧アラームが生産停止となった．ベンチレータ購入時に必ず気道内圧アラームが内蔵されること自体は大きな改善ではあるが，例えば，自発呼吸や用手換気などのベンチレータを作動させない状態で，気道内圧のアラーム機構が動作しないのでは安全上大きな問題がある．気道内圧アラームはベンチレータの駆動とは関係なく，常に監視できる存在でなければ意味がない．そこで，調節呼吸，自発呼吸共に対応でき，麻酔システムとは独立した気道内圧アラームを作成したので紹介する．

方　法

1) 気道内圧アラームの作動原理

気道内圧アラームは気道の圧力を検知し，陽圧呼吸するのに十分な気道内圧が一定の期間内に得られれば良し，そうでなければ無呼吸と判断して警報音を発信する装置である．この動作を実現するには，圧検出トランスデューサ，圧アンプ，有効な圧に対して ON/OFF を制御する電圧コンパレータ，再トリガー可能な単安定マルチバイブレータ，そして警報音発信装置が必要である．図１に気道内圧の変化とアラーム閾値の関係を示す．従来の方法では破線 B で示す最低陽圧レベルと実際の気道内圧とを比較する

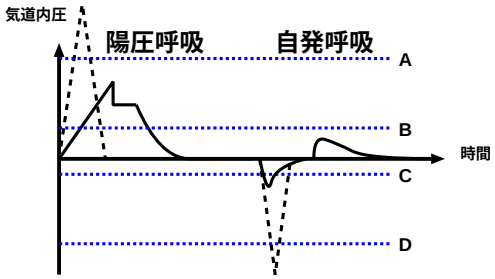


図１ 陽圧呼吸，自発呼吸における気道内圧と圧力監視設定レベル

気道内圧の時間変化を示す．破線 A は異常気道内圧，B は有効な陽圧レベル，C は有効な吸気圧，D は異常気道内陰圧を示す．

方法であった．つまり，B の圧力閾値を越えれば有効な陽圧が得られていると判断されてパルスが発生する．そして一定の時間内にパルスが得られなければ，無呼吸と判断されてアラーム音が発生する方式である．

この気道内圧を検出する方式は色々な応用が考えられる．まず図１の破線 A に示す異常な気道内圧を設定すると，気管チューブが閉塞したり折れ曲がったりして有効な換気ができない状態で警報音を発することができる．また，気道内陰圧で検出するレベル C および D を設定すると，それぞれ有効な吸気圧，そして吸気不能状態が検出できる．このような電圧コンパレータを増設し．論理回路を組み合わせるだけで，調節呼吸，自発呼吸共に動作する呼吸回路監視装置となる．

2) 気道内圧アラームの全回路．

気道内圧アラームの全回路を図２に示す．回路の左上端に破線で囲われた抵抗群は圧力トラ

京都府立医科大学麻酔科学教室

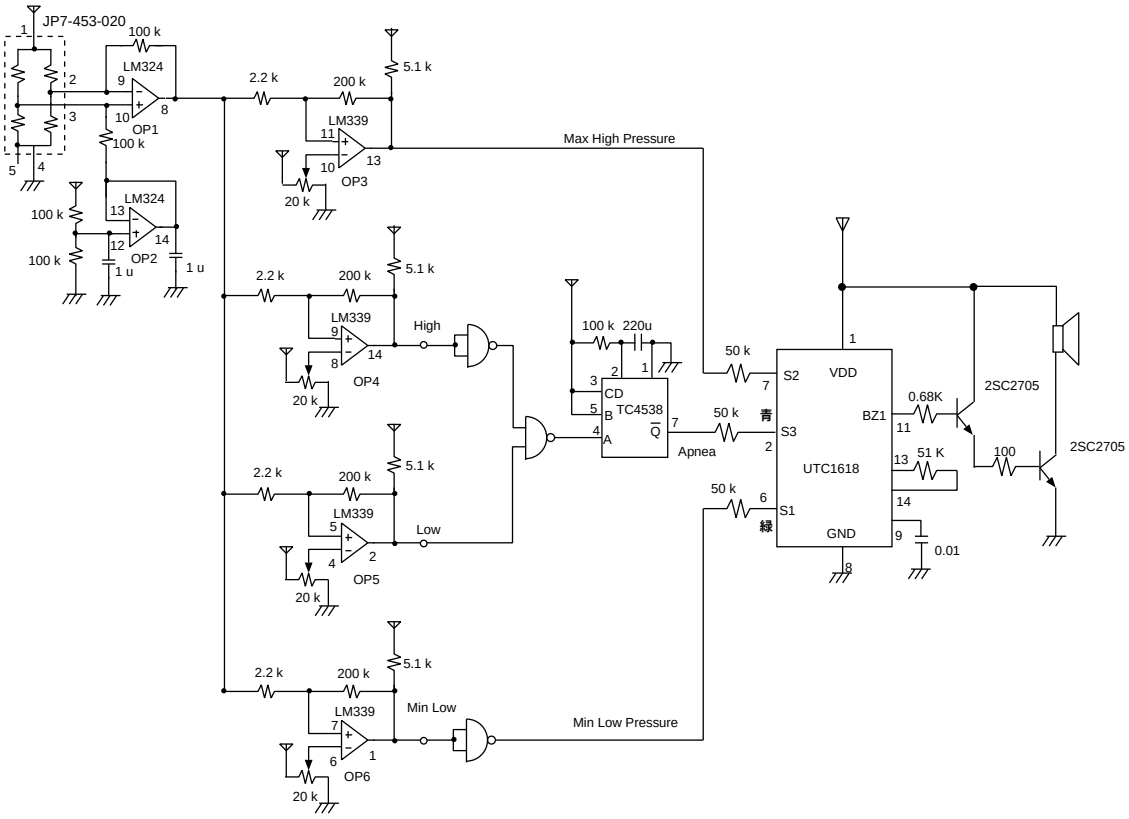


図 2 気道内圧アラームの全回路．
5V 単一電源を使用．JP7-453-020 は Edwards Lifesciences 製の観血的圧力測定トランスデューサを示す．回路説明は本文参照．

ンスデューサを示している．今回はディスポーザブル動脈圧トランスデューサを使用した．これには 5 本のリード線が接続されているが，5 番と 4 番が短絡されており，トランスデューサと圧アンプの接続を示す信号線に利用されている．抵抗群ハブリッジに組まれており，1 番に 5V，4 番に GND を接続し，2 番と 3 番の圧力変化による電位差を差動アンプ OP1 で増幅している．

OP1 の＋入力端子に接続されている 100 k の抵抗は，本来ならば GND レベルに接地されるべきであるが，本回路は+5V 単一電源で動作するために OP2 のボルテージフォロア回路で 2.5V にシフトアップした．そのため OP1 は 2.5V が信号原点電位となっている．OP3 から OP6 までがヒステリシス付きコンパレータとして作動する．OP3 は最高許容圧力より高くなるとおよそ+5V の high レベルを出力し，それ以下の圧力では low レベルを出力する．同様に OP4 は有効な陽圧になれば high，それ以下であれば low となる．また OP4 の出力には NOT 回路を付けて，論理を反転している．

OP5 は有効な陰圧レベル以上であれば high，それ以下の陰圧であれば low となる．つまり，OP4 と OP5 はそれぞれ有効な換気レベルであれば low を出力する．両者の信号は，その次の NAND 回路で OR され，正論理になって出力する．その結果，陽圧，陰圧共に有効な換気レベルであれば low から high レベルに向かうパルス信号となる．このパルスは TC4538 の再トリガー可能な端安定マルチバイブレータに接続される．1 度でも正論理のパルスが入力されると CR の組合せで 20 秒間 ON 状態が続く．その負論理を

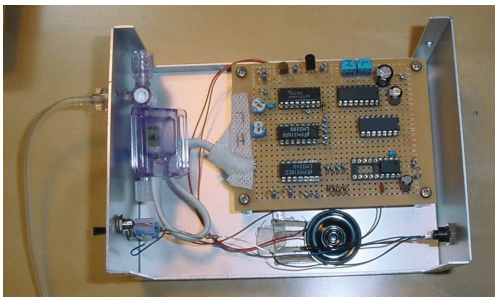


図 3 自作の気道内圧監視装置
左パネルは気道内圧チューブ接続端子と電源SW，電源表示および警報表示 LDD を配置し，右パネルには 5V 電源入力端子のみを配置している．

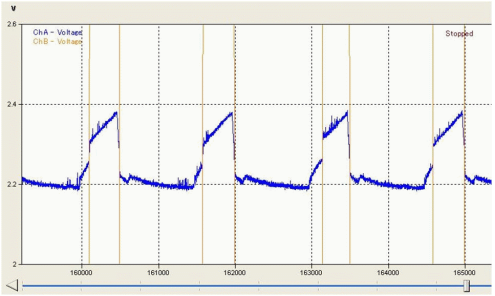


図 4 有効陽圧検出コンパレータの入力信号
2.2V を 0 圧としてリニアな気道内圧を表示してる．飛び跳ねた信号はコンパレータの干渉による．

利用して警報発生 IC である UTC1618 の S3 に入力させている．

警報発生 IC は S1 より S6 までの 6 種類の音色が発生するが，音色の優先順位は S1 -> S2 -> S6 -> S4 -> S5 -> S3 となっている．そこで危険な最高気道内圧，危険な最低陰圧，無呼吸の順になる組合せ順位を考慮して，好みの三種類を決定した．オーディオ出力は 2SC2705 をダーリントン接続し，8 オームの小型スピーカを駆動している．

結果および考察

装置は万能基板にラッピング配線，一部ハンダ付けを行い，市販のアルミケースに組み込んだ．天蓋を取り去った全貌を図 3 に示す．図 4 にレスピレータ駆動時の有効陽圧検出コンパレータの入力信号を示す．一応リニアな気道内圧が表示されるが，シュミットトリガーによる干渉を受けて飛び跳ねた信号になっている．コンパレータで ON/OFF 制御を行うだけであるから語動作などの問題は生じなかった．作れば必ず動作する回路である．

近年，レスピレータやベンチレータの性能はコンピュータを内蔵することにより飛躍的に進歩したが，チューブトラブルや操作ミスなどの人が介在する事故，ニヤミスは決して無くなるものではない．特に問題になる例は，ベンチレータの駆動忘れ，ジョイント接続不良，チューブの折れ曲がりなど人の操作が介在するうっかりミスであろう．最近ではパルスオキシメータ，呼気炭酸ガスモニター，それに換気装置内蔵型気道内圧警報装置などが普及し，安全性が飛躍的に向上したが，これらの装置も正しく操作できる知識と技術を習得して始めて価値ある患者監視が可能となる．

人工呼吸を例にすると，最近のベンチレータは，有効な気道内圧が生じなければ無呼吸と判断して警報を発する気道内圧監視装置が内蔵されている．しかしこの警報装置はベンチレータを駆動させている間だけ監視するのであって，麻酔から覚醒に移行するときにはベンチレータを停止させ，用手換気を行うために気道内圧監視装置は動作しない．麻酔器は用手換気モードになっているのに，麻酔医はベンチレータモードだと勘違いすると事故が発生する．気道内圧監視装置が動作しないためである．始めに呼気炭酸ガスモニターが異常を知らせ，次にパルスオキシメータが異常値を示すであろう．最後に心電図が徐脈を示す警報を発するに違いないが，手遅れになる場合もある．一方，麻酔状態でベンチレータを自発呼吸と陽圧換気との混合換気モー

ドである SIMV モードに設定すると，気道内圧監視装置は陽圧換気をセンスする構造であるために警報を発することがない．このようにベンチレータ内蔵型気道内圧警報装置は麻酔操作から考えると決して十分な安全性を保証するものではないといえる．

このようなベンチレータ内蔵型気道内圧監視装置の欠点を克服するには，単純にベンチレータとは独立した外部気道内圧監視装置を装着するだけで解決する．我々は日本メディコ社製気道内圧アラームを長年使用してきたが，最近ではベンチレータ内蔵型が行政指導になったために外部取り付け型の装置は製造中止となり，入手できない状況となった．そこで，本稿に紹介する外部増設気道内圧アラームの設計開発に至ったが，本装置は気道内圧コンパレータを 4 個，そして論理回路を付加することにより，陽圧呼吸のみならず自発呼吸である陰圧平圧呼吸にも対応できるようにした．警報設定レベルはチューブの折れ曲がりなどで作動する異常高圧を最優先順位，換気ガス供給不足などで生じる異常陰圧を次の順位，そして 20 秒間の無呼吸で発する最低順位とし，それぞれ警報の音色を変えた．日本メディコの製品では表パネルに無呼吸時間を設定するボリュームがあるが，これは麻酔医が勝手に調節できるので，安全性の低下の原因にもなっている．そのために，本装置の表パネルは気道内圧の入力チューブ接続孔と電源 SW のみとして，無調整システムにした．

本装置に使用した部品は観血的動脈圧測定用圧トランスデューサ，警報音発生用音源 IC 以外に特殊なものはない．全て町の電気ショップで入手可能である．また，電気的に厳しいタイミングもなく，テスター一つで調整可能である．気道内圧アラームは呼気炭酸ガス検出装置やパルスオキシメータが異常を知らせる前に警報を発する装置であるから，安全な麻酔を遂行する上で各麻酔器に必ず装備すべき存在である．本装置の今後の検討課題としては警報音を音色で判別するのではなく，警報原因を音声で示すこと，バッテリーバックアップにすること，陽圧換気による自動駆動などが実現できれば，電源 SW は無くなり，表パレルは気道内圧のチューブ接続孔とアラームリセットボタンのみとなって，麻酔の安全性に一層の向上が得られると考えている．

参考資料

1. UTC1618, 6 KEYS SIREN/ALARM SOUND GENERATOR. UNISONIC TECHONOLOGIES CO., LTD.

ABSTRACT

An Airway pressure alarm system, which can detect on both ventilator mode and spontaneous breezing mode.

Yoshifumi Tanaka MD, PhD

FDA and Japanese ministry of health care prohibited to sale such ventilator without airway pressure alarm unit in the system and also we could not obtain the pressure alarm unit as the accessory attachment recently. But the airway pressure alarm unit should turn on the alarm during anesthesia in both ventilator and mandatory breathing mode. Hence an airway pressure alarm system, which can detect alarms on both ventilator mode and spontaneous breezing mode was made. The device showed higher performance than low-pressure monitoring devices on the market.

PDF ファイルの使い方工夫

諏訪邦夫

最近，インターネットの発表を中心にPDF ファイルの使用が増えている．この形式は利点も多いが，問題点も少なくない．その問題点を指摘して私の解決策を述べるが，会員諸氏のお知恵も拝借したい．

PDF ファイルの問題点をまずいくつか指摘する．

画面で読みにくい

1. 縦と横の比率の問題：
   パソコン画面は横長なのにファイル画面は縦長である．この点で横 2 段組・縦書き論文・縦書き書籍などでは，不都合の度合いが特に大きい．
2. ソフトウェアの速度が遅くて，高速で読むのに適さない．
3. 目次と本文との関係:
   目次と本文のリンク機能が貧弱で，無料の”Reader” には作成機能はもちろんない．有料ソフトウェアでも使いにくく，私はマスターできていない．インターネット提供の PDF のデータで，目次がついていながら目次から本文へのリンク機能を使っていないデータが多い．
   PDF ファイルの場合，画面に元の印刷頁数が残っている場合が多いが，ほとんどの場合に元の印刷頁と PDF ファイルの頁数とはずれている．それによって，ファイルはさらに使いにくくなる．
   極端にひどい例では，1 面が 2 頁ずつになっている場合がある．ごく小さな本などの例である．これは「見る」には便利だが,「読む」にも印刷にも不便である．
4. メモしにくい:
   文章を読めばメモを書き込みたいが，それが「使いやすい」とは言えない．

個人的解決策

PDF ファイルの使いにくさを，個人的には一応下のように工夫して解決している．

1. 文字のテキスト化:
   文字がテキスト化されていれば，テキストファイルを作成してエディターで読む．テキストの冒頭に PDF ファイルの名を書き込んでリンクする．文章部分が圧倒的に多いファイルには有用である．作成が簡単な場合と，やや困難な場合がある．
2. テキスト化されていない場合：
   画面で眺めるだけにとどめ,「ダウンロードして精読」は避ける．どうしても読みたい場合は不便だが画面で読むか，あきらめて印刷する．ファイルが小さければ，自分で改めて電子化する．
3. PDF を読むソフトウェアはフリーソフトウェアも含めて数多くあるので，高速なものを探したいが満足できるものはまだ見つかっていない．

結　語

PDF ファイルの欠点を指摘し，自分の工夫を述べた．会員諸氏のサジェスチョンを御願いします．

会場から「大きな画面を使えば読みにくくはない」とのコメントがあった．有効なサジェスチョンではあるが，ノートパソコンに頼る私の使い方には役立たない．

帝京医学技術専門学校

ABSTRACT

Pdf files are increasingly common on the Internet and in other environment. While this type of file has many advantages, it also possesses various disadvantages as well. I try to explain such disadvantages and propose a few solutions.

Disadvantages of Pdf files

Difficult to read on the display Ratio of longitudinal to lateral length.

The display of a PC is wide in lateral and narrow in longitudinal length. Yet just about all pdf files are long in longitudinal and narrow in lateral, causing great difficulties in reading the screan.

The software for reading (Abobe reader) is so slow in its performance that its use causes enough frustration.

The software does not have a capacity for linkage between table of contents and its bodies. It is tough to add memos while reading.

Personal solutions

If practical, I copy and paste the text portion and make a corresponding text file. Then the content is easier to read using a text editor of daily use. The original pdf file is made to link to this text file so that it could be referred at any time. If the text part of the pdf files does not allow to make a text file, then I may simply give up reading it altogether or, if it is sufficiently attractive, I may simply read it with hesitation.

“R” の統計以外への応用

萩平　哲，高階　雅紀*，内田　整，森　隆比古†

はじめに

“R” は Windows, Macintosh, Linux などのマルチプラットフォームで動作する強力な統計処理ソフトウェアであり，フリーソフトウェアとして配布されている．“R” を用いれば分散分析から多変量解析までほとんどの統計処理を行なうことができる．また，必要に応じて関数を追加することも可能であるように設計されている．さらに他のソフトウェアとの連携にも優れたインターフェイスを有しており，処理した結果を２次元や３次元のグラフとして表示したりすることもできるように設計されており，そのようなソフトウェアも実際にインターネット上から取得することが可能である．本来は統計処理用に作られたソフトウェアであるが，その強力な演算機能は統計計算以外にも応用できるものである．“R” のコンソールは言わば UNIX の shell のようなものであり，実装されている関数を利用すればデータ変換のためのフィルタを作成することも容易である．また，“R” にはベクトルや行列の計算を行なう関数も多数組み込まれており，最小２乗法による直線回帰なども簡単に行なえる機能を有している．こういった統計処理以外のデータ処理にも ”R ”は有用であると思われる．

ここでは，“R” を用いて DNA シーケンスを蛋白コードに変換するフィルタや，ベクトルおよび行列演算機能を用いて最小２乗法により直線回帰を行なう方法などを試みたので紹介する．

大阪大学大学院医学系研究科　麻酔・集中治療医学講座
*大阪大学医学部附属病院　手術部
†大阪府立急性期・総合医療センター　麻酔科

コード変換フィルターへの応用

簡単な応用例として DNA の trplet を蛋白質コードに変換するフィルターを作成した．もちろんこの程度の処理は UNIX の shell script や awk script などでも容易に作成可能である．“R” を使う利点は操作を単純化できる点であろう．元のデータを Excel で作成しておけば，簡単に “R” 上に読み込むことも可能である．“R” では作成したコマンドも起動時に読み込むことができるため，組み込みコマンドと同じように扱うえるため便利である．

```
# for MAC OSX
# df <-read.table(stdin(), header=TRUE,
     as.is=TRUE)
# For Windows
# df <-read.table("clipboard",
     header=TRUE, as.is=TRUE)
# df2 <- read.table("~/codon_table2.txt",
     header=T, as.is=T)
#
#  Let : 1= 1 letter, others= 3 letters
#
aa_code2 <-function(idx,let=2){
  x <- df[idx,1]
  y <- df[idx,2]
  z <- df[idx,3]
  n <- length(df2$L1)
    for(i in 1:n){
      if( df2[i,3] == "X" ){
        if( (x==df2[i,1])&&(y==df2[i,2]) ){
          if( let==1 ){
        return(df2[i,5])
          }
          return(df2[i,4])
```

```
        }
        next
      }
      if( (x==df2[i,1])&&(y==df2[i,2])&&
          (z==df2[i,3]) ){
        if( let==1 ){
          return(df2[i,5])
        }
        return(df2[i,4])
      }
    }
    return("-")
}
AA <- function( )
{
  aa_res <- sapply(rep(1:length(df$L1)),
     aa_code2)
  return( aa_res )
}
```

リスト 1.

また，プログラムコードとは別に triplet とアミノ酸のコード表を作成しておく．コード表の一部を示す．

L1	L2	L3	C3	C1
T	T	T	Phe	F
T	T	C	Phe	F
T	T	A	Leu	L
T	T	G	Leu	L
C	T	X	Leu	L
A	T	T	Ile	I
A	T	C	Ile	I
A	T	A	Ile	I
A	T	G	Met	M
G	T	X	Val	V
T	C	X	Ser	S
C	C	X	Pro	P
A	C	X	Thr	T
G	C	X	Ala	A

リスト 2.

1 から 3 列目に triplet，4 列目に 3 文字のアミノ酸コード，5 列目に 1 文字のアミノ酸コードを並べている．ここでは A,T,G,C の 4 種の塩基コードに，そのどれもに該当するという意味で X というコードを加えている．

線形最小 2 乗法への応用

機能	関数名
データフレームを行列に変換	data.matrix(df)
転置行列	t(m)
対角行列	diag(m)
行列の積	X%*%Y
行列式	det(m)
固有値および固有ベクトル	eigen(m)
特異値分解	svd(m)
逆行列	solve(m)

表 1 “R” が実装している主な行列演算関数

さて，“R” には表 1 に示されるような種々の行列演算関数が実装されている．これを利用すれば線形最小 2 乗法による最適解を簡単に計算することも可能である．

線形最小 2 乗法は観測されたデータセット $(a_{j1}, a_{j2}, \cdots, a_{jn}, y_j)$, $j = 1, 2, \cdots, m$ に対して

$$y = x_1 \cdot a_1 + x_2 \cdot a_2 + \cdots + x_n \cdot a_n \quad (1)$$

との 2 乗和を最小にする $\vec{x}$ $(x_1, x_2, \cdots, x_n)$ を求めるものである．ここで a_{ij} を要素とする Jacobian 行列を A(m 行 n 列) とすると $\vec{x}$ は以下の正規方程式から得られる．

$$\vec{x} = ({}^tA \cdot A)^{-1} \cdot {}^tA \cdot \vec{y} \quad (2)$$

数値演算的には正規方程式を直接解くのは不利 (演算誤差が大きくなる可能性がある) であるため，一般的には Jacobian 行列の分解を用いて演算を行う．行列分解の方法にもいくつかの手法がある．QR 分解 (上三角行列と対角行列に分解) を用いる方法，固有値分解を用いる方法，特異値分解を用いる方法，Cholesky 分解を用いる方法などが知られている．ここでは，“R” に実装されている行列演算関数を考慮し，特異値分解を用いる方法を示す．

特異値分解では Jacobian 行列を以下のような形に分解する．

$$A = U \cdot D \cdot {}^tV \quad (3)$$

U, V は正規直交ベクトルを列ベクトルとする行列，D は対角行列である．特異値分解を正規方程式に適応すると

$$\vec{x} = V \cdot D^{+} \cdot {}^{t}U \cdot \vec{y}$$

で最小 2 乗解が得られる．ただし $D \cdot D^{+} = I$, (I は単位行列) である．“R” に実装された特異値分解を用いると V, D, U が求まり，さらに転置行列，逆行列計算から $D^{+}, {}^{t}U$ も算出できる．これらの関数を用いることによりわずが数行で $\vec{x}$ が得られる．これを C などの言語で組めば相当な工程になる．以下に実際のコードを示す．1,2 行目はコメントアウトしているが，このような形でデータフレーム df および $df2$ に観測された $\vec{x}$ と $\vec{y}$ を読み込む．後はデータフレームを行列に変換して特異値分解を行い最小 2 乗解を算出するだけである．

```
# df <- read.csv("~/tmp/Sample_x.csv",
     header=F)
# df2 <- read.csv("~/tmp/Sample_y.csv",
     header=F)
A <- data.matrix(df)
Y <- data.matrix(df2)
ans <- svd(A)
X <- (ans$v)%*%solve(diag(ans$d))%*%t
   (ans$u)%*%Y
```

リスト 3.

サンプルデータを作成し，“R” による解析結果と，以前 C++を用いて開発した汎用最小二乗法ソフトウェア LSR Ver1.41 による解析結果を比較したところ同等の結果であった．

まとめ

“R” は各種の統計処理を行う事ができる強力なソウフトウェアであるが，その能力は単に統計処理に留まらず，種々の演算を効率よく行える機能も備えている．特に行列やベクトル演算が手軽に行えるためこの方面の応用も考慮すべきと考えられた．

参考資料

1. R Development Core Team. A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2006.
2. 中川 徹，小柳 義夫．最小二乗法による実験データ解析．プログラム SALS．東京大学出版会．東京．pp55–93.
3. 萩平 哲，高階 雅紀，森 隆比古，妙中 信之，吉矢生人．薬物動態解析のための最小二乗法プログラムの作成と応用．麻酔・集中治療とテクノロジー 1999,pp22–5.

ABSTRACT

Applications of a software for statistical analysis “R” to other than statistical analysis.

[1]Hagihira S, [2]Takashina M, [1]Uchida O, [3]Mori T.

“R” is a powerful free software for statistical analysis. It also equips many functions for vector and matrix calculations. Here we showed some apllications of “R” other than the statistical analysis. First, we made a filter software to convert DNA triplets to codes of amino acids. As the “R” console was just like the shell on UNIX system, such small filter software could easily be used on “R”. Second, we made a script for linear least-sqaure calculation using some built-in functions for vector and matrix calculation. It required many steps when we developed the software for linear least-sqaure calculation. However, using “R”, we could described it by only several lines.

We should pay much more attension to the performance of “R” in vector and matrix calculations.

[1] *Department of Anesthesiology, Osaka University Graduate School of Medicine*
[2] *Surgical Center, Osaka University Hospital*
[3] *Department of Anesthesiology, Osaka Prefectural Osaka General Medical Center*

CICR アッセイ用シーケンサーシステムの開発

岩瀬良範，市原靖子*，菊地博達

はじめに

悪性高熱症 (Malignant Hyperthermia; 以下 MH) は，重篤な麻酔合併症として知られるが，その診断は容易ではない．近年脚光を浴びている免疫および遺伝子診断は，MH においては発展途上にあり，また MH そのものの遺伝の多形性のため現時点では現実的な方法にはなっていない．その中で，筋生検による CICR(Calcium Induced Calcium Release) 法は，MH の検査法として最も精度が高いとされる [1,2] が，その実施は必ずしも容易ではない．今回我々は，CICR アッセイ用シーケンサーを開発したので報告する．

CICR による悪性高熱症診断の概要

1. 筋生検によって得られた骨格筋を顕微鏡下に筋線維まで分離する (skinned fiber 作製)．
2. 多種 (16 種類) の実験溶液を正確なタイミングで投与と同時に微細な筋収縮力 (等尺性収縮) を記録する．
3. それぞれの実験溶液に対する反応から判定を行う．カルシウム濃度に依存するカルシウム放出速度を計算し，正常群と亢進群に統計的に分離する．亢進群を悪性高熱症素因ありと判定する．

システムそのものは以前より存在したが PC-9800 の終焉とともに役目を終えた．新システムにおいては,「多種 (16 種類) の実験溶液を要求どおりの正確なタイミングで実験槽に投与」することが求められた．この投与スケジュールは自由に変更できるようにした．

埼玉医科大学麻酔学
*東京臨海病院麻酔科 (現:埼玉医科大学)

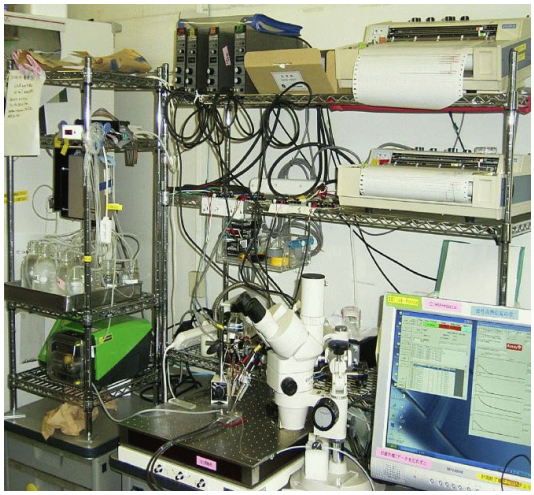


写真 1 CICR アッセイ用シーケンサーシステム全景
PC 画面左側に機器制御パネル，右側に各筋線維の反応が表示されている．

CICR アッセイ用シーケンサーシステム

16 種類の実験溶液をあらかじめ定めた順序とタイミングで実験槽に送り込み，筋収縮を記録し，排液する．これを実現するために，RS-232C 駆動の切り替えバルブ (Valco, Italy)，電圧駆動式ポンプ (Watson-Marrow, USA)，電磁弁を，AD/DA コンバーター (Contec: Japan) 内蔵の PC(Hewlett-Packard) から制御するシステムを構築した．

方　法

上記機器の制御には，Visual BASIC 5.0J と AD/DA コンバーター付属の VB 用ライブラリを用いた．実験溶液の切り替えバルブ (Valco) は，RS-232C によるデジタル制御であるが，バルブ位置の誤りは重大な結果を招くので，ソフトウェ

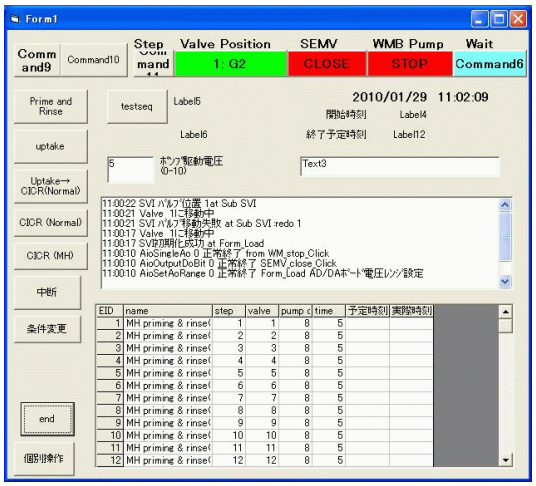


図 1 機器制御パネル画面

この画面で機器の状況監視と制御を行う．上・中段に各機器の作動状況モニター，下段にタイムテーブル，左段に各種スイッチを配置した．

ア上で完全な確認を行ってから次のステップに進めた．電圧駆動式ポンプ (Watson-Marrow) は D-A コンバーターで段階的に電圧を発生し，滑らかなローラーポンプの回転開始と停止を実現した．電磁弁は，同コンバーターのデジタル出力を利用し，筋収縮も同コンバーターのアナログ入力 4 ポートを使用して，4 本の筋線維が同時に検査できるようにし，記録はグラフと CSV ファイルで行った．制御と記録は一台の PC で同時に行われる．

結　果

CICR によるアッセイ用シーケンサーシステムを実現した．(写真 1)

(1). 制御系 (図 1)

機器制御パネルからは個々の機器を個別に制御できた．実験シーケンスでは，この制御ルーチンをタイムテーブルから呼び出す形式をとった．

(2). 記録系

4 本の筋線維の反応を溶液別に記録できた．この結果から MH の診断が行えた．

考察と結語

このシステムは，現在では実地稼動している．機器制御のソフトウェアの基本骨格と論理は，約半年をかけてデバッグし精度を向上させた．繊細な実験系に対して，使いやすく信頼性の高いソフトウェアを開発することは容易ではないが，その意義は大きい．

参考文献

1. Endo M, Yagi S, Ishizuka K, et al. : Changes in the Ca -induced Ca release mechanism in the sarcoplasmic reticulum of the muscle from a patient with makignant hyperthermia. Biomed. Res. 4 : 83-92, 1983.
2. 遠藤實．II 細胞内カルシウム動態と悪性高熱の発症機序　菊地博達編．悪性高熱症．東京：克誠堂；2006.　p19—42.

ABSTRACT

CICR(Calcium Induced Calcium Release) assay sequencer system for malignant hyperthermia diagnosis

Yoshinori Iwase, Yasuko Ichihara and Hirosato Kikuchi

The measurement of CICR (Calcium-induced Calcium release) rate is the most reliable diagnostic test for MH susceptibility, while its implementation was not easy because of the necessity for precise study sequence. We developed a CICR assay sequencer system with voltage controlled roller pump (Watson-Marrow, USA), RS-232C serial interface controlled multiple channel valve (Valco, Italy), voltage gated electromagnetic valve (Shimada-Rika, Japan) and a vacuum pump controlled by a single Windows Xp computer with multipurpose AD/DA converter (Contec, Japan). Software was written in Visual Basic 5.0J (Microsoft) with Contec device driver library. It consisted the device controlling part and study recording part from isotonic skinned fiber tension transducer through AD converter. The study sequence timetable is easily defined by

typical CSV file for future alteration according to academic progress. This system currently works properly after several feedbacks between diagnostic physician and software developer. Our experience showed a usefulness of PC technology for a specific and precise medical purpose.

Department of Anesthesiology, Saitama Medical University Hospital, Moroyama, Saitama 350-0495, Japan

当院での手術部門システム導入について

寺井岳三，藤井　崇

はじめに

大阪労災病院は病床数 734 床の総合病院で，手術室は 10 室，2006 年度の年間手術件数は 7944 件，このうち麻酔科管理の手術件数は 3516 件である．麻酔科医の定員は常勤医 9 名，後期研修医 2 名であるが，2007 年は常勤医 5.5 名と欠員があり，非常勤医の応援は 2 名/日，研修医は麻酔科後期研修医が 1 名と 2 ヶ月間ローテーションの前期研修医が 2〜3 名である．麻酔科医不足が背景にあり，麻酔科医の業務軽減も期待して，病院情報システム (HIS) の更新に合わせて手術部門システム導入を病院上層部と交渉したところ，幸いにも予算が認められた．2007 年 4 月，HIS と同時に新規導入した手術部門システムの特徴，今後の構想について報告する．

手術部門システム導入以前の問題点

1)HIS 更新前の旧オーダリングシステムには手術予約オーダ機能はあるが，紙伝票で手術申し込みをする科もあり，手術予定表の完成は前日の夕方であった．2) 麻酔申込はすべて紙伝票であった．3) 麻酔記録は手書きであり，担当麻酔医による JSA 麻酔台帳への入力間違いや漏れがあった．4) 医事伝票は手書きであり請求漏れの可能性があった．5) 患者モニターは 3 社 4 種類が混在して各手術室で異なり，麻酔科控室で全室の集中監視ができなかった．

手術部門システムの導入と運用フロー

これらの問題点を解消するために，当初は手術部門システムとして JSA 麻酔台帳/手術室管理システム 2006 を用いる予定であったが，年間手術件数が約 8000 件の手術室にとって機能は不十分であると思われた．前述したように予算が認められたので，各社の手術部門システムを比較検討した結果，麻酔器および患者モニターをジーイー横河メディカルシステム株式会社 (以下 GE 社) 製に順次更新中であったため，モニターとの接続が確実に行えることを考慮して同一メーカーの手術部門システム CIS-OR™ を採用した (図 1)．HIS は NEC 社製オーダリングシステム Mega-Oak™ が採用された．電子カルテの導入

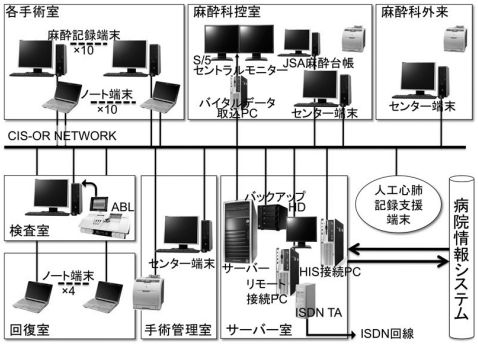


図 1 手術部門システムの構成

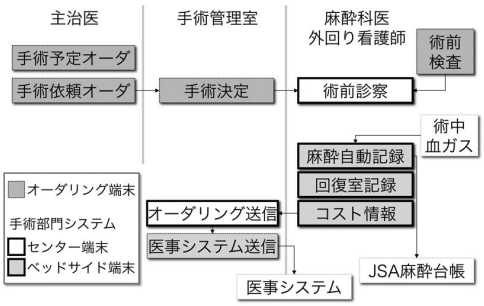


図 2 オーダリングシステムと
手術部門システムの運用フロー

大阪労災病院麻酔科

図３ 手術室の麻酔記録端末と看護師用ノート端末，手術管理室のオーダリング端末とセンター端末

は見送られた．

運用フローは図２に示したように，主治医はオーダリング端末で手術申込 (手術予定オーダ・依頼オーダ) を行い，手術室看護師はオーダリング端末上で手術３日前に手術決定操作を行い，決定後は患者情報が CIS-OR^{TM} に取込まれて手術予定表ができ上がる．以前に使用していた麻酔申込用紙は廃止した．麻酔術前診察は手術 2 日前に麻酔科外来の CIS-OR^{TM} センター端末を用いて行い，麻酔担当医の割当は 2 日前に可能となった．各手術室の麻酔記録端末では，患者モニターのデータが自動記録される．手術室内の検査室の血液ガス分析装置 ABL505^{TM}(ラジオメーター社) と CIS-OR^{TM} の接続により，血液ガス検査結果は自動で取り込まれ麻酔記録上に表示される．看護師はノート端末で医事請求入力を行う (図３)．回復室に入室後は回復室のノート端末に患者モニターのデータが自動で取り込まれて回復室記録が作成できる．麻酔科控室と手術管理室のセンター端末で患者情報，手術進捗状況，麻酔記録を参照できる．麻酔科控室のセントラルモニターでは，GE 社製 S/5^{TM} 患者モニターがある手術室は集中監視が可能である (図４)．他社の患者モニターはセントラルモニターに接続できないため今後 S/5^{TM} 患者モニターに順次更新すれば全室の集中監視が可能となる．JSA 麻酔台帳システムに接続している．

手術部門システムの利点

麻酔記録の電子化により，麻酔科医は麻酔記録用紙に記入する作業から開放され麻酔管理に集中できるようになった．麻酔終了時には自動的に麻酔台帳データベースが完成されているため，麻酔台帳への入力は不要であり，かつ誤入力や漏れがなくなった．麻酔記録情報の検索，集計，分析が容易となった．日本麻酔科学会への

図 4 麻酔科控室のセンター端末とセントラルモニター

麻酔偶発症報告が可能である．手術棟運営委員会への手術件数・麻酔件数の各科別集計，退室時刻の分布，予定時間延長症例件数などの報告書が短時間で完成できるようになった．手術室在室時間が予定より延長した症例に対して，入退室，麻酔開始終了，手術開始終了などの時刻を「在室延長・入室遅延報告書」として出力する機能を組み込んだ (図５)．手術室在室が予定より１時間以上延長した症例の担当医には，延長理由と今後の対策を記入して報告してもらうことにより，麻酔科医や手術室看護師の時間外業務を減らし，かつ手術室の有効利用を図ることを期待している．

今後の構想

麻酔記録に入力したプロポフォール、フェンタニル、レミフェンタニルなどの静脈麻酔薬の投与情報に自動連動して，静脈麻酔薬の予測血中・効果部位濃度を麻酔記録画面中にグラフ表示させる機能を希望しているが実現には課題が多い．すなわち，1) 品質保証の問題：予測濃度シミュレーション結果はあくまでも予測濃度であり実測値ではないため，信頼性があるかどうかの検証が必要であるが実証は困難である．2) 法務面の問題：シミュレーション結果の過信によるトラブルが発生する可能性があるため，医療行為に対する判断は全て使用者の自己責任であるとするソフトウェア使用許諾契約などでメーカーの責任の回避が必要である．3) 薬事面の問題：麻酔記録は医療機器扱いではなく薬事規制対象外であるが，予測濃度グラフ機能を麻酔記録システムに組み込んで製品として販売すると医療機器扱いとなる可能性があり，薬事規制の対象となれば，柔軟な仕様の追加やバージョン

在室延長・入室遅延報告書

ページ数 1/1
2007/12/04 12:46

診療科	整形外科	主治医	[redacted]		執刀医	[redacted]	
病棟	中5階	患者ID	[redacted]	性別	女	手術実施日	2007/12/03
予定術式	人工肘関節置換術						
実施術式	人工肘関節置換術						
＜予定＞		＜実施＞					
予定入室時刻	11:50	入室時刻	11:50	入室遅延	0h00m		
		麻酔開始時刻	11:56				
		手術開始時刻	12:41				
		手術終了時刻	16:52				
		麻酔終了時刻	17:18				
予定退室時刻	14:50	退室時刻	17:18				
予定在室時間	3h00m	在室時間	5h28m	在室延長	2h28m		
		入室−麻酔開始	0h06m				
		麻酔開始−手術開始	0h45m				
		手術時間	4h11m				
		手術終了−麻酔終了	0h26m				
		麻酔終了−退室	0h00m				
		麻酔時間	5h22m				

延長理由

今後の対策

備考

手術日から3日以内に、手術棟管理室まで提出して下さい。

図 5 在室延長・入室遅延報告書

アップが困難となる．

予測濃度グラフの組み込みが困難であるなら，麻酔記録端末上に既存のシミュレーションソフトウェアを別に起動させることはすぐに実現可能であるが，静脈麻酔薬の投与量と時刻を二重入力しなければならないのは煩わしい．そこで電子麻酔記録システムから自動でデータが書き出され，シミュレーションソフトウェアにデータが自動で取込まれるような機能があれば便利である (図６)．どのメーカーの麻酔自動記録装置でも互換性があるように，データ取込みインタフェイスの仕様を決める必要がある．

現在のシステムでは，麻酔術前診察や術後診察で病棟に往診したとき，病棟には手術部門システムがないためにすぐに入力ができない．従って手書き用紙で運用することが多く，往診後にあらためて麻酔科外来や麻酔科控室のセンター端末でデータ入力する必要があり不便である．手術システムと HIS を，Web サーバーを介して接続

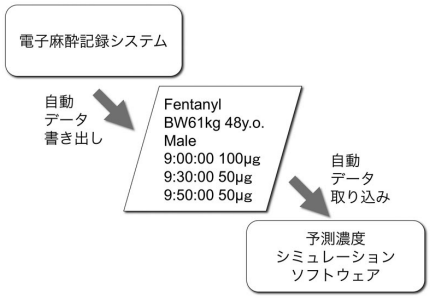


図 6 麻酔記録から予測濃度シミュレーションソフトウェアへのデータ受け渡しインタフェイスの概念

すれば，HIS 端末上で，または HIS の無線 LAN を介して PDA などの携帯端末やノート PC 上に，院内どこからでも手術システムに術前・術後診察情報を入力できるようになる．さらに病棟の HIS 端末上に麻酔記録を表示でき，手術進捗状況をリアルタイムに確認することができるようになる．この機能は予算が認められれば実現可能である．

まとめ

手術部門システムを新規に導入したので，その経緯，特徴，今後の構想について報告した．手術部門システムの導入により，麻酔申込用紙を廃止でき，手術予定表の完成が早くなった．麻酔記録の電子化により麻酔科医は患者の麻酔管理により多く専念できるようになり，さらに麻酔記録の検索，集計，分析が容易となった．セントラルモニターで各手術室を集中監視することが可能となった．

今後の展望として，静脈麻酔薬の予測濃度をグラフ表示する機能を組み込みたいが品質保証，法務，薬事に関して課題は多い．手術システムを HIS に Web サーバーを介して接続すれば，病棟の HIS 端末上で，または HIS の無線 LAN を利用してモバイル端末上で，手術部門システムを参照し入力することができるので便利である．

ABSTRACT

Introduction of operation-related system in our hospital

Takekazu Terai, Takashi Fujii

Following the introduction of the new operation-related system, we here report on the process of the introduction, characteristics and future design of the system. The introduction of the operation-related system has contributed to the abolition of anesthesia application forms and the more rapid completion of the operation schedule. Anesthesiologists are now able to devote more time to the anesthetic management of the patient as a result of anesthesia record computerization, and the search, counting, and analysis of anesthesia records has been made easier. We are now able to intensively monitor each operating room by means of a central monitor.

In the future, we want to incorporate a function that will show a graph of the predictive concentration of intravenous anesthetic, but there are many problems in relation to guaranteeing quality, as well as legal and pharmaceutical issues. By referring to an operation-related system and inputting it with hospital information system (HIS) terminals or with mobile terminals via wireless LAN of HIS in a ward, convenience can be greatly increased. This will be made possible by connecting the operation-related system to HIS through a web server.

key word:
hospital information system, operation-related system, anesthesia record system

Department of Anesthesiology,
Osaka Rosai Hospital
1179-3, Nagasone-cho, Kita-ku,
Sakai-shi, Osaka 591-8025

麻酔科領域における Scalable Vector Graphics の利用

斎藤 智彦

はじめに

われわれ麻酔科医が日常使用する画像データには，ビットマップや JPEG・PNG に代表されるラスター系のグラフィックデータと直線やベジエ曲線によるドロー系のグラフィックデータがある．一般に，写真には JPEG，PC 画像のキャプチャやアイコンには PNG や GIF，ビットマップが用いられる．

Power Point でプレゼンテーションを行う場合には，写真などラスター系のグラフィックの他に，矢印・ボックス・サークルなどのドロー系の描画データを扱うことが多い．ドロー系の画像データは，ビット毎に表現される画像には不向きであるが，描画をベクトル単位で扱うことが可能で，図形の拡大・縮小に伴う画質の劣化が見られないことや，格納データのサイズが小さくて済む等の利点がある．

しかし，Power Point などの市販ソフトで作成されるベクトルデータは，アプリケーション固有フォーマットのバイナリ形式であり，格納形式も公開されていないため，数値データを元に自分でベクトルデータを作成することは難しい．

Scalable Vector Graphics（以下 SVG）は，XML 形式のテキストファイルであり，テキストファイルを出力できるソフトウェアを用いれば，数値データから SVG ファイルを作成することも可能である．SVG の主な機能とマクロによる SVG 形式の簡易チャート作成プログラムについて報告する．

独立行政法人国立病院機構南岡山医療センター麻酔科

Scalable Vector Graphics とは

SVG はベクトル形式の画像を記述するための言語であるが，Web 標準仕様を推奨する World Wide Web Consortium (W3C) が提唱する Web グラフィックス用の XML ベースの言語であり，W3C により仕様がすべて公開されている．

SVG の記述は HTML と同じく XML 形式のテキストファイルであるため，一般のテキストエディタで簡単に編集することができる．専用のビュアーの他，Adobe 社の Illustrator，FireFox，Opera などの WWW ブラウザが対応している他，Internet Explorer でもプラグインを用いることで，簡単に表示することができる．（図１）

また，他種類のドロー系グラフィック形式への変換も容易である他，SVG は動画やインターラクティブなページを記述する機能も有している．

SVG によるチャート作成プログラムについて

麻酔記録など，数値データからグラフィックデータを作成する場合，Excel でデータを処理し，Excel のグラフ機能を用いてチャートを作成することが多いが，Excel の仕様に制限される部分も多く，薬剤投与などのデータを記述するには，多くの部分を手作業で行う必要がある．

SVG によるチャート作成プログラムでは，Excel のグラフ機能は使用せず，Excel で編集した数値データを元にしてマクロプログラムにより SVG を記述したテキストファイルを出力するようにした．血圧データは 1. マーカ形式，2. 折れ線形式，を選択可能とし，薬剤投与量は上部に

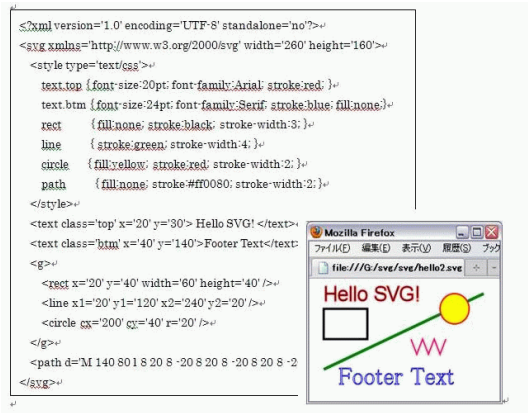


図 1 SVG ファイルの記述と描画イメージ

幅を持ったバーで示すようにした．Power Point などのプレゼンテーションで利用することを前提としたため，基本のチャート部分をシンプルに作成することを目的とし，複雑な描画機能は実装しなかった．

結果ならびに考察

図 1 に SVG による基本画像の描画サンプルとソースコードを示す．sVG は，XML 形式のヘッダと <svg … >〜</svg> タグで囲まれた部分に，描画命令をタグとして記述する．スタイルシートを用いることで，それぞれの描画コンポーネントの書式を一括して指定することが出来る．よく用いられる描画用タグは，<text> タグ：文字列表示，<rect> タグ：矩形表示，<line> タグ：直線表示，<circle> タグ：円表示，<path>：多角形表示 であり，各々のパーツを <g>〜</g> タグでくくることによりグループ化することが出来る．

図 2 に Excel のマクロで作成した SVG 形式のチャートを示す．SVG は XML 形式であるため，日本語を扱うには UTF-8 形式でエンコードする必要があるが，Excel は直接 UTF-8 形式での文字出力が出来ないため，今回は英数字のみの表示とした．この点は ShiftJIS 形式で出力した

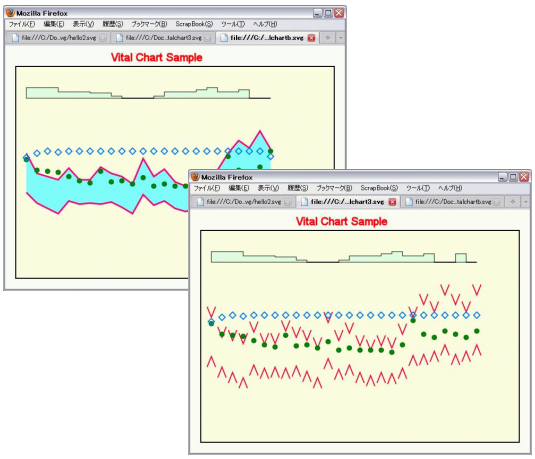


図 2 ．作成した麻酔記録の例

ファイルを，エディタなど適当なツールで UTF-8 に変換することで対応可能である．また，チャートに記述する文字列は，Power Point などプレゼンテーションソフトの上で編集した方が簡単であり，自由度も高い．

SVG Viewer や FireFox で表示したチャートを，そのままコピー＆ペーストすると，Power Point では PNG 形式の画像として扱われる．適当なサイズで描画した場合これでも十分な画質が得られるが，必要な部分を拡大したり，チャートデータを編集したりするためには，ベクトル形式で扱う必要がある．SVG 形式から Windows で一般的なベクトル形式 EMF(Enhanced Metafile) にするには，Adobe 社の Illustrator やフリーソフトの Inkspace を用いると簡単に変換できる．

プレゼンテーション用のチャートデータ作成ツールとしてではなく，Web 上でのグラフ作成ツールとして考えた場合，SVG によるグラフ表示は，サーバ上で PNG や JPEG 形式のグラフィックイメージを作成する必要がないため，サーバへ画像作成の負荷もかからない上，ネットワークトラフィックも小さい．FireFox 以外に，Internet Explorer でも標準機能として実装されれば，用

途はさらに拡大するものと思われる．

まとめ

麻酔チャート作成にSVG形式のベクトル画像を利用した．SVGファイルはExcelやテキストエディタなど、使い慣れたツールで容易に作成可能であり，バイタルチャートや検査データのグラフなど、プレゼンテーション用の画像作成に有用であった．SVGの動画機能やWeb上での高画質画像の作成については，今後の課題である．

参考 URL

1. http://www.w3.org/Graphics/SVG/
2. http://www.adobe.com/jp/svg/
3. http://www.adobe.com/jp/svg/viewer/install/
4. http://www.inkscape.org/

ABSTRACT

Using Scalable Vector Graphics
in the field of anesthesiology

Tomohiko Saito

Using Scalable Vector Graphics in the field of anesthesiology The Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics and graphical applications in XML. We used SVG format for drawing the anesthesia records. The SVG was useful for the vector graphics such as the anesthesia records. And, it was easy to describe the SVG scripts using with the text editors or Excel macro.

Key Words
Scalable Vector Graphics, SVG, XML

National Hospital Organization Minami-Okayama Medical Center, Okayama, 701-0304

第 25 回日本麻酔・集中治療テクノロジー学会
プログラム・抄録集
会長： 風間 富栄（防衛医科大学校 麻酔学講座）
会期： 2007 年 12 月 8 日（土）
会場： 所沢ミューズ ザ・スクウェア
第 25 回日本麻酔・集中治療テクノロジー学会プログラム

開会の挨拶：風間富栄（9：30〜9：35）
セッション 1（9：35〜10：35） 座長：岩瀬 良範 （埼玉医科大学 麻酔科）
1 超音波希釈法による心拍出量測定と熱希釈式心拍出量測定の成人手 術患者における精度比較
　筒井 雅人（防衛医科大学校 麻酔科）
2 静脈麻酔薬予測濃度表示装置 (PkBox) を研修医が使ってみて ― 感想と開発者からのコメント ―
　尾崎 眞（東京女子医大 麻酔科）
3 シリンジポンプ動作記録装置の開発
　内田 整（大阪大学大学院医学系研究科 麻酔・集中治療医学講座）
4 脈波の反射
　杵淵 嘉夫（東海大学 開発工学部 医用生体工学科）
5 陽圧・陰圧呼吸，共に作動する気道内圧アラーム
　田中 義文（京都府立医科大学大学院医学研究科 麻酔学教室）

セッション 2（10：3511：35） 座長：内田 整 （大阪大学 麻酔科）
6 pdf ファイルの使い方工夫
　諏訪 邦夫（帝京医学技術専門学校）
7 “ R ”の統計以外への応用
　萩平 哲（大阪大学大学院医学系研究科 麻酔集中治療医学講座）
8 ビデオ気道確保器具を支えるビデオ技術の進化
―DLNA および関連技術とメモリレコーダーによる小型化―
　岩瀬 良範（埼玉医科大学 麻酔学）
9 CICR アッセイ用シーケンサーシステムの開発
　岩瀬 良範（埼玉医科大学 麻酔学）

ランチョンセミナー（12：0013：00）
「一酸化炭素、非侵襲的モニタリング」
　演者：前山 英男（マシモジャパン） 座長：梅田 英一郎（防衛医科大学校 麻酔科）

特別講演（13：0014：00）
「テクノロジーからみた血液ガス測定の歴史」
　演者：諏訪 邦夫（帝京大学 八王子キャンパス） 座長：風間 富栄（防衛医科大学校 麻酔科）

セッション 3（14：0015：00） 座長：重見 研司（福井大学 麻酔科）
10 当院での手術部門システム導入について
　寺井 岳三（大阪労災病院 麻酔科）
11 麻酔科領域での Scalable Vector Graphic の応用
　斎藤 智彦（独立行政法人国立病院機構 南岡山医療センター 麻酔科）
12 電子化麻酔記録システム paperChart の数値/文字列計算メカニズムにつ いて
　越川 正嗣（神戸海星病院 麻酔科）
13 国際比較出来る電子カルテ導入
　浅山 健（エイ・エス・エイ会）

閉会の辞：風間富栄（15：0015：05）

特別講演
テクノロジーからみた血液ガス測定の歴史
帝京大学 八王子キャンパス 諏訪邦夫
時代を遡るスタイルで、テーマの問題を提示する I. 現代の血液ガス装置 全自動コントロール：コンピュータ制御 演算装置:コンピュータで。記録も当たり前 分光光度計組み込み機器が増加:本来は別系統 極端な小型装置も開発 II.1970 年頃：アナログ機の完成 電極系と電気系は分離、ディスプレイだけディジタル、プリンターなし 計算はすべて手作業 (電卓、計算尺、ノモグラム、グラフ) 較正も手作業 III.1955 年頃：Po2 &Pco2 電極の創始 二つの電極がほぼ同時に開発された。偶然？プラスティック産業？ Po2 電極：クラーク Pco2 電極:ストウセブリングハウスの果した役割 「Pco2 電極なしでの Pco2 測定」：アストラップ IV.pH 測定と塩橋 ブジェルムによる塩橋採用 1950 年→血液 pH 測定可能に ベックマンによる pH 計商品化と真空管アンプ 1930 年：血液以外のｐ H 測定容易に ハーバーのガラス電極：1909 年、実用性は乏しい 水素電極 (ネルンストら)：1900 年、初めて測定可能に V. Po2 測定の試み ダニールの発見：1897 年。実用にならず ヘイロフスキーのポーラログラフィー：1922 年、水銀滴下 電極で組織 Po2 測定：1930 年以降、血液には不可能、較正も困難 VI. 附：意外に永いカプノメトリーの歴史 1952 年:医学応用開始 浅山健先生らの日本での使用 1980 年以降、麻酔モニターとして確立 機器の改良：価格低下、小型化、較正容易不要 「換気している」ことの確認、「麻酔のモニターの標準」 VII. 経皮測定の問題：較正と価格がネック

一般演題
1 超音波希釈法による心拍出量測定と熱希釈式心拍出量測定の成人手術患者におけ る精度比較
筒井 雅人、風間 富栄 防衛医科大学校 麻酔科
はじめに：心血管系のリスクのある患者の麻酔管理において、心拍出量などの循環動態を正確に把握することは重要である。肺動脈カテーテルを使用した熱希釈式心拍出量（COtd）測定法は正確であるが、挿入により様々なリスクが生じる。炭酸リチウムやインドシアニングリーンなどの色素を用いた低侵襲的な心拍出量測定法が開発されてきたが、投与することによる副作用など、それぞれに短所もある。超音波希釈式心拍出量測定（COud）は生理食塩水を中心静脈から投与することにより、血液の超音波透過速度の変化を利用する低侵襲的な測定法である。COud と COtd との精度比較を行った。方法：29 人の全身麻酔患者にて比較を行った。麻酔導入後、左右の橈骨動脈から観血的動脈圧測定ラインを挿入し、シースおよび肺動脈カテーテルを内頚静脈より挿入した。右橈骨動脈カテーテルおよび内頚静脈シースに超音波希釈式心拍出量測定システムを接続した。血行動態が落ち着いていると思われる時にランダムに測定を施行。各測定は 30 分以上の間隔を空けて行った。結果：患者 29 人から取得したデータから適正なものを抽出し、計 142 のデータを比較検討した。心拍出量は 2.679.80L/min の範囲で、直線回帰で r=0.91、Bland-Altman 解析で bias は 0.02、limitsofagreement は-1.04 1.08 となり、パーセンテージエラーは 23.53%であった。結語：COud は広い範囲において COtd に追従した。この測定法は改良を重ねることにより、より簡便かつ正確に心拍出量を測定できる可能性がある。

2 静脈麻酔薬予測濃度表示装置 (PkBox) を研修医が使ってみて―感想と開発者からのコメント―
尾崎 眞（東京女子医大麻酔科） 内田 整（大阪大学大学院医学系研究科 麻酔・集中治療医学講座）
数年前までは、例えばプロポフォール投与開始 10 分間は、10 μ g/kg/h で投与し、その後 10 分は 8 μ g/kg/h そして最終的には 6 μ g/kg/h で投与するという投与方法が推奨されていたものが、Target Controlled Infusion がプロポフォールで利用できるようになった現在では、一般的には 4 μ g/ml を目標にスタートし、就眠時の効果部位濃度を確認しておき、それが 1.8 μ g/ml であれば、それを目安にしてその後は維持濃度を決めて投与するという考え方が日常的になった。従って 2007 年になって我が国にやっと導入されたレミフェンタニルであるが、その添付文書やガイドラインには 0.5 μ g/kg/分で投与開始し、気管挿管後に 0.25 μ g/kg/分で投与すると書かれていても、例えば各種文献からレミフェンタニルの気管挿管刺激を抑制するための 95 %有効効果部位濃度は、6.0ng/ml であることから、例えば循環系の予備力の少ないと判断された症例では、気管挿管時に効果部位濃度が 6.0ng/ml 辺りになるように、もっと長い時間をかけてその濃度を達成するように 0.2 μ g/kg/分で開始後 10 分前後で挿管しようという計画を立てることに違和感を覚えない。すなわち、我々の基本的な薬物投与の概念の中に TCI による効果部位濃度、そしてそこには血中濃度との時間的ずれが存在することや、薬剤投与の目標とするのは効果部位の濃度であって、体重当たりの投与量が効果を規定しているのではないという投与方法が根付きつつあるのだ。以上のことをプロポフォール以外では TCI ポンプが現在利用できない日本の現状でも、簡便にかつ効果的に麻酔科研修中の卒後 23 年の医師に教育するために工夫をした。すなわち、シリンジポンプの投与情報をシリアルポート経由で吸い上げることで、静脈麻酔薬の薬物動態・薬力学計算を行い、予測濃度を液晶画面に表示する装置（PkBox）を麻酔科研修教育に活用した。この際に、レミフェンタニル投与における PkBox の臨床使用により研修医からのフィードバックを収集したので、PkBox 開発者からのコメントを交えて報告する。

3 シリンジポンプ動作記録装置の開発

内田 整 1，萩平 哲 1，高階雅紀 2
大阪大学大学院医学系研究科 麻酔・集中治療医学講座 1) 大阪大学医学部付属病院 手術部 2)

現在の静脈麻酔では，静脈麻酔薬の血中あるいは効果部位濃度を意識して投与量を調節する概念が重要であり，臨床麻酔の教育ではそのような投与方法を指導している．通常，投与量の変更は麻酔記録に記載されるが，短時間に行う頻回の変更やボーラス（早送り）などは麻酔記録に正確に反映されないこともある．従って，このような麻酔記録からは，研修医がシリンジポンプをどのような手順で操作したかを再現することは困難である．本研究では，マイクロコンピュータを応用して，シリンジポンプの動作状況を記録する装置を試作した．試作した装置は 16 ビットマイクロコンピュータ H8/3664（ルネサステクノロジー）を搭載したマイコンボードで開発した．本装置はシリンジポンプと RS-232C で通信を行い，30 秒ごとの流量および総投与量，また，早送りやアラームなどのイベントを記録する．これらのデータは EEPROM（Electronically Erasableand Programmable Read Only Memory）に記録されるため，電源をオフにしても消去されることはない．記録したデータの取り出しは RS-232C 経由で可能で，テキストファイルを出力するように設計した．なお，本装置に接続可能なシリンジポンプは Graseby およびテルモの各シリーズである．本装置が記録したデータと麻酔記録を対比して解析することにより，手術の進行や薬物動態を考えて適切な静脈麻酔薬の管理ができているかどうかを客観的に評価できるようになる．また，記録したデータから，血中および効果部位濃度の変化を再現できることも教育的価値がある．

4 脈波の反射

杵淵嘉夫、＊嶋田勝斗 東海大学開発工学部医用生体工学科、＊埼玉大学大学院理工学研究科

血管内を血圧脈波が伝搬していくと、脈波形は次第に変形する。ピーキング現象やスティープニング現象に与える末梢からの反射波の影響について実験的に検討した。【1. 方法】弾性率 (硬度) の異なる (柔 C2.5 硬 C12.5)、内径 3mm、外径 5mm のウレタンチューブを作成し、全長 40cm の血管モデルとした。拍動ポンプから血圧波形やインパルス波形を印加して波形の変化を観察した。硬度のレンジは血管系のそれと概ね同等である (血管半径方向の弾性率分布は無視する)。また、ウレタンチューブの物理定数 (抵抗、慣性、弾性) を回路定数 (R、L、C) に翻訳した後、回路シミュレータ上 (PSpice V2.0) で各部の電圧変化を詳細に検討した。【2. 結果】(1) ウレタンチューブ末梢端から発生する反射波 (血圧波) が、進行する入射波に重畳するのを避けるためダミーチューブを付加すると、弾性率分布を反映したピーキング現象やスティープニング現象、CPB 後の圧較差現象が確認された。(2) これらの現象はシミュレータ上で血管抵抗に相当する抵抗成分をゼロにしても変わらないので、弾性率分布によるものとしてよい。(3) ダミーチューブを短くすると反射波が進行波に重畳する。(4) 進行波、反射波ともに、弾性率分布に従って脈波伝搬速度が変化する。ピーキング・スティープニング現象と圧較差現象では進行波に対する反射波の重畳の仕組み (時間差) が異なる。【3. 考察】ウレタンモデルでは弾性率分布とそれに依存する脈波伝搬速度および系の長さ等から、観測する部位において進行波に対する反射波の位置を特定することができる。したがって脈波の形成と両成分の寄与の度合いをある程度まで知ることができる。大動脈から橈骨動脈に至る血管内圧の変化には弾性率分布の影響のみならず末梢からの反射波の影響も確実と思われるが、反射部位を特定することが難しいので定量化はできていない。現在、動脈系の入力インピーダンスの位相周波数特性から反射部位を推定する方法について検討中である。

5 陽圧・陰圧呼吸，共に作動する気道内圧アラーム

田中 義文 京都府立医科大学大学院医学研究科

麻酔学教室数年前より全身麻酔器のベンチレーターは気道内圧アラームを装備しなければ市販できなくなり，それに伴って独立した気道内圧アラームも生産中止となった．我々の教室では今でも日本メディコ社製気道内圧アラームを使用している．もちろんベンチレータには気道内圧アラームが装備されているが，警報装置を二重に備えることは無駄ではないとの考えてある．しかしこのアラームも経年変化で故障することも多くなり，最近は製造元に修理を依頼しても修理できないと言われるようになった．そこで，従来の内圧アラームより性能の優れた気道内圧アラームを自作することにした．従来の装置はある一定の時間，陽圧が検出できなければ警報が鳴る．また，異常高圧になれば警報が鳴るなどの 2 機能が備えられているが，これでは自発呼吸下の陰圧平圧時ではアラームが鳴りっぱなしとなり，警報装置として役にたたない．そこで，オペレーションアンプで過剰陽圧，有効陽圧，有効陰圧，過剰陰圧の 4 点の圧選択レベルのコンパレータを作成し，そこから得られるパルス信号をロジック回路を介して，3 種の警報音発生 IC を駆動させた．気道内圧測定素子はディスポーザブル動脈圧測定トランスデューサを用いた．アラームとしては良好な結果が得られたが，警報音の音色識別に混乱が生じることも明らかになった．音声による警報の説明が必要であることも判明した．

6 PDF ファイルの使い方工夫

諏訪邦夫 帝京医学技術専門学校

最近、インターネットの発表を中心にｐｄｆファイルの使用が増えている。この形式は利点も多いが、問題点も少なくない。その問題点を指摘して私の解決策を述べるが、会員諸氏のお知恵も拝借したい。PDF ファイルの問題点をまずいくつか指摘する。1. 画面で読みにくい 1-1 縦と横の比率の問題：パソコン画面は横長なのにファイル画面は縦長である。この点で横 2 段組・縦書き論文・縦書き書籍などでは、不都合の度合いが特に大きい。1-2 ソフトウェアの速度が遅くて、高速で読むのに適さない。1-3 目次と本文との関係:目次と本文のリンク機能が貧弱で、無料の”Reader”には作成機能はもちろんない。有料ソフトウェアでも使いにくく、私はマスターできていない。インターネット提供の PDF のデータで、目次がついていながら目次から本文へのリンク機能を使っているデータはごく少数である。PDF ファイルの場合、画面に元の印刷頁数が残っている場合が多いが、それは PDF ファイルの頁数とは当然ずれている場合が多い。それがさらにファイルを使いにくくする。極端にひどい例では、1 面が 2 頁ずつになっている場合がある。ごく小さな本などの例である。これは「見る」には便利だが、「読む」にも印刷にも不便である。1-4 メモしにくい:文章を読めばメモを書き込みたいが、それが「使いやすい」とは言えない。2. 個人的解決策 PDF ファイルの使いにくさを、個人的には一応下のように解決している。2-1. 文字のテキスト化:文字がテキスト化されていれば、テキストファイルを作成してエディターで読む。テキストの冒頭に PDF ファイルの名を書き込んでリンクする。文章部分が圧倒的に多いファイルには有用。作成が簡単な場合と、やや困難な場合がある。2-2. テキスト化されていない場合：画面で眺めるだけで「ダウンロードして精読」しない。どうしても読みたい場合は不便だが画面で読むか、あきらめて印刷する。ファイルが小さければ自分で改めて電子化する。2-3.PDF を読むソフトウェアはフリーソフトウェアも含めて数多くあるので、高速なものを探したいがまだ見つかっていない。結語：PDF ファイルの欠点を指摘し、自分の工夫を述べた。会員諸氏のサジェスチョンを御願いします。

7 “ R ”の統計以外への応用

1 萩平 哲，2 高階 雅紀，1 内田 整，3 森 隆比古
1 大阪大学大学院医学系研究科 麻酔集中治療医学講座 2 同附属病院 手術部
3 大阪府立急性期総合医療センター 麻酔科

”R”は Windows, Macintosh, Linux などのマルチプラットフォームで動作する強力な統 計処理ソフトウェアであり，フリーソフトウェアとして配布されている．”R”を用いれば 分散分析から多変量解析までほとんどの統計処理を 行なうことができる．さらに他の ソフトウェアとの連携にも優れたインターフェイスを有しており，処理した結果を 2 次元 や 3 次元のグラフとして表示したりすることもできるように設計されている． ”R” のコンソールは言わば UNIX の shell のようなものであり，実装されている関数を 利用すればデータ変換のためのフィルタを作成することも容易である．さらに”R”には ベクトルや行列の計算を行なう関数も多数組み込まれており， 最小 2 乗法による 直線 回帰なども簡単に行なえる機能を有している．こういった統計処理以外のデータ処理 にも”R”は有用であると思われる． ここでは，”R”を用いて DNA シーケンスを蛋白コードに変換するフィルタや，ベクトル および行列演算機能を用いて最小 2 乗法により直線回帰を行なう方法などについて 紹介したい.

8 ビデオ気道確保器具を支えるビデオ技術の進化 ―DLNA および関連技術とメモリレコーダーによる小型化―

岩瀬良範、菊地博達 埼玉医科大学 麻酔学

ビデオ気道確保機器は、ようやく普及の兆しを見せている。これらの機器の使用には、ビデオ機器による中継・投影と記録が不可欠である。我々は、本機器使用の早期からビデオ技術に着目し、最適なビデオ環境の構築によって大量の経験を蓄積してきた。本年に入り、民生用の 2 つの技術が我々の環境に大きな影響を与えたので紹介したい。1.DLNA(Digital Living Network Alliance) とその関連技術大容量の HDD レコーダーは、大量の録画が可能な利点の反面、その整理方法が問題になる。もし、外部記憶媒体として DVD に限らず、ネットワーク上の PC や HDD を使用できたら、その利便性は大幅に向上する。また、ネットワーク上の PC からサーバーとしての HDD レコーダーにアクセスして、機器の操作、編集作業、記録画像の閲覧などができれば、さらに快適だ。東芝の VARDIA シリーズには、これらのサーバー機能が搭載されている。DLNA 技術により、録画ファイルは mpeg 動画として、ネットワーク上のクライアント PC にコピーやストリーミングが可能である。同シリーズに組み込まれた JAVA ベースの Web サーバーで、QuickTime によるストリーミングや細かな操作が可能になっている。我々は、さらに本機 (RD-E300) に無線 LAN アクセスポイント機を接続して、無線 LAN からのアクセスを可能とした。この結果、ビデオ画面はそのまま無線中継が可能になった。2. メモリレコーダーによる小型化 HDD レコーダーは、機能は進化する一方で、機器の大きさは変わらない。しかし、ビデオ気道確保機器の使用は、緊急事態も含まれ、小型でフットワークの良好なビデオ記録が求められる。我々は miniDV テープによる超小型デッキ (GV-1000:Sony) を愛用してきたが、本機はすでに製造中止品目である。ASA(2007

サンフランシスコ) の展示会場にて、Supacam(http://supacam.com) という超小型ビデオカメラ・デッキ兼デジタルカメラ兼 MP3 プレイヤー兼ゲーム機を$328 で購入した。大きさは 122 × 66 × 25mm で、ビデオライン入力可能 (解像度 640 × 480、30fps)、ビデオファイルは USB2.0 または SD カードで PC 処理可能である。記録形式は、Xvid コーデックによる AVI ファイルである。本機により、Airwayscope(Pentax) が純正バッグとイントロックだけで録画可能になった。しかし、本機への AVI-mpeg4 ファイルの書き戻しと再生は困難で、単純にアナログビデオ信号の録画と再生に割り切った方が使いやすいようである。【考察と結語】HDD レコーダーは LAN への接続によって利便性が高まり、さらに無線化で便利になった。本発表は DLNA による「デジタル家電」の利便性が、そのままビデオ気道確保機器に応用できた実例といえよう。しかし、現在の衛星および地上デジタル放送はほぼすべてがコピー不可になっており、このような技術が活用できないジレンマがある。メモリレコーダーの発展は XVID 等のビデオ圧縮技術の関与が大きい一方で、まだ発展途上の感が拭えず安定期とはいえない。しかし今回紹介するストリーミング等の技術は、医用画像をさらに利便に応用できる可能性がある一方で、セキュリティーの問題も考慮する必要がある。

9 CICR アッセイ用シーケンサーシステムの開発

岩瀬良範、市原靖子*、菊地博達 埼玉医科大学麻酔学、東京臨海病院麻酔科*

CICR(Calcium Induced Calcium Release) は、悪性高熱症診断の検査法として最も精度が高いとされるが、その実施は容易ではない。今回我々は、CICR アッセイ用シーケンサーを開発したので報告する。【CICR による悪性高熱症診断の概要】1. 筋生検によって得られた骨格筋を顕微鏡下に筋線維まで分解し等尺性収縮を測定するための準備を行う。2. 多数 (16 種類) の実験溶液を正確なタイミングで投与と同時に微細な筋収縮力を記録する。3. それぞれの実験溶液に対する反応から判定を行う。【CICR アッセイ用シーケンサーシステム】16 種類の実験溶液をあらかじめ定めた順序とタイミングで実験槽に送り込み、筋収縮を記録し、排液する。これを実現するために、RS-232C 駆動の切り替えバルブ (Valco)、電圧駆動式ポンプ (Watson-Marrow)、電磁弁を、AD/DA コンバーター (Contec) 内蔵の PC(Hewlett-Packard) から制御するシステムを構築した。【方法】上記機器の制御には、VisualBASIC5.0J と AD/DA コンバーター付属の VB 用ライブラリを用いた。実験溶液の切り替えバルブ (Valco) は、RS-232C によるデジタル制御であるが、バルブ位置の誤りは重大な結果を招くので、ソフトウェア上で完全な確認を行ってから次のステップに進めた。電圧駆動式ポンプ (Watson-Marrow) は D-A コンバーターで段階的に電圧を発生し、滑らかなローラーポンプの回転開始と停止を実現した。電磁弁は、同コンバーターのデジタル出力を利用し、筋収縮も同コンバーターのアナログ入力 4 ポートを使用して、4 本の筋線維が同時に検査できるようにした。【結果】CICR によるアッセイ用シーケンサーシステムを実現した。【考察と結語】このシステムは、現在では実地稼動している。機器制御のソフトウェアの基本骨格と論理は、約半年をかけてデバッグし精度を向上させた。繊細な実験系に対して、使いやすく信頼性の高いソフトウェアを開発することは容易ではないが、その意義は大きい。

10 当院での手術部門システム導入について

寺井岳三、藤井 崇 大阪労災病院麻酔科

大阪労災病院では 2007 年 4 月、病院情報システム（HIS）の更新と同時に手術部門システムを新規導入した。システム導入の経緯と特徴、今後の構想について報告する。HIS 更新前は、1) オーダリングシステムに手術予約オーダ機能はあるが、紙伝票で手術申込する科もあり、手術予定表の完成は前日の夕方であった。2) 麻酔申込はすべて紙伝票であった。3) 麻酔記録は手書きであり、担当麻酔医による JSA 麻酔台帳への入力間違いや漏れがあった。4) 医事伝票は手書きであり請求漏れの可能性があった。5) 生体情報モニタが各手術室で異なり、麻酔科控室で集中監視ができなかった。これらの問題点を解消するために、当初は手術部門システムとして JSA 麻酔台帳／手術室管理システム 2006 を用いる予定であったが、機能は不十分であった。幸いにも予算が認められたので GE 社製 CentricityCIS-ORSystem を採用した。HIS は NEC 社製オーダリングシステム Mega-Oak が採用された。手術申込と決定は Mega-Oak 上で行い、決定後 CIS-OR に患者情報が取込まれる。麻酔術前診察は麻酔科外来の CIS-OR センター端末で行い、麻酔申込の伝票を廃止した。麻酔担当医は 2 日前に割り当て可能となった。麻酔科控室と手術室管理室のセンター端末で患者情報、手術進捗を参照できる。各手術室の麻酔記録端末では、モニタのデータが自動記録され、血液ガス分析装置と接続し検査結果が取り込まれる。看護師はノート端末で医事請求入力を行う。回復室のノート端末で回復室記録が作成できる。JSA 麻酔台帳システムに接続し、学会への偶発症報告が可能である。GE 社製 S/5 モニタがある手術室のみ集中監視が可能となり、今後モニタを更新予定である。麻酔記録の電子化により検索、集計、分析が容易となった。手術室在室時間が予定より延長した症例では、入退室、麻酔開始終了、手術開始終了などの時刻を「在室延長・入室遅延報告書」として出力し、担当医に延長理由と今後の対策を報告してもらうことで、手術室の有効利用を図っている。今後の構想として、静脈麻酔薬の予測血中・効果部位濃度をグラフ表示させる機能を希望しているが実現には課題が多い。無線 LAN を介して PDA から手術システムに接続し、院内どこからでも麻酔記録を表示でき、術前・

術後診察情報を入力できる機能があれば便利である。

11 麻酔科領域での Scalable Vector Graphic の応用
斎藤 智彦 独立行政法人 国立病院機構 南岡山医療センター麻酔科

われわれ麻酔科医が使用する画像データには，ビットマップや JPEG・PNG に代表されるラスター系のグラフィックデータと直線やベジエ曲線によるドロー系のグラフィックデータがある．ドロー系の描画データは，写真などの表現には不向きだが，描画をベクトル単位で扱うことが可能で，図形の拡大・縮小に伴う画質の劣化が見られないことや，格納データのサイズが小さくて済む等の利点があり，プレゼンテーションで用いられるシェーマなどに使用されている．しかし，パワーポイントなどの市販ソフトで作成されるベクトルデータは，アプリケーション固有フォーマットのバイナリ形式であり，格納形式も公開されていないため，数値データを元に自分でベクトルデータを作成することは難しい．Scalable Vector Graphic(以下 SVG) は，ベクトル形式の画像を記述するための言語であるが，仕様は公開されており，XML 形式のテキストファイルであるため，エディタでも簡単に編集することができる．専用のビュアーの他，Adobe 社の Illustrator，FireFox，Opera など一部の WWW ブラウザが対応している他，Internet Explorer でもプラグインを用いることで，簡単に表示することができる．テキストファイルを出力できるソフトウェアを用いれば，数値データから SVG ファイルを作成することも可能である．SVG の主な機能とマクロによる SVG 形式の簡易チャート作成プログラムについて報告する．

12 電子化麻酔記録システム paperChart の数値/文字列計算メカニズムについて
越川正嗣 神戸海星病院麻酔科

以前本会で発表した麻酔記録ソフトウェアを全面的に見直し，内部に次のような 3 つの仮想機械を備えた構造に書き換えた．1．各設定ファイルを読み込み，階層化された辞書を構築する部分．2．画面の描画（シンボル，文字列，ボタン，座標変換など）を行う部分．3．各種の値（数値，書式付き数値，その他の文字列）の計算機構．その中でも諸値の計算機構は本システムの中でも重要な位置を占め，次に述べるデータを取り扱う．A．患者属性（患者名，生年月日など，入退院を通して普遍なもの）B．イベント情報（麻酔開始/終了時刻，麻酔終了時刻の麻酔終了状態など）C．イベント付帯情報（硬麻の穿刺位置，挿管のチューブサイズなど）D．薬剤単品ごとの使用量とグループ（漿質液，膠質液，血液など）別の使用量 E．バイタルサイン値の参照 A E はそれぞれが密接に関係している．たとえば“ 人工心肺前漿質輸液量 ”を求める場合は患者入室時から人工心肺中の輸液量途中残量入力時点までの範囲で漿質輸液グループに分類されている薬剤の総計を求めることになる．あるいは“ 麻酔終了時 NIBP 値と SpO2 値 ”は麻酔終了時刻から遡って最後に測定された NIBP 測定値と，その時刻近辺の SpO2 測定値（複数点）の中央値または最低値を算出する必要がある．JSA 麻酔台帳項目の中で，もっとも算出困難なものは“ 麻酔中最低血圧 ”である．麻酔記録ソフトウェアの動作終了時の 12 秒の間に麻酔開始から終了までのバイタルサインをスキャンし，artifact の影響を除いた値を算出しなければならない．この“ 麻酔中最低血圧 ”だけはまだ実用的なものになっていない．麻酔台帳はかなりの施設で電子化されていると思われるが，麻酔記録が手書きで行われることが多い．麻酔記録が電子化されたら，当然そこから麻酔台帳へは可能な限り手入力なしでデータを転送したい．今回フリーソフトとして公開する paperChart は，この点にかなりの比重をおいて製作したので，このコンパイラ/インタープリタ系について発表する．

13 国際比較出来る電子カルテ導入
浅山 健 エイ・エス・エイ会

始めに部外者が理解難しい基幹病院の手術室で行われる麻酔・手術の診療を公開できる形に出来る麻酔科電子カルテ導入を提言目的 1．手術室の診療を透明化して、手術室看護婦は勿論、麻酔科医師・手術を行う各科医師、臨床工学士は勿論、環境を整備する職員、事務職員、更に関連する薬剤師に対して、仕事内容を、地元医師会、経営者、所轄官庁の外部が理解する方法として、電子カルテは有用 2．関連各科に所属する患者に、麻酔科が診療する責任範囲を明文化するには、電子カルテは役立つ 3．明文化に基く責任を果たす必要人員を確保するに電子カルテは有用 4．電子カルテ導入を提言する最大理由として、責任責任を経済的価値で表す診療報酬を支払基金へ請求出来ます方法 1．米国麻酔科学会 (ASA) が採用する RelativeValueGuide を導入する電子カルテ。これを導入する作業に従う時、手術に対応する麻酔科診療の技術度・麻酔科診療に従事する時間・患者の年齢と予定手術適否の補正の、三因子に基く技術が先ず単位で表現される 2．次いで、この単位に対して地域、病院、時代に応じる係数を掛け合わせるとき、金額で表されます 3．但し、この作業には MicrosoftCo. の協力が必須です。この協力を得るには、私共の学会の社団法人・麻酔科学会の決定が必須です電子カルテに基く麻酔科の診療報酬額を支払基金が公表するには、日本医師会・病院協会・厚生労働省の合意が必須です。考察現在、支払基金が公表する診療科目別支払金額に麻酔科金額の記載はない事実がある。筆者は支払基金の事務局に問い合わせて、確認しています。理由には、患者が所属する診療科の麻酔科の請求金額に含まれると説明します。解決には、先ず、新設の社会医療法人制度に麻酔科が当て嵌まる実績が必要です。麻酔科の場合、相手先の基幹病院との診療契約が必要。基幹病院が麻酔科の電子カルテを採用して初めて、目的は達成に大きく近くなりま

す。電子カルテ導入で予想する事態麻酔科の電子カルテ導入に基く手術室診療の質・量が透明化されます。この結果、内視鏡手術など患者に負担少ない手術に対応する静脈点滴麻酔が普及する条件が整いましょう。しかし、麻酔科医師に対する静脈点滴麻酔の精神的ストレス負担は大きく、負担に対応する麻酔科の勤務体制が必要です。電子カルテ導入で関係者の理解を得られると考えます。

特別寄稿：換気力学における数学的モデルに関して

萩平　哲

はじめに

Lung mechanics の理解や種々の人工呼吸法の simulation による研究などの目的でこれを等価電気回路で置き換えた数学的モデルがよく用いられている．しかしながらこれまで報告されているモデルは現実に即していないところもある．今回，より現実に即した数学モデルを構築し換気力学上の各パラメータがどのように表現されるかを示し，さらに基礎的解析の具体例を示しモデルの応用法を示した．

モデルの構築

等価電気回路で肺を表現する場合，気道の各要素は電気回路では以下のように表現される．ガスの流れ (気流) は電流 (i) であり，吸気によって増加した肺容量は電荷 (Q) に相当する．つまり，これはコンデンサに蓄えられた電荷ということになる．気道抵抗は電気抵抗 (R) であり，自発呼吸もしくは人工呼吸によって生み出される吸気駆動力は電圧 (E) で表され，回路上では電源として示される．

これまでよく用いられていた最も単純なモデルはコンデンサと抵抗１個ずつによる RC 回路である．しかしながら RC 回路モデルは単純化されすぎていて現実にそぐわない．例えば，dynamic Auto-PEEP 測定法として気道閉塞法があるが，この方法で気道閉塞を行った時の気道内圧曲線は図１に示されるように上に凸の曲線を示し，やがて定常状態に近付く形を示す．気

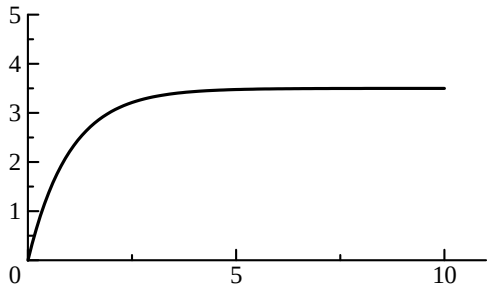


図 1 気道閉塞時の気道内圧変化

道閉塞という操作は口元に当たる位置で回路を open にすることに相当するが，RC 回路でこの状態をシミュレートすると，この場合瞬時に電流は流れなくなるため肺がこのモデル通りであるなら気道内圧は気道閉塞直後から一定の圧を示すことになり図１のような変化は示さないことになる．

これは RC 回路モデルが肺胞レベルしか考慮していないためであり，実際には中枢気道に相当する回路を付加する必要がある．これを付加した回路が図２である．図のごとく RC 回路が並列に接続されていることになる．直列ではない点に注意する必要がある．直列回路では外部から圧力をかけた場合定常状態でも肺胞にその圧力がすべてかからないこと，肺胞と中枢気道のガス容量が常に等量になってしまうことから矛盾が生じる．

このモデルでは中枢気道のコンプライアンスおよび肺胞のコンプライアンスがそれぞれコンデンサの静電容量 C_1, C_2 に，中枢気道と末梢気道の気道抵抗が電気抵抗 R_1, R_2 に相当する．ま

大阪大学大学院医学系研究科　麻酔・集中治療医学講座

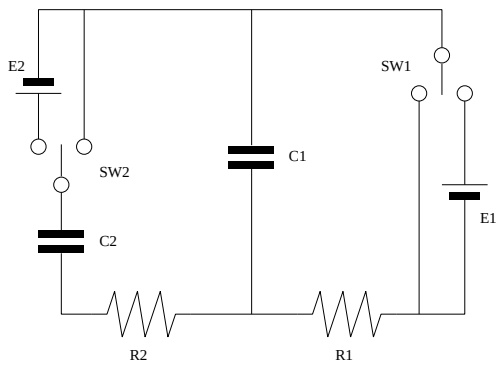


図 2 肺モデルの等価電気回路
C1:中枢気道コンプライアンス，C2:肺胞コンプライアンス，R1:中枢気道抵抗，R2:末梢気道抵抗，E1:人工呼吸器陽圧，E2:自発呼吸陰圧，人工呼吸時は SW2 をショート，SW1 の開閉で陽圧呼吸モード，自発呼吸時は SW1 をショート，SW2 の開閉で陰圧平圧呼吸になる．

た電源 E_1 は人工呼吸器による陽圧を示し，スイッチ SW_1 の切替えにより回路端がショートされると呼気の状態となることを表している．一方回路左側の肺胞側に付加されている電源 E_2 は自発呼吸の駆動力を表すものであり，自発呼吸のみの場合にはスイッチ SW_1 はショートされた状態でスイッチ SW_2 の開閉により自発呼吸が行われることになる．人工呼吸器による陽圧は中枢気道を介して肺胞に作用する外力であるから，E_1 は C_1, C_2 を並列に並べた形で作用するように配置されるが，一方自発呼吸の吸気圧は内力として肺胞に作用するため E_2 は C_2 に対し直列に配置される．また，気道が閉塞した状態ではガスの流れが止まるため，電気回路では電流が流れなくなる状態に相当する．つまりこれはスイッチをオープンにするという動作に相当することになる．例えば呼気時に口元を閉塞させる操作は SW_1 をオープンする動作になる．つまり各スイッチにはショート，オープン，電源接続の３通りの状態があり，これらはそれぞれ実際の肺では気道開放，気道閉塞，加圧という状態に相当することになる．

中枢気道および肺胞における容量の増加分はそれぞれ Q_1, Q_2 である．またそれぞれにおけるガス流量は単位時間当たりの容量の変化率であり圧は容量をコンプライアンスで除した値となる．さてこれまでに報告されているモデルにおいて AutoPEEP に相当する圧力を数式の中にあらかじめ設定しているものが見られるが，これは適切とは言えない．例えば Marini ら [1] は

$$P_T = \frac{V_T}{C} + P_{ex} + \dot{V}R_i$$

として呼気終末肺胞内圧 P_{ex} を定義しているが，AutoPEEP は肺内に残存するガスによって生み出される圧力であるから，ある状態での V_T/C の値であるべきである．等価電気回路で考えれば正しくない理由が理解できる．このような圧力を数式内に置くことは等価回路では自発呼吸の駆動力と同様の位置に電源を置くことになる．定常状態では確かに肺胞に一定量のガスが残存し正しいように見えるが，このガスをなんらかの手段でなくした場合のことを考えてみよう．本来なら一度なくした残存ガスは吸気で再び肺に入ってこない限り 0 のままのはずである．Marini ら [1] のモデルでは呼気終末状態の電気回路は図 2 では E_2 が AutoPEEP を生み出す電源である．この場合，SW_1 はショートされた状態であるから一度 $Q_2 = 0$ としても定常状態では常に $Q_2 = C_2E_2$ が成立することになり現実にそぐわない．これは本来残存ガスによって生み出される受動的な圧力を能動的な圧力として定義したことに起因している．AutoPEEP とは呼気終末の時点における Q_2/C_2 として表すべきものである．この定義自体は気道閉塞の有無にかかわらず定義できるものである．

呼気中に気道閉塞が生じる場合には電気回路上

では別に扱う必要がある．気道閉塞と気道抵抗が非常に大きい状態では実際の肺では大きな差はないように考えられるが電気回路上では気道閉塞は呼気のある時点 (Q_2 がある値になった時点) で SW_2 がオープンになる状態に相当する．吸気では自発呼吸の場合にはここから SW_2 が E_2 側に切替えられることになり，人工呼吸では Q_1/C_1 が気道閉塞時の Q_2/C_2 に等しくなった時点で SW_2 がショートされることでシミュレートできる．一方気道閉塞を伴わない場合には回路操作はそのままでよい．また現実には気道閉塞と気道抵抗が非常に大きい状態を判定することは困難であるがシミュレーションを行う場合にはそれぞれの場合を考えた方がよいと考えられる．

なお人工呼吸において人工呼吸器からかける PEEP は人工呼吸器の構造によって電気回路での表現が異なる．active に人工呼吸器側から加圧している場合には E_1 の位置に電源を置くことと等価である．一方 PEEP valve のように肺胞内の残存ガス量を調節しているような場合には，E_1 が PEEP と同等の電位になると抵抗が極端に大きくなるダイオードをここに入れるようなものである．このように $\mathrm{PEEP_i}$ と $\mathrm{PEEP_e}$ は本来の性質が異なることに注意しておく必要がある．

等価電気回路の数式化

図 2 の等価電気回路を数式化すると一般に以下の (4) 式, (5) 式が成立する．もちろん SW_1, SW_2 によって式は変わる．

$$E_1 = \frac{Q_1}{C_1} + R_1\left(\frac{dQ_1}{dt} + \frac{dQ_2}{dt}\right) \quad (4)$$

$$\frac{Q_1}{C_1} = E_2 + \frac{Q_2}{C_2} + R_2\frac{Q_2}{dt} \quad (5)$$

これより

$$\frac{dQ_1}{dt} = \frac{-(R_1+R_2)}{R_1R_2C_1}Q_1 + \frac{1}{R_2C_2}Q_2 + \frac{E_1}{R_1} + \frac{E_2}{R_2} \quad (6)$$

$$\frac{dQ_2}{dt} = \frac{1}{R_2C_1}Q_1 - \frac{1}{R_2C_2}Q_2 - \frac{E_2}{R_2} \quad (7)$$

この式はこのモデルの元ではいかなる条件下でも成立する．R, C を定数とした場合にはこの方程式は 2 元連立線形微分方程式となる．E_1, E_2 を時間 t の関数で表せばこれらの関数が特定の条件を満たす場合には解析的に解くことも可能となる．解析的に解けないような場合でも多くの場合，コンピュータを用いて近似解を得ることが可能である．以下では特にことわりのない限り R, C は定数として扱うものとする．

では人工呼吸の場合についてこの式を解いてみよう．

吸気時

人工呼吸の場合，吸気時には上述のように電池が加わった回路となっている．SW_2 はショートのままであり，$E_2 = 0$ である．ここでは PCV (pressure control ventilation) を行った場合の肺胞容量の変化をみてみよう．この場合 E_1 は定数になるので以下の式が成立する．

$$E_1 = \frac{Q_1}{C_1} + R_1\left(\frac{dQ_1}{dt} + \frac{dQ_2}{dt}\right) \quad (8)$$

$$\frac{Q_1}{C_1} = \frac{Q_2}{C_2} + R_2\frac{Q_2}{dt} \quad (9)$$

呼気時

吸気のための時間が十分にあり吸気が完全に完了した後に呼気が開始されるとすれば，呼気の開始時にはガスの流れは 0 であるから，呼気開始時の容量 $Q(t_{ei})$ は

$$Q_1(t_{ei}) = C_1E_1 \quad (10)$$

$$Q_2(t_{ei}) = C_2E_1 \quad (11)$$

また，呼気時では (8) 式の E_1 が 0 となるから，(8), (9) 式から以下のようになる．

$$\frac{dQ_1}{dt} = \frac{-(R_1+R_2)}{R_1R_2C_1}Q_1 + \frac{1}{R_2C_2}Q_2 \quad (12)$$

$$\frac{dQ_2}{dt} = \frac{1}{R_2C_1}Q_1 - \frac{1}{R_2C_2}Q_2 \quad (13)$$

(12), (13) 式の連立線形微分方程式は Laplace 変換 (Appendix 2 参照) を用いて解くことができる．ここでは解のみを示す．$(R_1C_1 - R_1C_2 - R_2C_2)^2 + 4R_1R_2C_1C_2 = A$ とすれば

$$Q_1 = \frac{C_1E_1}{2\sqrt{A}}[\{(R_1C_1 - R_1C_2 + R_2C_2) + \sqrt{A}\}e^{\beta_1 t} - \{(R_1C_1 - R_1C_2 + R_2C_2) - \sqrt{A}\}e^{\beta_2 t}]$$

$$Q_2 = \frac{C_2E_1}{2\sqrt{A}}[\{(R_1C_1 + R_1C_2 + R_2C_2) + \sqrt{A}\}e^{\beta_1 t} - \{(R_1C_1 + R_1C_2 + R_2C_2) - \sqrt{A}\}e^{\beta_2 t}]$$

ただし，呼気の開始を $t = 0$ としている．また，

$$\beta_1 = \frac{1}{2R_1R_2C_1C_2}\{-(R_1C_1 + R_1C_2 + R_2C_2) + \sqrt{A}\}$$

$$\beta_2 = \frac{1}{2R_1R_2C_1C_2}\{-(R_1C_1 + R_1C_2 + R_2C_2) - \sqrt{A}\}$$

$$A = (R_1C_1 - R_1C_2 - R_2C_2)^2 + 4R_1R_2C_1C_2$$

$$= (R_1C_1 + R_1C_2 + R_2C_2)^2 - 4R_1^2C_1C_2$$

であるから $\beta_2 < \beta_1 < 0$ となり，Q_1, Q_2 は常に単調減少する．

自発呼吸の場合

一般に自発呼吸の呼吸パターンは trigonometric function(三角関数) でシミュレートされている．従って (6), (7) 式において E_2 を以下のごとく定義しこれを解くとよい．

$$E_2 = \frac{V_2}{2}(1 - cos\omega t) \quad (14)$$

なお，ω は 2π/T だから 1 分間に 10 回の呼吸であれば $\pi/3$ である．

E_1, E_2 がこの程度単純な関数である場合には連立微分方程式を変形して適当な定数 a,b によって $a \cdot Q_1 + b \cdot Q_2$ を一つの関数とした 1 階線形微分方程式とする．ここで Appendix 1 に示した 1 階線形微分方程式の解法を適応すると，定数 a,b を 2 組求めて同じ解法を用いれば Q_1, Q_2 が求まる．

モデル方程式を用いたシミュレーション

(4) 式，(5) 式にパラメータを代入して Lunge-Kutta 法でシミュレーションした例を図 3 に示す．ここでは吸気圧 15cmH_2O で I:E=1:2，吸気時間 2 秒という一般的な PCV での換気を設定している．また，便宜上安静時呼気終末の容量を 0 としている．ここに示した以外のパラメータは体重 50kg の成人を想定して設定した．Q_1, Q_2 の単位は ml，C_1，C_2 の単位は ml/cmH_2O, R_1，R_2 の単位は cm$H_2O \cdot$ sec/ml である．実際の正常値を設定すれば，そのまま生理的状態のシミュレーションが可能である．

次に，病的肺の場合を考えてみよう．COPD 患者では肺胞のコンプライアンスは高く一方で末梢気道抵抗も特に呼気時には著明に上昇している．重症の COPD 患者を用手換気すると吸気

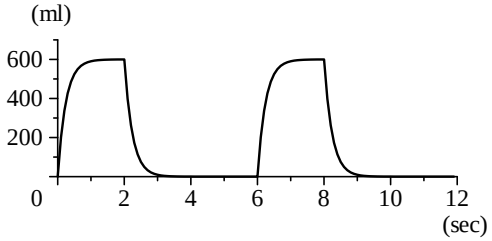


図 3 正常人における PCV 時の肺容量の変化のシミュレーション

時には換気バックがスポンジのように軟らかく非常に低い圧でも吸気が行える．一方で呼気時にはバッグを手放してもバッグの膨らみは非常に悪く，徐々にしか膨れて来ない．またいつまでも呼気が終らない．このような状態をシミュレーションするために C_1 を大きくし，R_1 を呼気時と吸気時で別個に設定して特に呼気時の R_1 を大きくした．先のシミュレーションと同じ人工呼吸を行った場合を正常の場合と比較した (図 4)．COPD 患者では従圧式換気の場合であっても肺コンプライアンスが大きいために容易に肺容量は大きくなる．つまり，吸気圧は低く設定する必要があることが判るだろう．また，呼気は著明に延長しこの換気条件では呼気終末にも肺に 230–240 ml もの残気が存在するようになっており，autoPEEP が生じている．先に述べたようにここでは安静時呼気終末の肺容量を 0 と設定して計算しているためここで言う残気量は呼吸生理学で定義される「残気量」とは異なることに注意して頂きたい．

そこで吸気圧 10cmH_2O で I:E=1:4，吸気時間 1.5 秒という設定での換気に変更してシミュレーションを行った (図 5)．この設定の場合には呼気終末の残気量は 75 ml 前後にまで減少しているが，それでもまだ 0 にはなっていない．一方で分時換気量は 3.2 L とかなり少ない状況である．このような患者の全身麻酔管理では酸素化を術

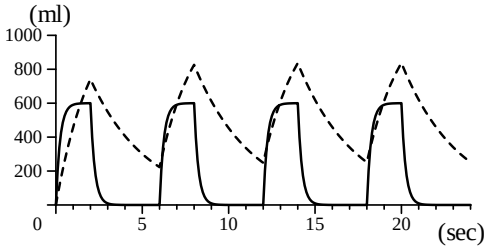


図 4 COPD 患者の PCV 時の肺容量の変化のシミュレーション

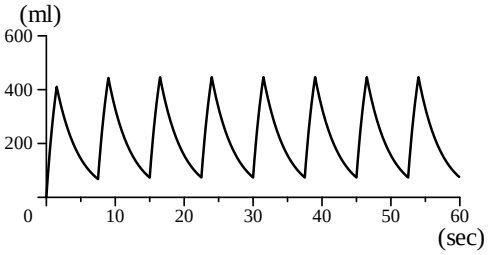


図 5 COPD 患者に合わせた PCV 時の肺容量の変化のシミュレーション

前の状態に近い程度に吸気酸素濃度をコントロールしながら，P_aCO_2 に関しては高くなっても許容するといういわゆる permissive hypercapnea の概念での管理を考慮することになる．COPD 患者の管理において圧損傷に対して注意を喚起している文献は多い．確かに重症 COPD 患者の気胸は致命的であることも多いため注意が必要であるのは間違いないが，それには換気条件よりもバッキングなど思わぬ高圧が肺胞に掛かるのを防ぐことに主眼を置く必要があるだろう．シミュレーションで示したように吸気圧はむしろ低くできるからである．吸気圧を適切に低く保っていれば通常の換気で肺胞に強い力が掛かる可能性は低いのである．

先に自発呼吸のシミュレーションを (14) 式で示したが，これに具体的な数値を代入してグラフ化すると図 6 のようになる．自発呼吸の場合には E_2 を逆向きに置いているため計算上は Q_1 は負になるが，このグラフでは正負を入れ換えて表示している．

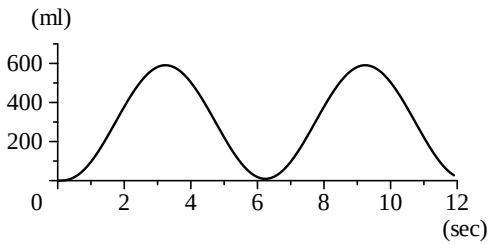


図 6 健常人の自発呼気のシミュレーション

Auto-PEEP に関して

Rossi ら [2] は人工呼吸中の静的コンプライアンス (C_{rs}) の測定において Auto-PEEP が存在する場合 $C_{rs} = V_T/(\mathrm{P_{plateau}} - \mathrm{PEEP_e} - \mathrm{PEEP_i})$ で計算する必要があるとしているが，これは正しくない．$\mathrm{PEEP_e} < \mathrm{PEEP_i}$ なら分母は $(\mathrm{P_{plateau}} - \mathrm{PEEP_i})$ であり，$\mathrm{PEEP_e} >= \mathrm{PEEP_i}$ なら $(\mathrm{P_{plateau}} - \mathrm{PEEP_e})$ とするべきである．

また呼気終末気道閉塞法による dynamic Auto-PEEP 測定方法は本来計測すべき Q_2/C_2 ではなく $(Q_1 + Q_2)/(C_1 + C_2)$ を測定していることが本モデルから示される．現実には $Q_1 << Q_2$ かつ $C_1 << C_2$ を考えると Q_2/C_2 を近似しているとは考えられるが，条件を考慮しておく必要がある．

さらに，本方法では static Auto-PEEP(気道閉塞を伴う場合) が高い症例では Auto-PEEP が全く計測できないことも判る．SW_2 がオープンの状態では SW_1 をオープンしてもこの時点で $Q_2/C_2 > Q_1/C_1$ であり SW_2 が再びショートされる状態は生じないためである．

一般化に関して

これまでは人工呼吸回路を無視していたが人工呼吸をシミュレートする場合には回路図 2 の右側にさらに人工呼吸回路に相当する RC 回路を付加する必要がある．この場合には上記のような解析は難しくなる．もちろん三元の連立線形微分方程式となるから解析的に解くことは可能であるがここでは省略する．また肺の一部に病変がある場合にはこの部分を別に分ける必要がある．これは等価回路上では C_1 の両端にさらに RC 回路が R_1C_1 と並列に並ぶことになる．R, C の異なる肺胞が多数存在する場合には同様に複数の RC 回路が C_1 の両端に付加される．

これらのモデルは一般的には Laplace 変換 (Appendix 2. 参照) を用いると以下の形になることが知られている．

$$Q_i = A_1 e^{\alpha_1 t} + A_2 e^{\alpha_2 t} ... + A_n e^{\alpha_n t} + f(t) \quad (15)$$

但し，α_i はすべて負数である．(実際の厳密解は n 次方程式が解けない限り求められない．) また $f(t)$ は E_1, E_2 で示される呼吸様式の関数に依存した形になる．

さらにここではコンプライアンスや気道抵抗が不変であると仮定して微分方程式を解いたが現実にはこれらもまた Q の関数であり現実はより複雑である．しかし微分方程式自体は常に成立するものであるから種々のシミュレーションを行う元にすることができる．

また現実の人工呼吸では回路のリークというものがしばしば問題となる．リークは等価回路ではリーク箇所とアースを抵抗で結んだ形で表現できる．もちろん一般的にはこの抵抗値もそこの流速の関数であることもあるが，これらは定義しだいでシミュレーションは可能である．

おわりに

このように正しいモデルを構築することは肺の換気力学の理解の助けとなりまた種々の現象を考察する上で欠かせないものである．また実際の種々の換気力学上のパラメータがモデル上で

どのように表現されるかを正しく認識することは理論構築およびそれらの測定方法を検討する上で必須である．
本モデルは肺の換気力学を考察する基本的なモデルであると考えられる．

Appendix 1.
１階線形微分方程式の解について

$$y' + P(x) \cdot y = Q(x)$$

上記のような１階線形微分方程式の解は以下のようになる．

$$y = e^{-\int P(x)dx} \cdot \{\int Q(x) \cdot (e^{\int P(x)dx})dx + C\}$$

また，１つの解 (y_1) が既知である時には

$$y = y_1 + C \cdot e^{-\int P(x)dx}$$

となる．

Appendix 2.
Laplace 変換とは

$$F(s) = \int_0^\infty e^{-st} f(t)dt \qquad (16)$$

(16) 式を $f(t)$ の Laplace 変換と言う．$f(t)$ を原関数，$F(s)$ を像関数と呼ぶ．
Laplace 変換は以下のようにも記述される．

$$F(s) = L(f(t))$$

同様に逆 Laplace 変換は以下のようにも記述される．

$$f(t) = L^{-1}(F(s))$$

Laplace 変換と逆 Laplace 変換は互いに逆変換である．

代表的な関数の Laplace 変換は以下のようになる．

$$\begin{aligned}
f(t) &= 1 \rightarrow F(s) = \frac{1}{s} \quad (s > 0) \\
f(t) &= t \rightarrow F(s) = \frac{1}{s^2} \quad (s > 0) \\
f(t) &= e^{\alpha t} \rightarrow F(s) = \frac{1}{s-\alpha} \quad (s > a) \\
f(t) &= \cos(\alpha t) \rightarrow F(s) = \frac{s}{s^2+\alpha^2} \quad (s > 0) \\
f(t) &= \sin(\alpha t) \rightarrow F(s) = \frac{a}{s^2+\alpha^2} \quad (s > 0)
\end{aligned}$$

Laplace 変換では線形性，相似性が成立する．つまり，

$$\begin{aligned}
L(\alpha f(t) + \beta g(t)) &= \alpha F(s) + \beta G(s) \\
L(f(\alpha t)) &= \frac{1}{\alpha} F(\frac{s}{\alpha}) \quad (\alpha > 0)
\end{aligned}$$

が成立する．
なぜ，Laplace 変換が微分方程式を解くのに有用なのか解説する．
$f(t)$ を微分してから Laplace 変換を行うと

$$\begin{aligned}
L(f'(t)) &= \int_0^\infty e^{-st} f'(t)dt \\
&= [e^{-st} f(t)]_0^\infty + s\int_0^\infty s^{-st} f(t)dt \\
&= [e^{-st} f(t)]_0^\infty + sL(f(t)) \\
&= \lim_{t\to\infty} e^{-st} f(t) - f(0) + sL(f(t)) \qquad (17)
\end{aligned}$$

通常 $L(f(t))$ が収束するなら (17) 式の右辺の第１項も 0 に収束する．従って，

$$L(f'(t)) = -f(0) + sL(f(t))$$

これを Laplace 変換の微分法則と呼ぶ．いわば Laplace 変換は微分を乗除算に置き換えるものであると言える．例えば次の微分方程式を Laplace 変換を用いて解いてみる．

$$y' + 2y = e^{-t}, \quad y(0) = 2 \qquad (18)$$

両辺を Laplace 変換すると

$$
\begin{aligned}
-y(0) + sL(y) + 2L(y) &= \frac{1}{s+1} \\
(s+2)L(y) &= 2 + \frac{1}{s+1} \\
L(y) &= \frac{2}{s+2} + \frac{1}{(s+1)(s+2)} \\
&= \frac{1}{s+1} + \frac{1}{s+2} \\
&= L(e^{-t}) + L(e^{-2t})
\end{aligned}
$$

両辺を逆 Laplace 変換すると

$$
y = e^{-t} + e^{-2t} \tag{19}
$$

となり，微分方程式の解が得られる．(19) 式を (18) 式に代入してみれば解が正しいことが確認できるだろう．このように Laplace 変換を用いれば微分方程式を乗除算として解くことができるのである．

参考文献

1. Marini JJ, Crooke PS 3rd. A general mathematical model for respiratory dynamics relevant to the clinical settings. Am Rev Respir Dis 1993;147(1):14–24.
2. Rossi A, Polese G, Brandi G, Conti G. Intrinsic positive end-expiratory pressure (PEEPi). Intensive Care Med. 1995;21(6)522–36.

特別寄稿：Python GUI 入門 –Tkinter から PIL まで–

田中義文

なぜ Python か？

今日では多くの優秀なプログラムが公開または市販され，誰でもが通常の業務に利用できるようになった．特に Microsoft Office に代表される Word や PowerPoint は世界共通製品となり，USB メモリーチップにデータを書き込めば，どの様な場所に出張しようと，そこに PC が有れば仕事が継続できる．

一方，データベースや表計算，そして CPU 間の LAN 操作などの機能を一体化し，さらに目的に特化したプログラムとなると状況は一変する．こまごまとした仕様を作りあげてソフト業者に発注しなければならない．費用も莫大となり，満足する状況までにはなかなか至らない．その原因は，利用者が，専用プログラムには少なくとも Microsoft Office 程度の汎用機能を持った上で，種々の専用機能を実現できているだろうと勝手に想像するからである．

専用プログラムの不備な部分は，業者に訴えて，改善してもらうより他に方法がないが，時間も費用もかかる難問である．結局，個人プログラマーになって解決するのが得策である．そこで，どのプログラム言語を利用すれば最も効率的にプログラムが開発できるかを考える必要がある．

1. 言語が読みやすいこと，
2. すぐに実行できること，
3. 仕様の変更や拡張が容易であること，
4. GUI (Graphic User Interface) を装備し，利用しやすいこと，
5. データベースとしてハードディスクが取り扱えること，
6. LAN 通信ができること，
7. リモート CPU 操作ができること，
8. Linux, Windows, MAC などのプラットホームで使用できること，

などがあげられよう．これだけ出来れば，インターネットを介して，世界規模でデータベース回線が構築できる．

プログラム開発において，C 言語は最も動作が速いプログラムを作成するが，完成までに何度もコンパイル，デバグを繰り返さねばならない．また，GUI は X window システムを呼び出す必要がある．C 言語はプロのプログラマーでないかぎり，実用言語として利用すべきではない．

今日よく利用される Microsoft Basic は，実はオプションが複雑で使いづらいものの一つである．JAVA や Tcl/Tk は非常に優れた言語であり，Linux, Windows, MAC の各プラットホームで動作する．しかしながら，言語は冗長で type miss の繰り返しといって過言ではない．

筆者が推薦する言語は Python である．文末やブロック文を示す「;」や「 」がなく，書きやすくてタイプミスが少ない．C 言語と Python 言語を比較してみよう．以下のプログラムは”Hello World.”を 5 回繰り返し端末に表示するだけのプログラムである．

京都府立医科大学大学院医学研究科麻酔科学教室

C 言語

```
#include <stdio.h>
main()
{
   int i;
   for(i=0;i<5;i++) {
      printf("Hello World.\n");
   }
}
```

Python 言語

```
#! /usr//bin/env python
for i in range(0,5):
   print "Hello World."
```

[1]

C 言語では 8 行必要であるが，Python では 3 行で実現できる．しかもインタープリータ (対話方式) であるから，コンパイル，リンクの手間が不要である．というわけで Python は上記の全ての条件を満たしている言語だといえる．Python でプログラムの見通しを立て，必要なら C 言語に書き直せばよい．始めから C 言語でのプログラム開発は労が多すぎて成功する可能性は低い．CPU に同じ仕事をさせるのに，プログラムの行数が 1/2〜1/3 になれば，それだけ間違いが少なくなり，開発効率も飛躍的に上昇する．その意味でも Python 言語を用いてオブジェクト指向に慣れることが重要である．オブジェクト指向については後述するが，プログラムのグループ化，継承ができ，短いプログラムピースを積み重ねて全体のシステムを構築する手法である．

Python のインストール

Python は無料のパブリックドメインプログラムであるから，インターネット接続できる機種であれば，いつでも導入できる．また多くの解説がホームページ上に公開されているから，悩むことは少ないと思われる．

Windows と MAC:

Google 検索で「Python インストール」を行い，そのサイトの指示に従う．コンパイル済みの実行環境が入手できるから容易である．GUI モジュールである Tkinter は装備されている．追加して，別の GUI モジュールの wxPython も「wx-Python インストール」を検索してインストールする．wxPython は Tkinter よりも洗練された GUI が作成できる．最も容易にインストールできる．MAC も同様にインストール出来るはずである．

ubuntu:

ubuntu は近年最も活発に活動している Linux デベロッパーで，雑誌の付録の CDROM，ネット上で ubuntu インストーラを導入すれば容易に ubuntu Linux が動作する．Windows のディスクスペースに ubuntu を構築する方法と，ディスクを 2 分割 (パーティティション) して Windows と ubuntu とのデュアルブートを行う方法との 2 通りの選択ができる．どちらも快適に動作するが，当然 Windows 上で動作すれば使用できるディスク容量は少ない．最大 30 G バイトしか占有できないが Python の動作には十分である．ubuntu に慣れるまではこの方法で十分である．Ubuntu では Python 言語はバイナリーファイルで提供され容易に動作する．

Python インストールは端末 (gnome-terminal) を開け，以下の命令を入力する．

```
sudo apt-get install python
sudo apt-get install python-tk
```

また wxPython もバイナリーファイルで動作可能である．

[1] 枠で囲まれた部分はソースプログラム

Python 言語仕様

書式

Python のプログラム書式は C 言語と違って，改行に意味が有る．そして行頭からのインデント (空白) に意味があり，同じインデントは同一のブロック文と解釈される．このルールにより C 言語の「;」と「{ }」記号から開放されることができる.「#」文字より右側はコメント，行末の「\」は継続行である．

for 文，if 文，while 文などでは条件式の後に「:」で改行し，次の行からインデントを行う．関数指定である def 文，クラス指定である class 文も同様に「:」で改行し，次の行からインデントを行ってブロック文の範囲を定める．

定数と変数

定数には整数，実数，複素数，文字列を受け付ける．これらの定数を受け付ける変数は前もっての宣言が不要である．前項で示した C 言語と Python 言語との違いがここにある．定数 i についての宣言をしていない．定数を「=」つまり，等号で変数に代入するときに，定数がどのような種類であるかを Python は自動的に判断して変数領域に書き込む．なお，文字列は「'」，または「"」で囲み，\は\\，改行は\n と記述する．複素数は 3+2j のように記述する．

実は Python では変数そのものがオブジェクトである．オブジェクトとは，変数がもつ定数の値，属性 (性質) を含み，更にその演算方法まで用意されている．従って，"a = 1"とプログラムに記述すると，変数 a のメモリー領域に値，属性，演算法の全てが記録される．従って，何バイトのメモリーを変数 a が消費したか利用者にはわからない．

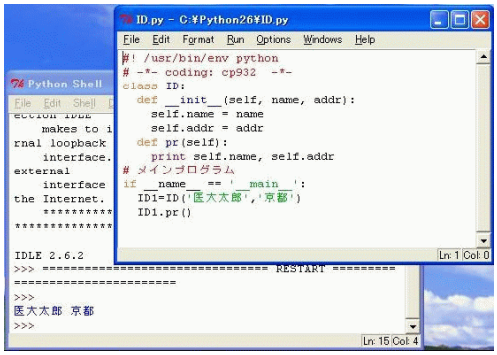


図 1 Windows での Pytho 駆動画面

オブジェクト指向のプログラムであるために，複素数どうしの演算も通常の「+, -, * /」演算子を用いてできる．変数が文字列どうしであれば，「+」記号は文字列の連結を意味する．当然文字列の引き算は意味がないからエラーが返る．このように同じ演算子を用いて，その要求に見合った仕事ができることをオブジェクト指向では「多態性」と呼んでいる．この性質により，プログラマーは処理関数の名称を覚える量が少なくなり，エラーも防止できる．

いろいろな変数

Python では整数，実数，複素数，文字列などの単純変数の他に，リスト，タプル，辞書形式と呼ばれている複合変数を取り扱うことができる．この説明のために対話形式で Python を駆動させよう．

Linux なら端末を開き，python ⏎ と入力する．Windows であれば Python 2.6 IDLE をスタートプログラムで選択する．色々表示され，最後に「>>> 」とプロンプトが表れる．これでプログラムを入力すると対話形式で実行する．簡単なプログラムでも別に編集用ウィンドウを開いてそこにプログラムを書き，コピーペーストで行入力すると，入力作業が楽になる．また編集用ウィンドウで Run Module を選択すると Shell

画面に結果やエラーが表示される．

1) 単純変数

まず単純変数の演算から始めよう．

```
>>> 2*3.14
6.2800000000000002
>>> a='abc'
>>> b='def'
>>> a+b
'abcdef'
>>> (1+1j)*(1-1j)
(2+0j)
>>> from math import *
>>> log(10.0)
2.3025850929940459
```

[2]

前述の単純変数の演算は全て思う通りの結果になっている．「from math import *」は数学演算用モジュールの読み込み命令で通常使用する数学関数は全て用意されている．

2) リスト

リストはプログラミングで最もよく利用される変数領域であるから，十分に慣れてもらいたい．

a) リストの標準的な使用法

リストは順序だった変数の集まりである．変数 (リスト項目) は何でもよい．リストは項目ごとに「,」で区切り，全体を「[ ]」で括る．「+」演算子はリストの連結ができる．

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> c=a+b
>>> c
[1, 2, 3, 4, 5, 6]
>>> a.append(b)
>>> a
[1, 2, 3, [4, 5, 6]]
```

となる．最後の行で，変数 c ⏎を行うとその内容が返る．なお，append メソッドを使うと最後の項目にリスト b が 1 項目として追加される．

リスト項目の選択はリスト変数に [m:n] の添字をつける．m は始まりを指定し，先頭番号は 0 である．末尾は n-1 になる．上記のリスト変数 c をもちいて，

```
>>> print c[1:4]
[2, 3, 4]
>>> print c[0]
1
```

となる．

リストは単純変数に代入できる．

```
>>> a=[1,2,3]
>>> x,y,z=a
>>> print x,y,z
1 2 3
```

となる．但しリスト項目と変数の数が合わないとエラーになる．

リストには追加，削除，リストの個数，検索，リスト整数の作成，など色々のメソッドが備わっている．help() ⏎と入力すると，簡単な説明が表示されるが，成書を参考にした方がよい．

また，リストはリストのネスティングができ，一次元配列，二次元配列であるマトリックスも表示，演算ができる．

リストを重ねて表記する．

```
>>> a=[[1,2,3],[4,5,6],[7,8,9]]
>>> a
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> a[2][1]
8
```

0 行，0 列から始まり，行，列の順に記述すると配列因子が指定できる．ベクトル，マトリックスの演算は関数演算プログラムを作成しなければならない．

b) 関数 range

関数 range は容易にリストが作成できるから for 文などによく利用される．range(5) は [0, 1, 2, 3, 4] が返る．

関数 range の引数は (初期値，最終値，増分) と

[2]影枠で囲まれている部分は Python 対話モードの表示を示す

なっており，初期値を指定しなければ 0，増分を指定しなければ 1 がデフォルト値になる．したがって，range(5,1,-1) は [5, 4, 3, 2] が返る．for 文を利用して，

```
for i in range(1, 5):
```

は，変数 i に 1，2，3，4 を順に代入して作業を行えというループである．また，

for i in range(1, 5, 2):

は，2 おきに順となるから i に 1, 3 が代入される．

c) リストの参照

リスト項目は 0 番から始まり,「[ ]」の添字 (インデックス) で各項目を参照する．

```
>>> a=range(10)
>>> a[3]
3
```

また，添字には「[開始値:終了値:増分]」が指定できる．開始値を省略すると 0，終了値を省略すると最大値，増分値を省略すると 1 がデフォルトになる．

```
>>> a=range(10)
>>> a[0:5:2]
[0, 2, 4]
>>> a[1:6:2]
[1, 3, 5]
```

さらに，負の開始値を指定すると，リスト項目の終端から開始値の方へ参照する．

```
>>> a[-1]
9
>>> a[-1:-6:-2]
[9, 7, 5]
```

d) 関数 zip

関数 zip はリストの組合せをリストで返す．

```
>>> a=range(8)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7]
>>> zip(a[0::2],a[1::2])
[(0, 1), (2, 3), (4, 5), (6, 7)]
>>> zip(a[0::3],a[1::3],a[2::3])
[(0, 1, 2), (3, 4, 5)]
```

e) リストループ (1)

リストは「[ ]」で初期設定するが，関数値や大量のデータの場合は for 文と append で初期化する．

```
x=[];y=[]
for i in range(5):
  x.append(i)
  y.append(i+10)
print x, y
```

は，x=[0, 1, 2, 3, 4], y= [10, 11, 12, 13, 14] となり，それぞれ出力する．

f) 関数 append と insert

append はリストの最後に追加するメソッドであるが，insert はリストの途中に追加するメソッドである．しかし，insert の index を 0 にすると，リストの最初に項目が追加できる．

```
>>> x=[1,2,3]
>>> x.insert(0,'a')
>>> x
['a', 1, 2, 3]
```

となる．

g) 関数 map と lambda

組み込み関数に map 命令がある．呼び出し法は map(function, list) で，リスト項目に対して関数 function を行ってリストを作成する．例えば，

```
>>> List = ['1', '2', '3', '4']
>>> a=map(int, List)
>>> a
[1, 2, 3, 4]
```

と，文字列の数字に対して，int 関数で数字のリストに変換する．しかし，int+1 のように式

を記述することはできない．このような場合は map(lambda x:x+1,a) と lambda 関数を使う．

```
>>> a=[1,2,3,4]
>>> b=map(lambda x:x+1,a)
>>> b
[2, 3, 4, 5]
```

また，map 関数は None を利用してタプルのリストを作成する．

```
>>> x=range(3)
>>> y=range(10,13)
>>> z=range(20,23)
>>> p=map(None,x,y,z)
>>> p
[(0, 10, 20), (1, 11, 21), (2, 12, 22)]
```

となる．リスト p の各項目の始めだけを抽出する場合も q=map(lambda x:x[0],p) と lambda 関数を使うと出来る．

```
>>> q=map(lambda x:x[0],p)
>>> q
[0, 1, 2]
```

となる．

リストを文字列に変換する方法は

```
>>> data="%s" % [1,2,3]
>>> data
'[1, 2, 3]'
```

でで き，文字列に変換されたリストの復元は string モジュールを import して、

```
>>> map(int,split(data[1:-1],','))
[1, 2, 3]
>>> map(float,split(data[1:-1],','))
[1.0, 2.0, 3.0]
```

のようにする．

e) リストループ (2)

for 文をリスト内に入れ込むことができる．

```
>>> L=[1,2,3]
>>> L1=[i*2 for i in L]
>>> L1
[2, 4, 6]
>>> L1=[3.0 for i in range(3)]
>>> L1
[3.0, 3.0, 3.0]
```

となる．このような記述法は始めは戸惑うが，append を使用すれば少なくとも 3 行必要で，さらにインデントが必要になるところを 1 行で解決できるから，知っておくべきである．

3) タプル

タプルはリストに似ているが，項目を変更することができない．リストは「[ ]」で括るが，タプルは「()」で括る．関数の引数並びはタプルと解釈される．

```
>>> (a,b)=(1,2)
>>> a
1
>>> b
2
>>> d=(a,b)
>>> d[0]
1
>>> d[1]
2
```

と，見れば分かる．1 項目だけのタプルは d=(1,) と最後に「,」が必要である．d = 1,2,3 は括弧が省略されたタプルである．したがって，a,b,c = 1,2,3 と記述すれば，一行で 3 項目の代入になる．

タプルをリストへの変換は list 関数，逆にリストからタプルへは touple 関数を使う．

```
>>> a=(1,2,3)
>>> list(a)
[1, 2, 3]
>>> tuple(list(a))
(1, 2, 3)
```

タプルを辞書のキーとして使うことで，多次元の配列も擬似的に辞書を使って実現することが

できる

```
>>> m = {}
>>> m[(1,2)] = 3
>>> m[(5,4)] = 6
>>> p=5
>>> q=4
>>> m[(p,q)]
6
```

4) 辞書

辞書変数は見出しと内容を示す連想配列でハッシュ法を用いている．表記法は

{ 見出し:内容，...}

である．見出しは数値，文字列などが適当で，リストなど複合変数は指定できない．内容は確定した変数であれば何でもよく，リストを入力してもよい．

```
>>> d1={}                          #空辞書
>>> d1={'a':1,'b':2,'c':3}
>>> print d1['a'], d1['c']
1 3
>>> d1.keys()
['a', 'c', 'b']
>>> d1.values()
[1, 3, 2]
>>> d1.items()
[('a', 1), ('c', 3), ('b', 2)]
```

と，見れば分かる．items() メソッドは見出しと内容とでペアーのタプルを形成し，全体の辞書がリストで返る．

辞書やリストで注意しなければならないことが一つある．それは項目が変数名のままで記憶されるのではなく，全て展開されるということである．

リストの場合：

```
>>> # リスト
>>> p1=1;p2=2;p3=3
>>> P=[p1,p2,p3]
>>> p1=2
>>> P
[1, 2, 3]
```

辞書の場合：

```
>>> # 辞書
>>> p1=1;p2=2;p3=3
>>> d1={'a':p1,'b':p2,'c':p3}
>>> d1
{'a': 1, 'c': 3, 'b': 2}
>>> p1=2
>>> d1
{'a': 1, 'c': 3, 'b': 2}
```

つまり，p1 という変数を項目に設定したのち，p1 の値を変更しても先に作成した辞書やリストの項目は変更されない．

辞書オブジェクトには，追加，削除，検索などのメソッドが備わっている．名前と住所とで住所録を作成できる．内容にリストを用いれば，住所，電話番号，職業など，写真ファイル名などいくらでも複雑なデータベースが構築できる．しかし，辞書を書き込み可能な連想配列とみれば，タプルの項で示したように辞書は大変便利なベクトルや配列の記憶領域として利用できる．

せっかく作ったデータベースの保存はというと，次に示す pickle(漬物) と shelve(本棚) というディスク入出力ルーチンがある．

a) pickle

辞書機能を系統立てて複雑にしていくとデータベースに発展する．しかし，このデータベースをハードディスクに記録できなければ実用化ができない．pickle はこのためのユーティリティで，クラスで作成したインスタンスがディスク読み書きできる．

例：家族構成を辞書型とリストで示す．

A 家には A1, A2 で構成され，B 家では B1, B2, B3 で構成される．そのデータを d とすると，

`d='A':['A1','A2'],'B':['B1','B2','B3']`

と表記できる．もちろん操作性のよい GUI 入力プログラムを作り，スケールの大きなリストを完成できるが，今は，この完成したデータを pickle

でディスクに書き込む．

プログラム： pickle 書き込み

```
import pickle
d={'A':['A1','A2'],'B':['B1','B2',
  'B3']}
object = d
file = open('picle.dic', 'w')
pickle.dump(object, file)
file.close()
```

プログラム： pickle 読み取り

```
import pickle
file = open('filename', 'r')
object = pickle.load(file)
d = object
print d
file.close()
```

pickle はファイルをオープンしたままでの追加書き込みはできない．全てを読み取り，一旦ファイルを閉じて，内容を変更し，再度ファイルを開いて，書き込む操作が必要がある．これが shelve との違いである．つまり，pickle は全てのオブジェクトを一度に書き込み，また読み込む．キー検索はできないから，(漬物) と名付けたと思われる．

b) shelve

shelve は pickle と同様に辞書のディスク保存ができるが，キーインデックスによる直接保存だと思えばよい．

例：shelve 書き込み

```
import shelve
dbase = shelve.open('filename')
object = ['A1', 'A2']
dbase['A']=object
dbase.close()
dbase = shelve.open('filename')
object = ['B1', 'B2','B3']
dbase['B']=object
dbase.close()
```

と直接ディスクに書き込む．

例：shelve 読み取り

```
import shelve
dbase = shelve.open('filename', 'r')
object=dbase['A']
print object        # ['A1', 'A2'] が
返る
dbase.close()
```

とキーインデックスを使って，直接ディスクから内容が取り出せる．

ループ制御

a) for 文

for 文は C 言語などの for 文と違い，数値設定はできない．リスト項目の内容が代入される．もし数値ステップが必要であれば while 文を使用する．

```
for i in range(5):
  if i==3:
    break
  print i
```

は各行に数字の 0 から 2 までを出力し，i が 3 になった段階でループから脱出する．break の代わりに continue を使うと，3 だけが出力しない．

b) while 文

while 文は答えが真の間はループを維持する．

```
i=1
while i<10 :
 if i>6:
   break
 print i
 i=i+2
```

は各行に 1, 3, 5 と出力する．従来の for 文は while 文で代用できる．break と continue の動作は for 文と同じである．

以上で関数やクラスをのぞいて，Python プログラミングのための基本知識は終了するが，リストと shelve のサポートのおかげて，簡便に住所録

程度のデータベースが構築できるようになったことを理解してほしい．import できるモジュールは math，pickle，shelve を使用したが，これ以外に多くのモジュールがネット上に公開されている．

関数操作法

関数の概念

少し大きなスケールのプログラムを行って困ることは変数名 (ラベル) である．作成当初はその意味を記憶しているが，数日も経つと忘れるからだ．そこで，一時的に使用する変数名は全体に影響が生じないように関数の概念が生まれた．関数の使用法さえ覚えておけば．関数内の変数名は全体のプログラムに影響をあたえない．すると，一連の長いメインプログラムをステップ 1，ステップ 2，… と区切りのよい所で関数登録し，関数の呼び出しだけでメインプログラムになる．

このようにプログラムを関数で区切ると，例えば，日時や，人名など関数を越えて，引用できる変数定義が欲しくなる．そこで生まれたのがグローバル変数でメインプログラムに登録する変数名である．それに対して関数内で登録する変数を局所変数という．

1) 関数の書式

```
def 関数名 (引数 1, ..., 引数 n):
  文
  文
  return 返り値
```

と記述する．return は返り値がなければ必要ない，引数は無くてもよい．但し，関数の呼び出し側と関数定義と引数が一致しなくてはならない．例を上げると，

```
def pr():
  print 'aaa'
# メインプログラム
pr()
print 'bbb'
```

とプログラムすると，端末に

```
aaa
bbb
```

と表示される．つまり始めに pr() を実行して aaa が表示され，次にメインプログラムの print 'bbb' を出力したわけである．

引数を出力するプログラムは，

```
def pr(x):
  print x
# メインプログラム
pr('abc')
pr(123)
```

で確かめられる．つまり，メインの関数呼び出しで引数 x に対応する部分にデータを記述すると，始めに abc 次に 123 と出力する．文字変数も数値も同じ print 文でよい．これもオブジェクト指向のよい所である．引数を増やす場合は「,」で区切る．この ( ) 構造は前述のタプルである．したがって,「pr(x)」と書いても「pr (x)」と書いてもよい．

関数内の局所変数とグローバル変数について述べる．メインプログラムで定義した変数は関数内で引用できる．

```
def pr():
  print x
# メインプログラム
x=3
pr()
```

は 3 を返す．しかし，

```
def pr(x):
  x=x+1
  print 'sub',x
# メインプログラム
x=3
pr(x)
print 'main',x
```

は，始めに sub 4 を出力し，次いで，main 3 を返す．関数内の変数は局所変数であって，メインプログラムに影響を与えない．しかし，global

x を関数内で宣言すると，関数内の x が加算され，メインプログラムの x を書き直す．
例：global 宣言

```
def pr():
  global x
  x=x+1
  print 'sub',x
# メインプログラム
x=3
pr()
print 'main',x
```

は，始めに sub 4 を出力し，次いで，main 4 を返す．上記の例で def pr(x) と記述すると，局所変数とグローバル変数が同じだとのエラーを知らせてくれる．

2) 様々な引数操作

関数の引数は前述のごとく数と順番を会わせる必要があるが，Tkinter などの GUI ソフトでは引数のデフォルト値を前もって準備したり，自由にオプション指定を行ったりしている．そこで，3 種類の引数設定法を述べる．

a) 可変長引数

可変長引数は引数をいくらでも受け付ける．書式は引数名の前に「*」(アスタリスク) を付ける．例を示す．

```
def fun(*arg):
  for i in arg:
    print i
# メインプログラム
x = 1; y = 2; z = 3
fun(x, y, z)
```

は，数値 1 2 3 を出力する．通常の引数と可変長引数とが混在する場合は通常引数を先に書く．

b) デフォルト引数

例えば GUI で背景色は白がデフォルトであるが，それを赤に指定する場合などに利用できる．

```
def fun(a,bg='white'):
  print a, 'bg =',bg
# メインプログラム
fun(3)
fun(5,bg='red')
```

は，始めに 3 bg = white，次いで，5 bg = red と出力する．可変長引数の場合と同じ様に通常の引数は先に指定する．

c) キーワード引数

キーワード引数はキーワードとなる変数とその値を関数に与える．書式を次に示すようにアスタリスク 2 個を引数の前に書く．

```
def fun(a,**b):
  print a,b
# メインプログラム
fun(3)
fun(5,bg='red')
```

は，始めに 3 {}，次いで，5 {'bg': 'red'} と辞書形式で出力する．可変長引数の場合と同じ様に通常引数は先に指定する．
通常の引数を含めて 4 種の引数を示したが，これらの記述法の順番は，通常引数，デフォルト引数，可変長引数，キーワード引数の順である．引数の戻り値についても複数になればリスト形式，そして辞書形式にすれば任意のキーワードをメインプログラムに返すことができる．

クラス操作法

クラス

クラスとは，データおよび関数の上位に存在するオブジェクトであり，始めに，クラス宣言でクラスの雛型を作成する．次いで代入操作によりインスタンスを作成する．クラスで作成されたオブジェクトは具体的な存在であるからインスタンスと呼ばれる．また，クラスに属するデータを処理する関数をメソッド (処理法) という．

ここまでの話しをプログラムで示すと，

```
class ID:
  def __init__(self):
    pass

# メインプログラム
if __name__ == '__main__':
  ID1=ID()
```

ということになる．つまり，class ID:でクラス宣言を行い，def __init __() 関数でクラスの内容を初期設定する．クラス ID には何の引数もないので，self と汎用のクラス名称だけを付ける．その初期設定は無しであるから，pass と何もしない形式だけの命令を記述しておく．

メインプログラムではクラスインスタンスを代入文で作成し，その名称を ID1 とする．メインプログラムは代入文 ID1=ID() で，クラス ID を評価する．その評価法は__init __() 関数である．引数もなく，pass だけだから，ID1 という変数領域に ID のクラスポインターを記述して終了する．

クラス構造体

次にクラスのデータ構築法について述べる．例えば C 言語で構造体とか，レコードと呼ばれているデータ構造がある．これは，いくつかのデータを集めて一つの構造体データとして扱うことができる．例えば，名前と住所は対の関係になっており，ID1.name とすると，1 番目の名前が呼び出され，ID1.addr とすると，その人の住所がわかる．ID1 とは上記の構造体名となる．この機能を Python のクラスで実行すると，

```
class ID:
  def __init__(self, name, addr):
    self.name = name
    self.addr = addr
# メインプログラム
ID1=ID('医大太郎','京都')
print ID1.name
print ID1.addr
```

となる．まず，class はクラス宣言であり，空白を置いてクラス名を書く．クラス名の次に「()」を記述してもよく，括弧の中に上位のクラス名を記述すると，指定したクラスの全機能が継承される．(上記の例では使っていないが，Tkinter の説明では他用している) 行末には「:」を記述する．次行は__init__関数を定義する．

今回ははクラス構造体を構築する．引数を見ると，self に始まり，name と addr が続く．self は具体的なインスタンス名が引用される仮の名称として使われる．self.name と self.addr は前述の構造体メンバーに対応し，引数の name と addr がそれぞれクラスメンバーとして代入される．

メインプログラムを見ると、始めに

ID1=ID('医大太郎','京都')

と書き，インスタンス ID1 を作成する．これがクラスの代入操作である．引数はクラス構造体を作るだけだから，引用する名前と住所に対して self の接頭語を付けて代入する．__init__関数を実行することにより，この 2 行でインスタンスにデータが記録される．

次の 2 行はインスタンスの確認のための端末出

力である．インスタンスメンバーは「インスタンス名.メンバー名」とピリオッドがオブジェクト書法の分離子となる．メソッドの呼び出しも同様の表記である．

対話形式で実行すると，

```
>>> class ID:
...   def __init__(self, name, addr):
...     self.name = name
...     self.addr = addr
...
>>> ID1=ID('医大太郎','京都')
>>> print ID1.name
医大太郎
>>> print ID1.addr
京都
>>>
```

と結果が返る．

クラスメソッド

クラスメソッドの作成は容易である．クラスメンバーを表示する関数 pr() を作成する．関数のルールに従って記述すればよい．しかし，第 1 引数に self を忘れずに書く必要がある．また，self で始まる変数はクラス内でグローバル変数として引用できるが，__init__関数の中で，引数や式の右辺に書かれている name や addr などは，定義した関数内でしか引用できない局所変数である．従って，pr() 関数内で print name, addr と書いても変数名が引用できないとエラーが返る．

```
class ID:
  def __init__(self, name, addr):
    self.name = name
    self.addr = addr
  def pr(self):
    print self.name, self.addr
# メインプログラム
ID1=ID('医大太郎','京都')
ID1.pr()
```

とすれば，

```
>>> class ID:
...   def __init__(self, name, addr):
...     self.name = name
...     self.addr = addr
...   def pr(self):
...     print self.name, self.addr
...
>>> ID1=ID('医大太郎','京都')
>>> ID1.pr()
医大太郎 京都
>>> ID2=ID1
>>> ID2.pr()
医大太郎 京都
>>> ID3=ID('医大次郎','東京')
>>> ID3.pr()
医大次郎 東京
>>>
```

と，端末に'医大太郎 京都' と表示する．また，ID2=ID1 のようにクラス構造体のコピーは容易である．ID3 の定義のようにいくらでもインスタンスの作成ができる．

プログラムの保存と実行

クラスの継承に話しを移す前に，クラス ID を実行するプログラムを保存して，バッチ操作ができる正しい書法を紹介する．

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
class ID:
  def __init__(self, name, addr):
    self.name = name
    self.addr = addr
  def pr(self):
    print self.name, self.addr
# メインプログラム
if __name__ == '__main__':
  ID1=ID('医大太郎','京都')
  ID1.pr()
```

第 1 行は Python インタプリータを働かすための呼び出し行である．第 2 行はこのファイルコーディングが euc-jp を使用していることの宣言である．Windows の文字コードは Shift-Jis であるから，この行は以下のように変更する．

```
# -*- coding:  cp932 -*-
```

続いて，class 定義を行う．

#の行はコメント行と解釈されて，何を書いてもよい．メインプログラムと記述しているコメント行の次の if 文は，もし，これがメインプログラムであれば，以下の文を実行せよ，という意味である．そこで，先ほど実行した命令をインデントをつけて記述する．

上記のデータに対して，クラス名と同じファイル名に拡張子「.py」を付けることが習慣的に決まっている．すなわち「ID.py」となる．Linux であればプログラムファイルに実行命令をつける．

```
chmod +x ID.py
```

として，端末から ID.py と入力すれば実行する..

Windows であれば，Python26 のディレクトリに保存し，Run Module を行えば実行する．

クラスの継承

クラス ID に職業項目（job）を追加するクラス IDJ を作成する．既に，名前，住所のクラス ID を作成しているから，そのリソースを利用して新しいクラス IDJ を作成すると開発効率がよくなる．この方法を継承という．なお，継承元となるクラス ID はスーパークラスと呼ばれている．

クラス IDJ を正しく書くと，

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
from ID import *
class IDJ(ID):
  def __init__(self, job, *ext):
    self.job=job
    ID.__init__(self, *ext)
  def pr(self):
    print self.job, self.name,
      self.addr
# メインプログラム
if __name__ == '__main__':
  ID1=IDJ('MD','医大太郎','京都')
  ID1.pr()
```

とすれば，端末に'MD 医大太郎 京都' と表示され，ほんの少しの変更だけで，職業欄が追加される．また，同じ pr() 関数で，項目表示ができている．つまり，継承元の関数がオーバーライトされて，新しくなる．このようなメソッドの変更は以前に紹介した多態性を利用している．

新しく作成した IDJ クラスを見ると，始めに，

```
from ID import *
```

と記述している．これはモジュール ID から全てのファイルを読めという意味である．次に，

```
class IDJ(ID):
```

となって，クラス ID を継承して新しいクラス IDJ を宣言している．次の初期設定関数__init__の引数では，job，name,addr としてもよいが，せっかく継承を利用しているから，job だけを宣言し，残りの引数を*ext として可変長引数を利用した．そのことにより name，addr は*ext にタプルとして挿入される．

job をクラス変数（self.job）に代入し，次の行は

```
ID.__init__(self, *ext)
```

となっている．この意味は，継承元の ID クラスに引数を渡して，初期設定せよという意味である．この操作でスーパークラスである ID が初期化され，self.name，self.addr がクラス内で有効になる．ext のアスタリスクを外すとエラーになる．初期設定はここまでで，pr() 関数は ID クラスを利用せずに新しく print 文を作成した．

クラス ID と IDJ の例を見て，たった 1 項目の追加のためにこのように大がかりの変更をしなくても良いのでは，と思われるかもしれないが，実用にしているプログラムは少なくとも数百行になる．この時，1 項目を増やすために全体のプ

ログラムを再度点検して設計変更するか，それとも継承ができるオブジェクト指向で設計当初よりプログラミングしておくかを考えれば答えは自ずからきまるであろう．また，多くのプログラマーと協力して一つの巨大システムを作り上げる場合，この方法なくして，プログラム開発はあり得ない．次章で紹介する Tkinter GUI では文書エディター，図形エディターにメニューボタンを継承して利用している．短いプログラムステップの割には豊富なバリエションが実現できるから，この際クラスと継承の概念は是非習得してもらいたい．

クラス継承の落し穴

クラス ID を作成したが，そのメソッドを追加したい．しかし，本来のクラスの引数はそのままにしておきたいという要望がある．もちろんオリジナルのクラスプログラムに新しいメソッドを追加すればよいのであるが，既に種々のプログラムでオリジナルクラスは利用されており，いまさらの変更は許されないという場合である．

この場合，あきらめて新しいクラス名 IDN を作成する．rev.2 と思えばよい．しかし，継承が機能するから，改めて def __init__関数を記述する必要ない．例を示すと，

```
from ID import *
class IDN(ID):
  def pr2(self):
    print self.name, self.addr
    print self.name, self.addr
# メインプログラム
if __name__ == '__main__':
  ID1=IDN('医大太郎','京都')
  ID1.pr2()
```

となる．クラス ID はそのままで，新しいメソッド pr2() を作成した．単に pr() を 2 度実行するだけである．メインプログラムでは新しいクラス名 IDN を利用してインスタンス ID1 を作成しているが，IDN には__init__関数が存在しない．この場合，継承元の ID まで戻って初期化を行う．したがって，クラス名が変更されても，オリジナルで定義されたクラスの機能が全て継承される．

演算子としてのクラスの利用

今までメソッドの第 1 引数に self を用いてインスタンスの作成と，メソッドの利用に当ててきたが，第 1 引数は同じクラスのインスタンスであれば self と異なる変数を指定してもよい．クラス ID の pr() 文のあとに prname(x,y) を追加して、以下のようにプログラムする．

```
class ID:
  def __init__(self, name, addr):
    self.name = name
    self.addr = addr
  def pr(self):
    print self.name,self.addr
  def prname(x,y):
    print x.name
    print y.name
```

プログラムを走らせると，

```
>>> a=ID('医大太郎','京都')
>>> b=ID('医大次郎','大阪')
>>> ID.prname(a,b)
医大太郎
医大次郎
>>> a.prname(b)
医大太郎
医大次郎
```

となる．つまり，a.xxx(b) という形式の演算処理が出来ることがわかる．これはリストに新たな項目を追加する append() メソッドと同じ形式である．この手法は色々な場面で応用できる．繰り返しになるが，def __init__() 関数でクラスの性質を規定し，メソッドでそのクラス演算処理を定めることができる．この応用として，x-y 平面の 2 点間の距離を求めるクラスを作成する．

```
import math
class Point:
  def __init__(self, x1, y1):
    self.x = x1
    self.y = y1
  # 2 点間の距離を求める
  def distance(p1, p2):
    dx = p1.x - p2.x
    dy = p1.y - p2.y
    return math.sqrt(
      dx * dx + dy * dy)
```

実行させると，

```
>>> a=Point(0,0)
>>> b=Point(1,1)
>>> a.distance(b)
1.4142135623730951
```

と確かに 2 点間の距離が求められている．三角形の面積，立体図形の距離，など種々の応用が可能である．

よく利用する関数

これまで Python の骨格となるプログラム書法を述べたが，実際は sys，os，string，math などのモジュールを取り込んで実用的なプログラムに仕上げることになる．本項ではよく利用する関数を取り上げて使い方を述べる．

★ 標準入出力

標準入力はキーボード（sys.stdin），標準出力は DISPLAY(sys.stdout)，標準エラーは DISPLAY(sys.stderr) が sys モジュールで対応する．

a）組み込み関数を利用した標準入力

```
s=raw_input('% ')
```

はプロンプト「%」を出力してキー入力でき，改行で終了する．

b）標準入力をリダイレクト

コマンドラインに「命令 < ファイル名」と記述してファイルデータを標準入力にする場合がよくある．python では，

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import sys
s=sys.stdin.readline()
while len(s)!=0 :
  print s,
  s=sys.stdin.readline()
```

とすればよい．

c）標準出力をファイルに記録

```
import sys
tmp_stdout=sys.stdout
sys.stdout=open('output','w')
print 'abc'
...
sys.stdout.close()
sys.stdout=tmp_stdout
print 'abc'
```

はディスプレー出力をファイル名 output に記録する．元の標準出力に戻るため，あらかじめ，sys.stdout を tmp_stdout に記録しておくとよい．同様の方法で標準エラーもファイル出力できる．

★ 便利なリスト

入力データの個数が不定の場合，リストにデータを記憶させる．

```
adr=[]
file=open('add', 'r')
while 1:
  line=file.readline()
  if not line: break
  adr.append(line[:-1])
print len(adr)
for i in range(len(adr)):
  print adr[i]
```

1 行目はリスト adr の初期設定，ファイル「add」を開き，append() で改行記号を除いた読み取りデータを記憶さす．リストの個数は len() でわかり，range() でリスト内容を出力する．len() 以外に max()，min() などの組み込み関数が使え， append，count，extend，index，insert，pop，remove，reverse，sort などのリスト本来のメソッドが使用できる．

★ 数字を文字列にする

```
>>> str(123)
'123'
```

★ ファイルの存在確認

```
import os
if os.path.exists('status') :
  print 'yes'
```

★ ファイルを読む

a）1 行毎に読む方法

```
file=open('ID.py', 'r')
while 1:
  line=file.readline()
  if not line: break
  print line,
```

line の後に「,」を付けると print 文の改行が抑止される．

b）ファイルをまとめて一気に読む．

```
file=open('ID.py', 'r')
for line in file.readlines():
   print line,
```

★ ファイルに書く

a）ファイル書き込み:

```
fn=open(fname,"w")
fn.write(data)
fn.close()
```

b）日本語 euc コード書き込み：

```
fname='tmp.txt'
fn=open(fname,'w')
for i in range(3):
  data=u'医大太郎\n'
  data1=data.encode('euc_jp')
  fn.write(data1)
fn.close()
```

こんな書き方もできる．

```
fname='tmp.txt'
fn=open(fname,'w')
for i in range(3):
  fn.write(
    u'医大太郎\n'.encode('euc_jp'))
fn.close()
```

★ 関数の代入と間接呼び出し

```
def echo(var):
  print var
x=echo
x('医大太郎')
```

★ コマンドラインの取り扱い

命令の引数はよく利用する．

```
#!/usr/bin/env python
import sys
for item in sys.argv:
   print item
```

★ 文字列の数値計算 (電卓)

```
z=eval("2+3+4")
print z
9
```

```
from math import *
z=eval('sin(1.0)')
print z
0.841470984808
```

★ 文字列やファイルスクリプトの実行

a）メモリーにスクリプトを作成

```
cmd='''
for i in range(3):
  print 'あいうえお'
'''
exec cmd
```

とすると,「あいうえお」が 3 回端末表示する．

b）ファイルにスクリプトを作成

```
vi a.txt
for i in range(3):
  print 'あいうえお'
```

とソースファイルを作成する．python を対話モードで駆動し

```
execfile('a.txt')
```

とすると,「あいうえお」が 3 回端末表示する．

c）バッチファイルを対話形式で運用

実は最もデバグに効化的な方法を紹介する．

```
vi Ved3.py
最後の行のイベント待ち受け行
mainloop() をコメント行にする．
#c.mainloop()
```

[3]

[3] イベントは Tkinter などの GUI プログラムで使用されている．

```
execfile('Ved3.py')
```

とすると，対話形式で実行でき，デバグがしやすくなる．

応用その 1

Python 言語の説明だけでは面白くもないので，これまでの説明で応用できる問題を提示する．

UNIX では従来より編集作業に ed や sed が広く使われてきた．ed は 1 行単位の編集機能で，ロールペーパに出力するテレタイプで利用してきた経緯がある．それが時代の進歩で CRT 端末に代ると，ed はディスプレー画面で編集できる vi に主役の座が代わった．一方の sed は stream editor の略であるが，文字や行を行単位で読み，必要があれば編集して出力するシステムで，単なるフィルターだと理解できる．その性質から，sed は要求されるメモリー容量が少なくて済み，また高速度で編集できることから，大規模のプログラムのバージョンアップなどに今日でも使われている．しかし複数行を訂正するブロック単位の編集命令は大変難しくて簡単には編集命令が作れない．

以下に示す lsed.py は行単位でソースを読み，複数行のパターンが一致すればそのブロックを別のパターンに置き換えるブログラムであり，筆者が Tex 原稿のマクロ命令を変更するときに用いているものである．

File name: lsed.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import sys
LS=[]
file=open(sys.argv[1], 'r')
while 1:
  line=file.readline()
  if not line: break
  LS.append(line[0:-1])
LE=[]
file=open(sys.argv[2], 'r')
while 1:
  line=file.readline()
  if not line: break
  LE.append(line[0:-1])
nLS=len(LS)
nq=0
file=open(sys.argv[3], 'r')
while 1:
  line=file.readline()
  if not line: break
  if line[0:-1]==LS[nq]:
    nq=nq+1
    if nq==nLS:
      for i in range(len(LE)):
        print LE[i]
      nq=0
  else:
    if nq!=0:
      for ix in range(nq):
        print LS[ix]
      nq=0
    print line,
```

命令：
lsed.py a b python.tex
a，b はそれぞれ旧マクロと新マクロ．

プログラム解説

プログラムはコマンドラインを用いて，まず旧マクロファイルを読み，それをリスト LS とする．次いで新マクロファイルを読み，そのデータをリスト LE として記憶する．旧マクロのリスト長を nLS にする．次いで解析するデータを読み LS と比較するが，データを 1 行づつ読み取り，もしデータが無ければプログラムを終了，有ればデータを line に記録する．読み取りデータとパターンが一致すれば nq に 1 を加算し，全てのパターン行が一致すればリスト LE を出力する．そうでなければ一致した範囲のパターンを書き出し，最後に一致しなかった入力データを書き出し，再びデータ読み取りを行う．

line 文字列には [0:-1] と添字が指定されているが，ファイルからの読み取りデータの行末に改行記号が含まれているため，それを除いてい

る．およそ 8000 行の Tex 原稿を 0.5 秒以内に書き換えてくれるから Python の威力は絶大である．

応用その 2

消えた写真メモリーの復活法

ディジタルカメラの重要な写真を操作ミスで消去した経験は一度や二度は必ずある．Python はバイナリーデータの読み取りや 2 進化 16 進の表示は組み込み関数に含まれており，容易にディスクダンププログラムを作成することができる．そのルーチンを利用して消去された写真の復活プログラムを作成する．

通常の写真メディアは MS-DOS 形式が採用されており，写真消去といわれている操作はディレクトリー領域にデータ領域の空きマークを書き込むだけであり，真のデータは重ね書きをされない限り生きていると思われる．従って，ディレクトリ領域とは関係なくデータ領域から jpg データを拾い読みできれば消去された写真は復活できるはずである．

手続きとしては 1）写真メディアをバイナリーのままディスクにコピーする，2）そのデータを目で読める 2 進化 16 進表記ファイルに変換する，3）jpeg プリアンブル毎に別の画像ファイルだと思って，その部分を独立したファイルとしてコピーする，4）ディレクトリー領域は間違った内容が記入されているから復活させる必要はない，

1）バイナリーデータのコピー法

通常のファイルはメディアのディレクトリ領域にデータの記入場所が記載され，その情報を元にデータ交換が行われる．ところが，削除したデータに対してすでに記載場所はないわけだから，メディアの先頭から末尾まで全てバイナリーデータとしてコピーして CPU から一つのファイルとして認識させる必要がある．UNIX では dd 命令で容易にできる．

```
/etc/fstab に
/dev/hdc1 /flush msdos defaults 1 1
と記述する．
# mount /flush
# ls -x /flush
```

とすればファイル内容がリスト表示されるが，すでに消去されているため表示できない．

そこで、バイナリーデータを何も加工せずにコピーできる dd 命令をもちいて、

```
# dd if=/flush of=flush.dd
```

を実行する．ダンププログラムは，

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
# 日本語 EUC を通すため、
# 第一パラメータのバイナリーファイルを
# 16 進ダンプする．
# 但しダンプアドレスを
# 0x4000000 で打ち切る．
import sys
b=open(sys.argv[1],"r")
add=0
c=b.read(16)
while c:
  n=len(c)
  data=''
  cata=''
  for i in range(0,n):
    data=data+"%02x" % ord(c[i])
    if (c[i] >=' ') and (c[i] <= '~'):
      cata=cata+c[i]
    else:
      cata=cata+'.'
  data2=data[0:16]+' '+data[16:]
  print "%08x " % add,
  print data2,cata
  add=add+n
  if add>=0x4000000:
    break
  c=b.read(16)
b.close()
```

解析の結果，jpg ファイルは 512 バイト単位で連続的に記録されていること，また 0xffd8ffe1 の 2 バイトがプリアンブルになっていることが明らかになった．そこで，先頭から 512 バイト毎に 0xffd8ffe1 と記録されているファイルを抽出するプログラム jpgrec.py を作成した．

File name: jpgrec.py

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
# メモリースティック削除 jpeg データを拾い
# 読みする．
# 2003 年 11 月 14 日（金）14:28:34 JST
import sys
mark=chr(0xff)+chr(0xd8) \
+chr(0xff)+chr(0xe1)
b=open(sys.argv[1],"r")
add=0
n=1
f=0
c=b.read(512)
while c:
  d=c[0:4]
  if d==mark:
    if f==1:
      jout.close()
    fname=("%04d" % n)+".jpg"
    print "%010x %s" % (add,fname)
    n=n+1
    jout=open(fname,"wb")
    f=1
  if f==1:
    jout.write(c)
  add=add+512
  c=b.read(512)
b.close()
```

たったこれだけのプログラムで貴重な画像ファイルが復活できれば有り難いことである．ただし，削除したメディアで再び写真撮影すると以前の画像が復活できないのは当然である．

応用その 3

大量のメール発送

同一文書を複数のメンバーに発送することは cc を付ければ容易にできる．また alias や，メーリングリストを立ち上げてもよい．しかし，学会誌の校正原稿のように発送先の添付文書が相手によって異なる場合は，個々にメーラーのマウス操作をしなければならない．手作業だから，時には他人の原稿を発送する間違いも生じる可能性がある．このような場合に宛先と原稿名が対応しているスクリプトを作成し，メーラーがそれを読み込み，自動発送すれば安心である．Python にはメール操作のモジュールが提供されており，それを利用すれば解決できる．

1）単純なメールプログラム例

Python ライブラリリファレンスの使用例より，テキストファイルを読み込んでそれをメールの文面にする例を以下に示す．メール内容はファイル名「textfile」とし，中に「123」とだけ記述している．

File name: e1.py

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
import smtplib
from email.MIMEText import MIMEText
# 送信原稿読み取り
textfile='textfile'
fp=open(textfile,'rb')
msg=MIMEText(fp.read())
fp.close()
# 表題
msg['Subject']='The contents of \
  %s' % textfile
# 発信元と宛先
me='AAA@xxx.ac.jp'
msg['From']=me
you='BBB@xxx.ac.jp'
msg['To']=you
# 送信
s=smtplib.SMTP()
s.connect()
s.sendmail(me,[you],msg.as_string())
s.close()
```

[4]

メールが送信できることを確認した上で，メー

[4] http://www.python.jp/doc/2.4/lib/node571.html

ル本体は以下の内容を送信していた．

File name: e1.mail

```
>>> print msg
From nobody Tue Jan 12 16:34:07 2010
Content-Type: text/plain; \
charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: The contents of textfile
From: AAA@xxx.ac.jp
To: BBB@xxx.ac.jp

123
```

対話モードで作動させ，print msg にすると送信内容が表示する．重要な項目は，本当は 1 行で書かれているが，

```
Content-Type: text/plain; \
charset="us-ascii"
```

と記述されていることで，添付書類がないことを示している．
また，

```
charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
```

と記述されていることで，7 bit の ascii コードである．次いで，

```
Subject: The contents of textfile
From: AAA@xxx.ac.jp
To: BBB@xxx.ac.jp
```

となっていることから，メールヘッダーと本文との間に 1 行の空行が必要である．この空行はは単純テキストメールでは本文の最後にも必要である．

2）添付書類付き複合メール

添付書類付き複合メールは MimeWriter モジュールを利用してメールを作成する．このモジュールについて説明は省略し，動作するプログラムを以下に示す．

File name: e2.py

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
import sys
import mimetools as mto
import mimetypes as mty
import MimeWriter
import base64,pykf
import smtplib
def send(me,you,title,*ext):
  out=open('output.txt','w')
  writer=MimeWriter.MimeWriter(out)
  writer.addheader('From', me)
  writer.addheader('To', you)
  s=title
  s=pykf.tojis(s)
  s=base64.encodestring(s)
  s='=?ISO-2022-JP?B?'+s[:-1]+'?='
  writer.addheader('Subject', s)
  writer.addheader('MIME-Version',
    '1.0')
  writer.startmultipartbody('mixed')
  writer.flushheaders()
  # 添付ファイルを読み込む.
  for file in ext:
    s=writer.nextpart()
    out.write('MIME-Version: 1.0\n')
    if file==ext[0]:
      out.write('yes')
      out.write(
        'Content-Type: text/plain;\n')
      out.write(
        ' charset=ISO-2022-JP\n')
      out.write(
        ' Content-Transfer-Encoding:\
          7bit\n')
      infile=open(file,'rb')
      s=infile.read()
      s=pykf.tojis(s)
      out.write(s)
      infile.close()
    else:
      type,encoding \
        =mty.guess_type(file)
      if encoding:
        s.addheader(
```

```
            "content-encoding",encoding)
        else:
          s.addheader(
            'Content-transfer-encoding',
              'base64')
        if type:
          pout=s.startbody(type,[('name',
            file)])
        else:
          pout=s.startbody('text/plain',
            [('name',file)])
        infile=open(file,'rb')
        mto.encode(infile,pout,'base64')
        infile.close()
    writer.lastpart()
    out.close()
  # メール送信
    textfile='output.txt'
    fp = open(textfile, 'rb')
    msg = fp.read()
    fp.close()
    s = smtplib.SMTP()
    s.connect()
    s.sendmail(me, you, msg)
    s.close()
# メインプログラム
title='テスト'
send('AAA@xxx.ac.jp','BBB@xxx.ac.jp',
  title,'a1.txt','a2.docx','a3.gif')
```

プログラム解説

モジュール pykf は ShiftJIS，EUC-JP，JIS コードを相互に変換するためのモジュールで，Atsuo Ishimoto<ishimoto@gembook.org>の作品である．メインプログラムを見ると，引数は差出人，宛先，表題 (title)，そしてメール本体である平文 (a1.txt)，添付書類 (a2.docx，a3.fig) と続く，関数 send の引数は，me，you，可変長引数 ext になっており，添付書類の数の制限はない．

MimeWriter はこれらの添付ファイルをうけて，自動的に MIME 文を作成し，必要なら，JIS 変換，base64 エンコーディングを行って，出力ファイル output.txt を作成する．メール送信は output.txt を読み込み，sendmial を行う．output.txt は送信内容の記録になるから，必要なら，別名で保存するとよい．

また，メインプログラムを見ると，単純な引数の並びであるから，上記の send 関数をコピーしさえすれば，CSV 送信依頼スクリプトを作り，それを読み込んで自動メール送信をするプログラムは容易に作成できると思う．

オブジェクト指向の考え方

従来のプログラム書法は動詞である命令と目的語が分離されており,「何をどうするか」という考えが中心であったが，オブジェクト指向のプログラミングでは動詞部分がメソッドに置き換えられて,「何がどうなるか」という形式に変更されたと思えばよい．分離子「.」を介して,「何が，ああなり，こうなる」と記述する．目的語を必要とする他動詞中心の英語的発想ではなく，自動詞中心の日本語的発想，もしくは文章の最後に動詞が列ぶ日本語文法だといえる．

また，オブジェクト指向とは整数の「+」である足し算記号が，リストに対して項目の追加演算子になり，また文字列の連結子になる．変数がそのような演算情報を組み込んだ存在だと思えばよい．

なお，われわれプログラムの利用者は，言語学者ではないのだから，オブジェクト指向を大上段に構えて，始めから種々のメソッドを構築する必要はない．その場面に出会った時，問題解決のためのプログラムを作成し，継承を利用してオブジェクトの幅を広げていくだけで十分である．

GUI Tkinter の使い方

なぜ Tkinter か？

Tkinter は Tc/Tk の GUI 部分（Tk）を Python で動作するモジュールである．Tcl/Tk のプログラミングではラベル ，ボタン，スライダーなど全ての GUI ウィジェットはグローバル変数で登録しなければ画像表示ができない欠点がある．そのため少し複雑な GUI 画面を作成すると，ウィジェット変数名の混乱に陥る．その欠点を克服する手段がクラスの概念で，オブジェクト指向でプログラミングができる．このインターフェースとなるモジュールが Tkinter である．また，Tcl/Tk 独特の癖である$記号を使わず，set 命令でなく，= 記号で代入操作ができるため，Tcl 言語に比べてプログラムが読みやすくなる．

実は，まだ Tkinter についての詳細なテキストがないが，Tcl/Tk の on line マニュアルと Tkinter の変換方式に慣れれば，書法は検討がつく．インターネットでも多くの資料が公開されている．また，今のところ，Linux 系の日本語処理はコピー，ペーストが不安定であるが，Windows では CTRL+C，CTRL+V の組合せで快適にコピー，ペーストができる．労力を惜しまなければ自作のエディターも作成できる．もし，Tcl/Tk のプリグラミング経験が無ければ単純な画像 GUI が作成できる程度までは習得してほしい．

Tkinter の書法

ラベルとボタンを表示し，ボタンクリックでプログラムが終了するだけのプログラムを Tcl/Tk と Python とで比較する．まず背景色を赤色にする Frame をつけ，その輪郭を 10 ポイントと指定する．その中にプログラム終了ボタンを配置する．また，終了ボタンをクリックすると端末に終了と表示してプログラムが終了する．単に

図 2 ボタン
10 ポイントの赤枠の中に終了ボタンを表示している．ボタンクリックで端末に終了と表示し，プログラムは終了する．

ボタンだけなら，ここまで冗長にする必要はないが，クラス設計に Frame コンテナーは重要な役割りを担うため，あえて，このような例を選んだ．仕上りの GUI を図 2 に示し，プログラムを以下に示す．

Tcl/Tk プログラム：

```
#!/usr/bin/env wish
frame .f -bg "red" -bd 10
button .b -text "終
了" -command goout
pack .b -in .f
pack .f -fill both -expand 1
proc goout {} {
  puts "終了"
  exit
}
```

以下は同じ内容を Python で記述したプログラムである．Tcl/Tk をできるだけ忠実に python に変更したから，ウィジェット名はグローバルのままメインプログラムに残る．

Python Tkinter プログラム：

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
def goout():
  print u'終了'
  exit()
root=tk.Tk()
f=tk.Frame(root,bg='red',bd=10)
b=tk.Button(f, text=u'終了',
  command=goout)
```

```
b.pack()
f.pack(fill='both',expand=1)
root.mainloop()
```

となる．

プログラム解説：

最初の 2 行は従来通りで，Windows であれば

```
# -*- coding:  cp932 -*-
```

とする．次の import 文は Tkinter のモジュールの読み込み命令であり，接頭語 tk を付けて，ネーム空間の衝突を防ぐ．root は基本ウィジェットで Tcl/Tk の「.」に相当するものである．Tcl/Tk でボタン作成は button .b としているが，Python では b=tk.Button(root, ...) と表示する．root は親ウィジェットの名前で，どこに表示するかを指定している．上記のプログラムでは tk.Tk() が指定されているから，ウィンドウマネージャ(wm_) のコントロールが可能であるが，None と指定するとデフォルトトップレベルとなり，そこからのウィジェットではウィンドウマネージャのコントロールができない．

Tcl/Tk でのオプションは「-」パラメータ 値となっているが，Python では「=」等号が使用できる．また，テキストの u はユニコードを指定するものであって，Windows でも必要である．pack 命令はウィジェットのメソッドとして登録されている．ボタンクリック処理関数 goout() はその呼び出しより前に書かねばならない．Tcl/Tk ではプログラム終了時に自動的にイベントループが作動するが，Python では明示的に mainloop メソッドを駆動しなければならない．Windows でこの行が無ければ，プログラムは瞬時に終了して GUI 画像は表示さない．

クラスで GUI を作成する．

クラスの概念で上記のプログラムを書き直すと，メインプログラムでのウィジェット変数はたった一つになる．そのプログラムを以下に示す．ボタン処理も継承できるようにファイル名を B.py とクラス名と同じにした．

File name: B.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
class B(tk.Frame):
  # クラス初期設定
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master,bg='red',bd=10)
    f.pack()
    b=tk.Button(f,text=u'終了',
      command=self.goout)
    b.pack()
  def goout(self):
    print u'終了'
    exit()
# メインプログラム
if __name__ == '__main__':
  g=B()
  g.mainloop()
```

確かにメインプログラムから見えるウィジェット変数は g 一つである．Frame の f，Button の b，さらに，ボタン処理関数 goout も class の中の局所変数になっている．

プログラム解説

初めての GIU クラスの説明だから，詳細に述べる．import 文まで含めて始めの 3 行は前例と同じである．次に class 定義を行う．クラス B は tk.Frame を利用するから括弧内に記述する．tk.Frame は tk.Button を包括するからこれ以上書く必要はない．クラスの初期設定は__init__関数に記述する．その中身は，まずウィジェットコンテナーである tk.Frame の初期設定，つまり，tk.Frame.__init__と Frame 初期設定を行う．続いて，tk.Frame を設定するが，親ウィジェットに

master を選ぶ．master はその上の行の__init__関数で指定され，None が指定されている．もし，root と指定すると，それを引数に反映するために相当複雑なプログラムになる．

さて，GUI 本体であるが，前述のプログラムと殆んど変わらない．ただ，command=self.goout のみが変更されている．self が接頭語になっている変数はクラス内グローバル変数だと思ってよい．この場合は goout メソッドを示している．goout 関数は前述のプログラムと同じだが，class 内に閉じ込められている．f，b のウィジェット変数は，外部から引用される場合には self を付けるが，局所変数にとどめるだけならその必要はない．

クラス文の書き方をまとめると，始めに Frame を初期設定する．次いで，GUI 本体ウィジェットと配置を記述する．必要ならメソッド関数を記述する．これだけである．

メインプログラムをみると，g=B() となって，インスタンス g を作成している．g はクラス呼び出しにより，self の部分が b に名前が変更される．GUI の表示はクラス初期設定の pack でなされるから，mainloop を走らせて待機状態にすればよい．プログラム終了はボタンクリックで関数 goout が動作する．

プログラムの実行順は始めに import 文を読み，次いで class 文を発見し，サブルーチンアドレスとして記憶する．その後，if 文でメインプログラムを知る．そこで，g=B() を見つけ，クラス B の処理を実行する．クラス内では__init__関数の処理を行う．つまり Frame の初期設定，Frame の表示，Button の Frame への張り付けを行い，Button command の登録をして，class B の処理を終えて，メインプログラムに戻る．メインプログラムではクラス B のインスタンスとしてそのポインターを g に記録する．そしてクラス B の既存メソッドの一つである mainloop を回してイベントの発生を待つ．

以上が B.py の実行順であるが，プログラムの途中に print 文で，メッセージを表示する命令を挿入すると，プログラムの流れがよくわかる．繰り返しになるが，class 文はコンパイルされて実行命令がメモリー配置される宣言文ではなく，実行されるサブルーチンであることに注意をしてもらいたい．

B.py の継承

ボタンクリックで，GUI 本体はそのままにして，メソッドだけを変更したり，新たなメソッドを追加したい場合がよくある．そこで B.py では「終了」とだけ端末に表示したが,「extend 終了」とメソッドの多態性を利用する継承プログラムを作成する．

File name: BE.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from B import *
class BE(B):
  # メソッド再設定
  def goout(self):
    print u'extend 終了'
    exit()
# メインプログラム
if __name__ == '__main__':
  g=BE()
  g.mainloop()
```

プログラム解説

import 文は継承元の拡張子「.py」を除いたファイル名を書く．Tkinter は継承元に記述されているため，不要．新しいクラス名 BE を明記し，括弧のなかに継承するクラス名 B を記述する．これで B の全てが継承される．

メソッドは新しく変更するから，それを示すため

に，同じメソッド関数名 goout を書き直す．本例は，簡単に「extend 終了」として，継承元と異なる表示にしただけである．

メインプログラムでは新たなクラス名を引用する．ただ，これだけで以前のプログラムを変更して，新たなプログラムに仕上るから，継承とは便利なツールである．

GUI 継承の落し穴

前に示した B.py の GUI を継承して，新たなクラスを作成してみよう．単純に赤色のフレームの回りに緑のフレームを作成するだけである．簡単に以下のプログラムが考えられる．ファイル名を BE2.py として，

File name: BE2.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from B import *
class BE(B):
  # クラス初期設定
  def __init__(self, master=None):
    B.__init__(self, master)
    f=tk.Frame(master,bg='green',
      bd=10)
    f.pack()
    l=B(f)
    l.pack()
  # メソッド初期設定
  def goout(self):
    print u'extend 終了'
    exit()
# メインプログラム
if  __name__ == '__main__':
  g=BE()
  g.mainloop()
```

仕上りを図 2a に示す．何と，GUI が 2 個立てに出現する．しかも，上の図は継承元の B であって，終了ボタンをクリックすると「extend 終了」と多態性が働いたメソッドになり，下の緑の枠で囲まれた新しい GUI は元の「終了」と

図 2a ボタンの継承
10 ポイントの赤枠の周りに 10 ポイントの緑の枠を表示する継承プログラム BE2.py の結果．本文参照のこと．

だけ返る．よく考えると，先ほどの継承プログラム BE.py は class 文を記述しただけであり，GUI は古いまま，新しいメソッドに置き換わった．しかし BE2.py は，まず B.__init__でその継承作業をして，次に緑のフレームを作成し，その中に継承元の GUI をはめこんだ．しかも，それは古いメソッドポインターも変更されることなく同時に引用されたと考えられる．

ここから得られる教訓は，メソッドの継承はよいが，安易には GUI の継承はできないということである．

観光バス乗客窓の開閉コントローラクラスの効化的な使用例

例えば観光バスには乗客用に左右それぞれ 6 個のパワーウインドウが装備されているとする．これらの窓の開閉を運転席からのコントローラで開閉する場合，制御パネルは左右にそれぞれ 6 個のボタンを備えることになる（図 3 参照）．

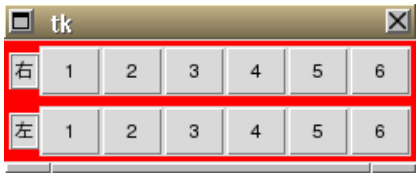


図 3 バス パワーウインドウコントローラー
ラベル名が入力できる Frame クラスを作成し，上下 2 段構成で表示している．ボタンをクリックすると，左右の何番目か端末に表示される．

これを従来通りのプログラムを行うとラベルを含めて 14 個のウィジェット変数が必要になる．しかし，クラスで設計すると 30 行程度でプログラムが仕上り，外部から見えるウィジェット変数は一つもない．プログラムを以下に示す．

File name: Bus6.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# クラス作成
class Frame(tk.Frame):
  def __init__(self, var,
    master=None):
    self.var=var
    tk.Frame.__init__(self, master)
  # クラス画面作成
    f=tk.Frame(master,bd=3,bg='red')
    f.pack()
    # Label
    l = tk.Label(f, text=self.var,
      relief=tk.RIDGE, bd=2)
    l.pack(side="left")
    for i in range(1,7):
      b=tk.Button(f, text=i,
        command=self.make_it(
          self.var, i))
      b.pack(side="left")
  # ボタンメソッド
  def make_it(self,side, i):
     def cmds():
       print u'%s %s 閉じる' % (
         side,i)
     return cmds

# メインプログラム
if __name__ == '__main__':
  fr=Frame(u"右")
  fl=Frame(u"左")
  fl.mainloop()
```

クラス設計では，新たに左右の区別を示すパラメータを受け入れる必要がある．．ボタンの割り込み番号は必要だが，ボタンウィジェット名は引用されることはないから不用である．仕上り図は図 3 である．

ボタンクリックを実行すると，

```
>>> 右 1 閉じる
左 2 閉じる
右 3 閉じる
左 4 閉じる
```

と端末表示される．

端末表示の代わりに本物の制御装置を作動させるインターフェースを作れば実用になる．

プログラム解説：

始めの 3 行は定型どおり．

class Frame の継承は tk.Frame だけでよく，tk.Label や tk.Button は自動的に組み込まれる．__init__関数の引数 var は左右の区別を行うラベルを受け入れる．メソッド関数で引用されるから self.var としてクラス内グローバル変数を作成し，コピーする．続いて Frame の初期設定を行う．親ウィジェットの None 指定は root 指定と同じ効果である．__init__関数からここまでの一連の書式は定型的な書式であるから，いつでも使えるようにメモしておくとよい．特に引数を伴ったインスタンス設計での self の使い方に注意してほしい．

ラベル名を l として引数 self.var を text に指定し，pack している．pack の親元は必ず f とする．Frame コンテナーを作らずにデフォルトで直接ウィジェットを張り付ける例が多数紹介されているが，複雑なウィジェット配置になると思いもしない所に pack されることもあるから，必ずしも行儀がよい方法とはいえない．明示的に Frame を作成し，そこにウィジェットを張り付けることが大切である．ラベル text オプションは self.var として，クラス内グローバル変数を引用している．

ボタンは 6 個必要であるから，range を利用し，for loop で作成する．ボタン名は全て b である

が，重複しても今後外部から引用されることがないから問題は生じない．ボタン命令 make_it は 6 個作られる．

次に，ボタンクリックで端末に番号を表示させるボタンメソッドを記述している．実は，このインスタンス作成時にプログラムが実行される．つまり，ボタン命令 make_it も実行されて端末にボタンラベル名が表示されてしまう．その誤動作を回避するために，make_it 関数の中で更に，cmd 関数を作成し，make_it 関数の return 命令で cmd 関数を実行させた．引数 side と i は局所変数のスコープの性質を利用して上位の関数データを引用している．本来のボタン command オプションは実行関数名だけを入力しなければならない（「()」をつけてはならない）のに，引数を利用ために作ったトリックである．

Frame を新しく特化した Frame に変えたのだからもう従来の Frame の概念はない．また，l や b などのウィジェット名もメインプログラムから見えることはない．

Bus6.py の継承

練習のために Bus6.py のメソッドの継承プログラム Bus6E.py を作成する．例によって,「extend」を追加出力させる．

プログラム解説

ほとんど解説の必要がない．新しく出た項目は，print 文の行末の\は次行への継続符である．

クラス設計の雛型

GUI デザインでフォントの字体，大きさ，前景やや背景の色を変更すると見栄えも変わる．その基本命令は config と cget であり， 単純なラベルウィジェットでこれらのメソッドプログラムを紹介する．対話形式で駆動すれば容易に変更できる．調整できればオプションとして組み

File name:　Bus6E.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from Bus6 import *
# クラス作成
class FFrame(Frame):
  # ボタンメソッド
  def make_it(self,side, i):
     def cmds():
       print u'%s %s extend 閉じる' \
         % (side,i)
     return cmds
if __name__ == '__main__':
  fr=FFrame(u"右")
  fl=FFrame(u"左")
  fl.mainloop()
```

入れればよい．雛型は Frame をもちいて bd=5, bg='green' で枠を明確に表示した．メインプログラムではクラスを 2 回引用したために外枠に段差がついた，そのことからも，独立してインスタンスが作成されていることがわかる．仕上りを図 4 に示す．

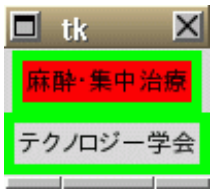


図 4 単純なラベルウィジェットクラス
フレームの中にラベルをパック配置している．上段のラベルは config メソッドで背景を赤色に変換している．

File name:　Prototype.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
class Frame(tk.Frame):
  # クラス初期設定
  # 引数があれば self. 変数を作る.
  def __init__(self, var,
    master=None):
    self.var=var
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master,bd=5,
```

```
      bg='green')
    f.pack()
    self.l = tk.Label(f,
      text=self.var)
    self.l.pack(side="top")
  # 基本メソッド
  def cget(self,data):
    return self.l.cget(data)
  def config(self,pos,data):
    self.l[pos]=data
# メインプログラム
if __name__ == '__main__':
  l1 = Frame(u'麻酔・集中治療')
  l2 = Frame(u'テクノロジー学会')
  l1.config('bg','red')
  print l1.cget('text')
  l2.mainloop()
```

プログラム解説：

__init__関数でラベル表示される引数をグローバル変数に記憶させる．フレームウィジェットを作成する．

明示的にフレーム f を作成し，属性変更を行っている．ラベルウィジェットを作成し，フレームにパックさせている．特定ののラベルインスタンスを認識させるために self を付けている．

メソッドの cget も同様に，どのインスタンスが必要であるかを指定するために self を付ける．config の書式は「bg='red'」の書式では bg がわからないとエラーが発生するためにオプション名も文字列に変更して，上記の記述法を行う．

メインプログラムでは l1 と l2 のインスタンスを作成しパックしている．クラス名が Frame であるが，もはやフレーム機能はない．ラベルが中に入っているフレームである．config メソッドで l1 インスタンスの背景を赤色に変更し，また l1 の text 文を cget で端末に出力している．この雛型は冗長だという意見があるかもしれないが，フレームや init の省略はいつでもできるから，まずはこの雛型をコピーしてから目的とする GUI に少しづつ変更していくことを強く進める．

テキストウィジェット

テキストウィジェットは行，列，フォント設定など多くのオプションが指定でき最も複雑なウィジェットの一つである．さらに，立て，横のスライダーを必要とする．従って，どのような class を作成するのが良いかを考えよう．まずは，クラスを使わずに直接表示する方法である．

フレームを f，テキスト画面を t，X 軸スクロールバーを sx，Y 軸スクロールバーを sy にしてフレーム f に張り付けている．フレームは 3 ピクセルで赤色で明示的に示した．長い一本のメインプログラムになる．プログラムを以下に示し，GUI の仕上りを図 5 に示す．

1) クラスを使わないスクロールドテキスト

スクロールバーのプログラムは Tcl/Tk の例題を参考にして Python に変換するのがよい．

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
root=tk.Tk()
f=tk.Frame(root, bg="red",
  relief="groove",borderwidth=3)
# テキスト画面作成
t=tk.Text(f,width=24, height=8,
  wrap=tk.NONE, setgrid=tk.TRUE,
  font=('FixedSys', 14))
t.grid(column=0, row=0, sticky='nsew')
# X 軸 スクロールバー作成
sx=tk.Scrollbar(f,orient='horizontal',
   command=t.xview, width=10)
t["xscrollcommand"]=sx.set
sx.grid(column=0, row=1, sticky='ew')
# Y 軸 スクロールバー作成
sy=tk.Scrollbar(f,orient='vertical',
  command=t.yview, width=10)
t["yscrollcommand"]=sy.set
sy.grid(column=1, row=0, sticky='ns')
f.pack(fill=tk.BOTH,expand=1)
# 画面を拡大するとテキストだけが拡大する
f.grid_columnconfigure(0,weight=1)
f.grid_rowconfigure(0,weight=1)
f.mainloop()
```

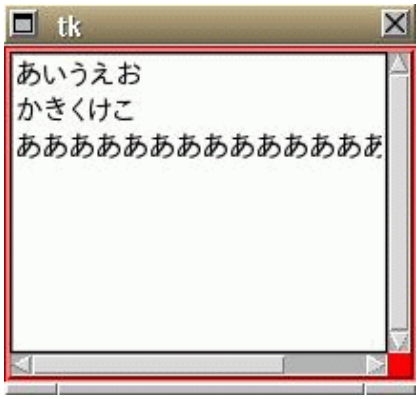


図 5 スクロールバー付きテキストウィンドウ
フレームの中にグリッド配置でテキスト，x 軸スクロール y 軸スクロールを設定している．

プログラム解説：

Frame の中に grid 配置でテキスト，x 軸スクロール y 軸スクロールを設定している．grid 命令は pack 命令と同じく，ウィジェットメソッドに組み入れられている．スクロールバーの設定と接続法は考えても思い付かないから例を真似するとよい．テキスト名，スクロール名がグローバルであるために，ウィジェット名の衝突に注意しなくてはならない．

上記だけのプログラムだと，テキスト画面の書き込みはできるが，プログラム終了と同時にデータは失われる．実際は，新規，読み取り，書き込みなどのメニューバーを作成してテキストデータとリンクするのであるが，その基本命令であるテキストデータの書き込み，読み取り命令を以下に示す．対話形式で Python を走らせて，各自試みてもらいたい．テキストウィジェットの命令は大変多く，これ以外の命令は割愛する．

```
書き込み：
t.insert(tk.END,'abc')
改行書き込み：
t.insert(tk.END,'\n')
日本語書き込み：
t.insert(1.0,u' 日本語 abc\n')
```

```
読み取り：
print t.get(1.0,tk.END)
行は 1 行目から始まり，
列は 0 列から始まる．
削除：
t.delete(2.0,2.2)
2 行目 0 列から 2 文字を削除
```

2) クラスを用いたスクロールドテキスト

スクロールドテキストは結構ややこしいプログラムになるから class でまとめる効果は大きい．クラスを使わない方法のプログラムを，前述のクラス雛型に準じて，また，テキスト，水平スクロール，垂直スクロールバーのウィジェット名はそのままにして，変更を加えた．

本質的な話しではないが，プログラム作成にあたってネスティングがひとつでも増えると，コーディングを失敗する確率が高くなる．筆者のコーディング方法は，まず，雛型をコピーし，def init 以下，すなわち GUI 本体を消す．次いで，クラスを使わない方法で作成したプログラムをコピーし，インデントの調整を行う．root で都合の悪いところを修正し，クラス内局所変数で画像が表示されるところまで努力する．それが済めば，何がクラス内グローバル変数かをよく考えて，変数に self を付ける．この例ではテキスト変数 t 以外に必要な変数はない．メソッドのコーディングは正しく GUI が表示されてからの話しである．

File name: TS.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
class TS(tk.Frame):
  # クラス初期設定
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master, bg="red",
      relief="groove",borderwidth=3)
    # Text 作成
    self.t=tk.Text(f,width=24,
      height=8, wrap=tk.NONE,
      setgrid=tk.TRUE,
      font=('FixedSys', 14))
    self.t.grid(column=0, row=0,
      sticky='nsew')
    # X 軸 スクロールバー作成
    sx=tk.Scrollbar(f,
       orient='horizontal',
       command=self.t.xview,
       width=10)
    self.t["xscrollcommand"]=sx.set
    sx.grid(column=0, row=1,
      sticky='ew')
    # Y 軸 スクロールバー作成
    sy=tk.Scrollbar(f,
      orient='vertical',
      command=self.t.yview,
      width=10)
    self.t["yscrollcommand"]=sy.set
    sy.grid(column=1, row=0,
      sticky='ns')
    f.grid_columnconfigure(0,
      weight=1)
    f.grid_rowconfigure(0,
      weight=1)
    f.pack(fill=tk.BOTH,expand=1)
# メインプログラム
if __name__ == '__main__':
  t=TS()
  t.t.insert('1.0',u' あいうえお\n')
  print t.t.cget('bg')
  t.t['bg']="green"
  t.t.delete('1.0','1.1')
  t.mainloop()
```

プログラム解説：
grid によるウィジェット配置はクラスでも可能である．クラス名を TS として tk.Frame を継承するだけで tk.Text や tk.Scrollbar が取り込まれる．テキストウィジェットのみが他の関数で引用されるから self を付ける必要があるが，スクロールバーに関しては引用されないから不要である．

メインプログラムでは cget，config の代わり

の辞書設定，insert，delete の例を示した．テキストウィジェットのみが self で公開されているから．呼び出しは t.t. と t を重ねればよい．スクロールウィジェットがクラス化されたから，これでいくつでもテキスト画面を作成することができる．例に示したテキスト画面では立て 8 行，横 24 文字の小さな画面であるが，コーナーをドラグすると，いくらでも広がるようにプログラミングしている．

フォント設定は各システムで異なり，難しい問題であるが，例に示されている「FixedSys」は Windows も含めて True Type で使用できるようである．サイズは 14 としているが，任意のサイズが指定できる．他に「Helvetica，Times」などを指定すると日本語では明朝体表示になる．さらに，

```
from tkFont import *
```

と tkFont をプログラムに導入すると，

```
font1=Font(family="Helvetica",
weight=NORMAL, size=18)
t.configure(font=font1)
```

という使い方ができ，

```
font1.configure(size=24)
font1.configure(weight=BOLD)
```

と記述できる．

スクロールドテキストの継承

前述の tkFont を import し，フォントを変更する継承プログラム TSE.py を作成する．

File name: TSE.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from TS import *
from tkFont import *
class TSE(TS):
  # クラス初期設定
  def __init__(self, master=None):
    TS.__init__(self, master)
    font1=Font(family="Helvetica",
      weight=NORMAL, size=8)
    self.t.configure(font=font1)
# メインプログラム
if __name__ == '__main__':
  t=TSE()
  t.t.insert('1.0',
    u'テクノロジー学会\n')
  t.mainloop()
```

プログラム解説

tkFont はクラス属性でないのか上記の方法以外に import できない．つまり，名前空間はないようである．TS の継承クラス名を TSE にした．クラス内では self.t と指定すると configure 関数が通過する．

このプログラムの流れは、メインプログラムで TSE クラスを駆動する．TSE の初期設定で継承元の TS を初期設定する．そこでテキスト画面が表れ，次いで，font1 設定，self.t.configure を実行し，メインプログラムに返る．最後に，メインプログラムの t.insert を実行し，mainloop に入る．

以前 GUI 本体の継承は無理と述べたが，今回の TSE は継承が成功している．その理由は GUI 本体そのものの変更ではなく，単に GUI 属性の追加，変更であるためである．つまり，pack 命令の必要な変更はダメで，メソッドを含むそれ以外の GUI 属性の変更は継承すると考えればよい．

次項ではテキストウィジェットのデータを読み書きできるプルダウンメニューを述べる．

プルダウンメニュー

プルダウンメニューは種々のオプションがあり，相当複雑なプログラムになる．またどの程度ま

でクラス内に閉じ込めて汎用性を保つかという問題もある．とりあえず従来通りの方法を以下に示す (fm.py)．

1) クラスを使わないプルダウンメニュー

メニュー命令はいろいろなボタン操作で種々のコマンドを呼び出し，また継承を利用することが多い．

プログラム解説

プルダウンメニューは Menubutton が分かりやすいので，それを利用する．上記の例はフレー

File name: fm.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
#ボタンクリック処理
def callback():
  print 'exit'
  exit()
# メインプログラム
root=tk.Tk()
f=tk.Frame(root,bg='red',bd=3)
f.pack(anchor='nw')
file=tk.Menubutton(f, text='File',
  relief=tk.RAISED)
file.pack()
file.m =tk.Menu(file, tearoff=0 )
file['menu']=file.m
file.m.add_separator()
file.m.add_command(label='Quit',
  command=callback)
f.mainloop()
```

ムに「File」と表示されたプルダウンメニューを作成し，クリックするとセパレータと「Qiit」と表示するボタンが表れる.「Quit」をクリックすると callback 関数が呼び出され，端末に「exit」と記述してプログラムが終了する．

プルダウンメニューのメインが file ボタンで，各メニュー file.m になり，それに add_命令を加えていると読めば意味がわかる．Tcl/Tk では Menubutton で-menu file.m が認められるが，Tkinter では file.m が存在しないとエラーが発生する．そのためにそのパラメータ無しで file を作成し，その後，file["menu"]=file.m と記述する．

メニュー全体が Frame コンテナーに乗っていることの確認に，bg='red',bd=3 で見えるようにしたが，bd=0 にすれば簡単に消せる．

2) クラスを用いたプルダウンメニュー

前述のプログラムは Tcl/Tk プログラムに準じて作成したので，長い一連のプログラムになり，見通しが悪い．クラスで記述すると，内容はより複雑になるが，メインプログラムからはクラス内はブラックボックスであるため，見通しは大変よくなる．また，callback 関数の多態性も利用できる．FM はクラス名であるために，前述の fm.py を FM.py にプログラム名を変更してクラスで書き直す．

File name: FM.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
class FM(tk.Frame):
  # クラス初期設定
  # 引数があれば self. 変数を作る.
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master,bg='red',bd=3)
    f.pack(anchor='nw')
    file=tk.Menubutton(f,
      text='File', relief=tk.RAISED)
    file.pack()
    file.m =tk.Menu(file, tearoff=0)
    file['menu']=file.m
    file.m.add_separator()
    file.m.add_command(label='Quit',
      command=self.callback)
# メソッド
  def callback(self):
    print 'exit'
    exit()
```

```
# メインプログラム
if __name__ == '__main__':
  f=FM()
  master=None
  g=tk.Button(master,text=u'終了',
    command=exit)
  g.pack(side='left')
  f.mainloop()
```

プログラム解説：

雛型に準じて，先の例を書いただけである．ただし，tk.Frame の親ウィジェットを root でなく master に変更している．それ以降の各ウィジェットに self はない．考えれば，同じクラスを数多く作る必要がないためである．しかし，ボタンメソッドは継承に必要であるから，self を付けている．なお，クラス名の FM は File Menu のイニシャルを選んだ．

3) 実用になるプルダウンメニュー

ファイル選択ダイアログに tkFileDialog があり，その動作を対話モードで調べると．

```
>>> import tkFileDialog as D
>>> dir(D)
['Dialog', 'Directory', 'Open',
'SaveAs', '_Dialog', 'askdirectory',
'askopenfile', 'askopenfilename',
'askopenfilenames', 'askopenfiles',
'asksaveasfile', 'asksaveasfilename']
```

などがメソッドとして登録されている．そこで askopenfilename と検討をつけて，

```
>>> f=D.askopenfilename
>>> f=D.askopenfilename()
>>> print f
/home/tanaka/a1.py
```

と思う通りに選択したファイル名が f に記録された．この結果を考慮して同じファイル名でプルダウンメニューを作成する．

File name: FM.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import tkFileDialog as D
# グローバル変数
fname='noname.txt'
# ファイルメニュー
class FM(tk.Frame):
  # クラス初期設定
  # 引数があれば self. 変数を作る.
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    self.f=tk.Frame(master,bg='red',
      bd=3)
    self.f.pack(anchor='nw')
    file=tk.Menubutton(self.f,
      text='File', relief=tk.RAISED)
    file.pack(side='left')
    file.m =tk.Menu(file, tearoff=0 )
    file['menu']=file.m
    file.m.add_command(label='New',
      command=self.new_f)
    file.m.add_command(label='Open',
      command=self.open_f)
    file.m.add_command(label='Save',
      command=self.save_f)
    file.m.add_command(
      label='Save As',
      command=self.saveas_f)
    file.m.add_separator()
    file.m.add_command(label='Quit',
      command=self.exit_f)
# メソッド
  def new_f(self):
    fname='noname.txt'
    print 'FM',fname
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    print 'FM',fname
  def save_f(self):
    print 'FM',fname
  def saveas_f(self):
    fname = D.asksaveasfilename(
      event=None)
    print 'FM',fname
  def exit_f(self):
    print 'exit_f'
```

```
    exit()
# メインプログラム
if __name__ == '__main__':
  f=FM()
  f.mainloop()
```

プログラム解説：

メニューフレームは将来の機能追加のために公開にした．また，左から右へとボタンが並ぶように pack した．ファイルメニューは New，Open，Save，Save As，Quit の順に作成している．メニュー選択をクリックすると外部関数に作業が移動する方式にしている．オープンするファイル名はメインプログラムで fname とし，新規 (New) は'noname.txt' と初期設定する．このプログラムはプロトタイプなので，単純に端末にファイル名を表示するだけであるが，実際はテキストウィジェットの画面をクリアーする作業が続く．保存 (Save や Save As) ならテキストウィジェットのデータを書き込む作業になる．当然終了なら，その前に注意喚起ダイアログを引き出す必要があろう．次項では，前述のスクロールドテキストとメニュー画面を含めてエディタープログラムを作成するが，その前に FM.py のメソッドの継承を行う FME.py を作成して，継承の仕方を明らかにする．

FM.py の継承プログラム FME.py

メニューの基本構成は祖のままに，メニュー選択による callback 関数だけを変更してプログラムの柔軟性を上げることはオブジェクト指向の最も得意とするところである．FM.py をスーパクラスとして Edit メニューの追加と，File メニューのメソッドだけを変更する FMJ.py を作成する．単純な例題であるから，単にメニュークリックで端末出力の行頭に「extend 」と打ち出すだけのプログラムである．

File name: FME.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
# ファイルメニュー 継承
class FME(FM):
# メソッド
  def new_f(self):
    fname='noname.txt'
    print 'extend FM',fname
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    print 'extend FM',fname
  def save_f(self):
    print 'extend FM',fname
  def saveas_f(self):
    fname = D.asksaveasfilename(
      event=None)
    print 'extend FM',fname
  def exit_f(self):
    print 'extend exit_f'
    exit()
# メインプログラム
if __name__ == '__main__':
  f=FME()
  f.mainloop()
```

プログラム解説

継承元のプログラムを from 命令で import する．File メニューのメソッド関数は全て書き直おすだけである．多態性を利用して確認に必要な「extend」だけを追加した．メインプログラムのクラス名は FME に書き直し，後はそのままである．

テキストエディター作成

Tkinter のテキストウィジェットはそれだけで十分な編集機能が備えている．従って，テキストウィジェットのデータの入出力をメニューボタンで操作できればエディターとしての基本機能は備わることになる．そこで，これまで作成した TS クラス，FM クラスを継承するエディター

Ed.py を製作する．

作業としては，二つのリソースを継承し，一つの GUI を作成すること，FM クラスのメソッドを変更し，テキストウィジェットとのデータの入出力が対応できるメソッドを製作することになるが，よく考えると FM.py の継承版 FME.py のコピーファイルを Ed.py に名前を変えて，その class 名を Ed に変更するだけでよい．終了メッセージダイアログを import しておく．また，新しい GUI 構築は継承できないので，メインプログラムで TS を呼び出す必要がある．仕上り GUI を図 6 に示す．

まず，骨格となる基本プログラムを示し，次にデータ授受のためのメソッド部分については，順次明らかにしていく．

File name: Ed.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
from TS import *
import tkMessageBox as M
# エディター
class Ed(FM):
# ファイルメニュー 継承
# メソッド
  def new_f(self):
    fname='noname.txt'
    print 'extend FM',fname
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    print 'extend FM',fname
  def save_f(self):
    print 'extend FM',fname
  def saveas_f(self):
    fname = D.asksaveasfilename(
      event=None)
    print 'extend FM',fname
  def exit_f(self):
    print 'extend exit_f'
    exit()
```

図 6 簡易エディター Ed.py
メニュー FM，スクロールドテキスト TS を継承した．

```
# メインプログラム
if __name__ == '__main__':
  master=None
  f=Ed()
  t=TS()
  t.t.insert('1.0',u'あいうえお\n')
  f.mainloop()
```

画面表示して，画面の各コーナーをドラグしてもテキスト画面だけが拡張してメニューは左上端に固定されたままであることを確認する．

それではメニューメソッドの継承プログラムを class Ed 初期設定の下に継ぎ足すことにする．最初は new_f 関数である．

new_f 関数は，テキスト内の内容を全て消し，記録ファイル名を「nonmae.txt」にする．プログラムは

```
  def new_f(self):
    fname='noname.txt'
    t.t.delete(1.0,'end')
```

となる．Ed.py を実行させ，適当に画面に文字を入力し，New を指定して，画面の「あいうえお」が消えれば成功である．

open_f 関数は既存のファイルを読み込むモードであるから，読み取りダイアログを開け，指定ファイルを読み取り，テキスト画面に書き込む．既存のデータの後に書き込むことにした．最初からファイルだけを書き込む場合は一旦，New を実行させ，続いて Open を実行する．

```
def open_f(self):
  global fname
  fname = D.askopenfilename(
    event=None)
  fn=open(fname,"r")
  t.t.delete(1.0,tk.END)
  for line in fn.readlines():
    t.t.insert(tk.END,line[:-1]. \
      decode('euc_jp')+"\n")
  fn.close()
```

となる．

save_f 関数は，テキストデータ全てを fname で指定したファイルに書き込む操作である．

```
def save_f(self):
  data=t.t.get(1.0,tk.END-1)
  data1=data.encode('euc_jp')
  fn=open(fname,"w")
  fn.write(data1)
  fn.close
```

となる．

saveas_f 関数は，始めに書き込みファイルダイアログを開き，ファイル名を指定し，そこに書き込む操作である．

```
def saveas_f(self):
  global fname
  fname = D.asksaveasfilename(
    event=None)
  data=t.t.get(1.0,tk.END)
  data1=data.encode('euc_jp')
  fn=open(fname,"w")
  fn.write(data1)
  fn.close
```

となる．

exit_f 関数はエディターを終了すればよいが，注意喚起をした方が親切である．ダイアログの呼び出しは，tkMessageBox モジュールが簡単である．python 対話モードで調べると，

```
>>> import tkMessageBox as M
>>> dir(M)
['askokcancel', 'askquestion',
'askretrycancel', 'askyesno',
'showerror', 'showinfo',
'showwarning']
>>>
```

となっている．askokcancel を選択すると，'OK' なら True，'Cancel' なら False が返る．これで exit_f 関数が出来る．

```
def exit_f(self):
# メッセージボックス
  ans=M.askokcancel(u'メッセージ',
    u'終わりますよ．')
  if ans==True:
    exit()
  else:
    return
```

となる．

以上で簡易エディター Ed.py のプログラムは完成であるが，Windows のための encode，decode オプションは'shift_jis' である．カット/ペーストを行う selection 機能は Linux ではマウス位置が大変クリティカルで使いにくく，更に他のディスプレー画面の日本語文字が通過しないなどの欠点がある．しかし，同じテキスト画面内であれば，Ctrl+C，Ctrl+X，Ctrl+V キーで快適に操作できる．デフォルト画面を小さくした理由はいくつもの簡易エディターをディスプレーに表示させるためである．表示画面の邪魔にならないことが重要であろう．また，画面のコーナーをドラグすれば，いくらでも大きく拡張できる．

世の中に vi を始めとして立派なエディターがあるのに，なぜこのような簡易エディターを作成

したのかという質問に対して，汎用エディターでは出来ない機能，例えば，2 行にまたがる検索/置換，上付き，下付き文字の検索/置換などに役立てることができる．これらの作業は Tex ソースの作成時に必要で，私的な編集メニューで利用している．また，Python の継承能力を確かめる意味もある．分散プログラミングと継承の概念があればこそ，この程度の行数で実用可能なエディターが作成できることを理解してもらいたい．

画像ウィジェット

今日では立派な画像ソフトが数多く公開されているが，出来上がった画像ファイル（jpg，png，gif など）からその X-Y 座標を数値として読み取るソフトは筆者は知らない．例えば心電図のグラフを数値として読み取ることができれば，心電図解析のための強力なツールになる．画像処理とはそのような可能性を持つプログラムである．

Tcl/Tk の拡張版である Tkinter では bitmap，gif ファイルしか対応できていない．しかし，ImageTk を import することにより，jpg，png 画像をキャンバスに表示することができる．その基本となるプログラムと結果を以下に示す．

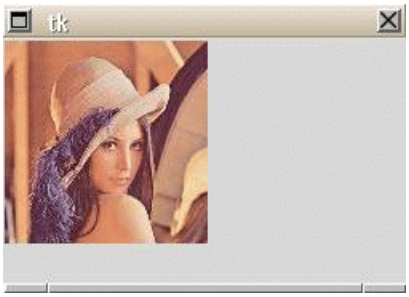


図 7 キャンバス画像基本表示
ImageTk.Photoimage で jpg 画像をキャンパスに表示している．

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import Image as I
import ImageTk as Itk
c = tk.Canvas(width=300, height=200)
p=Itk.PhotoImage(file='lena.jpg')
c.create_image(0,0,anchor=tk.NW,
  image=p)
c.pack()
c.mainloop()
```

プログラム解説：

画像キャンバス c を作成し，ファイル「lena.gif」はプログラムと同じディレクトリにあるとして，Itk.PhotoImage で Tkinter で表示できる画像に変換する．そしてキャンバス c の X-Y 座標の原点に画像の左上端を合わせて表示している．Image モジュールは Python Imaging Library(PIL) で公開されている．また ImageTk は PIL と Tkinter との変換モジュールで公開されている．

練習を兼ねて，前述のプログラムをクラスでプログラミングをしよう．例により Frame にキャンバスを張り付ける．似たような構造はボタンウィジェットであったことを思い出してほしい．

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import Image as I
import ImageTk as Itk
class C(tk.Frame):
  # クラス初期設定
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master,bg='red',bd=3)
    f.pack()
    self.c=tk.Canvas(f,width=250,
      height=150)
    self.c.pack()
# メインプログラム
if __name__ == '__main__':
  c=C()
  p=Itk.PhotoImage(file='lena.jpg')
  c.c.create_image(0, 0,
    anchor=tk.NW, image=p)
  c.mainloop()
```

プログラム解説：

無事に赤枠の中にキャンパス，画像が表示されれば正解である．何がクラスの中に入り，self をどこに付けるか，メインプログラムとクラスの仕分けが重要である．

1) クラスを使ったスクロールドキャンバス

テキストと同様にキャンパスもスクロールできな

ければ大きな画像は表示できない．方法は，今までに作成したファイルメニューを継承し，テキストウィジェットの代わりにキャンバスウィジェットを利用すれば画像表示は可能である．問題は画像ファイルの X-Y 座標から色データの抽出であるが，Photoimage の get，put 命令で色データの抽出，書き込みができる．まずはスクロールドキャンバスの作成である．TS.py を変更すればよく，ファイル名は，CA.py とする．

File name: CA.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import Image as I
import ImageTk as Itk
class CA(tk.Frame):
  # クラス初期設定
  def __init__(self, master=None):
    tk.Frame.__init__(self, master)
  # クラスウィジェット作成
    f=tk.Frame(master, bg="red",
      relief="groove",borderwidth=3)
    # キャンバス作成
    self.c=tk.Canvas(f,width=250,
      height=150,
      scrollregion=(0,0,400,400))
    self.c.grid(column=0, row=0,
      sticky='nsew')
    # X 軸 スクロールバー作成
    sx=tk.Scrollbar(f,
       orient='horizontal',
       command=self.c.xview,width=10)
    self.c["xscrollcommand"]=sx.set
    sx.grid(column=0, row=1,
      sticky='ew')
    # Y 軸 スクロールバー作成
    sy=tk.Scrollbar(f,
      orient='vertical',
      command=self.c.yview, width=10)
    self.c["yscrollcommand"]=sy.set
    sy.grid(column=1, row=0,
      sticky='ns')
    f.grid_columnconfigure(0,weight=1)
    f.grid_rowconfigure(0,weight=1)
    f.pack(fill=tk.BOTH,expand=1)
  # 基本メソッド
  def cget(self,data):
    return self.c.cget(data)
  def config(self,pos,data):
    self.c[pos]=data
  def insert(self,pos,data):
    self.c.insert(pos,data)
  def get(self,pos1,pos2):
    return self.c.get(pos1,pos2)
  def delete(self,pos1,pos2):
    return self.c.delete(pos1,pos2)
# メインプログラム
if __name__ == '__main__':
  c=CA()
  p=Itk.PhotoImage(file='lena.jpg')
  c.c.create_image(0, 0,
    anchor=tk.NW, image=p)
  c.c.create_line(0,0,60,30,
    tags="t1")
  c.mainloop()
```

となる．上記プログラムの仕上りを図 8 に示す．メインプログラムに画像と直線を表示する命令を書き込んでいるから，スクロール機能が確かめられる．

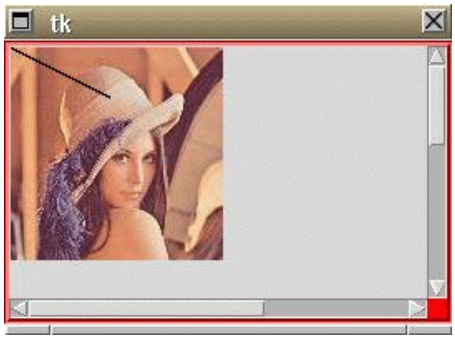


図 8 スクロールドキャンバス CA.py
スクロールドキャンバスは TS.py を変更しただけで作成できる．

2) スクロールドキャンバスとメニューの結合

キャンパスには新規画像を作成する機能，画像を読み取る機能，記録する機能が必要である．そこで，メニュー FM.py を import し，Ed.py と同様の画像エディター Ied1.py を作成する．実際は Ed.py をコピーし，メソッドの変更を加えるだけである．Ied1 の 1 は改定番号で，変更を

加えるごとに番号を上げ，最終版は Ied.py の名前に落ち着くことにする．

File name: Ied1.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
from CA import *
import tkMessageBox as M
# 図形エディター
class Ied(FM):
# ファイルメニュー 継承
# メソッド
  def new_f(self):
    fname='noname.txt'
    t.t.delete(1.0,'end')
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    fn=open(fname,"r")
    t.t.delete(1.0,tk.END)
    for line in fn.readlines():
      t.t.insert(tk.END,line[:-1]. \
        decode('euc_jp')+"\n")
    fn.close()
  def save_f(self):
    data=t.t.get(1.0,tk.END-1)
    data1=data.encode('euc_jp')
    fn=open(fname,"w")
    fn.write(data1)
    fn.close
  def saveas_f(self):
    global fname
    fname = D.asksaveasfilename(
      event=None)
    data=t.t.get(1.0,tk.END)
    data1=data.encode('euc_jp')
    fn=open(fname,"w")
    fn.write(data1)
    fn.close
  def exit_f(self):
  # メッセージボックス
    ans=M.askokcancel(u'メッセージ',
      u'終わりますよ．')
    if ans==True:
      exit()
    else:
      return
# メインプログラム
if __name__ == '__main__':
  f=Ied()
  c=CA()
  p=Itk.PhotoImage(file='lena.jpg')
  c.c.create_image(0, 0, anchor=tk.NW,
    image=p)
  c.c.create_line(0,0,60,30,tags="t1")
  f.mainloop()
```

とりあえず，Ied1.py はグラフィクスに関係する import 文の追加，class 文を変更，そして，メインプログラムを変更しただけである．メソッドは変更を加えていないから，テキストエディター Ed.py の最終版のままであり，その参考にしてもらいたい．

さっそく新規画像 (new_f) メソッドの作成であるが，デフォルトの名前設定と，キャンバスに表示されている画像部品の全てを消し去る仕事を行う．find_all 関数は表示されている部品番号を返すからそれを利用する．delete 関数は部品の消去である．プログラムは

```
def new_f(self):
  fname='noname.txt'
  for i in c.c.find_all() :
    c.c.delete(i)
```

となる．

画像入力 (open_f) メソッドは現在のキャンパスに画像を追加する機能である．どの位置に画像を追加させるかは次の課題として，とりあえず (0,0) に表示させる．プログラムは

```
def open_f(self):
  global fname
  fname = D.askopenfilename(
    event=None)
  self.p=Itk.PhotoImage(file=fname)
  c.c.create_image(0,0,
    anchor=tk.NW,
    image=self.p, tags='latest')
```

となる．部品番号は自動的に配付されるが，最

新という意味で「latest」とタグを付けている．Itk.PhotoImage 変換したイメージ p には self を付けなければ create_image が引用できない．画像記録（save_f）メソッドは Tkinter では Postscript(PS) ファイルしかサポートされていない．そのためにプログラムは

```
  def save_f(self):
    c.c.postscript(file='a.ps')
```

となる．PIL で ps ファイルより別の画像に変換することは可能だが，現バージョン（PIL 1.16）では画像が縮小劣化するので，これ以上は深入りしない．

名前を付けての保存（saveas_f）メソッドもダイアログを表示してファイル名を定めるだけである．プログラムは

```
  def saveas_f(self):
    global fname
    fname = D.asksaveasfilename(
      event=None)
    c.c.postscript(file=fname)
```

となる．fname はグローバル変数であるから self の必要はない．

終了（exit_f）メソッドは変更がない．ここまでの変更をまとめて Ied2.py とすると，プログラムは

File name: Ied2.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
from CA import *
import tkMessageBox as M
# 図形エディター
class Ied(FM):
# ファイルメニュー 継承
# メソッド
  def new_f(self):
    fname='noname.txt'
    for i in c.c.find_all() :
      c.c.delete(i)
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    self.p=Itk.PhotoImage(file=fname)
    c.c.create_image(0,0,
      anchor=tk.NW,
      image=self.p, tags='latest')
  def save_f(self):
    c.c.postscript(file='a.ps')
  def saveas_f(self):
    global fname
    fname = D.asksaveasfilename(
      event=None)
    c.c.postscript(file=fname)
  def exit_f(self):
  # メッセージボックス
    ans=M.askokcancel(u'メッセージ',
      u'終わりますよ．')
    if ans==True:
      exit()
    else:
      return
# メインプログラム
if __name__ == '__main__':
  master=None
  f1=tk.Frame(master,bg='green',bd=3)
  f1.pack(anchor='nw')
  f=Iedd(f1)
  c=CA()
  c.c.create_line(0,0,60,30,tags="t1")
  p=tk.PhotoImage(file='lena.gif')
  c.c.create_image(50,0,
    anchor=tk.NW,image=p)
  f.mainloop()
```

となる．

Tcl/Tk の知識で Tkinter でのキャンバスウィジェット命令の検討はつくが，正確ではない．課題を解決する前に，どの様なキャンバスウィジェットメソッドがサポートされているか，正確に知る必要がある．その方法は，対話もードで python を呼び出し，

```
>>> from FM import *
>>> from CA import *
>>> c=CA()
>>> dir(c.c)
['_Misc__winfo_getint',
..... 省略 .....
ew_moveto', 'yview_scroll']
>>>
```

とすると，莫大な数のメソッドが端末に表示される．また PIL の正式情報は，

http://www.pythonware.com/library/index.htm

に公開されている．

画像の移動

読み取った画像を (0,0) に張り付けるのでは役に立たないので，マウスで画像を選択し，任意の部位に移動するルーチンを付加する．そのためにはマウスの位置情報を取得するバインド機能が必要になる．

さっそく最も簡単なバインドプログラムを以下に示す．キャンバスを作成し，そこにボタン 1 を押してドラグすると x-y 座標が端末に表示されるだけのプログラムである．

File name: pr_xy.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# B1 Motion バインド
def b1motion(event):
  print 'x = ',event.x, \
    'y = ', event.y
# メインプログラム
c = tk.Canvas(width=300, height=200)
c.pack()
c.bind('<B1-Motion>',b1motion)
c.mainloop()
```

プログラム解説

バインド関数は b1motion で，引数 event を浮け付ける．event は構造体になっており，カーソルの X-Y 座標は event.x，event.y の変数名になっており、それらを表示している．キャンバスインスタンス c には bind メソッドがあり，イベント条件とイベント処理関数を登録する．

次は実際に，サークルを作成し，それにマウスを当ててドラグするとサークルが移動するプログラムを作成する．

File name: mv_circle.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# マウスバインド処理
# タグ 'a1' 移動処理
def mv(event):
  data=c.coords('a1')
  x1,y1,x2,y2=data
  oldx=(x1+x2)/2.0
  oldy= (y1+y2)/2.0
  dx=event.x-oldx; dy=event.y-oldy
  c.move('a1',dx,dy)
# メインプログラム
c = tk.Canvas(width=300, height=200)
c.pack()
r1=c.create_oval(10, 10, 30, 30,
  fill='red', tag='a1')
#c.bind('<B1-Motion>',mv)
#c.bind('<Button-1>',mv)
c.tag_bind('a1','<Button-1>',mv)
c.tag_bind('a1','<B1-Motion>',
  mv, '+')
c.mainloop()
```

プログラム解説

キャンバスに直径 20 のサークルを (20,20) の位置に作成し，タグを'a1' と文字列で指定する．マウス左クリックで'<Button-1>'，ドラグで'<B1-Motion>' イベントが発生し，共に mv 関数を呼ぶ．

mv 関数の move メソッドはサークルを dx，dy だけ移動する命令を行うが，event で示される現在のマウスの位置，そして coords で示される

サークルの位置情報から計算する．oldx，oldy はサークルの中点座標になる．

バインドプログラムは引数が (event) に限られるため汎用性が望めず，残念ながら mv 関数はメインプログラムと切り離して記述はできない．コメントマーク (#) が付いている c.bind 命令ではキャンバスのどこでもクリックすれば mv 関数が作動する．一方の c.tag_bind では，タグで示めされるウィジェットにマウスを移動しなければ mv 関数が作動しない．

もう一つの例として，いくつかのサークルがキャンバスに有り，その一つを左ボタンで指定し，ドラグで移動させるプログラムを作成する．仕上りを図 9 に示す．

File name:　mv_circles.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# マウスバインド処理
def mv(event):
  try:
    id=c.find('closest',
      event.x,event.y)
    tag,now=c.gettags(id)
    data=c.coords(tag)
    x1,y1,x2,y2=data
    oldx=(x1+x2)/2.0
    oldy= (y1+y2)/2.0
    dx=event.x-oldx; dy=event.y-oldy
    c.move(tag,dx,dy)
  finally:
    return
# メインプログラム
c = tk.Canvas(width=300, height=200)
c.pack()
x=10; y=30
for i in range(3):
  t='a'+str(i)
  c.create_oval(x, x, y, y,
    fill='red', tag=t)
  x=x+20; y=x+20
c.bind('<B1-Motion>',mv)
c.bind('<Button-1>',mv)
c.mainloop()
```

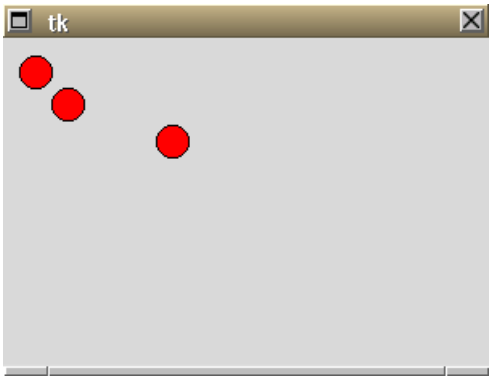


図 9 サークルの移動
サークルを 3 個キャンバス上に作成し，3 番目のサークルをマウスドラグで移動した．

プログラム解説

メインプログラムで range(3) を利用してサークルを 3 個作成する．サークルのタグ名は'a' を接頭語にして range の数字を利用している．サークルの配置は x，y にそれぞれ 20 を加算して斜め下に並べた．

マウスバインドの mv 関数は find 命令でマウス座標に最も近い部品 ID を得る．gettags で ID のタグ名を得る．coords でその部品の位置を知り，dx,dy を計算して move 命令で部品をマウス位置に移動させている．

もし，マウス位置に部品が見当たらない場合はエラーが発生する．その場合，何も動作をせずにイベントを強制終了させるために try/finally 例外処理を行った．bind 命令と tag_bind 命令との使い分けは，特定のタグが明らかであれば tag_bind 命令，そうでなければウィジェット名を find 命令で探し，bind 命令を実行する．

バインド処理と部品移動の理解が深まったところで本論に戻り，画像ファイルの移動機能および部品どうしの重なりを変更する機能を付加したプログラムを最終版 Ied.py として，プログラムと仕上りを以下に示す．

File name: Ied.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
from CA import *
import tkMessageBox as M
# 図形エディター
class Ied(FM):
# ファイルメニュー 継承
# メソッド
  def new_f(self):
    fname='noname.txt'
    for i in c.c.find_all() :
      c.c.delete(i)
  def open_f(self):
    global fname
    fname = D.askopenfilename(
      event=None)
    self.p=Itk.PhotoImage(file=fname)
    c.c.create_image(0,0,
      anchor=tk.NW,
      image=self.p, tags='latest')
  def save_f(self):
    c.c.postscript(file='a.ps')
  def saveas_f(self):
    global fname
    fname = D.asksaveasfilename(
      event=None)
    c.c.postscript(file=fname)
  def exit_f(self):
  # メッセージボックス
    ans=M.askokcancel(u'メッセージ',
      u'終わりますよ．')
    if ans==True:
      exit()
    else:
      return
# メインプログラム
if __name__ == '__main__':
  # マウスバインド
  # 部品移動
  def mv(event):
    try:
      id=c.c.find('closest',event.x,
        event.y)
      idn=id[0]
      c.c.tag_raise(idn)
      data=c.c.coords(idn)
      x1=data[0]
      y1=data[1]
      dx=event.x-x1; dy=event.y-y1
      c.c.move(idn,dx,dy)
    finally:
      return
  # 部品重なり：下の部品を上へ
  def up(event):
    try:
      id=c.c.find('closest', event.x,
        event.y)
      idn=id[0]
      c.c.tag_raise(idn)
    finally:
      return
  # キャンバスメインプログラム
  f=Ied()
  c=CA()
  c.c.create_line(0,0,60,30,tags="t1")
  p=Itk.PhotoImage(file='lena.jpg')
  c.c.create_image(50,0,
    anchor=tk.NW,
    image=p, tag='abc')
  c.c.bind('<B1-Motion>',mv)
  c.c.bind('<Button-1>',mv)
  c.c.bind('<Button-3>',up)
  f.mainloop()
```

プログラム解説

前述のごとく，イベント処理プログラムは import で読み込むことができないためにメインプログラム領域に書き込む．左クリックおよび左ドラグは mv 関数で対応し，右クリックは指定された部品を画面前方に移動する．これらのキャンバスバインドは c.c.bind として mainloopu() のすぐ上に記述している．

上記プログラムではキャンバスでマウスクリック，ドラグ操作で必ずバインドイベントが動作する．この動作を停止するには

```
c.c.bind('<B1-Motion>','-',mv)
c.c.bind('<Button-1>','-',mv)
c.c.bind('<Button-3>','-',up)
```

とイベントと実行関数との間に「-」を挿入すればよい．再度，別の実行関数をバインドに登録

することができる．

mv 関数は左クリックを行ったとき，部品が発見できなければエラーでプログラムが停止する．それを防止する目的で try:を使用した．また，タグの取扱いも面倒なので，部品 ID を利用した．直線などの coors は x-y 座標の左上，右下を示す 4 要素リスト情報が得られるが，画像イメージは左上端だけの 2 要素リストで返る．そのため，左上端の x 座標を x1，y 座標を y1 として，マウス移動を左上端に合わすプログラムに変更した．また，部品が他の部品の下に存在すると正確な位置調整ができないため，移動する部品は前面に位置するように c.c.tag_raise 関数を使用した．

up 関数は mv 関数の上の部分だけであるから，理解できると思う．仕上りを図 10 に示す．

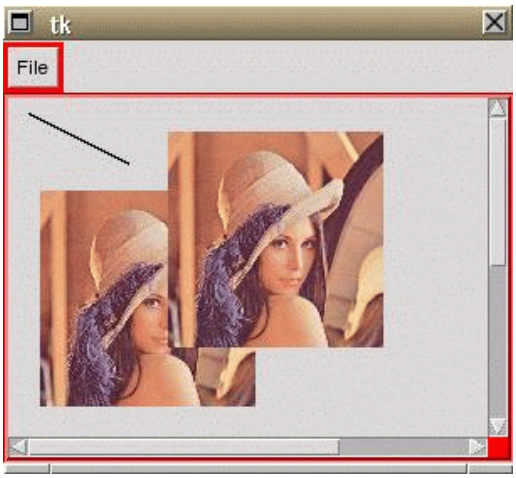


図 10 Ied.py による画像入力と部品移動
直線と'lena.gif' はプログラム駆動時に表示されているもので，新たに'lena.gif' を加えた．それぞれの部品はマウスドラグで移動できる．

トップレベル作成法

前述の例では使用しなかったが，ダイアログと同じように root ウィンドウとは別にトップレベルでウィンドウを作成することができる．プログラムの雛型は，

```
# トップレベル
def top():
  t1=tk.Toplevel()
  t1.title('Toplevel')
  w = tk.Canvas(t1,width=150,
    height=100,bg='white')
  w.pack()
  w.create_line(0,0,100,100)
```

とすればよい．ボタン機能などを用いて top 関数を呼び出せば，幅 150，高さ 100 のキャンバスに (0,0,100,100) の直線が表示される．

ボタン機能の追加は FM.py の Frame を横並びにすること，および Frame 名を公開することでできる．変更点は，

```
  # クラスウィジェット作成
    self.f=tk.Frame(master,bg='red',
      bd=3)
    self.f.pack(anchor='nw')
    file=tk.Menubutton(self.f,
      text='File',relief=tk.RAISED)
    file.pack(side=tk.LEFT)

# Ied.py メインプログラム追加
# キャンバスメインプログラム
f=Ied()
b=tk.Button(f.f,text='Top',
  command=top)
b.pack(side=tk.LEFT)
c=CA()
```

となる．つまり，赤で示す Frame 名 f に self を付け，メニューボタンの親ウィジェットを self.f にする．これでフレーム名 f は公開される．メニューボタン名 file のパックは side=tk.LEFT を指定するだけである．Top と表示されているボタンは FM.py で公開されたフレームを使用するから，親ウィジェットに f.f が指定でき，横並びに pack できる．仕上りを図 11 に示す．

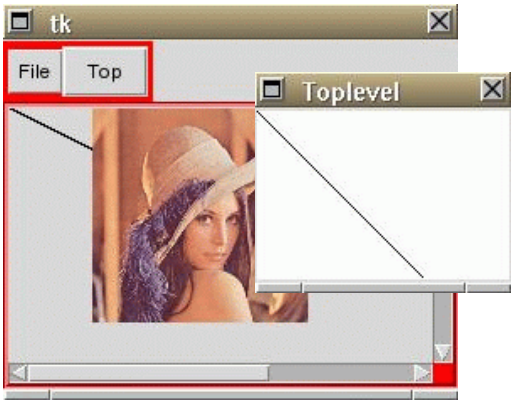


図 11 グラフィクエディター Ied.py
メニューボタンの横並びにトップレベル呼び出しボタンを追加している．

以上で，Tkinter，クラスを用いた GUI 基本ソフトの作成を終える．テキストエディター，グラフィクエディター共にもっと機能を増やすことができるが，一般に公開されている機能まで近づけることに意味はないと思うし，またプログラムの見通しが悪くなり，容易に変更できなくなる．この程度のプログラムを基本ソフトとして常時たづさえており，特別な処理プログラムが必要なときに，これらのプログラムをベースにして，開発を進めるのがよいと思う．

心電図ベクトル波形作成システム

心電図ベクトル波形作成システムとは心電図 12 誘導で測定した検査結果を画像スキャナーで読み取り，その画像よりベクトル心電図波形をつくり出すプログラムである．画像読み取り作業は市販のプリンタースキャナーシステムを利用することにして，1) 得られた画像をトリミングする，2) 心電図波形だけを抽出する，3) その波形より X-Y 座標を検出する，そして最後に 4) ベクトル心電図を合成する作業が必要になる．これらの作業を編集メニューに組み入れるか，それともファンクションボタンを横に列べるのが良いかなどの使い勝ては後で判断することにして，まずはファイルメニューの右横にボタンを配置し，下に画像編集画面を配置するだけのプログラムだけを考えて，プログラム名を Ved1.py とする．FM.py のフレーム f は前項で self を付けて横並び公開を行った．

ファンクションボタンの設定

Ied.py のコピーファイルを Ved1.py として，バインドルーチンを削除した骨格プログラムを以下に示す．

File name:　Ved1.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from FM import *
from CA import *
import tkMessageBox as M
class Ved(FM):
  pass
# キャンバスメインプログラム
if __name__ == '__main__':
  master=None
  f=Ved()
  b1=tk.Button(f.f,text=u'終了１',
    command=exit)
  b1.pack(side='left')
  b2=tk.Button(f.f,text=u'終了２',
    command=exit)
  b2.pack(side='left')
  b3=tk.Button(f.f,text=u'終了３',
    command=exit)
  b3.pack(side='left')
  c=CA()
  c.c.create_line(0,0,60,30,
    tags="t1")
  p=tk.PhotoImage(file='lena.gif')
  c.c.create_image(50,0,
    anchor=tk.NW,
    image=p, tag='abc')
  f.mainloop()
```

となり，結果を図 12 に示す．

図 12 ベクトル心電図処理画面 Ved1.py
メニューボタンのフレームは FM.py のフレームを横並びに指定，また，self を付けて公開している．

上記のプログラムはオリジナルの FM.py のフレームを横並びに設定することこと，フレーム名を公開することにより容易に作成できる．またそれを継承して新たな Ved クラスを作成した．どの様なメソッドが必要になるのか，不定であるため，とりあえず，pass を入力しておき，Python プログラム書法を守ることにした．もしフレームの枠が邪魔ということであれば，bd=0 とすることで隠すことができる．また，改良予定のファンクション機能のためにボタンを 3 個並べている．

キャンバスサイズの変更

いよいよ心電図画像ファイルの貼り付けになる．新規作成は編集画面の部品削除であるから問題はない．画像読み取りも読み取りダイアログと編集画面の貼り付けは問題がない．しかし，画像データがデフォルト設定より大きい場合にはスライダーが対応できなくなる．

PhotoImag を p とすると，入力画面の縦，横のサイズを求めるには，p.width()，p.height() で得られる． またキャンバスのサイズ変更は，c.c['width']=x，c.c['height']=y で変更できるから，前述プログラムで pass と記述した部分に新しく継承メソッドを書き直すと，

```
class Ved(FM):
# ファイルメニュー 継承
# メソッド
  def new_f(self):
    self.fname='noname.txt'
    for i in c.c.find_all() :
      c.c.delete(i)
  def open_f(self):
    for i in c.c.find_all() :
      c.c.delete(i)
    self.fname = D.askopenfilename(
      event=None)
    self.p=Itk.PhotoImage(
      file=self.fname)
    c.c.create_image(0,0,
      anchor=tk.NW,
      image=self.p, tags='latest')
    #画像サイズ
    x=self.p.width()
    y=self.p.height()
    #キャンバスサイズ変更
    c.c['width']=x
    c.c['height']=y
```

となる．

画像トリミング

終了 1 ボタンの表示ををトリムに変更して，トリミング枠を表示させる．マウス左ボタンのドラグでトリミング枠が確定し，フラッシュさせると，見やすくなる．ドリミングの実行は終了 2 ボタンで行い，自動的にトリミングを受けた画像サイズに変更させる．

この作業は伸縮自在の四角形ラバーアングル作成と呼ばれている Tcl/Tk のテクニックの一つで，まずは独立したプログラムを紹介し，次に Ved2.py に導入する．

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# 四角形を作る.
def r_create(event):
  global sx,sy
  c.create_rectangle(event.x,
    event.y,event.x,event.y,
    outline='black',width=2,
    tags='rubbershape')
  sx=event.x; sy=event.y
# 四角形をドラグする.
def r_drag(event):
  global sx,sy,x2,y2
  data=c.coords('rubbershape')
  x1,y1,x2,y2=data
  x2=event.x; y2=event.y
  if (x2>sx) and (y2>sy):
    c.coords('rubbershape',sx,y1,
      x2,y2)
  else:
    c.coords('rubbershape',x2,y2,
      sx,sy)
# 四角形を消す
def r_end(event):
  global sx,sy,x2,y2
  print sx,sy,x2,y2
  c.delete('rubbershape')
# メインプログラム
c = tk.Canvas(width=300, height=200)
c.pack()
c.bind('<Button-1>', r_create)
c.bind('<B1-Motion>', r_drag)
c.bind('<ButtonRelease-1>', r_end)
c.mainloop()
```

プログラム解説

始めの 3 行は定型通り．バインド関数 r_create は左ボタンを押した時点で線幅 2 の四角形を作成，タグ名を'rubbershape' とする．バインド

関数 r_drag は'rubbershape' の左上端，右下端の X-Y 座標を得，ドラグしている間，その四角形の coords を変化させる．四角形が伸縮しているように見えるのはバインド割り込みで何度も書き直しているためである．バインド関数 r_end は左ボタンを放すことにより実行し，四角形の座標を端末表示し，'rubbershape' の四角形をキャンパスより削除する．本例は四角形がキャンバスに表示されるだけだから図示しない．読者はプログラムを作成して確かめてほしい．

ラバーアングルの作成法が理解できたところで Ved1.py のコピーを Ved2.py としてプログラム修整を行う．

new_f メソッドおよび，open_f メソッドは先ほど示した．トリミング開始ボタンは，左ボタンドラグで黒色ラバーアングルを表示させる．その関係するプログラムは全てメインプログラムのバインド領域に書き込む．したがって，Ved2.py のメインプログラムは，

File name:　Ved2.py (main)

```
# メインプログラム
if __name__ == '__main__':
  master=None
  f=Ved()
  # マウスバインド
  # 四角形を作る.
  def r_create(event):
    global sx,sy
  # find_withtag
    c.c.delete('rubbershape')
    c.c.create_rectangle(event.x,
      event.y,event.x, event.y,
      outline='black',
      width=2, tags='rubbershape')
    sx=event.x; sy=event.y
  # 四角形をドラグする.
  def r_drag(event):
    global sx,sy,x2,y2
    data=c.c.coords('rubbershape')
```

図 13 ラバーアングルでトリム画像を抽出 Ved2.py
File メニューの横のトリム開始で編集画面にラバーアングルを作成し，トリム終了でトップレベルで選択領域の画面が表示される．

```
    x1,y1,x2,y2=data
    x2=event.x; y2=event.y
    if (x2>sx) and (y2>sy):
      c.c.coords('rubbershape',
        sx,y1,x2,y2)
    else:
      c.c.coords('rubbershape',
        x2,y2,sx,sy)
  # トリミング開始
  def trim_on():
    c.c.bind('<Button-1>', r_create)
    c.c.bind('<B1-Motion>', r_drag)
  # トリミング終了
  def trim_off():
    im=I.open(f.fname)
    x1,y1,x2,y2=c.c.coords(
      'rubbershape')
    im2=im.crop((int(x1),int(y1),
      int(x2),int(y2)))
    im2.show()
  b1=tk.Button(f.f,text=u'トリム開
始',
    command=trim_on)
  b1.pack(side='left')
  b2=tk.Button(f.f,text=u'トリム終
了',
    command=trim_off)
  b2.pack(side='left')
  b3=tk.Button(f.f,text=u'終了３',
    command=exit)
  b3.pack(side='left')
  c=CA()
  f.mainloop()
```

となる．

トリミング終了ボタンはプログラムが正しく動作するかを確かめるために，一次的に PIL の show() 命令を使用した．仕上りを図 13 に示す．

プログラム解説

前に述べたラバーアングルの雛型プログラムをトリム開始ボタン command にコピーし，c.create が c.c.create になっただけで，説明の必要はないだろう．

トリミング領域は改めて coords で得ることにした．その理由は右から左にドラグしたときの四角形の指定が逆になっているため，再度確かなデータを得ることにした．また，図形ファイルの表示が (0,0) から始まっているから，PIL の im 波形と同じ座標になり，容易に crop ができる．

トリミングが成功し，show() 命令で画像が表示できれば，トリム終了命令の第 2 段階に移る．crop で作成したトリム画像を'tmp.jpg' に保存し，再度キャンバスに表示する．PIL でのイメージファイル呼び出しが f.fname で，tmp.gif のファイル呼び出しが fname のままになっている理由は前者はクラスで self.f と指定したファイル名であり，後者は trim_off 関数の局所変数を使用して区別する．また，イメージ画像 p はグローバル宣言しなければ表示されない．常に画像表示ポインターは self またはグローバル宣言が必要である．trim_off 関数の完成したプログラムを以下に示す．

```
# トリミング終了
def trim_off():
  global p
  global maxx,maxy
  x1,y1,x2,y2=c.c.coords(
    'rubbershape')
```

図 14 トリミング後の心電図解析画面
トリミング終了後直接キャンバスに張り付けている．

```
im=I.open(f.fname)
im2=im.crop((int(x1),int(y1),
  int(x2),int(y2)))
im3=im2.convert('RGB')
im3.save('tmp.gif')
for i in c.c.find_all() :
  c.c.delete(i)
fname='tmp.gif'
p=Itk.PhotoImage(file=fname)
c.c.create_image(0, 0,
  anchor=tk.NW, image=p)
maxx=p.width()
maxy=p.height()
c.c['width']=maxx
c.c['height']=maxy
c.c.bind('<Button-1>','-',
  r_create)
c.c.bind('<B1-Motion>','-',
  r_drag)
```

となる．トリミング画像は im2，それを'RGB' にモード変換し，im3 を tmp.jpg ファイルに保存する．再度キャンバス内をクリアーし，保存ファイルを読み込む．これらの作業は Tkinter だけでは処理できず，PIL の助けを借りている．トリミング後は，最大画像で表示できるようにキャンバスの縦，横のサイズを変更している．またこの縦，横の値はカーソルや，スキャン表示などに利用するため，グローバル変数 maxx，maxy とした．最後にバインドイベントをクリアーする．同様の変更は open_f でも行った．そのようにしてできあがった GUI を図 14 に示す．

心電図波形の基線設定

図 15 に示す心電図は上段より第 I 誘導，第 II 誘導，第 III 誘導であり，それらの基線 (X 軸) を指定しなければ，数値としての波形座標が計算できない．そこで，本例では目視により 3 チャネルの基線を設定することにして，カーソル表示，移動法，位置決定のプログラムミングを考える．Ved2.py を Ved3.py にコピーしてプログラムを改良する．

メインプログラムの終了 3 ボタンの位置に追加のカーソル表示ボタン，また，基線 1，基線 II，基線 III の位置決定ボタンを作成する．

```
# カーソル移動
def cur1_move(event):
  global old_cur1y
  c.c.move('cur1',0,
    event.y-old_cur1y)
  old_cur1y=event.y
# カーソル作成
def base():
  global old_cur1y
  global maxx
  old_cur1y=5
  c.c.create_rectangle(0,0,10,10,
    fill='green', tag='cur1')
  c.c.create_line(0,5,maxx,5,
    fill='green',
      width=2, tag='cur1')
  c.c.bind('<Button-1>',
    cur1_move)
  c.c.bind('<B1-Motion>',
    cur1_move)
# 基線記録
def baseI():
  global old_cur1y,baseI_y
  baseI_y=old_cur1y
def baseII():
  global old_cur1y,baseII_y
  baseII_y=old_cur1y
def baseIII():
  global old_cur1y,baseIII_y
  baseIII_y=old_cur1y
# キャンバスメインプログラム
b1=tk.Button(f.f,text=u'トリム開始',
```

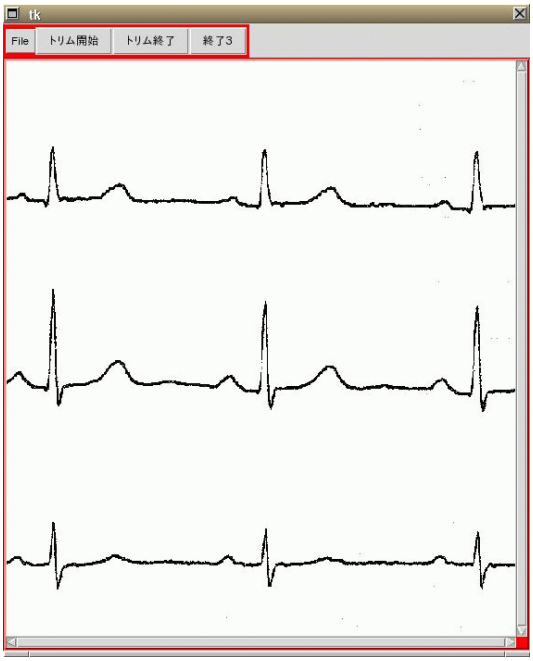


図 15 心電図解析画面
スキャナーで取り込んだ心電図チャート．
上段より I，II，III 誘導を示す．

```
  command=trim_on)
b1.pack(side='left')
b2=tk.Button(f.f,text=u'トリム終了',
  command=trim_off)
b2.pack(side='left')
b3=tk.Button(f.f,text=u'カーソル',
  command=base)
b3.pack(side='left')
b3=tk.Button(f.f,text=u'基線 I',
  command=baseI)
b3.pack(side='left')
b3=tk.Button(f.f,text=u'基線 II',
  command=baseII)
b3.pack(side='left')
b3=tk.Button(f.f,text=u'基線 III',
  command=baseIII)
b3.pack(side='left')
b3=tk.Button(f.f,text=u'終了３',
  command=exit)
b3.pack(side='left')
c=CA()
f.mainloop()
```

プログラム解説

一気にファンクションボタンが増えたが複雑なことはない．メインプログラムから説明すると，「カーソル」ボタンは base を呼び，カーソルを表示する.「基線 I」は baseI を呼び，第 I 誘導の基線の Y 座標を記憶，以下「基線 III」まで同じである．ボタンウィジェット名が全て b3 となっているが，個別ボタンとして引用しなければ問題ない．

カーソルを作成する base 関数は，心電図チャート幅 maxx をグローバルで呼ぶ．また始めのカーソル Y 座標 old_cur1y を 5 に指定する．次いで (0,0,10,10) の位置に緑の四角形を作成，タグを'cur1' とする．また，(0,5) から，(maxx,5) まで緑の幅 2 の直線を作成，タグを同じ'cur1' として四角形とグループ化した．そして，マウスボタン 1 のドラグでカーソルの移動関数 cur1_move を呼び出している．マウスバインドは常に最終命令に対応するから，重複して指定しても問題ないことはバスの乗客窓コントローラで実証すみである．

実際にカーソルを移動させる関数は cur1_move 関数である．move 命令は dx,dy，すなわち x,y 方向の変化分だけを移動させるために初期値が必要である．そのため old_cur1y をグローバル変数にして，Y 軸の初期値を得た．x 軸方向にカーソルは移動しないので，dx 方向の変化は 0 である．dy は現在のカーソル位置，event.y との差を指定し，移動タグは'cur1' と指定しているから，四角形も直線も同じ量だけ移動する．移動後に，old_cur1y の値を現在値に更新している．I，II，III 誘導の各基線の Y 値は baseI，baseII，baseIII 関数で求める．単に，old_cur1y の値を記録しているだけだから簡単である．新しく追加したボタンとトリミングされた心電図波形を図 16 に示す．

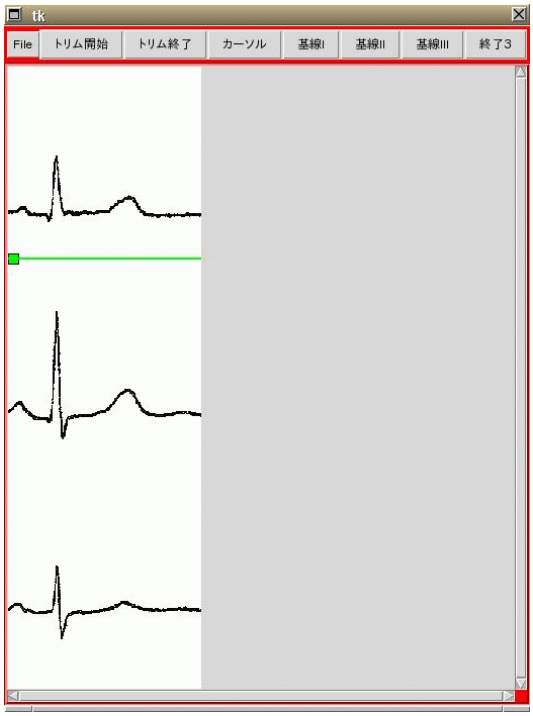


図 16 心電図解析画面 Ved3.py
カーソル表示，基線 I，基線 II，基線 III 設定ボタンを追加．心電図は 1 心拍にトリミングを行った．上段より I，II，III 誘導を示す．

心電図波形の座標抽出

記録用紙に描かれた心電図は一見画像が美しく見えていても，詳細に分析すると汚れやが数多く有り，そのグラフから X-Y 座標を自動的に読み取ることは難しい．

筆者の行った方法はカラーのチャートを濃淡のある白黒チャートに変換し，灰色になったグラフの目盛を灰色の閾値を移動することにより，出来るだけ灰色目盛を白色に変換し，心電図波形のみを浮き立つように調整した．使用したソフトは xv と gimp である．

Tkinter の PhotoImage は get 命令でピクセル単位で色を取得し，put 命令で色の指定ができる．今回は PIL の PhotoImage を利用しているため，画像データを p とし，その X-Y 座標の色は，

```
data=p._PhotoImage__photo.get(x,y)
```

で得られ，data には RGB の文字列リストが記録される．これを利用して黒点の数を調べる．

次にトレースの方法であるが，X 軸方向にトレースすることにして，Y 軸方向には 3 本の心電図波形が存在する．単純に Y 軸を 3 分割して，それぞれの分画でこの Y 軸の高さの間であればこの誘導の黒点だと if 文で検出することにした．そして，それぞれの分画で，黒点に相当するピクセルの平均値を計算し，その平均値を心電図の Y 座標と判断する．このようにして作成したのが trace 関数で，以下に示す

```
# トレース
def trace():
  global AA,BB,CC
  # トップレベル作成
  t1=tk.Toplevel()
  t1.title('Toplevel')
  w = tk.Canvas(t1,width=maxx,
    height=maxy,bg='white')
  w.pack()
  # チャネル下限設定
  sep=maxy/3
  sep2=2*sep
  # キャンバスにカーソル作成
  c.c.create_line(1,0,1,maxy,
    fill='green',tag='cursor')
  curx=0
  AA=[]; BB=[]; CC=[]
  for i in range(maxx):
    AA.append(0); BB.append(0)
    CC.append(0)
  # x 軸スキャン
  for i in range(maxx):
    # カーソル移動
    curdx=i-curx
    c.c.move('cursor',curdx,0)
    curx=i
    c.c.update()
    # 黒点 y 座標検出
    A=[]; B=[]; C=[]
    c.c.coords('cursor',i,0,i,maxy)
    for j in range(maxy):
      data=p._PhotoImage__photo.get(
        i, j)
      r=string.split(data,' ')
      x=int(r[0])
      if x< 50:
        if j<sep :
          A.append(j)
        elif j<sep2 :
          B.append(j)
        else:
          C.append(j)
      # AA チャネルの平均
      An=len(A)
      sum=0
      if An!=0 :
        for k in range(An):
          sum=sum+A[k]
        AA[i]=sum/An
      # BB チャネルの平均
      Bn=len(B)
      sum=0
      if Bn!=0 :
        for k in range(Bn):
          sum=sum+B[k]
        BB[i]=sum/Bn
      # CC チャネルの平均
      Cn=len(C)
      sum=0
      if Cn!=0 :
        for k in range(Cn):
          sum=sum+C[k]
        CC[i]=sum/Cn
    if i>0:
      w.create_line(i-1,AA[i-1],i,
        AA[i], width=2)
      w.create_line(i-1,BB[i-1],i,
        BB[i], width=2)
      w.create_line(i-1,CC[i-1],i,
        CC[i], width=2)
```

プログラム解説：

トレース結果を視覚に訴えるためにトップレベルのウィンドウを作成し，そこに結果を表示することにした．また，キャンバスには Y 軸方向のカーソルを表示し，処理中の X 座標を示している．AA，BB，CC の各リストは I，II，III 誘導の Y 座標が記録される．このデータは後の解

析に利用されるためグローバル宣言をしておく．Y 軸分画値は sep と sep2 とし，とりあえず AA，BB,CC の各データ領域には 0 を記入した．

X 軸のスキャンであるが，i を変数として最大値 maxx までの for 文でループを作る．カーソル移動を行い，黒点の Y 座標を収納する Y 軸のスキャンは j を変数として最大値 maxy までの for 文でループを作る．リスト変数 A，B，C を作成する．もしピクセルの値が 50 以下であれば黒点とみなして，Y 座標の高さに従って，A，B，C のどれかに Y 値を記録する．変数 j のループが終了した所で，A，B，C の平均値を計算し，AA，BB，CC の Y 座標として格納し，トップレベルのキャンバスに直線で検出した心電図座標を描く．if 文はもったいないように思うが，そのための処理時間は 1msec 以下であることと，0 以下のリスト値は無いために付けた．およそ 1 秒でグラフは完成する．

trace 関数のファンクションボタンを新しく作成する必要がある．フレーム名を f1 として次の段を作成し，その中にトレースボタンを収納した．メインプログラムの追加文は以下のようになる．また string モジュールを利用するためにプログラムの始めに import 文を追加した．

```
b3=tk.Button(f.f,text=u’終了３’,
  command=exit)
b3.pack(side=’left’)
f1=tk.Frame(master,bg=’red’,bd=3)
f1.pack(anchor=’nw’)
b3=tk.Button(f1,text=u’トレース’,
  command=trace)
b3.pack(side=’left’)
c=CA()
f.mainloop()
```

結果を図 17 に示す．新しいフレームとトレースボタンが追加され，III 誘導の基線が表示され，トリミング画像の横にトップレベルでの心電図折れ線座標が表示されている．よく観察すると心電図のピークが幾分低くなっているが，黒点の平均値を採用した影響が表れている．

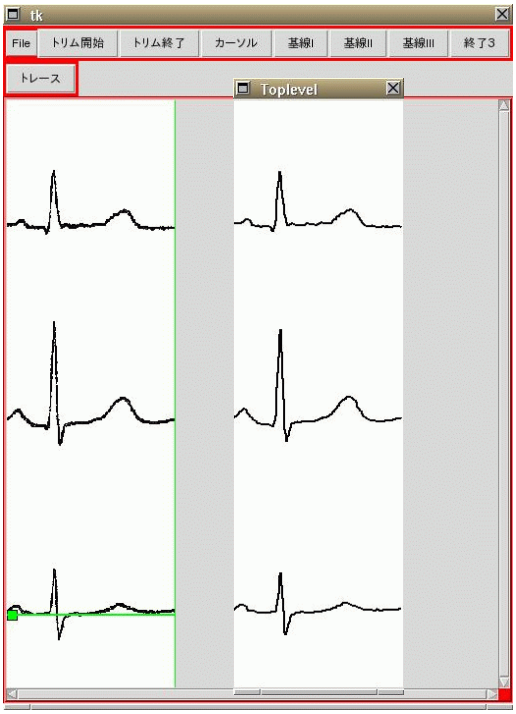


図 17 心電図波形のトレース
トレースボタン用に新しくフレームを作成．トレース結果を新しくトップレベルに表示している．

画像イメージの色操作

画像ピクセルの色は put，get メソッドで容易に操作ができる．

```
from Tkinter import *
c = Canvas(width=128, height=128)
p=PhotoImage(file=’lena.gif’)
c.create_image(0,0,anchor=NW,image=p)
c.pack()
# 赤の四角を作る．
p.put("#ff0000", to=(0,0,30,30))
# (29,0) の色を見る．
p.get(29,0)
u’255 0 0’  <- が返る．
p.get(30,0) <- が返る．
u’201 128 102’
```

プログラム解説：

キャンバス c を作成し，gif 形式画像を PhotoImage を作り，それをキャンバスに貼り付ける．次いで，座標 (0,0) から (29,29) までのピクセルを赤色に変換する．これは画像データの変換であって，キャンバスに直接色を指定しているのではない．しかし画像データが変化するとキャンバスは自動的に新しい画像データに更新する．

色データの指定は RGB の順で，16 進数字を#を付けて表示する．RGB データを独自の r，g，b などの変数にする場合は，

```
color='#%02x%02x%02x' % (r,g,b)
p.put(color, to=(0,0,30,30))
```

とすればよい．また r，g，b の数値を得るにはカラーコードがユニコード文字列で返るために

```
import string
data=p.get(29,0)
r=string.split(data,' ')
int(r[0])
255    <- と返る.
```

とすれば，後の計算ができるようになる．結果を図 18 に示す．

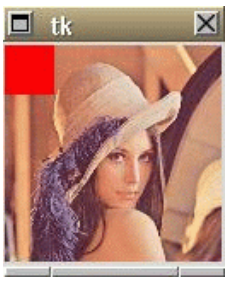


図 18 get, put メソッド
PhotoImage は picel 単位で色情報を指定し，また読み取りができる．変更するのは画像データであって，キャンバスデータではない．キャンパス画像は自動的に更新される．

上記の例では画像が GIF ファイルであるために Tkinter 付属の PhotoImage メソッドを使用したが，PIL の場合は，

```
data=p._PhotoImage__photo.get(x,y)
r=string.split(data,' ')
```

とすればよい．繰り返しになるが，これらの色データ処理は画像データに対して行われるものであって，キャンバスに表示されている画像ではない．

第 II 誘導心電図波形の合成

心電図の話しは Python プログラミングに直接関係するものではないが，コンピュータを利用して科学すると言う意味で話題性があると思う．

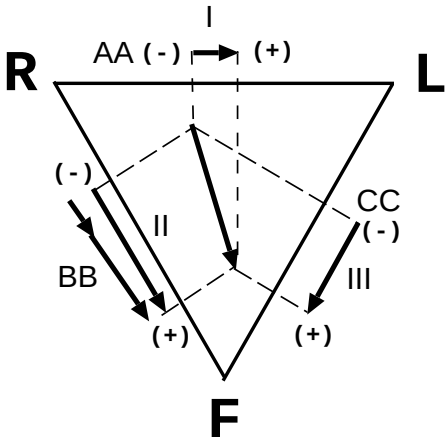


図 19 アイントーベンの正三角形
右手，左手，左足に電極を貼り，体表心電図を計測すると，それらの電位は正三角形のベクトルに投影できる．

心電図を発見したアイントーベンは右手，左手，左足に電極を装着し，各電極間の電位計測を行ったところ，図 19 に示すように正三角形に投影できることを発見した．本来なら 3 点の計測であるからその起電力となる心電図は $f(x, y, z)$ となるはずである．ところが 2 次平面の正三角形に投影できたことはヒトの体形に運が良かったとしか言いようのない出き事である．従って体表心

電図の電気軸という概念が生まれた，また，このことから第 II 誘導は図に示すように第 I 誘導と第 III 誘導の絶対値の和で表される．したがって，この第 II 誘導心電図波形を先ほどトレースした心電図座標をもちいて再合成しようというのが次に示す関数 second である．

Ved3.py をコピーして Ved4.py を作成する．新たに関数 second を作動するボタンの作成，second も結果をトップレベルで表示することにして，第 I 誘導，第 II 誘導の絶対値は AA，CC より基線の Y 座標値を引けばよい．プログラムは，

```
# 第 II 誘導合成
# 第 II 誘導は第 I 誘導と
# 第 III 誘導の絶対値の和
def second():
  global AA,BB,CC,SdII
  global dI,dII,dIII,SdII
  global baseI_y,baseII_y,baseIII_y
  # キャンバスにカーソル作成
  c.c.delete('cursor')
  c.c.create_line(1,0,1,maxy,
    fill='green',tag='cursor')
  curx=0
  t2=tk.Toplevel()
  t2.title('II Image')
  w2 = tk.Canvas(t2,width=250,
    height=250,bg='white')
  w2.pack()
  dI=[]; dII=[]; dIII=[]; SdII=[]
  for i in range(maxx):
    # カーソル移動
    curdx=i-curx
    c.c.move('cursor',curdx,0)
    curx=i
    c.after(1)
    c.c.update()
    dI.append(baseI_y-AA[i])
    dII.append(baseI_y-BB[i])
    dIII.append(baseI_y-CC[i])
    SdII.append(-(dI[i]+dIII[i])
      -250)
    if i>0:
      w2.create_line(i-1,SdII[i-1],
        i,SdII[i],width=2)
```

第 II 誘導合成ボタンは，c=CA() の前に，

```
  b3=tk.Button(f1,text=u'II 誘導合
成',
    command=second)
  b3.pack(side='left')
  c=CA()
  f.mainloop()
```

とすればよい．結果を図 20 に示す．第 II 誘導にかなり近似した波形が算出できている．

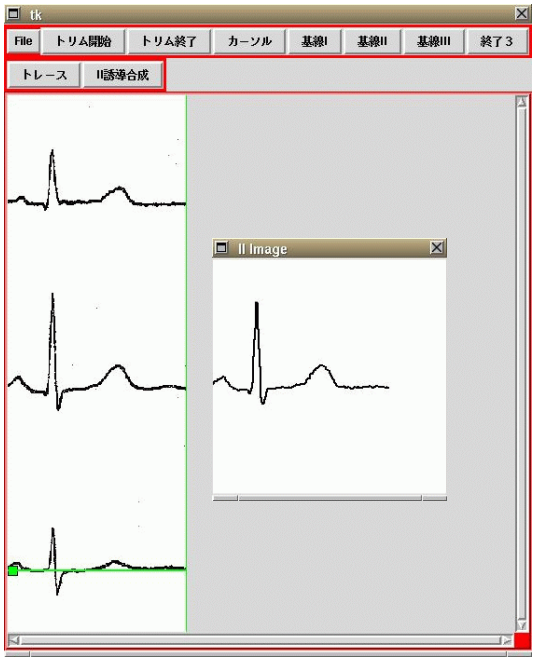


図 20 第 II 誘導の合成
第 I および第 III 誘導の絶対値を加算することにより第 II 誘導の波形が得られる．

本プログラムの最終段階であるベクトル心電図の表示に進む．Ved4.py をコピーし Ved5.py にする．ベクトル心電図は 3 波形のうち 2 波形の X，Y 方向のベクトルを抽出すれば良いが，その頂点のリサージュを描くことにする．前項で計算した絶対値 dI および dII をもちいて，以下に示す関数 axis を作成した．

```
# 電気軸
def axis():
  global dI,dII,dIII,SdII
  global dx,dy
  # キャンバスにカーソル作成
  c.c.delete('cursor')
  c.c.create_line(1,0,1,maxy,
    fill='green',tag='cursor')
  curx=0
  t3=tk.Toplevel()
  t3.title('Axis')
  w3 = tk.Canvas(t3,width=250,
    height=250,bg='white')
  w3.pack()
  dx=[]; dy=[]
  for i in range(maxx):
    # カーソル移動
    curdx=i-curx
    c.c.move('cursor',curdx,0)
    curx=i
    c.after(100)
    c.c.update()
    dx.append((dI[i]+0.5*dII[i])
      +150)
    dy.append((dII[i]*0.866)+250)
    if i>0:
      w3.create_line(dx[i-1],
        dy[i-1], dx[i],dy[i],
        width=2)
```

電気軸算出ボタンは以下のように追加する．

```
b3=tk.Button(f1,text=u'電気軸',
  command=axis)
b3.pack(side='left')
c=CA()
f.mainloop()
```

結果を図 21 に示す．以上が画像波形から X-Y 座標を取得する手段である．

電気軸リサージュは見にくいように感じるが，実は p 波のリサージュ，QRS のリサージュ，T 波のリサージュが重なって表示されている．CPU の計算では一瞬にして表示が終了するために 0.1 秒毎にステップが進むように after 命令を組み入れた．このリサージュを詳しく分析するために，p，QRS，T で表示の色を変更するとか，スケールウィジェットを組み込んで，時間経過を自由に調節表示するなど，幾つか工夫，発展の余地がある．これらは難しい問題ではないので，ここまでつき合って頂いた読者の宿題としておこう．

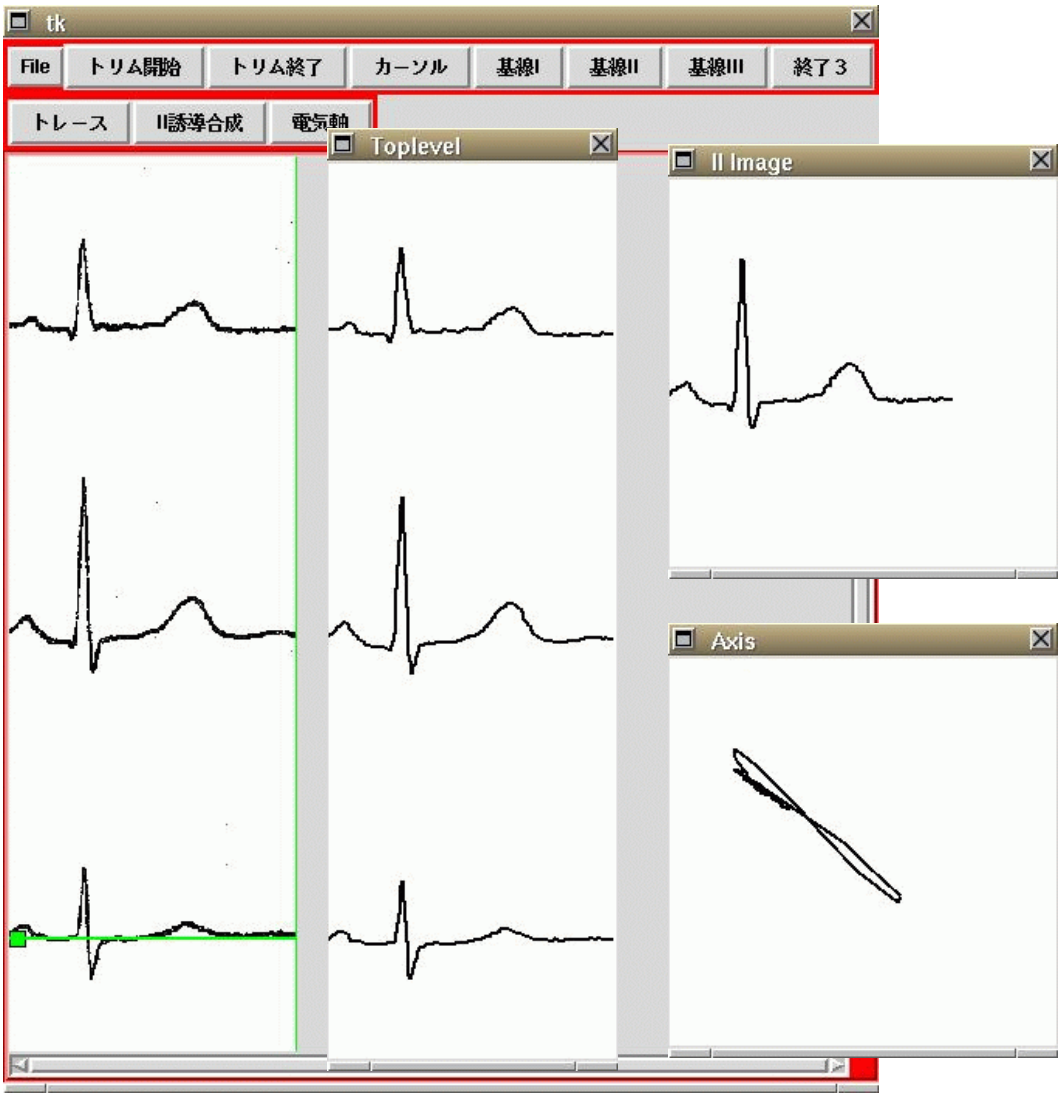


図 21 トレース，第 II 誘導の合成，電気軸リサージュ
心電図計測チャートの波形計測より，ベクトル心電図まで合成した.

補足：bind と event

Tkinter の GUI は画像を表示し，プログラムの最後に mainloop() を動かして，何らかのイベント待ちの状態でいる．例えば Entry ウィジェットなら名前を入力する．そして改行キー (Return) などを入力すると，次の処理プログラムが開始する．そのためにはウィジェットに bind 命令を行って，処理プログラムを設定する必要がある．また，キー入力に対して，改行コードだけがイベントとなるように設定しなければならない．以下に 1 行入力ウィジェットの例を示す．

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
from Tkinter import *
def cmd(event): # call back
   print e1.get()

root=Tk()
e1=Entry(root,width=12)
e1.bind("<Return>",cmd)  # bind
e1.pack()
e1.mainloop()
```

プログラム解説

GUI の root ウィンドウを作成し，そこに Entry ウィジェット名 e1 を貼り付ける．e1 には <Return> イベントで動作する cmd 関数を設定する．cmd 関数は引数 event を受け取り，get メソッドで内容を端末出力する．上記プログラムが bind と event の基本形である．

ウィジェットに設定するイベントは一つに限ったことではなく，例えば <Control-q> の行の下に，

```
e1.bind("<Control-q>",exit)
```

と記述すると，もし入力する文字が CTRL+q であればプログラムが終了できるように，複数個設定してもよい．使用できる各種イベントパラメータは次ページに一覧表示する．

仮想イベント

イベントはキーやマウスを動作しなくても仮想イベントとしてソフト的に発生できる．つまりソフト的にマウスポインターを移動したり，ボタンクリックが可能である．それにはバインドパラメータを <<Return>> のように不等号記号を 2 重に記述する．例えば

```
#! /usr/bin/env python
# -*- coding: euc-jp -*-
from Tkinter import *
def mv():
  e1.event_generate(
    "<<B1-Motion>>",x=0,y=0,warp=1)
root=Tk()
e1=Canvas(root,width=100, height=80)
e1.after(3000,mv)
e1.pack()
e1.mainloop()
```

として，マウスをキャンバスの任意の位置に置くと，3 秒後にマウスカーソルは (0, 0) に移動する．

その他の event 命令で，event_add，event_delete，event_info がある．event_add は仮想イベントに物理イベントを張り付ける．つまり，始めに仮想バインド命令を設定し，次いで物理イベントを event_add で設定する．例えば Entry ウィジェットで示すと，

```
root=Tk()
e1=Entry(root,width=12)
e1.bind("<Return>",cmd)  # bind
e1.bind("<<Control-q>>",exit)
e1.event_add("<<Control-q>>",
  "<Control-q>")
e1.pack()
e1.mainloop()
```

としておくと，本来仮想イベントの CTRL+q しか受け付けないソフトがキー入力の CTRL+q でプログラムは終了する．逆に，物理イベントに仮想イベントを貼り付けることはできない．

イベント
マウス関係：

<Button-1>	左マウスボタンが押された時に発生.
<Button-2>	真ん中マウスボタンが押された時に発生.
<Button-3>	右マウスボタンが押された時に発生.
<B1-Motion>	左ボタンを押しながらマウスを動かすと発生.
<B2-Motion>	真ん中ボタンを押しながらマウスを動かすと発生.
<B3-Motion>	右ボタンを押しながらマウスを動かすと発生.
<ButtonRelease-1>	左ボタンを押してそのボタンを放した時に発生.
<ButtonRelease-2>	真ん中ボタンを押してそのボタンを放した時に発生.
<ButtonRelease-3>	右ボタンを押してそのボタンを放した時に発生.
<Double-Button-1>	左ボタンをダブルクリックした時に発生.
<Double-Button-2>	真ん中ボタンをダブルクリックした時に発生.
<Double-Button-3>	右ボタンをダブルクリックした時に発生.
<Triple-Button-1>	左ボタンをトリプルクリックした時に発生.
<Triple-Button-2>	真ん中ボタンをトリプルクリックした時に発生.
<Triple-Button-3>	右ボタンをトリプルクリックした時に発生.
<Enter>	マウスポインターが widget に入った時に発生.
<Leave>	マウスポインターが widget から出た時に発生.

キーボード関係：

<FocusIn>	キーボードフォーカスが wiget に移った時に発生.
<FocusOut>	キーボードフォーカスを wiget が失った時に発生.
<Return>	Enter キーが押された時に発生.
<Cancel>	Break キーが押された時に発生.
<BackSpace>	BackSpace キーが押された時に発生.
<Tab>	Tab キーが押された時に発生.
<Shift_L>	左 Shift キーが押された時に発生.
<Shift_R>	右 Shift キーが押された時に発生.
<Control_L>	左 Control キーが押された時に発生.
<Control_R>	右 Control キーが押された時に発生.
<Control-q>	Control+q キーが押された時に発生.
<Alt_L>	左 Alt キーが押された時に発生.
<Alt_R>	右 Alt キーが押された時に発生.
<Pause>	Pause キーが押された時に発生.
<Caps_Lock>	Caps Lock キーが押された時に発生.
<Escape>	Escape,Esc キーが押された時に発生.
<Prior>	Page Up キーが押された時に発生.
<Next>	Page Down キーが押された時に発生.
<End>	End キーが押された時に発生.
<Home>	Home キーが押された時に発生.
<Left>	Left, 左矢印 キーが押された時に発生.
<Up>	Up, 上矢印 キーが押された時に発生.
<Right>	Right, 右矢印 キーが押された時に発生.
<Down>	Down, 下矢印 キーが押された時に発生.
<Print>	Print キーが押された時に発生.
< Insert >	Insert キーが押された時に発生.
< Delete >	Delete キーが押された時に発生.
<F1> 〜 <F12>	<F1> 〜 <F12> の各キーが押された時に発生.

<Num_Lock>	Num Lock キーが押された時に発生.
<Scroll_Lock>	Scroll Lock キーが押された時に発生.
<space>	space キーが押された時に発生.
<Win_L>	左 windows キーが押された時に発生.
<Win_R>	右 windows キーが押された時に発生.
<App>	アプリケーション キーが押された時に発生.
<Key>	キーボードのなんかのボタンが押されたら発生.
1	キーボードの 1 のボタンが押されたら発生.
<1>	マウスの左のボタンが押されたら発生.
3	キーボードの 3 のボタンが押されたら発生.
<1>	マウスの右のボタンが押されたら発生.
<Shift-Up>	Shift を押しながら上矢印が押された時に発生.
<Configure>	widget のサイズを変更した時に発生.

プロトコルイベント

WM_DELETE_WINDOW	ウィンドウが閉じられる時に発生.

イベントオブジェクト

event.widget	イベントを受け取った widget のインスタンス.
event.x	現在のマウスポインターの x 位置．(ピクセル)
event.y	現在のマウスポインターの y 位置．(ピクセル)
event.x_root	現在のマウスポインターの 画面上の x 位置．(ピクセル)
event.y_root	現在のマウスポインターの 画面上の y 位置．(ピクセル)
event.char	キーボードイベント時のみ，キャラクタを文字で返す.
event.keysym	キーボードイベント時のみ，そのキーのシンボル.
event.keycode	キーボードイベント時のみ，このキーのキーコード.
event.num	マウスボタンイベント時のみ，押されたマウスボタンのボタン番号.
event.width	Configure イベント時のみ，widget の幅．(ピクセル)
event.height	Configure イベント時のみ，widget の高さ．(ピクセル)
event.type	イベントのタイプ.

シミューレータ

シミューレータはモデルに種々の条件を設定し，それがどのように変化するかを数値計算し，グラフに結果を表示する道具である．種々のシミュレーションが想定されるが，とりあえず三角関数を X-Y 座標で表示するまでを目的にする．

シミュレーションの簡単な例として，$y = sin(x)$ を考える．この場合，振幅は± 1 であるが，それを 100 倍にし，y 軸の 0 値を 100 に移動すると，キャンバス座標で 0 から 200 までを表示できればよい．また，x 軸についても同様に 100 倍に拡大すれば，3.14 を 314 で表示できる．そのような考えで一応キャンバス絶対値の範囲を x 方向については 0〜500，y 方向については 0〜300 とした．x 軸，y 軸の表示は始めからシミュレーションに即したものでなくてもよく，絶対値表示の方がデバグに都合がよい．また，正確な目盛りを表示しておけば，最終的に tgif やイラストレータなどの図形エディターを使用して美しい座標軸に仕上げることができる．

シミュレータソフトの作成は先に紹介したキャンバスソフト CA.py の機能を改造し，ソフト名を SIM.py とする．プログラムは CA クラスを利用して、追加のメソッドを作成するだけであるから，新しいクラス CB を宣言するだけでよい．つまり，SIM.py は

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
from CA import *
class CB(CA):
```

の下に必要なメソッドを追加することになる．クラス CB の def __init__(self):は継承で CA が代行してくれる．

表示方法を直線表示とすると，p をペンの色番号として，mv(p,x,y) メソッドで，指定した色のペンが座標 (x,y) に移動し，draw(p,x,y) メソッドで，mv で移動した位置から新しく指定した座標まで直線を描く．直線の間隔を細かくすればそれだけ任意の曲線図形に近づくことになる．昔の X-Y プロッターはこのような命令で図形を描いていた．

まずは color メソッドから始める．

```
# ペンの色選択
  def color(self, p):
    c=['black','red','blue']
    return c[p]
```

と数値の引数で文字列を返している．色は何色でも構わないが，とりあえず 3 色にした．0 なら黒，1 なら赤，2 なら青になる．

mv メソッドは，

```
# 直線表示 開始点を指定
# 呼び出し法： mv ペン色
# x 開始点 y 開始点
  def mv(self, p, x, y):
    global xstart, ystart
    xstart[p]=x; ystart[p]=y
```

となる．p はペン番号で，xstart，ystart はそれぞれのペンの表示開始位置を記憶するための変数領域であり，グローバル変数にしておく．

直線表示 draw は，

```
# 直線表示 終点を指定し、描写する
# 呼び出し法： draw ペン色
# x 終点 y 終点
  def draw(self, p, x, y):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    x1=xorigin+xstart[p]
    y1=yspan-(yorigin+ystart[p])
    x2=xorigin+x
    y2=yspan-(yorigin+y)
    self.c.create_line(x1,y1,x2,y2,
      fill=self.color(p))
    xstart[p]=x; ystart[p]=y
```

となる．xspan と yspan はキャンバス全体の横幅と立て幅を示す．xorigin と yorigin はシミュレーション座標の原点を示す．したがって，キャンバス上の実際の開始位置 x1 は xorigin+xstart[p] となる．y 方向についても同様であるが，キャンバス上の原点は左上端であるために yspan からシミュレーションの値を

引くことにより実際の y 値を得ている．直線表示命令は self.c.create_line を使用している．作図が終ったところで，(x,y) の表示終了座標を xstart[p]，ystart[p] に代入する．ペンの色は fill オプションで指定するが，色オプションは color メソッドを使用する．上記のメソッドで global と self の使い方に注意してほしい．

x 軸，y 軸の表示はチックと呼ばれる目盛の作成から始める．

```
# x チック
  def xtic (self, p, x, y):
    self.mv (p, x, y)
    self.draw(p, x-5, y)
```

```
# y チック
  def ytic (self, p, x, y):
    self.mv (p, x, y)
    self.draw(p, x, y-5)
```

となる．mv や draw メソッドをすでに定義しているから作成は容易である．また，引数に「x-5」や「y-5」などの式を記述してもよい．

x 軸は，

```
# x 軸作成
  def xaxis(self, p):
    self.mv(p, 0, 0)
    self.draw(p, 400, 0)
    x=0
    while x<=400 :
      self.ytic(p, x, 0)
      self.drawstring(p, x, -15, x)
      x=x+50
```

となる．始めに原点から x 軸最大値の 400 まで直線を描く．次いで，原点から y 方向にチックを描く．そのチックに値を drawstring メソッドで記入する．間隔を 50 にして 400 に至るまでの繰り返しを行う．

y 軸についても同様に，

```
# y 軸作成
  def yaxis(self, p):
    self.mv(p, 0, 0)
    self.draw(p, 0, 250)
    y=0
    while y<=250 :
      self.xtic(p, 0, y)
      self.drawstring(p, -20, y, y)
      y=y+50
```

となる．

drawstring メソッドの呼び出しで，文字列の引数部位で単に x，または y と記述しているが，自動的に数値か，文字列かを drawstring メソッド側で判別して動作する．これも python の特徴の一つである．

```
# 文字列を表示
# 呼び出し法： drawstring
# ペン色 x y "文字列"
  def drawstring(self, p, x, y,
    string):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    x1=xorigin+x
    y1=yspan-(yorigin+y)
    self.c.create_text(x1, y1,
      text=string,fill=self.color(p))
```

となる．

最後にシミューレタ初期化メソッドである siminit を以下に示す．

```
  # シミューレター基本メソッド
  def siminit(self, a, b, c, d):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    xorigin=a
    yorigin=b
    xstart=[0,0,0]
    ystart=[0,0,0]
    xspan=c
    yspan=d

    self.config('width',c)
    self.config('height',d)
    self.config('scrollregion',
      (0,0,c,d))
```

となる．xorigin，yorigin などのグローバル

変数はメインプログラムで記述すれば，siminit で定義しなくてもメソッド内で引用はできるが，プログラムの判りやすさや変数を忘れないために初期設定で宣言している．xstart や ystart はリスト変数のペンの初期座標であるから，初期値 0 でリストを定義している．キャンバスのサイズを示す width や height は config メソッドで記述しているから記述方に注意してほしい．同様に，scrollregion もタプル形式で設定できる．この設定により，元の tk.Canvas のパラメータは全て変更できる．

いよいよメイン関数に入る．座標，折れ線と文字列だけの簡単な例を以下に示す．

```
# メインプログラム
if __name__ == '__main__':
  c=CB()
  b1=tk.Button(c.c,text=u'終了',
    bg='green',command=exit)
  c.c.create_window(400,10,
    window=b1,anchor="nw")
  c.siminit(50,30,500,300)
  c.xaxis(0)
  c.yaxis(0)
  c.mv(1,10,10)
  c.draw(1,100,100)
  c.draw(1,120,10)
  c.drawstring(2,120,10,u'テスト')
  c.mainloop()
```

となる．siminit で 500 × 300 のキャンバスを作成し，シミュレーション原点は (50,30) に底上げする．x 軸，y 軸を表示し，赤色で (10,10)，(100,100)，(120,10) の折れ線を描画し，(120,10) を中心に「テスト」と文字列を表示させている．また，キャンバスの右上にプログラム終了ボタンを追加した．単に，キャンバスに他のウィジェットを追加表示する例にすぎないが，クラス継承での呼び出し法の参考になる．結果を図 22 に示す．

次は sin(x) のグラフを示す．三角関数は math モジュールを読み込む．メインプログラムは，

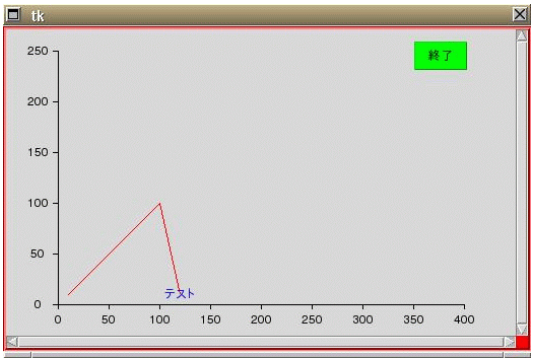


図 22 シミュレータ：折れ線と文字列

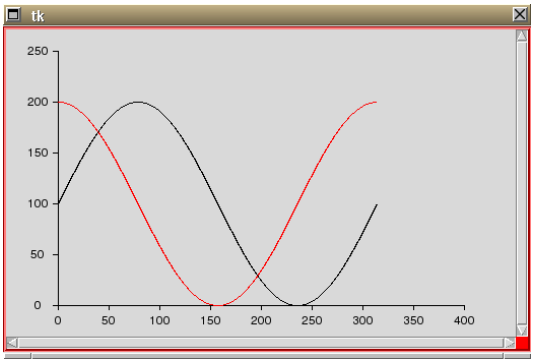


図 23 シミュレータ：正弦波形と余弦波形
黒色は正弦，赤は余弦波形を示す．

```
# メインプログラム
if __name__ == '__main__':
  import math
  c=CB()
  c.siminit(50,30,500,300)
  c.xaxis(0)
  c.yaxis(0)
  x=0
  c.mv(0,0,100)
  c.mv(1,0,200)
  while x<=314:
    y=(math.sin(x/50.0)+1)*100
    c.draw(0,x,y)
    y=(math.cos(x/50.0)+1)*100
    c.draw(1,x,y)
    x=x+1
  c.mainloop()
```

となる．黒，赤の 2 本のペンを使用しているが，表示のための変数は独立しているから，変数 x，y を共用しても問題ない．結果を図 23 に示す．

スプライン曲線 (1)

スプライン曲線は測定結果をスムーズな曲線で描く方法として広く利用されている．Tkinter では単純に $y = F(X)$ で表せる x に対する一価関数の B-スプライン補間ではなく，多価関数でもスムーズな曲線が得られるベジェ曲線を使用している．

スプライン曲線の例を画像の移動で紹介した mv_circles.py を改造して，サークルをノードにスプライン曲線を表示するプログラムを以下に示す．

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
# マウスバインド処理
def mv(event):
  global p
  try:
    id=c.find('closest',
      event.x,event.y)
    tag,now=c.gettags(id)
    c.delete('line')
    c.delete('line2')
    data=c.coords(tag)
    x1,y1,x2,y2=data
    oldx=(x1+x2)/2.0
    oldy= (y1+y2)/2.0
    dx=event.x-oldx; dy=event.y-oldy
    c.move(tag,dx,dy)
    tagn=int(tag[1:])
    p[tagn]=[oldx,oldy]
    c.create_line(p,tag='line')
    c.create_line(p,tag='line2',
      splinesteps=30,smooth='true',
      width=3)
  finally:
    return
# メインプログラム
c = tk.Canvas(width=300, height=200)
c.pack()
p=[]
x=20
for i in range(4):
  p.append([x,x])
  t='a'+str(i)
  c.create_oval(x-10, x-10, x+10,
    x+10, fill='red', tag=t)
  c.tag_bind(t,'<B1-Motion>',mv)
  c.tag_bind(t,'<Button-1>',mv)
  x=x+20
c.create_line(p,tag='line')
c.create_line(p,tag='line2',
  splinesteps=30,smooth='true',
    width=3)
c.mainloop()
```

プログラム解説

メインプログラムで，キャンバス c を作成，range(4) で 4 個の赤丸，その中心が節となる折れ線，スプライン曲線を作成する．また赤丸にはタグ名'a0'〜'a3' を付ける．
バインド関数 mv では find 命令で部品を選択し，折れ線とスプライン曲線を消去，赤丸の移動と共に，再度直線，曲線を表示する．この操作により，直線，曲線が移動するように見える．仕上りを図 24 に示す．

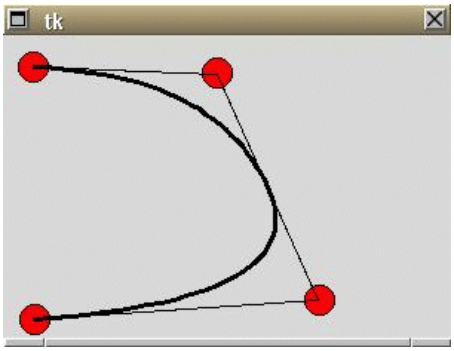


図 24 スプライン曲線
プログラムを開始すると，4 個の赤丸が右下方に並ぶ．赤丸はマウス左ボタンドローで移動し，細い直線と，太いスプライン曲線が表示される．

スプライン曲線 (2)

前項で示したように，Tkinter では多価関数でもスムーズな曲線が得られるベジェ曲線を採用している．この方法は美しい曲線が得られるが，スプライン曲線が測定値を通過しない欠点がある．そのために測定間の補間値を得ることができず，実験値からシミュレーションや推論ができない欠点がある．

幸い，インターネット[5] で，指定する X-Y 座標を通過する 3 次 B-スプライン補間プログラムが公表されており，それを前述ののシミュレーションプログラムに導入した．このための新しいメソッドは，測定値として円を表示する drawoval，スプライン関係の maketable と spline である．

以下にスプライン関係の maketable と spline メソッドを示す．

```
# スプライン補間
  def maketable(self, x, y, z):
    h=[0. for i in range(N)]
    d=[0. for i in range(N)]
    # 両端点での y''(x)/6
    z[0]=0;  z[N-1]=0
    for i in range(N-1):
      h[i]= x[i+1]-x[i]
      d[i+1]=(y[i+1]-y[i]) \
        /float(h[i])
    z[1]=d[2]-d[1]-h[0]*z[0]
    d[1]=2*(x[2]-x[0])
    for i in range(1, N-2):
      t=h[i]/float(d[i])
      z[i+1]=d[i+2]-d[i+1]-z[i]*t
      d[i+1]=2*(x[i+2]-x[i])-h[i]*t
    z[N-2] -= h[N-2]*z[N-1]
    for i in range(N-2, i-2, -1):
      z[i]=(z[i]-h[i]*z[i+1]) \
        /float(d[i])
```

```
  def spline(self, t, x, y, z):
    i=0;  j=N-1
    while i < j:
      k=(i+j)/2
      if x[k] < t:
        i=k+1
      else :
        j=k
    if i > 0:
      i -= 1
    h=x[i+1]-x[i];  d=t-x[i]
    return (((z[i+1]-z[i])*d/ \
      float(h)+z[i]*3)*d
      +((y[i+1]-y[i])/float(h)
      -(z[i]*2+z[i+1])*h))*d+y[i]
```

drawoval は drawstring を変更すれば容易に作成できる．

```
# 円を表示
# 呼び出し法： drawoval
# ペン色 x y 半径
  def drawoval(self, p, x, y, r):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    x1=xorigin+x-r
    x2=xorigin+x+r
    y1=yspan-(yorigin+y-r)
    y2=yspan-(yorigin+y+r)
    self.c.create_oval(x1, y1, x2,
      y2, fill=self.color(p))
```

スプライン補間の結果を以下に示す．

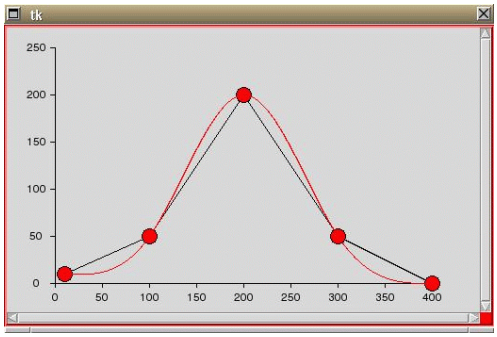


図 25 スプライン曲線
赤丸は測定点，黒折れ線は測定点間を直線補間し，赤線はスプライン補間した曲線である．

以下は図 25 を作成するためのメインプログラムである．

[5] http://f59.aaa.livedoor.jp/~ ookini/pukiwiki.php?spline.py

```
# メインプログラム
if __name__ == '__main__':
  c=CB()
  c.siminit(50,30,500,300)
  c.xaxis(0)
  c.yaxis(0)
  N=5
  x=[10, 100, 200, 300, 400]
  y=[10, 50, 200, 50, 0]
  z=[0. for i in range(N)]
  for i in range(N):
    c.drawoval(1,x[i],y[i],8)
    if i==0 :
      c.mv(0,x[i],y[i])
    else:
      c.draw(0,x[i],y[i])
  c.maketable(x, y, z)
  c.mv(1,x[0],y[0])
  for i in range(x[0],x[-1]-1):
    c.draw(1,i,c.spline(i, x, y, z))
  c.mainloop()
```

プログラム解説

クラス CB はクラス CA の継承で，スクロールドキャンバスを作成する．c.siminit でキャンバスサイズ，x-y 座標の原点を指定し，x 軸，y 軸を表示する．次いで，x，y，z でスプラインのための接点を指定する．これらのリストを利用して赤丸と折れ線を作成する．c.maketable でスプラインの係数を計算し，最後の for 文で x の開始座標から終端までのスプライン値を y として，c.draw で直線を描いている．

変化するスプライン曲線 (3)

前項のスプライン曲線では赤丸が固定されていたが，それが移動できると種々の曲線を作成することができる．また，3 次スプライン補間は接点の位置により思わぬ鋭い曲線が発生する危険性があり，その確認のためにも必要である．

マウス操作による接点移動と曲線の作成は tag_bind メソッドでマウス座標を検出し，赤丸をマウス位置に移動することと，既に表示されているスプライン曲線を消去し，新しくマウス位置に対応するスプライン曲線を描くことの連動で可能になる．したがって，これらの操作のためには赤丸と直線にタグ名の設定が必要になる．

まずは，赤丸の移動から始めよう．drawoval メソッドの変更は容易で引数にタグ名 t を追加するだけである．但し，タグ名は文字列を指定しなくてはならない．

```
# 円を表示
# 呼び出し法： drawoval
# ペン色 x y 半径 タグ名
  def drawoval(self, p, x, y, r, t):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    x1=xorigin+x-r
    x2=xorigin+x+r
    y1=yspan-(yorigin+y-r)
    y2=yspan-(yorigin+y+r)
    self.c.create_oval(x1, y1, x2,
      y2, fill=self.color(p), tag=t)
    self.c.tag_bind(t,'<B1-Motion>',
      self.bmv)
    self.c.tag_bind(t,'<Button-1>',
      self.bmv)
```

プログラムは引数の追加，self.c.create_oval に tag=t の追加，それにバインド処理メソッド bmv(bind move の略) の登録が必要で，さらにメインプログラムでの draoval 作図の前にタグ名の設定が必要になる．

スプライン曲線は draw メソッドで描いていたが，タグ名を付けると軸表示を含む基本図形も消去されるので，新たにタグ名を付加する drawt メソッドを作成する．これは draw メソッドをコピーし，creatge_line に tag='l' とタグオプションを付加するだけでよい．

バインド割り込みを以下に示す．

```
# マウスバインド処理
  def bmv(self,event):
    global xspan, yspan, xorigin, \
      yorigin, xstart, ystart
    global x,y,z,N
    try:
    # 赤丸特定
      id=self.c.find('closest',
        event.x,event.y)
      tag,now=self.c.gettags(id)
      if tag[0]!='a':
        return
      data=self.c.coords(tag)
      x1,y1,x2,y2=data
      oldx=(x1+x2)/2.0
      oldy= (y1+y2)/2.0
      dx=event.x-oldx
      dy=event.y-oldy
    # 赤丸移動
      self.c.move(tag,dx,dy)
      tagn=int(tag[1:])
    # キャンバス座標より SIM 座標の変換
      xnew=oldx-xorigin
      ynew=yspan-yorigin-oldy
    # x, y 座標リスト更新
      x[tagn]=xnew
      y[tagn]=ynew
      z=[0. for i in range(N)]
      self.maketable(x, y, z)
    # 折れ線，スプライン消去
      self.c.delete('l')
    # 折れ線作成
      self.mv(0,x[0],y[0])
      for i in range(1,N):
        c.drawt(0,x[i],y[i])
     # スプライン作成
      self.mv(1,x[0],y[0])
      for i in range(x[0],x[N-1]):
        self.drawt(1,i,
          self.spline(i, x, y, z))
    finally:
      return
```

プログラム解説

バインドは tag_bind を行っているから，赤丸と直線にマウスクリックすると bmv メソッドにプログラム割り込みが発生する．そこで，まず，try:，finally:で赤丸でない割り込みであれば何も動作せずに終了できるようにしておく．次いで，赤丸の座標と，マウスポインターの位置の差を検出し，赤丸の中心をマウスポインターの位置に移動する．常に dx，dy は 0 になるようにイベントが発生するから，赤丸はマウス位置に追従する．

赤丸のタグ番号を調べて，接点番号を知り，x および y の接点リストを更新する．今までの，折れ線，スプライン曲線を消去し，それらを新しい条件で描画して bmv を終了する．

メインプログラムの始めの部分はスプライン曲線 (1) と同じであるため，for 文以下を示す．

```
for i in range(N):
  tag='a'+str(i)
  c.drawoval(1,x[i],y[i],8,tag)
  if i==0 :
    c.mv(0,x[i],y[i])
  else:
    c.drawt(0,x[i],y[i])
c.maketable(x, y, z)
c.mv(1,x[0],y[0])
for i in range(x[0],x[-1]-1):
  c.drawt(1,i,c.spline(i, x, y, z))
c.mainloop()
```

プログラム解説

drawoval にタグ名を作成しているだけである．接点移動をさせた場合の作図を以下に示す．

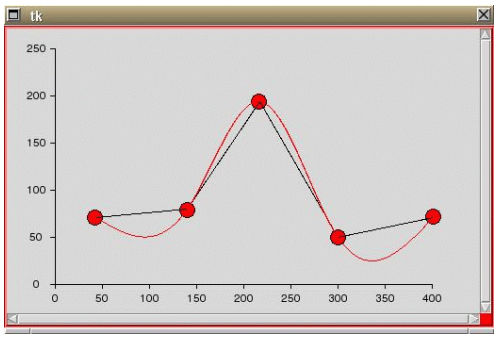


図 26 スプライン曲線
赤丸をドラグすると，それを通過する折れ線とスプライン曲線が再表示される．

ランチャー

ランチャー（launcher）とは発射装置という意味であるが，特にコンピュータソフトではディスプレーの隅に小さなボタンを並べて，日頃使用する電卓や電話番号帳などのソフトを起動させるボタン群を示す（図 27）．

通常の GUI プログラムはウィンドマネージャに依存，マウスクリックにより任意の位置に表示されるが，Python Tkinter の wm_命令を利用すると表示場所が指定できる．

ランチャープログラム (launcher.py)：

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import os
def vi():
  os.system('kterm -e vi vicmds &')
def calc():
   os.system('calc.py &')
def phone():
  os.system('phone.py &')
# ランチャーメインプログラム
root=tk.Tk()
root.title('launcher')
root.wm_geometry('+0+0')
f=tk.Frame(root, bd=2,bg='red')
f.pack(side='left')
b = tk.Button(f,text='vicmds',
  command=vi)
b.pack(side='left')
b = tk.Button(f,text='calc',
  command=calc)
b.pack(side='left')
b = tk.Button(f,text='phone',
  command=phone)
b.pack(side='left')
f.mainloop()
```

プログラム解説：

GUI は 3 個のボタンを並べただけであるが，新しい命令に wm_geometry で Display の左上端に GUI を表示させている．この命令の駆動には

図 27 ランチャーボタンボックス
Display の左上端に表示され，左より vi 命令，電卓，電話帳がクリックで駆動する．

Tk() クラスインスタンスである root を作成している．次に，ランチャーでは一般のシェル命令を Python 内より駆動させるため，os.system を利用した．エディター vi は端末の上で実行するプログラムだから，始めに Kterm(日本語端末) を表示させ，それに vi を駆動させる．vi 駆動時に特定のファイル vicmds を表示させるので -e オプションを使用している．また，&を用いて新たなセッションを作成し，多重処理を行わせている．

電卓プログラム (calc.py)：

電卓ソフトは通常の電卓の形式では一行しか表示されないため，結果を出力すると入力した内容が消されてしまう欠点がある．そのため，Entry ウィジェットを 2 段にして，入力用と出力用に分離している．また，行内の編集もでき，数学関数も組み入れている．

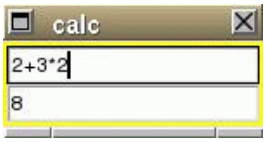


図 28 電卓 GUI
上段に式を入力し，下段に結果を示す．式は確認やカーソルを移動して編集ができる．

calc.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import os
from math import *
def cmd(event): # エンターコールバック
   data=eval(e1.get())
   e2.delete(0,'end')
   e2.insert(tk.END,data)
```

```
# 電卓メインプログラム
root=tk.Tk()
root.title('calc')
f=tk.Frame(root, bd=2,bg='yellow')
f.pack()
e1 = tk.Entry(f)
e1.pack()
# エンターバインド
e1.bind("<Return>",cmd)
e2 = tk.Entry(f)
e2.pack()
f.mainloop()
```

プログラム解説：

ext ウィジェットは複数行の文字入出力を行うが，Entry は 1 行入力・出力表示のテキストウィジェットである．メソッドの insert や delete 機能はテキストウィジェットと同じ動作をする．入力終了は改行キー <Return> イベントを利用し，cmd 関数を呼び出す．そこでは，e1 エントリーのデータを読み取り，式を評価し，e2 の表示している内容を消去してから計算結果を表示させている．

電話帳プログラム (phone.py)：

電話帳はデータベースになり得るから pickle か shelve のよい例と思ったが，データ入力の問題，同姓の場合の処理法，ソーティングの方法などを考えると解決しなければならない多くの問題を抱えることになる．

一方データの再利用の利便性考えると，text データに優るファイル形式はない．それであれば，grep で検索，vi による編集，uniq，sort，cut，paste などの unix 本来の命令を利用することができる．したがって，電話帳の GUI は 1 行エントリーとスクロールドテキストで構成することにした．

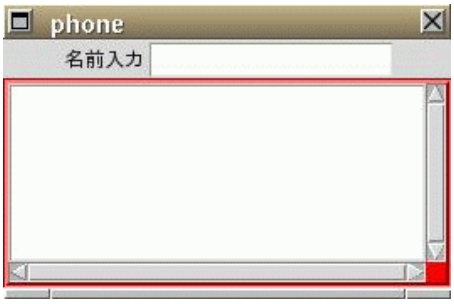


図 29 電話帳 GUI
上段に名前入力欄，下段はスクロールドテキストクラス TS() を利用している．

phone.py

```
#!/usr/bin/env python
# -*- coding: euc-jp -*-
import Tkinter as tk
import commands
from TS import *
from math import *
def cmd(event):
   global data,dbta,a
   data=e1.get()
   dbta='grep '+data+
     ' ./nenga/n04.txt'
   a=commands.getstatusoutput(
     dbta.encode('euc_jp'))
   t.t.delete(1.0,tk.END)
   t.t.insert(
     1.0,a[1].decode('euc_jp'))
# 電話帳メインプログラム
root=tk.Tk()
root.title('phone')
f=tk.Frame()
f.pack()
l1=tk.Label(f,text=u'名前入力')
l1.pack(side='left')
e1 = tk.Entry(f)
e1.pack(side='left')
e1.bind("<Return>",cmd)
t=TS()
```

プログラム解説：

メインプログラムを見ると，フレームを作成し，その中にラベルとエントリーを横向けに配置，それとは別に，スクロールドテキスト TS() を立てに配置している．エントリーの改行イベントで cmd 関数が駆動する．

cmd 関数では grep で検索を行う．Python 内より unix 命令を呼び出して結果を取得する方法は，commands モジュールの getstatusoutput 命令を使う．それは一つの引数で文字列を要求する．したがって，grep そのものを文字列に変換する．電話帳ファイルは ./nenga/n04.txt であり，euc コードが検索文字列に入るため，encode させる．

検索結果は変数 a に euc コードで記録される．その文字列をスクロールドテキストに表示させる．まずテキスト内容を削除し，文字列を decode して insert 命令を使用する．

以上，ランチャーを含めて 4 個の短いプログラムを紹介したが，30 行足らずのプログラムでこれほどの性能を引き出すプログラムは他に類を見ない．「長いプログラムは寿命が短く，短いプログラムは寿命が永い」というのが筆者の経験法則である．

終りに際して

筆者は日立製作所のわずか 8K byte しかメモリーを装備していない HITAC-10 による Assembler，Fortran の時代，次いで C 言語による X11R4 の時代，そして現在へと CPU プログラミングを経験してきたが，最も効率のよい言語は Tcl/Tk と思っていたし，今でもそのように思っている．しかし，プログラムが少々長くなると，変数扱いが突然難しくなり，数日も達つと何の変数であたったかも忘れ，プログラムの再利用や改良も困難になる．さらに，Tcl/Tk は jpg や png の画像が処理できないなどの欠点を克服するために Python+PIL の組合せを利用するに至った．

Python を使用して，クラスの作成，そしてメソッドの継承の素晴らしさに感動した．長い一連のプログラムではなく，短いプログラムを重ねていく作業で，システムとして成長していくことが理解できた．変数の問題も局所関数として遮蔽することができ，また別の関数で引用されることがなければ同じ変数を多用することもできる．これらの性質を利用することにより，初めて自作のプログラムがリソースとして再利用できる楽しみを知った次第である．

オブジェクト指向として C++や C#などの言語，JAVA などを否定するわけではないが，今までの経験から Python は開発効率のよい一級のプログラム言語だと思う．

本稿は Python について基本的な言語解説書ではない．それについては多くの解説書が出版されているので参考にしてほしい．むしろ，一通りの知識を知った上で，どのようにプログラミングに利用していくかという点を強調してきたつもりである．また，本項で紹介したプログラムは文献 4 に公開しているから自由にコピーして次のステップへと進んでほしい．最後に応用編として，心電図チャートよりベクトル心電図のリサージュ波形の構築に至った経緯は 2009 年日本麻酔・集中治療テクノロジー学会で Tcl/Tk を用いて発表したプログラムの焼き直しであることを付け加える．

参考書など

1. アラン・ゴールド著，松葉素子訳：Python で学ぶプログラム作法．ピアソン・エデュケーション．2001．
2. Mark Lutz，David Ascher 著，世紀太 章訳：始めての Python． O’ Reilly，2000．
3. 久野 晴：入門 tcl/tk． アスキー出版局．1997．
4. http://kansai.anesth.or.jp/gijutu/python/howto-python.php

索 引

編集後記

2009 年麻酔・集中治療テクノロジー学会理事・評議員会の席上、小生が来年定年退職になるので、事務局長が勤まるかどうか発議したが、学会財政困難な折りもあり、当分の間引続き小生が事務を担当することで了解をえた。各年度の学会は学会長にお願いすることになるが、冊子「麻酔・集中治療とテクノロジー」の発刊は「麻酔科医のためのパソコンセミナー」よりの原稿が滞り、急遽、小生が以前より備忘録としてノートしてきた Python 言語の使い方を解説書風にまとめ上げ、ようやく 100 ページを越える冊子に仕上げた次第である。この解説書は Python 言語の基本的な文法を解説しているのではないので、まずは書店で販売されている入門書を理解した上で、問題解決の参考にしていただければと願っている。

本冊子は従来より学会発表を雑誌「麻酔」の投稿規定に従った形式で出版することにしているが、投稿論文数が少なくなり、原著風ではなく、また、総説よりも読みやすい解説書、少しページ数が多くなってもよい、そのような主張を印刷物にできる重宝な存在として、今後も発展していけばよいのではないかと考えている。若い先生も、経験を積んだ先生も考えをまとめて頂き、会員、非会員を問わず、寄稿して頂ければ本誌発刊にとってこの上ない幸せである。

麻酔・集中治療テクノロジー学会
事務局長 田中 義文
2010 年 3 月 30 日

麻酔・集中治療とテクノロジー　2008　　〈検印省略〉

2010年5月30日　第1版発行
定価（本体3,000円＋税）

編集者　風　間　富　栄
　　橋　本　　悟
　　田　中　義　文
発行者　田　中　義　文
発　行　日本麻酔・集中治療テクノロジー学会
発行所　㈱北斗プリント社
〒606-8540　京都市左京区下鴨高木町38-2
電話（075)791-6125　FAX（075)791-7290

ISBN4-89467-158-1

ISBN4-89467-158-1
C3047 ¥3000E
9784894671584
定価（本体3,000円＋税）
1923047030009

麻酔・集中治療と
テクノロジー
2008