(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2013-211043
(P2013-211043A)
(43)公開日　平成25年10月10日(2013.10.10)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 19/20	(2011.01)	G06F	19/20		4B024
C12Q 1/68	(2006.01)	C12Q	1/68	Z	4B029
C12M 1/00	(2006.01)	C12M	1/00	A	4B063
C12M 1/34	(2006.01)	C12M	1/34	Z	
C12N 15/09	(2006.01)	C12N	15/00	A	

審査請求 有　請求項の数 1　ＯＬ　外国語出願　（全 60 頁）

(21)出願番号　特願2013-114005 (P2013-114005)
(22)出願日　平成25年5月30日 (2013.5.30)
(62)分割の表示　特願2008-555390 (P2008-555390)の分割
原出願日　平成19年2月15日 (2007.2.15)
(31)優先権主張番号　60/774,354
(32)優先日　平成18年2月16日 (2006.2.16)
(33)優先権主張国　米国 (US)

(71)出願人　504005932
４５４　ライフ　サイエンシーズ　コーポレイション
アメリカ合衆国　コネチカット　０６４０５，　ブランフォード，　コマーシャル　ストリート　２０
(74)代理人　100078282
弁理士　山本　秀策
(74)代理人　100113413
弁理士　森下　夏樹
(72)発明者　チェン　イー－フ
アメリカ合衆国　コネチカット　０６５１０，　ニュー　ヘイブン，　テンプル　ストリート　１５２，　アパートメント　６１０

最終頁に続く

(54)【発明の名称】核酸配列データのプライマー伸長誤差を補正するためのシステムおよび方法

(57)【要約】
【課題】核酸配列データのプライマー伸長誤差を補正するためのシステムおよび方法の提供。
【解決手段】本発明は、テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するための方法の一実施態様について、（ａ）配列決定反応において１つまたは複数のヌクレオチドの組込みに応答して生成された信号を検出するステップと、（ｂ）この信号の値を生成するステップと、（ｃ）第１パラメータおよび第２繰越パラメータを使用して位相同期誤差の値を補正するステップとを含むことを説明する。
【選択図】図１

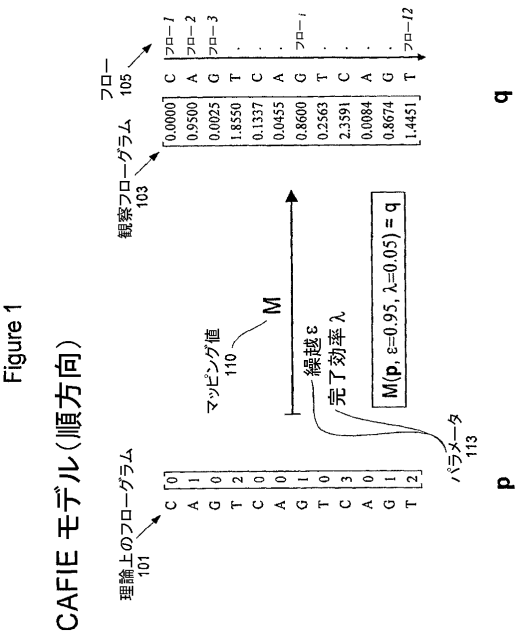


Figure 1

【特許請求の範囲】
【請求項１】
本願明細書に記載された発明。
【発明の詳細な説明】
【技術分野】
【０００１】
関連出願
本願は、２００６年２月１６日に出願された、米国仮特許出願第６０／７７４，３５４号（発明の名称「Ｓｙｓｔｅｍ ａｎｄ Ｍｅｔｈｏｄ ｆｏｒ Ｃｏｒｒｅｃｔｉｎｇ Ｐｒｉｍｅｒ Ｅｘｔｅｎｓｉｏｎ Ｅｒｒｏｒｓ ｉｎ Ｎｕｃｌｅｉｃ Ａｃｉｄ Ｓｅｑｕｅｎｃｅ Ｄａｔａ」）に関連し、この仮特許出願の優先権を主張し、この仮特許出願は、あらゆる目的のために、その全体が本明細書中に参考として援用される。
【０００２】
発明の分野
本発明は、分子生物学の分野に関する。詳細には、本発明は、「合成による配列決定」（ＳＢＳ）技術と一般に呼ばれる技術によって生成された核酸配列データの誤差の補正に関する。
【背景技術】
【０００３】
発明の背景
合成による配列決定（ＳＢＳ）は、一般に、核酸サンプル中の１つまたは複数のヌクレオチドの同一性または配列を判断するための方法であって、ヌクレオチド配列の組成が決定されるテンプレート核酸分子に対して相補的な単鎖ポリヌクレオチドの段階的な合成を含む方法を意味する。たとえば、ＳＢＳ技術は、一般に、単一核酸（ヌクレオチドとも呼ばれる）種を、対応する配列位置におけるテンプレート分子の核酸種に対して相補的な新生ポリヌクレオチド分子に添加することによって機能する。新生分子に対する核酸種の添加は、一般に、先行技術で公知の多様な方法を使用して検出され、こうした方法としては、パイロシーケンシングと呼ばれる方法、または蛍光検出法、たとえば可逆的ターミネーターを使用する方法挙げられるが、これらだけに限らない。一般に、このプロセスは、完了するまで（つまり、すべての配列位置が表現されるまで）、またはテンプレートに対して相補的な所望の配列長さが合成されるまで繰り返される。ＳＢＳ技術のいくつかの例は、特許文献１に記載されており、この特許は、本明細書で引用することにより、あらゆる目的で全体を本願に援用する；および米国特許出願第１０／７８８，５２９号；第０９／８１４，３３８号；第１０／２９９，１８０号；第１０／２２２，２９８号；第１０／２２２，５９２号に記載されており、これらの特許出願は各々、本明細書で引用することにより、あらゆる目的で全体を本願に援用する。
【０００４】
ＳＢＳのいくつかの実施態様では、オリゴヌクレオチドプライマーは、サンプルテンプレート分子の予め決められた相補的な位置までアニールするように設計される。プライマー／テンプレート複合体は、核酸ポリメラーゼ酵素の存在下で、ヌクレオチド種とともに提示される。ヌクレオチド種が、サンプルテンプレート分子上の配列位置に対応する核酸種に対して相補的であり、サンプルテンプレート分子が、オリゴヌクレオチドプライマーの３’末端に直接隣接する場合、ポリメラーゼは、ヌクレオチド種とともにプライマーを伸長する。あるいは、実施態様によっては、プライマー／テンプレート複合体は、対象となる複数のヌクレオチド種（一般に、Ａ、Ｇ、Ｃ、およびＴ）とともに同時に提示され、オリゴヌクレオチドプライマーの３’末端に直接隣接するサンプルテンプレート分子上の対応する配列位置において相補的なヌクレオチド種が組み込まれる。上記の実施態様の何れの場合も、さらに伸長するのを防止するために化学的にブロックされ（たとえば、３’－Ｏ位置において）、次回の合成の前にブロック解除する必要がある。上記のとおり、ヌクレオチド種の組込みは、先行技術で公知の多様な方法で検出することができ、ピロリン

酸塩（ＰＰｉ）の放出を検出することによって（例は、特許文献２；特許文献３；および特許文献４に記載されており、これらの特許の各々は、本明細書で引用することにより、あらゆる目的で全体を本願に援用する）、またはヌクレオチドに結合された検出可能な標識を介して検出することができる。検出可能な標識のいくつかの例としては、マスタグ、および蛍光、または化学発光標識が挙げられるが、これらだけに限らない。代表的な実施態様では、組み込まれていないヌクレオチドは、たとえば洗浄により除去される。検出可能な標識が使用される実施態様では、検出可能な標識は、一般に、次の合成サイクルの前に不活性化する必要がある（たとえば、化学分解または光退色により）。次に、テンプレート／ポリメラーゼ複合体内の次の配列位置は、上記のとおり、別のヌクレオチド種または複数のヌクレオチド種で問い合わせることができる。ヌクレオチド添加、プライマー伸長、信号取得、および洗浄というサイクルが繰り返されると、テンプレート鎖のヌクレオチド配列が決定される。

【０００５】

ＳＢＳの代表的な実施態様では、多量または集団の実質的に同じテンプレート分子（たとえば、10^3、10^4、10^5、10^6、または10^7分子）は、信頼するに足る強力な信号を達成するために、任意の１つの配列決定反応で同時に分析される。特定の反応の集団における実質的にすべてのテンプレート分子に関連する新生分子の「均質な伸長」と呼ばれる状態は、信号対雑音比を低くするために必要である。「均質な伸長」という用語は、本明細書で使用する場合、一般に、上記の実質的に同じテンプレート分子が、同じ反応ステップを均質に実行する伸長反応の関係または相を意味する。たとえば、テンプレート分子の集団に関連する各々の伸長反応は、関連する各々のテンプレート分子に関連する同じ配列位置で同じ反応ステップを行う場合、互いに同相または位相同期と説明し得る。

【０００６】

しかし、関連技術の当業者は、各々の集団内のテンプレート分子の小画分は、集団内の他のテンプレート分子との位相同期を失うか、またはこうした位相同期から脱落することを理解するであろう（つまり、テンプレート分子の画分に関連する反応は、集団に関して実行される配列決定反応で、他のテンプレート分子より先に進むか、または遅れる（いくつかの例は、非特許文献１に記載されており、これは、本明細書で引用することにより、あらゆる目的で全体を本願に援用する）。たとえば、１つまたは複数のヌクレオチド種を１つまたは複数の新生分子に適切に組み込んで、配列を１つの位置だけ伸長する反応が失敗すると、後続の反応は、集団の他の部分の配列位置より遅延するか、または位相が一致しない配列位置になる。この作用は、本明細書では「不完全な伸長」（ＩＥ）と呼ぶ。あるいは、集団の他の部分の配列位置より先にあるか、または位相が異なる配列位置に、１つまたは複数のヌクレオチド種を組み込むことによって、新生分子を不適切に伸長することは、本明細書では「繰越」（ＣＦ）と呼ぶ。ＣＦおよびＩＥの複合効果は、本明細書ではＣＡＦＩＥと呼ぶ。

【０００７】

不完全な伸長の問題に関連して、単独または何らかの組合せで生じ得るＩＥの一因となる可能性があるいくつかのメカニズムがある。ＩＥの一因となる可能性があるメカニズムの一例としては、テンプレート／ポリメラーゼ複合体の部分集合に提示されるヌクレオチド種が欠如していることを含むことができる。ＩＥの一因となる可能性があるメカニズムのもう１つの例としては、新生分子に組み込むために適切に提示されるヌクレオチド種を組み込むためのポリメラーゼ分子の部分集合が破損していることを含むことができる。ＩＥの一因となる可能性があるメカニズムのさらに他の例としては、テンプレート／ポリメラーゼ複合体におけるポリメラーゼの活量の欠如を含むことができる。

【０００８】

少なくとも部分的にＳＢＳ法におけるＩＥ誤差の原因になる可能性があるさらに別のメカニズムの一例としては、Ｍｅｔｚｇｅｒ（非特許文献２）が調査した循環可逆終了（ＣＲＴ）と呼ばれる状態を含むことができ、その内容は、本明細書で引用することにより、あらゆる目的で全体を本願に援用する）。ＣＲＴでは、ヌクレオチド種は、単一ヌクレオ

チド種の組込み後に、新生分子がさらに伸長するのを防止する変性３’－Ｏ基（通常、キャップ、保護基、または終了暗号と呼ばれる）を有する。これらの保護基は、化学処理または光処理を含む様々な方法の１つによって除去できるように設計される。３’－Ｏ位置の脱保護（および３’－ＯＨ基の形成）後、新生分子は、別のヌクレオチド種によって伸長させることができる。しかし、位相非同期は、一部分の新生分子が、不完全な脱保護効果（不完全脱保護）によって保護状態を維持している場合に生じる。その後のサイクルでは、保護状態を維持している新生分子のこの部分は伸長せず、その結果、残りの集団の配列位置から脱落して、位相が異なる。しかし、その後の脱保護ステップでは、以前に不適切に残され、伸長を再開させ、新生分子からの信号を生成し、集団の他の部分との位相非同期状態を継続する保護基の少なくとも一部分の除去に成功し得る。当業者は、ＩＥの一因となるその他の要素があり、したがって、上記の例に限定されないことを理解するであろう。

【０００９】

本発明について本明細書に記載するシステムおよび方法は、こうした任意の単独または複合の原因またはメカニズムから生じ得るＩＥ誤差の補正を目的とする。たとえば、不完全な脱保護および後続の成功した脱保護の結合によって生じるＩＥ誤差の補正は、本発明の目的の１つである。

【００１０】

ＣＦの問題に関しては、ＣＦの一因となる可能性があるメカニズムがいくつかあり、これらは、単独で、または何らかの組合せで生じ得る。たとえば、可能性のある１つのメカニズムとしては、前のサイクルから残存する過剰なヌクレオチド種が挙げられる。これは、あるサイクルの終わりに実行される洗浄プロトコルが、すべてではないが、大部分のヌクレオチド種をサイクルから除去するために生じる可能性がある。この例では、その結果は、「Ｇ」ヌクレオチド種サイクル中に存在する「Ａ」ヌクレオチド種の小画分を含み、相補的な「Ｔ」ヌクレオチド種が、テンプレート分子中の対応する配列位置に存在する場合、新生分子の小画分の伸長の原因になる可能性がある。繰越効果を生じる可能性があるメカニズムのもう１つの例としては、ポリメラーゼの誤差、たとえばテンプレート分子上のヌクレオチド種に対して相補的ではない新生分子中に、ヌクレオチド種が不適切に組み込まれることが挙げられる。

【００１１】

少なくとも部分的にＳＢＳ法のＣＦの原因になる可能性があるさらにもう１つのメカニズムの一例は、Ｍｅｔｚｇｅｒが調査した循環可逆終了（ＣＲＴ）（非特許文献２、これは上記で引用することにより援用する）が挙げられる。この例では、ＩＥに関して上記で述べたとおり、３’－Ｏ保護ヌクレオチド分子の調製は、ヌクレオチド分子のある画分が保護基を持たないか、または保護基を紛失した場合に使用し得る。保護基の紛失は、意図する脱保護ステップの前の配列決定プロセスでも発生し得る。こうした保護基が欠如している場合、新生分子によっては、一度に複数のヌクレオチド種によって伸長するであろう。新生分子の画分のこうした不適切な複数の伸長によって、これらの画分は、配列位置で先に移動し、集団の他の部分の配列位置と位相が異なることになる。したがって、保護されていないヌクレオチド、および／または早期に脱保護するヌクレオチドは、少なくとも部分的に、ＣＲＴを伴うＳＢＳ法のＣＦの一因となり得る。

【００１２】

本明細書について本明細書に記載する実施態様のシステムおよび方法は、こうした単独または複合的な原因またはメカニズムから生じ得るＣＦ誤差の補正を目的としている。たとえば、保護基の欠如によって生じるＣＦ誤差の補正は、本発明の目的の１つである。

【００１３】

さらに、本発明について本明細書に記載する実施態様のシステムおよび方法は、ＩＥ誤差およびＣＦ誤差の両方の補正を目的とし、この両方のタイプの誤差は、同じ配列決定反応のある集団のいくつかの組合せで生じ得る。たとえば、ＩＥおよびＣＦは各々、上記のとおり、単独または複合的な原因またはメカニズムから生じ得る。

【００１４】
当業者は、ＩＥおよびＣＦ誤差の両方の可能性が、伸長反応の際に各々の配列位置で生じ得るため、結果として得られる配列データに明らかな累積効果を有する場合があることを理解するであろう。たとえば、この効果は、「実行」または「配列決定実行」とも呼ばれる一連の配列決定反応の終わりに向かって特に著しくなる場合がある。さらに、ＩＥおよびＣＦ効果は、ＳＢＳ手法を用いて確実に配列決定し得るテンプレート分子の長さ（場合により「読込み長さ」と呼ばれる）に上限を与え、つまり、配列データの質は、読込み長さが増加するにつれて低下するからである。
【００１５】
たとえば、ＳＢＳの１つの方法は、代表的な１回の実行で２５００万を超える配列位置を有する配列データを生成することができ、これは、「Ｐｈｒｅｄ」の品質スコアの２０以上に相当する（Ｐｈｒｅｄの品質スコア２０は、配列データが、９９％以上の精度を有すると予測されることを意味する）。ＳＢＳ法に関してＰｈｒｅｄ２０の品質を有する全体の配列決定の処理量は、毛細管電気泳動技術を使用するＳａｎｇｅｒ配列決定法として当業者に周知されている方法によって生成される配列データと比べて著しく多量だが、いまのところ、ＳＢＳ法の実質的により短い読込み長さを犠牲している（非特許文献３；本明細書で引用することにより、あらゆる目的で全体を本願に援用する）。したがって、ＩＥおよびＣＦ誤差によって生じる配列データの劣化を防止または補正することによって、読込み長さの上限を増加すると、ＳＢＳ法の全体的な配列決定処理量を増加することになるであろう。
【先行技術文献】
【特許文献】
【００１６】
【特許文献１】米国特許第６，２７４，３２０号明細書
【特許文献２】米国特許第６，２１０，８９１号明細書
【特許文献３】米国特許第６，２５８，５６８号明細書
【特許文献４】米国特許第６，８２８，１００号明細書
【非特許文献】
【００１７】
【非特許文献１】Ｒｏｎａｇｈｉ，Ｍ．，ＰｙｒｏＳｅｑｕｅｎｃｉｎｇ ｓｈｅｄｓ ｏｎ ＤＮＡ Ｓｅｑｕｅｎｃｉｎｇ．Ｇｅｎｏｍｅ Ｒｅｓ．１１，３－１１（２００１年）
【非特許文献２】Ｍｅｔｚｇｅｒ，Ｇｅｎｏｍｅ Ｒｅｓ．２００５ Ｄｅｃ；ｌ５（１２）：１７６７－７６
【非特許文献３】Ｍａｒｇｕｌｉｅｓ等，２００５，Ｎａｔｕｒｅ４３７：３７６－８０
【発明の概要】
【発明が解決しようとする課題】
【００１８】
したがって、核酸配列決定の合成による配列決定法によって生成された配列データにおけるＩＥおよび／またはＣＦ誤差を補正することを目的としたシステムおよび方法を提供することが望ましい。
【００１９】
本明細書では、多くの参考文献を引用するが、その全体の開示事項は、引用することにより、あらゆる目的で全体を本願に援用する。さらに、これらのどの参考文献も、上記でどのように記載されているかに関わらず、本明細書で請求する主題に関する本発明の先行技術として認めるものではない。
【課題を解決するための手段】
【００２０】
発明の概要
本発明の実施態様は、核酸の配列決定に関する。詳細には、本発明の実施態様は、ＳＢ

Ｓによる核酸の配列決定時に得られたデータの誤差を補正する方法およびシステムに関する。

【００２１】

テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法の一実施態様について、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップ、（ｂ）この信号の値を生成するステップ、および（ｃ）第１パラメータおよび第２パラメータを使用して、位相同期の値を補正するステップを含む実施態様を説明する。

【００２２】

いくつのインプリメンテーションでは、ステップ（ａ）～（ｃ）は、テンプレート分子の各々の配列位置ごとに繰り返され、補正された各々の値は、テンプレート分子の表現に組み込むことができ、テンプレート分子の表現は、フローグラム表現を含んでよい。

【００２３】

また、テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法の一実施態様について、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップ、（ｂ）信号の値を生成するステップ、（ｃ）テンプレート分子の配列に関連する表現に値を組み込むステップ、（ｄ）テンプレート分子の各々の配列位置についてステップ（ａ）～（ｃ）を繰り返すステップ、（ｅ）第１パラメータおよび第２パラメータを使用して、表現の位相同期誤差の各々の値を補正するステップ、および（ｆ）補正値を使用して、補正表現を生成するステップを含む実施態様を説明する。

【００２４】

さらに、テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法の一実施態様について、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップ、（ｂ）信号の値を生成するステップ、（ｃ）テンプレート分子の配列に関連する表現に値を組み込むステップ、（ｄ）テンプレート分子の各々の配列位置についてステップ（ａ）～（ｃ）を繰り返すステップ、（ｅ）表現を複数の部分集合に分割し、各々の部分集合が、テンプレート分子の１つまたは複数の配列位置を含むステップ、（ｆ）各々の部分集合において第１パラメータおよび第２パラメータの同期誤差を概算するステップ、（ｇ）各々の個々の部分集合に関する第１パラメータおよび第２パラメータの同期誤差の概算を使用して、位相同期誤差に関する各部分集合の各々の値を補正するステップ、および（ｈ）補正値を使用して、補正部分集合を補正表現に結合するステップを含む実施態様を説明する。

【００２５】

さらに、テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するシステムの一実施態様について、コンピュータ上で実行するために記憶されたプログラムコードを含み、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して検出された信号の値を生成するステップ、および（ｂ）第１パラメータおよび第２パラメータを使用して、位相同期誤差の値を補正するステップを含む方法を実行するコンピュータを備える実施態様を説明する。

【００２６】

さらに、テンプレート分子の実質的に同じコピーの集団から生成された配列データに関連する誤差を補正するシステムの一実施態様について、コンピュータ上で実行するためにプログラムコードを含み、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して検出された信号の値を生成するステップ、（ｂ）値をテンプレート分子の配列に関連する表現に組み込むステップ、（ｃ）テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｂ）を繰り返すステップ、（ｄ）第１パラメータおよび第２パラメータを使用して、表現の位相同期誤差の各々の値を補正するステップ、および（ｅ）補正値を使用して、補正表現を生成するステップ含む方法を実行するコンピュータを備

える実施態様を説明する。

【００２７】

また、テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するシステムの一実施態様について、コンピュータ上で実行するために記憶されたプログラムコードを含み、前記プログラムコードが、（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して検出された信号の値を生成するステップ、（ｂ）値をテンプレート分子の配列に関連する表現に組み込むステップ、（ｃ）テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｃ）を繰り返すステップ、（ｄ）表現を複数の部分集合に分割し、各部分集合が、テンプレート分子の１つまたは複数の配列位置を含むステップと、（ｅ）各々の部分集合において第１パラメータおよび第２パラメータの同期誤差を概算するステップ、（ｆ）各々の個々の部分集合に関する第１パラメータおよび第２パラメータの同期誤差の概算を使用して、位相同期誤差に関する各部分集合の各々の値を補正するステップ、および（ｇ）補正値を使用して、補正部分集合を補正表現に結合するステップを含む方法を実行する実施態様を説明する。

【００２８】

本発明の実施態様により達成される利点としては、（ａ）配列データの品質が増加し、その結果、所望のレベルの共通配列精度を達成するのに必要な配列包括度の深さが減少する；（ｂ）有用な配列の読込み長さが伸長し、これは、１回の実行から高品質の配列データが得られることを意味する；（ｃ）有用な配列読込み長さが伸長するため、一定の配列包括度深さを達成するために必要な実行が減少する、（ｄ）有用な配列読込み長さが伸長されるため、一定の領域にわたる配列コンティグを組み立てるのに必要な配列が減少する、および（ｅ）特に繰返し配列領域において、重複する読込みを容易に集合させることが挙げられるが、これらだけに限らない。

【００２９】

上記およびさらに他の特徴は、以下の詳細な説明を添付の図面と併せて考慮すると、より明らかになるであろう。図中、類似の参照符号は、類似の構造、構成要素、または方法のステップを指示し、参照符号の一番左の桁は、基準の構成要素が最初に記載された図面の番号を指示する（たとえば、構成要素１６０は、図１に最初に記載されている）。しかし、これらの表記はすべて、代表的なものであるか、または具体的に示すためであり、制限する意図はない。

例えば、本願発明は以下の項目を提供する。

（項目１）

テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法であって、

（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップと、

（ｂ）前記信号の値を生成するステップと、

（ｃ）第１パラメータおよび第２パラメータを使用して、前記位相同期誤差の前記値を補正するステップと

を含む、方法。

（項目２）

（ｄ）テンプレート分子の各配列位置についてステップ（ａ）～（ｃ）を繰り返すステップをさらに含む、項目１に記載の方法。

（項目３）

（ｅ）各々の補正値を前記テンプレート分子の表現に組み込むステップをさらに含む、項目２に記載の方法。

（項目４）

前記表現がフローグラムを含む、項目３に記載の方法。

（項目５）

前記位相同期誤差が、前記テンプレート分子の各々の配列位置についてともに実質的に一

定であるものとして処理される不完全な伸長要素および繰越要素を含み、前記第１パラメータが、前記不完全な伸長要素を表し、前記第２パラメータが前記繰越要素を表す、項目２に記載の方法。
（項目６）
前記位相同期誤差が、前記テンプレート分子の各々の配列位置について実質的に一定であるものとして処理される繰越要素を含み、前記第２パラメータが前記繰越要素を表す、項目２に記載の方法。
（項目７）
前記信号が、前記１つまたは複数のヌクレオチドの前記組込みに応じて放射される光を含む、項目１に記載の方法。
（項目８）
前記光が、前記配列決定反応からの化学発光光を含む、項目７に記載の方法。
（項目９）
前記配列決定反応が、ピロリン酸塩配列決定反応を含む、項目８に記載の方法。
（項目１０）
前記光が、前記配列決定反応からの蛍光光を含む、項目７に記載の方法。
（項目１１）
前記配列決定反応が、可逆的ターミネーターを使用する配列決定反応を含む、項目１０に記載の方法。
（項目１２）
前記信号の前記値が、前記組み込まれたヌクレオチドの数を表す、項目１に記載の方法。
（項目１３）
前記第１パラメータの値、および前記第２パラメータの値が、前記第１および第２パラメータの各々のマトリックス方程式に対するベストフィットを検索することによって概算される、項目１に記載の方法。
（項目１４）
前記第１および第２パラメータの前記ベストフィットの前記概算が、テスト値間の間隔を使用して検索し、１つまたは複数の近似値を各々のテスト値におけるマトリックス構成演算に適用することを含み、前記近似値が、前記概算の改善された計算効率を提供する、項目１３に記載の方法。
（項目１５）
テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法であって、
（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップと、
（ｂ）前記信号の値を生成するステップと、
（ｃ）前記値をテンプレート分子の配列に関連する表現に組み込むステップと、
（ｄ）前記テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｃ）を繰り返すステップと、
（ｅ）第１パラメータおよび第２パラメータを使用して、前記表現の前記位相同期誤差の各々の値を補正するステップと、
（ｆ）前記補正値を使用して、補正された表現を生成するステップと
を含む、方法。
（項目１６）
（ｇ）ステップ（ｅ）の前の繰返しからの前記補正値を使用して、ステップ（ｅ）～（ｆ）を繰返し反復するステップをさらに含み、前記補正値の一部または全部が、各々の繰返しで品質を改善する、項目１５に記載の方法。
（項目１７）
前記位相同期誤差が、前記テンプレート分子の各々の配列位置についてともに実質的に一定であるものとして処理される不完全な伸長要素および繰越要素を含み、前記第１パラメ

ータが、前記不完全な伸長要素を表し、前記第２パラメータが前記繰越要素を表す、項目１５に記載の方法。
（項目１８）
前記位相同期誤差が、前記テンプレート分子の各々の配列位置について実質的に一定であるものとして処理される繰越要素を含み、前記第２パラメータが前記繰越要素を表す、項目１５に記載の方法。
（項目１９）
前記信号が、前記１つまたは複数のヌクレオチドの前記組込みに応じて放射される光を含む、項目１５に記載の方法。
（項目２０）
前記光が、前記配列決定反応からの化学発光光を含む、項目１９に記載の方法。
（項目２１）
前記配列決定反応が、ピロリン酸塩配列決定反応を含む、項目２０に記載の方法。
（項目２２）
前記光が、前記配列決定反応からの蛍光光を含む、項目１９に記載の方法。
（項目２３）
前記配列決定反応が、可逆的ターミネーターを使用する配列決定反応を含む、項目２２に記載の方法。
（項目２４）
前記信号の前記値が、前記組み込まれたヌクレオチドの数を表す、項目１５に記載の方法。
（項目２５）
前記第１パラメータの値、および前記第２パラメータの値が、前記第１および第２パラメータの各々のマトリックス方程式に対するベストフィットを検索することによって概算される、項目１５に記載の方法。
（項目２６）
前記第１および第２パラメータの前記ベストフィットの前記概算が、テスト値間の間隔を使用して検索し、１つまたは複数の近似値を各々のテスト値におけるマトリックス構成演算に適用することを含み、前記近似値が、前記概算の改善された計算効率を提供する、項目２５に記載の方法。
（項目２７）
前記表現および補正された表現がフローグラムを含む、項目１５に記載の方法。
（項目２８）
テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正する方法であって、
（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して生成される信号を検出するステップと、
（ｂ）前記信号の値を生成するステップと、
（ｃ）前記値をテンプレート分子の配列に関連する表現に組み込むステップと、
（ｄ）前記テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｃ）を繰り返すステップと、
（ｅ）前記表現を複数の部分集合に分割し、各々の部分集合が前記テンプレート分子の１つまたは複数の配列位置を含むステップと、
（ｆ）各々の部分集合において第１パラメータおよび第２パラメータの前記同期誤差を概算するステップと、
（ｇ）各々の個々の部分集合に関する前記第１パラメータおよび前記第２パラメータの前記同期誤差の概算を使用して、前記位相同期誤差に関する各部分集合の各々の値を補正するステップと、
（ｈ）前記補正値を使用して、前記補正部分集合を補正表現に結合するステップと
を含む、方法。

（項目２９）
前記位相同期誤差が、前記テンプレート分子の複数の配列位置で変動する不完全な伸長要素、および繰越要素を含み、前記第１パラメータが不完全な前記伸長要素を表し、前記第２パラメータが前記繰越要素を表す、項目２８に記載の方法。
（項目３０）
前記位相同期誤差が、前記テンプレート分子の複数の配列位置で変動する繰越要素を含み、前記第２パラメータが前記繰越要素を表す、項目２８に記載の方法。
（項目３１）
テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するためのシステムであって、
コンピュータ上で実行するために記憶されたプログラムコードを含むコンピュータを備え、前記プログラムコードが、
（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して、検出された信号の値を生成するステップと、
（ｂ）第１パラメータおよび第２パラメータを使用して、前記位相同期誤差の前記値を補正するステップと
を含む方法を実行する、システム。
（項目３２）
前記プログラムコードによって実行される前記方法が、
（ｃ）テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｂ）を繰り返すステップ
をさらに含む、項目３１に記載のシステム。
（項目３３）
前記プログラムコードによって実行される前記方法が、
（ｄ）各々の補正値を前記テンプレート分子の表現に組み込むステップ
をさらに含む、項目３２に記載のシステム。
（項目３４）
前記プログラムコードによって実行される前記方法が、
（ｅ）前記表現をユーザに提供するステップをさらに含む、項目３３に記載のシステム。
（項目３５）
テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するシステムであって、
コンピュータ上で実行するために記憶されたプログラムコードを含むコンピュータを備え、前記プログラムコードが、
（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して、検出された信号の値を生成するステップと、
（ｂ）前記値をテンプレート分子の配列に関連する表現に組み込むステップと、
（ｃ）前記テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｂ）を繰り返すステップと、
（ｄ）第１パラメータおよび第２パラメータを使用して、前記表現の前記位相同期誤差の各々の値を補正するステップと、
（ｅ）前記補正値を使用して、補正表現を生成するステップと
を含む方法を実行する、システム。
（項目３６）
前記プログラムコードによって実行される前記方法が、
（ｆ）ステップ（ｄ）の前の繰返しからの前記補正値を使用して、ステップ（ｄ）～（ｅ）を繰返し反復するステップ
をさらに含み、前記補正値の一部または全部が、各々の繰返しで品質を改善する、項目３５に記載のシステム。

（項目３７）
前記繰返し反復するステップが、実行する繰返しの数に関するユーザの選択に応答する、項目３６に記載のシステム。
（項目３８）
前記プログラムコードによって実行される前記方法が、
（ｆ）前記補正された表現をユーザに提供すること
をさらに含む、項目３５に記載のシステム。
（項目３９）
テンプレート分子の実質的に同じコピーの集団から生成された配列データの位相同期に関連する誤差を補正するシステムであって、
コンピュータ上で実行するために記憶されたプログラムコードを含むコンピュータを備え、前記プログラムコードが、
（ａ）配列決定反応における１つまたは複数のヌクレオチドの組込みに応答して、検出された信号の値を生成するステップと、
（ｂ）前記値をテンプレート分子の配列に関連する表現に組み込むステップと、
（ｃ）前記テンプレート分子の各々の配列位置について、ステップ（ａ）～（ｃ）を繰り返すステップと、
（ｄ）前記表現を複数の部分集合に分割し、各々の部分集合が、前記テンプレート分子の１つまたは複数の配列位置を含むステップと、
（ｅ）各々の部分集合において第１パラメータおよび第２パラメータの前記同期誤差を概算するステップと、
（ｆ）各々の個々の部分集合に関する前記第１パラメータおよび前記第２パラメータの前記同期誤差の概算を使用して、前記位相同期誤差に関する各部分集合の各々の値を補正するステップと、
（ｇ）前記補正値を使用して、前記補正部分集合を補正表現に結合するステップと
を含む方法を実行する、システム。
【図面の簡単な説明】
【００３０】
【図１】図１は、「完全な」理論的なフローグラムを観察された「ダーティな」フローグラムに変換するために、一実施態様の数学モデルを単純化したグラフ表現である。
【図２】図２は、図１のマッピングモデルの反転の一実施態様を単純化したグラフ表現である。
【図３Ａ】図３ａは、図１および２のマッピングモデルを含む順方向および逆マトリックスの計算用のモデルの単純化された一実施態様のグラフ表現である。
【図３Ｂ】図３ｂは、図３ａの順方向モデルを使用する順方向マトリックスの計算の単純化された一実施態様のグラフ表現である。
【図４Ａ】図４ａは、図３ａの逆モデルを使用する逆マトリックスの計算の単純化された一実施態様のグラフ表現である。
【図４Ｂ】図４ｂは、図３ａおよび４ａの逆モデルを使用する様々なレベルの繰返し補正を使用して得られる結果の単純化された一実施態様のグラフ表現である。
【図５】図５は、本明細書に記載する発明のＣＡＦＩＥ誤差の結果の単純化された一実施態様のグラフ表現である。
【図６】図６は、実質的に同じテンプレート分子の集団のサンプル全体のパラメータ値の分布に関する単純化された一実施態様のグラフ表現である。
【図７】図７は、ＩＥ補正のみの効果、およびＣＡＦＩＥ補正の効果の単純化された一実施態様のグラフ表現である。
【発明を実施するための形態】
【００３１】
発明の詳細な説明
本明細書に記載する発明の実施態様は、少なくとも部分的に、理論上、つまり「完全な

」フローグラムは、ＩＥおよびＣＦの数学モデルによって、実際に観察される「ダーティな」フローグラムに変換することができるという発見に基づいている。本明細書で使用する「フローグラム」という用語は、一般に、たとえば配列決定データのグラフ表現を含み得る配列決定実行から生成される配列決定データの表現を意味する。たとえば、完全つまり理論上のフローグラムは、配列決定実行から生成され、上記のＣＡＦＩＥメカニズムに由来する誤差、またはその他のタイプの背景誤差がないデータを表す。同様に、ダーティまたは観察されたフローグラムは、ＣＡＦＩＥおよび背景誤差要素を含む配列決定実行から生成されるデータを表す。本明細書の例では、誤差要素の一部または全部は、正確に概算して完全なフローグラムモデルに適用し、実際の配列決定実行から得られる実際のデータの表現を提供し得る。

【００３２】

重要なことに、本明細書に記載する発明は、少なくとも部分的に、上記の数学モデルの逆は、直接観察されたフローグラムから完全な理論上のフローグラムを概算するのに役立つ可能性があるという発見に基づいている。したがって、上記の例を継続すると、誤差の概算は、観察されたフローグラムに表現される実際の配列決定データに適用することができ、すべて、または実質的にすべての誤差要素が除去された完全、または実質的に完全な理論上のフローグラム表現が得られる。

【００３３】

当業者は、誤差をデータから正確に除去することによって、前記データのより効率的かつ正確な解釈が得られることを理解するであろう。したがって、たとえば、配列決定実行で生成されたデータから誤差を除去すると、配列実行、およびより高品質の配列情報から生成される配列において、各々の核酸種を識別する必要性をより正確に提示することができる。

【００３４】

本明細書に記載する発明のいくつかの実施態様は、配列決定装置上のＳＢＳ配列決定実行から生成されるデータを分析するシステムおよび方法を含む。ＳＢＳ装置および方法のいくつかの例は、ピロリン酸塩ベースの配列決定手法と呼ぶこともできる方法を使用することができ、こうした手法は、たとえば、電荷結合（ＣＣＤ）カメラ、マイクロ流体チャンバ、サンプルカートリッジホルダー、またはポンプおよび流量バルブなどの１つまたは複数の検出デバイスを含むことができる。ピロリン酸塩ベースの配列決定の例を考慮すると、装置の実施態様は、化学発光を検出方法として使用することができ、この場合、ピロリン酸塩配列決定では、本質的に低レベルの背景雑音を生成する。本明細書の例では、配列決定のためのサンプルカートリッジホルダーは、光ファイバフェースプレートから形成される「ｐｉｃｏｔｉｔｅｒｐｌａｔｅ」と呼ばれるものを含むことができ、このフェースプレートは、酸エッチングされて、各々が実質的に同じテンプレート分子の集団を保持することが可能な何十万もの非常に小さいウェルを形成する。実施態様によっては、実質的に同じテンプレート分子の各々の集団は、ビードなどの固体基板上に配置することができ、各々のビードは、前記ウェルの１つの中に配置することができる。この例を続けると、装置は、流体試薬をｐｉｃｏｔｉｔｅｒｐｌａｔｅホルダーに、およびｐｉｃｏｔｉｔｅｒｐｌａｔｅ上の各々のウェルから放出される光子を収集することが可能なＣＣＤタイプの検出デバイスに提供するための試薬送達要素を備えることができる。ＳＢＳタイプの配列決定、およびピロリン酸塩配列決定を実行するための装置および方法のさらに他の例は、米国特許出願第１０／７６７，７７９号；同第１１／１９５，２５４に記載されており、これらの特許はともに、本明細書で引用することにより、あらゆる目的で全体を本願に援用する。

【００３５】

さらに、本発明について本明細書に記載するシステムおよび方法は、コンピュータシステム上で実行するために記憶されたコンピュータ可読媒体におけるインプリメンテーションを含むことができる。たとえば、いくつかの実施態様について、コンピュータシステム上にインプリメント可能なＳＢＳシステムおよび方法を使用して検出される信号の誤差を

処理および補正する実施態様を以下で詳細に説明する。

【００３６】

コンピュータとしては、任意のタイプのコンピュータプラットフォーム、たとえばワークステーション、パーソナルコンピュータ、サーバ、または現在もしくは将来の任意のその他のコンピュータが挙げられる。コンピュータは、一般に、プロセッサ、オペレーティングシステム、システムメモリ、メモリ記憶デバイス、入出力制御装置、入出力デバイス、およびディスプレーデバイスなど、公知の構成要素を備える。当業者は、コンピュータには可能性のある多くの構成および要素があり、データバックアップユニット、および多くのその他のデバイスも含むことができることを理解するであろう。

【００３７】

ディスプレーデバイスは、視覚情報を提供するディスプレーデバイスを備えることができ、この情報は、一般に、画素のアレイとして論理的および／または物理的に構成することができる。インターフェース制御装置を備えることができ、こうした制御装置は、入力および出力インターフェースを提供するための公知または将来の多様なソフトウェアプログラムの何れかを含むことができる。たとえば、インターフェースは、一般に、「グラフィカルユーザインターフェース」（多くの場合、ＧＵＩと呼ばれる）と呼ばれ、１つまたは複数のグラフィック表現をユーザに提供するものでよい。インターフェースは、一般に、当業者が周知している選択または入力手段を使用して、ユーザ入力を受け入れることを可能にする。

【００３８】

同じか、または別の実施態様では、コンピュータ上のアプリケーションは、「コマンドラインインターフェース」（多くの場合、ＣＬＩという）と呼ばれるものを含むインターフェースを使用する。ＣＬＩは、一般に、アプリケーションとユーザとの間にテキストベースの対話を提供する。一般に、コマンドラインインターフェースは、ディスプレーデバイスを介して、テキスト行として出力を提示し、入力を受信する。たとえば、インプリメンテーションによっては、「シェル」と呼ばれるもの、たとえば、当業者が周知しているＵｎｉｘ（登録商標）シェル、またはオブジェクト指向タイプのプログラミングアーキテクチャ、たとえばＭｉｃｒｏｓｏｆｔ．ＮＥＴフレームワークを使用するＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）Ｐｏｗｅｒｓｈｅｌｌを含むことができる。

【００３９】

当業者は、これらのインターフェースが、１つまたは複数のＧＵＩ、ＣＬＩ、またはこれらの組合せを含み得ることを理解するであろう。

【００４０】

プロセッサとしては、Ｉｎｔｅｌ　Ｃｏｒｐｏｒａｔｉｏｎが製造するＩｔａｎｉｕｍ（登録商標）もしくはＰｅｎｔｉｕｍ（登録商標）、Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓが製造するＳＰＡＲＣ（登録商標）、ＡＭＤ　ｃｏｒｐｏｒａｔｉｏｎが製造するＡｔｈａｌｏｎ（商標）もしくはＯｐｔｅｒｏｎ（商標）などの市販のプロセッサが挙げられ、または現在もしくは将来入手可能なその他のプロセッサの何れかで良い。また、プロセッサの実施態様によっては、マルチコアプロセッサと呼ばれるものを備えることができるか、および／またはシングルもしくはマルチコア構成の並列処理技術を使用することを可能にする。たとえば、マルチコアアーキテクチャは、一般に、２つ以上のプロセッサの「実行コア」を含む。本明細書の例では、各々の実行コアは、複数のスレッドの並列実行を可能にする独立するプロセッサとして機能する。さらに、当業者は、プロセッサが、一般に、３２もしくは６４ビットアーキテクチャ、または現在公知であるか、もしくは将来開発されるその他のアーキテクチャ構成として構成し得ることを理解するであろう。

【００４１】

プロセッサは、一般にオペレーティングシステムを実行するが、オペレーティングシステムは、たとえば、Ｍｉｃｒｏｓｏｆｔ　Ｃｏｒｐｏｒａｔｉｏｎが市販するＷｉｎｄｏｗｓ（登録商標）タイプのオペレーティングシステム（たとえばＷｉｎｄｏｗｓ（登録商標）ＸＰもしくはＷｉｎｄｏｗｓ（登録商標）　Ｖｉｓｔａ）；Ａｐｐｌｅ　Ｃｏｍｐｕ

ｔｅｒ Ｃｏｒｐ．が市販するＭａｃ ＯＳ Ｘオペレーティングシステム（たとえば７．５ Ｍａｃ ＯＳ Ｘ ｖｌ ０．４「Ｔｉｇｅｒ」もしくは７．６ Ｍａｃ ＯＳ Ｘ ｖｌ ０．５「Ｌｅｏｐａｒｄ」オペレーティングシステム）；多くのベンダー、もしくはオープンソースから市販されているＵｎｉｘ（登録商標）もしくはＬｉｎｕｘ（登録商標）オペレーティングシステム；別の、もしくは将来のオペレーティングシステム；またはこれらの何らかの組合せでよい。オペレーティングシステムは、公知の方法でファームウェアおよびハードウェアとインターフェースし、プロセッサが、様々なプログラミング言語で書くことができる様々なコンピュータプログラムの機能を調整および実行することを促進する。オペレーティングシステムは、一般にプロセッサと協働して、コンピュータのその他の構成要素の機能を調整および実行する。オペレーティングシステムは、すべて公知の技術に従って、スケジューリング、入出力制御、ファイルおよびデータ管理、メモリ管理、並びに通信制御および関連サービスも提供する。

【００４２】

システムメモリは、公知または将来の様々なメモリ記憶デバイスの何れかを含むことができる。例としては、一般に入手可能なランダムアクセスメモリ（ＲＡＭ）、常駐ハードディスクもしくはテープなどの磁気媒体、読み書きコンパクトディスクなどの光媒体、またはその他のメモリ記憶デバイスが挙げられる。メモリ記憶デバイスとしては、公知もしくは将来の様々なデバイスの何れか、たとえばコンパクトディスクドライブ、テープドライブ、リムーバブルハードディスクドライブ、ＵＳＢもしくはフラッシュドライブ、またはディスケットドライブが挙げられる。こうしたタイプのメモリ記憶デバイスは、一般に、プログラム記憶媒体（図示しない）、たとえばそれぞれコンパクトディスク、磁気テープ、リムーバブルハードディスク、ＵＳＢもしくはフラッシュドライブ、またはフロッピー（登録商標）ディスケットから読み込むか、および／またはこれらに書き込む。これらのプログラム記憶媒体、または現在使用されているか、もしくは後に開発され得るその他の媒体の何れかは、コンピュータプログラム製品と考えることができる。評価されるとおり、これらのプログラム記憶媒体は、一般に、コンピュータソフトウェアプログラムおよび／またはデータを記憶する。コンピュータ制御論理とも呼ばれるコンピュータソフトウェアプログラムは、一般に、メモリ記憶デバイスと関連して使用されるシステムメモリおよび／またはプログラム記憶デバイスに記憶される。

【００４３】

実施態様によっては、コンピュータが使用可能な媒体を含むコンピュータプログラム製品であって、制御論理（プログラムコードを含むコンピュータソフトウェアプログラム）が内部に記憶された製品について説明する。プロセッサが制御論理を実行する場合、プロセッサは、制御論理によって、本明細書に記載する機能を実行する。その他の実施態様では、いくつかの機能は、たとえばハードウェア状態マシンを使用して、主にハードウェア内にインプリメントされる。本明細書に記載する機能を実行するためのハードウェア状態マシンインプリメンテーションは、当業者にとっては明らかであろう。

【００４４】

入出力制御装置は、ヒューマンであるかマシンであるか、ローカルであるかリモートであるかに関わりなく、ユーザからの情報を受け取って処理するための多様な公知のデバイスの何れかを含むことができる。こうしたデバイスとしては、たとえば、モデムカード、ワイヤレスカード、ネットワークインターフェースカード、音声カード、または公知の様々な入力デバイスの何れかのためのその他のタイプの制御装置が挙げられる。出力制御装置は、ヒューマンであるかマシンであるか、ローカルであるかリモートであるかに関わりなく、ユーザに情報を提示するための多様な公知のディスプレーデバイスの何れかを含むことができる。本明細書に記載する実施態様では、コンピュータの機能構成要素は、システムバスを介して互いに通信する。コンピュータの実施態様によっては、ネットワーク、またはその他のタイプのリモート通信を使用して、いくつかの機能構成要素と通信することができる。

【００４５】

当業者には明らかなとおり、機器制御および／またはデータ処理アプリケーションは、ソフトウェアにインプリメントする場合、システムメモリおよび／またはメモリ記憶デバイスにロードされ、これらから実行することができる。機器制御および／またはデータ処理アプリケーションの全部または一部も、メモリ記憶デバイスの読出し専用メモリ、または類似のデバイスに常駐することができ、こうしたデバイスは、入出力制御装置を介して、機器制御および／またはデータ処理アプリケーションを最初にロードする必要はない。当業者には、機器制御および／またはデータ処理アプリケーション、またはその一部は、公知の方法で、システムメモリ、またはキャッシュメモリ、またはこれらの両方にロードすることができ、有利であることは明らかであろう。

【００４６】

また、コンピュータは、システムメモリに記憶された１つまたは複数のライブラリファイル、実験データファイル、およびインターネットクライアントを含み得る。たとえば、実験データは、１つまたは複数の実験またはアッセイに関連するデータ、たとえば検出された信号値、または１つまたは複数のＳＢＳ実験またはプロセスに関連するその他の値を含むことができる。さらに、インターネットクライアントは、ネットワークを使用して、別のコンピュータ上のリモートサービスにアクセスすることを可能にするアプリケーションを含むことができ、たとえば、「ウェブブラウザ」と一般に呼ばれるものを含むことができる。本明細書の例では、通常使用されるいくつかのウェブブラウザとしては、Ｎｅｔｓｃａｐｅ Ｃｏｍｍｕｎｉｃａｔｉｏｎｓ Ｃｏｒｐ．から市販されているＮｅｔｓｃａｐｅ（登録商標）８．１．２、Ｍｉｃｒｏｓｏｆｔ Ｃｏｒｐｏｒａｔｉｏｎから市販されているＭｉｃｒｏｓｏｆｔ（登録商標）Ｉｎｔｅｒｎｅｔ Ｅｘｐｌｏｒｅｒ７、Ｍｏｚｉｌｌａ Ｃｏｒｐｏｒａｔｉｏｎから市販されているＭｏｚｉｌｌａ Ｆｉｒｅｆｏｘ（登録商標）２、Ａｐｐｌｅ Ｃｏｍｐｕｔｅｒ Ｃｏｒｐから市販されているＳａｆａｒｉ１．２、または現在公知であるか、もしくは将来開発されるその他のタイプのウェブブラウザが挙げられる。また、同じか、またはその他の実施態様で、インターネットクライアントは、ＳＢＳアプリケーション用のデータ処理アプリケーションなどのネットワークを介して、リモート情報にアクセスすることが可能な特殊化されたソフトウェアアプリケーションを含むことができるか、またはこうしたソフトウェアアプリケーションでよい。

【００４７】

ネットワークは、当業者が十分に周知している多くの様々なタイプのネットワークの１つまたは複数を含むことができる。たとえば、ネットワークは、通常プロトコルスィートと呼ばれるものを使用するローカルまたはワイドエリアネットワークを含むことができる。ネットワークとしては、一般にインターネットと呼ばれる相互接続されたコンピュータネットワークの世界的なシステムを含むことができるか、または様々なイントラネットアーキテクチャも含むことができる。当業者は、ネットワーク環境にあるユーザによっては、一般に「ファイアウォール」と呼ばれるもの（場合によりパケットフィルタ、またはボーダー保護デバイスと呼ばれる）を使用して、ハードウェアおよび／またはソフトウェアシステムとのまの情報トラフィックを制御することを好む場合があることも理解するであろう。たとえば、ファイアウォールは、ハードウェアもしくはソフトウェア、またはこれらの組合せを含むことができ、一般に、たとえばネットワーク管理者などのユーザがセキュリティ基本方針を導入するように設計される。

【００４８】

ＳＢＳ実施態様の例は、一般に、ヌクレオチド種を上記のテンプレート分子に連続的に、または繰り返し添加するサイクルを使用する。これらのサイクルは、本明細書では、「フロー」と呼ぶ。たとえば、各々のフローでは、４つのヌクレオチド種Ａ、Ｇ，ＣまたはＴの何れか１つが提示されるか（ピロリン酸塩（ＰＰｉ）配列決定法の場合）、または４つのヌクレオチド種すべてがともに、テンプレート／ポリメラーゼ複合体に提示される（たとえば、各々のヌクレオチド種に関連する異なる標識を使用する配列決定法の場合）。この例を続けると、フローは、合成される新生分子の３’末端に直接隣接する配列位置に

おいて、テンプレート分子のヌクレオチド種に対して相補的なヌクレオチド種を含むことができ、この場合、ヌクレオチド種は、新生分子内に組み込まれる。本明細書の例では、ヌクレオチド種の組込みは、光信号（たとえば、発光もしくは蛍光などのプロセスから生成される光を含み得る光信号）、またはマスタグなどのその他の信号の形式で検出することができる。ヌクレオチド種の流れの繰返し後、洗浄方法がインプリメントされて、組み込まれていない過剰なヌクレオチド種および試薬が除去される。洗浄段階が完了した後、フローの次の繰返しは、別のヌクレオチド種、またはヌクレオチド種の混合物をテンプレート／ポリメラーゼ複合体に提示する。実施態様によっては、「フローサイクル」は、４つのヌクレオチド種を繰返し、または並行して添加することを意味する場合があり、たとえば、１つのフローサイクルは、４つのヌクレオチド種すべてを添加することを含む。

【００４９】

フローグラムに記録する場合、各々の流れについて検出される光またはその他の信号の値は、約ゼロであるか（フロー中のヌクレオチド種が、次の配列位置において、テンプレートのヌクレオチド種に対して相補的ではなく、その結果組み込まれていないことを示す）、約１であるか（テンプレートのヌクレオチド種に対して相補的な正確に１つのヌクレオチド種が組み込まれていることが検出されたことを示す）、またはほぼ１より大きい整数（フロー中に提示され、テンプレートの２つの連続するヌクレオチド種に対して相補的なヌクレオチド種の２つ以上のコピーの組込みが検出されたことを示す）でよい。

【００５０】

上記のとおり、繰り返す一連のフローの理論上の結果は、各々のフローからの信号であり、この信号は正確にゼロであるか、または整数であり、完全なフローグラムに表現されなければならない。ＣＦおよびＩＥメカニズムを含む様々な実験の変動により、検出される実際の信号は、これらの予想理論値付近で変化量だけ変動する傾向がある。この変化量を含む検出信号は、ダーティまたは観察フローグラムとして表現される。

【００５１】

フローグラムおよびパイログラムという用語は、本明細書では互換可能に使用される。「完全なフローグラム」、「クリーンなフローグラム」、および「理論上のフローグラム」という用語は、本明細書では互換可能に使用される。「ダーティなフローグラム」、「現実のフローグラム」、および「観察されたフローグラム」という用語は、本明細書では互換可能に使用される。

【００５２】

さらに、本明細書で使用する場合、「読込み」は、一般に、単一の核酸テンプレート分子、または複数の実質的に同じテンプレート分子のコピーの集団から得られる全体の配列データを意味する。「新生分子」は、一般に、テンプレート分子の対応するヌクレオチド種に対して相補的なヌクレオチド種を組み込むことによりテンプレート依存ＤＮＡポリメラーゼによって伸長されるＤＮＡ鎖を意味する。本明細書で使用する「完了効率」は、一般に、一定のフロー時に適切に伸長される新生分子の割合を意味する。本明細書で使用する「不完全伸長率」は、一般に、すべての新生分子の数に対して、適切に伸長しない新生分子の数の割合を意味する。

【００５３】

本明細書に記載するいくつかの実施態様は、上記のＣＦおよびＩＥメカニズムを考慮に入れて、各々のフローの検出された信号を補正する。たとえば、本発明の一態様は、ＣＦおよびＩＥの一定のレベルを仮定して、任意の公知の配列に関する位相同期の損失程度を計算することを含む。

【００５４】

以下に示す表１は、ＩＥおよびＣＦに関して数学的にモデル化した閾値の一例を示し、様々な読込み長さに対して９９％以上の精度を提供する（つまり、読込みは、テンプレート分子の実際の配列の少なくとも９９％を表す。表１に示す予測値は、様々な読込み長さの配列決定精度、および約９９％の読込み精度を達成することを許容可能なＩＥおよびＣＦ誤差の程度に対するＣＦおよびＩＥ効果の影響を示す。表１は、補正されていない読込

みに関して１％を越えないＣＦ率が許容可能であり（ＩＥが、その集団でゼロに等しいと仮定する）、約１００の配列位置の読込み長さは９９％正確である（つまり、９９％以上の完了効率）ことを示す。さらに、０．２５％以下のＩＥ率が許容可能であり（ＣＦ率がゼロに等しいと仮定する）、約１００の配列位置の読込み長さは９９％正確である。

【００５５】

【表１】

表1. 様々な読込み長さにおいて、99%の精度を生じる誤差予測値

読込み長さ(塩基対)	100		200		400	
不完全な伸長	0.0	**0.0025**	0.0	**0.0013**	0.0	**0.0007**
繰越	**0.01**	0.0	**0.005**	0.0	**0.003**	0.00
予測精度	~99%	~99%	~99%	~99%	~99%	~99%

表１に記載する値は、単に具体的に示すためであり、制限するものであると解釈すべきではないことがわかるであろう。当業者は、いくつかの要素は、予測を明確するためのゲノム配列または基準配列、およびその他のパラメータなどの値の変動性の一因となり得ることを理解するであろう。たとえば、ＳＢＳ法の代表的な実施態様は、一般に、１～２％の範囲のＣＦ率を達成し、ＩＥ率は０．１～０．４％である（つまり、完了効率は９９．６～９９．９％の範囲である）。上記のとおり、ＣＦおよびＩＥの補正は望ましく、なぜなら、位相同期の損失が読込み長さ全体に蓄積効果を有し、読込み長さが増加するにつれて、読込みの品質を低下させるからである。

【００５６】

本明細書に記載する発明の一実施態様では、ＣＦおよびＩＥの両方を表す値は、実質的に同じテンプレート分子の集団、たとえばｐｉｃｏｔｉｔｅｒｐｌａｔｅシステムの単一ウェル内に存在するテンプレート分子の集団の読込み全体で、実質的に一定であると仮定される。その結果、テンプレート分子の実際の配列の何らかの先験的な知識がなくても、２つの単純なパラメータ「不完全な伸長」および「繰越」を使用して、全体の読込みにおける各々の配列位置の数値補正が可能になる。本発明について本明細書に記載するシステムおよび方法は、テンプレート分子の集団内で生じるＣＦおよびＩＥの量を判断し、補正する際に有用である。たとえば、本発明の実施態様は、各々のウェル内に存在して、ＣＦおよびＩＥの一因となる実質的に同じテンプレート分子の各々の集団に関して、各々のフローから検出される信号値を補正する。

【００５７】

本発明の実施態様は、位相同期の欠如を非線形マッピングとしてモデル化する：

方程式（１）：

$$M(p, \varepsilon, \lambda) = q$$

ここで：

－ Ｍは、ＣＡＦＩＥマッピングであり
－ ｐは、仮定上の「完全な」フローグラム［アレイとして］であり
－ λは、完了効率パラメータであり
－ εは、繰越パラメータであり
－ ｑは、「ダーティな」フローグラム［アレイとして］である。

【００５８】

理論上の「完全な」フローグラムは、方程式（１）で与えられるマッピングモデル式を

使用して、現実の「ダーティな」フローグラムに変換し、ＩＥおよびＣＦを概算することができる。こうしたマッピング式のモデルは、たとえば、公知の配列を有するポリヌクレオチドテンプレート分子を配列決定することによって、観察されたフローグラム（ｑ）に導入される誤差を分析して生成することができる。方程式（１）によって与えられる数学モデルの具体的な一例を図１に示す。

【００５９】

たとえば、図１の左側では、理論上のフローグラム１０１は、理論上の（完全または理想的な）フローグラム（ｐ）の実例表現であり、関連するヌクレオチド種に隣接して括弧内に表された理想化信号強度値を示す。理論上のフローグラム１０１の各々の理想化値は、ある整数またはゼロである。本明細書の例では、「１」の値は、単一のヌクレオチドの組込みによって誘発された１００％検出信号強度を表し、「０」は、０％信号を表す（たとえば、１００万の実質的に同じテンプレート分子、および１００万の新生分子の集団を含むウェル中、「１」は、すべての新生分子が、単一ヌクレオチドによって伸長する時に誘発される信号を表し、「２」は、すべての新生分子が、２つのヌクレオチドによって伸長する時に誘発される信号を表す）。

【００６０】

図１の右側では、観察されたフローグラム１０３は、観察された（または模擬ダーティ）フローグラム（ｑ）からの検出信号強度値の実例表現である。同様に、フローグラム１０３の各々の信号強度値は、関連するヌクレオチド種に隣接して括弧内に表される。また、図１の右側には、ヌクレオチド種および信号値に関連する繰返しフロー（ｆｌｏｗ）配列を表す代表的な数を提供するフロー１０５がある（たとえば、フロー１０５の各々の繰返しは、ヌクレオチド種の添加、およびその後の洗浄プロセスを表す）。たとえば、図１に示されているフロー１は、フロー１０５の前記の繰返しで導入される「Ｃ」ヌクレオチド種に関連し、理論上のフローグラム１０１、および観察されたフローグラム１０３の両方の信号値に対応する。

【００６１】

図１の例では、理論上のフローグラム１０１と、観察されたフローグラム１０３との間の信号強度値の差は、各々のフロー１０５の繰返しでは、少なくとも部分的に位相同期の損失を表す。たとえば、観察されたフローグラム１０３に表される信号値は整数ではなく、フロー１０５の同じ繰返しでは、一般に、理論上のフローグラム１０１に表される理想値よりわずかに大きいか、またはわずかに小さい。

【００６２】

「Ｍ」として表現されるマッピングモデル１１０は、パラメータ１１３の既知の値を使用して概算することができる。たとえば、パラメータ１１３は、ε（繰越）パラメータおよびλ（完了効率）パラメータを含む。パラメータ１１３は、マッピングモデル１１０を概算し、理論上のフローグラム（ｐ）１０１の信号値を観察値（ｑ）１０３に変換するために使用し得る。本明細書の例では、マッピングモデル１１０によって表される誤差値は、フロー１０５の各々の繰返しで蓄積し、指数関数的に成長する。

【００６３】

上記の例を続けると、誤差値によって表現される誤差は、理論的には、各々のフローとともに指数関数的に増加する。たとえば、実質的に同じテンプレート分子の各々の集団に関連する位相同期した配列決定反応は、フローの繰返し後、位相同期した３つの異なる亜集団になる。この亜集団は、フロー中のヌクレオチド種が、テンプレート分子に対して適切な配列位置に適切に組み込まれる位相同期反応の第１の亜集団と（たとえば、ＣＡＦＩＥ効果はない）、ＣＦメカニズムによる不適切な組込みが生じ、反応が、第１集団に対する配列位置より先に進む位相同期反応の第２亜集団と、ＩＥメカニズムによる不適切な組込みが生じ、反応が、第１集団の配列位置より遅れる位相同期反応の第３亜集団とを含む。本明細書の例では、次のフロー繰返しで、３つの亜亜集団は、上記の３つの亜集団の各々から生じるという具合である。当業者は、ｎ番目のフローの繰返しで、各々がフローｎで信号を生じる位相同期の3^{n}の集団があることを理解するであろう。

【００６４】
さらに上記の例を続けると、図２は、逆マッピングモデル２１０として図２に表されるマッピングモデル１１０の逆の実例表現を提供する。たとえば、パラメータ１１３の正確な値を概算することによって（たとえば、ε（繰越）およびλ（完了効率）パラメータの両方の値）、観察されたフローグラム（ｑ）１０３の信号値は逆にして、理論上のフローグラム（ｐ）１０１の信号値を与える。
【００６５】
当業者は、図１および２に表される信号値は、単に具体的に示すために記載されており、広範な信号値が可能であることを理解するであろう。したがって、これらの信号値は、制限するものであると解釈するべきではない。
【００６６】
本発明のいくつかの実施態様は、以下に概略を示す２つの連続的な段階（ｉ）および（ｉｉ）で、反転したマッピングを実行する：
各々のヌクレオチド種のフローｉについて：
（ｉ）　－　ヌクレオチド種の添加により、新生分子を伸長する：
【００６７】
【数１】

$$\left\{\begin{array}{c} q_i = \lambda \sum_j m_j p_j \\ (m_j, m_{j'}) \leftarrow (m_j, m_{j'}) + \lambda(-1,1)\, m_j p_j \end{array}\right\}$$

すべてのｊについて、$N_j = N_i$および$p_j > 0$
（ｉｉ）　－　前の添加から残るヌクレオチド種により、新生分子を伸長する：
【００６８】
【数２】

$$\left\{\begin{array}{c} q_i \leftarrow q_j + \varepsilon \sum_j m_j p_j \\ (m_j, m_{j'}) \leftarrow (m_j, m_{j'}) + \varepsilon(-1,1)\, m_j p_j \end{array}\right\}$$

すべてのｊについて、$N_j = N_{i-1}$および$p_j > 0$
ここで：
－　p_iは、ｉ番目のヌクレオチド種フローで、理論上の（クリーンな）フローグラムの信号値である
－　q_iは、ｉ番目のヌクレオチド種フローで、観察された（ダーティな）フローグラムの信号値である
－　m_iは、ｉ番目のヌクレオチド種フローのフローグラム配列位置で、組込みに使用できるヌクレオチド種分子の画分である
－　N_iは、ｉ番目のヌクレオチド種添加（Ａ、Ｃ、Ｇ、またはＴ）である
－　（ｊ，ｊ’）は対の指数であり、$p_{j'}$はフローグラム上のp_jの次の正の値である。
【００６９】
マッピングモデルは、これらの計算をフローごとに実行し（たとえば、フロー１０５の繰返し）、観察されたフローグラム（ｑ）、およびテンプレート分子の画分ｍを段階（ｉ）および（ｉｉ）により更新する。
【００７０】
図３ａは、マトリックスの計算に使用されるモデルの具体的な例を示す。たとえば、以下にさらに詳細に説明するように、順方向マトリックスモデル３１０は、逆マトリックスモデル３２０を導くために使用することができる。本明細書の例では、逆マトリックスモデル３２０を使用してマトリックスを計算することは、パラメータ１１３の概算を導くために使用することができる。たとえば、パラメータ１１３の様々な値は、マトリックスの

計算に適用して、観察されたフローグラム１０３に対する適合程度を評価することができる。一般に、観察されたフローグラム（ｑ）１０３に最適なパラメータ１１３は、パラメータ１１３の実効値として良い概算値であるように決定される。

【００７１】

さらに、図３ｂは、順方向マトリックスモデル３１０を使用する順方向マトリックスの計算の具体的な例を示す。本明細書の例では、観察されたフローグラム（ｑ）１０３は、完了効率値λ＝０．９５、および繰越値ε＝０．０５を含むパラメータ１１３を使用するマトリックスの計算によって生成される。マトリックスのフロー１０５の繰返しに関連する各々の行は、各々のヌクレオチド種のフローに関する再帰的な段階（ｉ、ｉｉ）の実施および結果を記録する。

【００７２】

方程式（１）および再帰的な段階（ｉ、ｉｉ）は、マトリックスアレイの演算として書き換えることができる：

方程式（２）：

$$[M(p', \varepsilon, \lambda)] * p = q$$

ここで：

－　［Ｍ（ｐ’，ε，λ）］は、マトリックスである

－　＊は、マトリックスアレイの乗算である

－　ｐ’＝ｓｇｎ（ｐ）は、理論上つまり「完全な」フローグラムの２進コード化である（たとえば、図１のフローグラムｐ、ｐ＝［０１０２００１０３０１２］は、ｐ’＝［０１０１００１０１０１１］）としてコード化されるであろう）。

【００７３】

方程式（２）の逆形式は逆のマッピングを与え、「ダーティな」観察されたフローグラム（ｑ）１０３を逆に理論上のフローグラム（ｐ）１０１に変換する：

方程式（３）：

$$p = [M^{-1}(p', \varepsilon, \lambda)] * q$$

ここで：

－　［M^{-1}（ｐ’，ε，λ）］は、（集合論的）逆マトリックスである。

【００７４】

繰返し法は、図３ａに逆マトリックスモデル３２０として示す逆方程式（３）を解き、各々の読込みに関して理論上のフローグラム（ｐ）１０１を得るために使用される。この繰返しは、ＣＡＦＩＥの反転に関してパラメータ１１３（ε，λ）の一定の対で実行される：

方程式（４）：

$$p^{(n+1)} = [M^{-1}(p'^{(n)}, \varepsilon, \lambda] * q$$

ここで、$p'^{(n)} \equiv sgn(p^{(n)} - 閾値)$および$p^{(1)} \equiv q$は、計算の種として使用される。閾値の値は、システムの信号対雑音比によって決まる。

【００７５】

図３ｂと同様、図４ａは、逆マトリックスモデル３２０を使用する逆マトリックスの計算の具体的な例を示す。本明細書の例では、理論上のクリーンなフローグラム（ｐ）１０１は、完了効率値λ＝０．９５、および繰越値ε＝０．０５を含むパラメータ１１３を使用して、観察されたダーティなフローグラム（ｑ）１０３から生成される。

【００７６】

たとえば、あるインプリメンテーションでは、固定値、閾値≡０．２が使用される。こうしたインプリメンテーションでは、フローグラムｐ’の２進コード化は、フローグラム値ｐが０．２より大きい場合に値「１」をコード化し、フローグラム値ｐが０．２以下の場合に値「０」をコード化する。本明細書の例では、閾値０．２は信号対雑音比の概算である。

【００７７】

あるいは、いくつかのインプリメンテーションは、０～１の範囲、たとえば０．０５、

０．１、または０．３の閾値を使用し得る。したがって、「ダーティな」観察されたフローグラム（ｑ）１０３は、パラメータ１１３の一定の対（ε、λ）に関する方程式（４）により、クリーンな「完全な」理論上のフローグラム（ｐ）１０１に反転させることができる。多くのインプリメンテーションでは、一般に、フローグラムの反転の単一の繰返しで十分である。インプリメンテーションによっては、フローグラムの反転の２回、３回、またはそれ以上の繰返しを実行することが望ましい場合があり、フローグラム表現の精度は、特に読込み長さがより長い場合、計算が所望の品質で解に収束するまで、各々の繰返しで改善することができる。好ましい実施態様では、フローグラムの反転の１回の繰返し、または２回の繰返しは、計算効率の点で実行することができる。また、コンピュータコードによってインプリメントされる本発明のいくつかの実施態様は、ユーザが、多くの繰返しを選択し、ユーザの選択に応じて各々の繰返しを実行するか、および／または連続的に実行することを可能にする。たとえば、ユーザは、１つまたは複数の領域に値を入力するか、またはＧＵＩで提示されるボタンを選択するなど、先行技術で公知の方法を使用して選択を行うことができる。本明細書の例では、ユーザは、実行する多くの繰返しを指示する値を入力し得るか、および／またはユーザは、本発明の繰返しを実行するボタンを選択し得る。さらに、ユーザは、データ品質の指示を選択し、本発明を繰り返してデータ品質のレベルを達成することができる。

【００７８】

図４ｂは、方程式（４）の方法を使用して、連続する繰返し数で結果を改善する方法の具体的な一例を示す。未処理のフローグラム４１０は、各々の繰返しがフローバー４０９によって表されるヌクレオチド種添加の３３６回の繰返しから、完了効率値λ＝０．９９７、および繰越値ε＝０．０３を含むパラメータ値１１３を有する観察されたフローグラム（ｑ）１０３の一実施態様を示す。たとえば、各々のフローバー４０９は、ヌクレオチド種のフローを表し、各々の種は、特に、バー４０９の色またはパターンによって表される。さらに、各々のフローに関連する検出または補正された信号値は、信号強度４０５によって与えられるスケールに対するバー４０９の高さによって表される。

【００７９】

当業者は、特に、読込み長さ４０７によって与えられるスケールに関して、配列位置が５０を超える読込み長さの場合、フローバー４０９の信号強度４０５の値に関して、未処理フローグラム４１０に強度の変動があることを理解するであろう。つまり、フローバー４０９の大部分の信号値は、整数である信号値を含まない。２回の繰返しフローグラム４２０は、本発明の一実施態様を使用する２回繰り返される補正後、観察されたフローグラム（ｑ）１０３の同じ実施態様を示す。フローバー４０９の信号強度４０５の一貫性は、特に、読込み長さ４０７の位置が１５０以下のフローバー４０９の場合に改善される。同様に、データ品質の改善は、それぞれ４回繰り返されるフローグラム４３０、および８回繰り返されるフローグラム４４０で実証され、フローグラム４４０は、実質的にすべてのフローバー４０９が一貫性および整数値を示すことを表す。

【００８０】

いくつの実施態様では、パラメータ１１３の値の概算は、方程式（４）を使用して決定される。たとえば、完了効率パラメータ（λ）に最適な値は、方程式（４）を使用してテスト計算を実行し、異なる値を完了効率パラメータとして入力し、固定値をＣＦパラメータとして使用することにより決定し得る。本明細書の例では、一定のＣＦ値ε＝０を有するλ＝１、０．９９９、０．９９８、．．．、０．９９０の値を連続して使用し、各々の結果を得ることができる。様々な実施態様では、入力λの値間の０．００１の間隔は、たとえば０．０５、０．０１、０．００５、０．００１、０．０００５、０．０００１などの間隔値など、他の間隔と置き換えることができる。

【００８１】

この例を続けると、計算した理論上のフローグラム（ｐ）のフローバー４０９の何らかの信号値４０５が、λの入力値を使用して方程式（４）を解いた後にゼロ未満になる場合、λの値は、最適完了効率パラメータの値として宣言される。λの最適値が決定された後

、実質的により小さいλ値を使用すると、「過剰適合」と呼ばれる状態になり、人為的に負のフロー信号を生成する。また、本明細書の例では、ホモポリマーを表すフローバー４０９の長い列（たとえば、一連の配列位置は、同じヌクレオチド種を含む）の後の一連の配列位置におけるいくつかのフローバー４０９の場合、補正された信号値４０５はゼロ未満になり得る。このゼロ交差点は、図５の楕円５０３内に示され、最適な完了効率は、以下λとして指示する。

【００８２】

同様に、実施態様によっては、ＣＦの作用は類似の手法で対処し得る。たとえば、ＣＦパラメータの値はテストすることができ、たとえば、完了効率パラメータλは、前に発見された値λ＊に定められた状態で、ε＝０、０．００２５、０．００５、０．００７５、０．０１、．．．、０．０４の値を含み得る。これは、図５にステップ２→３として表され、楕円５０３は、開始位置２（ε，λ）＝（０，λ＊）を指示する。本明細書の例では、εの入力値間の０．００２５という間隔は、具体的に示すために提示するのであり、たとえば０．０５、０．０１、０．００５、０．００１、０．０００５、０．０００１、０．００００１などの他の小さい間隔値に置き換えることができる。計算された理論上のフローグラム（ｐ）中のフローバー４０９の任意の信号値４０５が、εの入力値を使用して方程式（４）を解いた後にゼロ未満になる場合（たとえば、λ経路に沿った調査の際にゼロ未満になるフローバー４０９の信号値４０５以外の、フローバー４０９の任意の信号値４０５）、εの値は、最適なＣＦパラメータの値として宣言される。εの最適値が決定された後、その後より大きい値を使用すると、過剰適合の状態になり、人為的に負のフロー信号を生成する。また、本明細書の例では、ホモポリマーを表すフローバー４０９の長い列の前のある配列位置におけるフローバー４０９の場合、補正された信号値４０５はゼロ未満になり得る。このゼロ交差点は、図５の楕円５０５内に示され、最適なＣＦは、以下ε＊として指示する。

【００８３】

図５は、具体的な例を示し、たとえば、横座標は完了効率軸５２０を表し、縦座標はＣＦ軸５１０を表す。楕円５１０、５０３および５０５内のグラフは各々、上記のステップを表し、３つの信号を示すフローグラムの例示的な部分を含む。たとえば、中心のバーは、主信号バー５３７を表し、左の小さい信号（ＣＦバー５３５）、および右の小さい信号（ＩＥバー５３３）が側面に位置する。楕円５０１は、最初の観察されたフローグラム（ｑ）１０３のステップを示し、主信号バー５３７は位相非同期によって減少し、ＣＦバー５３５およびＩＥバー５３３の小さい信号は、位相非同期によって生じる雑音を表す。楕円５０３は、ＩＥが補正された時のステップを表し、ＩＥバー５３３ａに関連する信号はなくなり、中心の主信号バー５３７は相応に増加する。上記のとおり、ＩＥが補正された点は、たとえば、最適完了効率パラメータのゼロ交差点を含み、λ＊として指示することができる。楕円５０５は、ＣＦが補正されたさらに他のステップを表しており、ＣＦバー５３５ａに関連する信号は除去され、中心の主信号バー５３７は相応に増加する。上記のとおり、ＣＦが補正された点は、たとえば、最適完了効率パラメータのゼロ交差点を含み、ε＊として指示することができる。楕円５０５は、理論上の予測されたフローグラムの概算である補正の結果を示し、位相非同期の誤差に起因する雑音は実質的に除去されている。

【００８４】

したがって、ＣＦおよびＩＥの量は、基礎となるテンプレート分子の配列ｐは、先験的に未知であり、本発明の方法は、完全な新しい分析モードで使用することができる。本発明を実施するために、ポリメラーゼの組込み効率（つまり、λ）、またはヌクレオチド洗浄効率（つまり、ε）に関する事前の知識は不要であり、基準の何らかのヌクレオチド配列も不要である。

【００８５】

実施態様によっては、上記のパラメータ概算の検索プロセスは、すべてのεおよびλの入力検索間隔で段階（ｉ、ｉｉ）を通してマトリックス［Ｍ］を構成し、計算効率の点か

ら制限している。こうした制限は、少なくとも部分的に、マトリックス構成演算に概算を使用することによって克服することができる。たとえば、すべての検索間隔でマトリックスを再構成することを防止し、その結果、計算速度を大幅に改善することができる。２つのこのような方法について、以下で説明する：

方法１：

εおよび（１－λ）の小さい値では（たとえば、（１－λ）≦０．００１およびε≦．００２５）、マトリックス［Ｍ］は分解され、ある形式に近似される：

方程式（５）：

$$[M(p', \varepsilon, \lambda)] \sim [L(p', \Delta\lambda)]^{\phi} * [U(p', \Delta\varepsilon)]^{\omega}$$

ここで：

－　Δε＝０．００２５およびΔλ＝０．００１は、それぞれεおよびλ軸における間隔である

－　φおよびωはマトリックスパワーであり、ω～ε／Δεおよびφ～（１－λ）／Δλの特性を有する

－　［Ｌ（ｐ’，Δλ）］は下方の対角マトリックスであり、わずかな欠如ΔλにおけるＩＥの作用をモデル化する

－　［Ｕ（ｐ’，Δλ）］は、上方の対角マトリックスであり、わずかな欠如ΔεにおけるＣＦの作用をモデル化する。

【００８６】

この分解により、方程式（５）は、検索経路に沿って一度、下方の対角マトリックスＬおよび上方の対角マトリックスＵを構成し、検索グリッド（ε，λ）における不完全および繰越の程度は、マトリックスの倍率（ω，φ）によりモデル化される。検索間隔における小さい値Δε＝０．００２５およびΔλ＝０．００１は、他の小さい値、たとえば０．０５、０．０１、０．００５、０．００１、０．０００５、０．０００１などに置き換えることができる。

【００８７】

前に提示された（ε，λ）グリッドを検索する代わりに、本明細書の方法は、好ましくは正の整数である一連の（ω，φ）グリッドを通して実施し、マトリックスパワーの計算を促進する。最適な（ω’，φ’）は、ゼロ交差状態で画定され、対応する完了効率およびＣＦパラメータは、λ＊＝（１－φ＊Δλ）およびε＊＝ω＊Δεである。

【００８８】

方法２：

方程式（５）により、小さいεおよび（１－λ）の事例では、下方および上方対角パワーマトリックス［Ｌ］$^{\phi}$および［Ｕ］$^{\omega}$は、以下によってさらに概算される：

方程式（６）：

$$[L]^{\phi} \equiv ([I] + [l])^{\phi} \sim [I] + \phi[l]$$

方程式（７）：

$$[U]^{\omega} \equiv ([I] + [u])^{\omega} \sim [I] + \omega[u]$$

ここで：

－　［Ｉ］は、同一性マトリックスである

－　［ｌ］および［ｕ］は、それぞれ［Ｌ］および［Ｕ］の非対角マトリックスである。

【００８９】

これは、マトリックスパワーを計算する段階の迂回を公式化し、その結果、計算時間をさらに加速する（たとえば、短縮する）。したがって、（ω，φ）の検索空間は、すべて正の実数を含む。最適な（ω＊，φ＊）は、ゼロ交差状態で画定され、対応する完了効率およびＣＦパラメータは、λ＊＝（１－φ＊Δλ）およびε＊＝ω＊Δε。

【００９０】

本明細書に記載する実施態様は、マトリックスの構成および反転、並びに（ε，λ）平面における２次元検索に基づき、ＣＡＦＩＥパラメータの最適な対を探索する。これらの

計算は、実質的に同じテンプレート分子の各々の集団に関して行われ、たとえば、ｐｉｃｏｔｉｔｅｒｐｌａｔｅタイプのシステムにおけるウェルごとの分析を含む場合がある。実施態様によっては、マトリックスは、最適なＣＡＦＩＥ値（ε＊，λ＊）を生成するために、各々の集団／ウェルごとに構成される。図６は、上記の反転／検索方法１を使用して計算するように、数十万の集団／ウェル６０３のサンプルにおける完了効率パラメータ６０５の値λ＊およびＣＦパラメータ６０７の値εの分布の具体的な例を示す。上記の方法２を使用する計算は、方法１より計算時間が少なく、類似の結果を提供する。

【００９１】

また、上記の実施態様は、一定の完了効率λおよびＣＦεパラメータに関連する率が、配列決定実行全体で一定であることを仮定している。この仮定は、数回のフローサイクルを含むフローグラム内で「フローウィンドウ」と呼ばれる場合があるものにＣＡＦＩＥ検索および反転手順を適用することによって緩和することができる（この場合、「数回」は、１とフローサイクル全体の回数との間の任意の数を意味する）。たとえば、各々のフローウィンドウは、フローグラムに表現されるフローサイクルの完全な集合のうちの部分集合であり、１対のＣＡＦＩＥパラメータおよび対応するクリーンな理論上のフローグラム１０１を発見する必要がある。本明細書の例では、フローウィンドウは、配列決定実行に関連するフローグラム内の最初のフローから開始して、フローグラム内のフローサイクル全体の長さより短いかまたはこの長さに等しい一定のフローで終了し、各々のより小さいフローウィンドウは、より大きいフローウィンドウ内に入れ子状態になるように配置される。各々のフローウィンドウｎでは、検索および反転プロセスは個々に行われ、一連のＣＡＦＩＥパラメータ１１３を生成し、これは、ウィンドウ指数の関数ｎ：ε＊＝ε＊（ｎ）およびλ＊＝λ（ｎ）になる。計算されたクリーンな理論上のフローグラム１０１、ｐ（ｎ）は、やはり入れ子状になっており、指数ｎに応じてＣＡＦＩＥパラメータのこれらの変数値の結果である。「切換え」プロセス：ウィンドウ（ｎ－１）およびｎ間のフローに関するｐ＝ｐ（ｎ）は、フローウィンドウ配列ｐ（ｎ）を最終のクリーンなフローグラム（ｐ）１０１に再構築する。

【００９２】

同じ実施態様、または別の実施態様では、λおよびεの一定の値の仮定は、別の方法で排除することができる。たとえば、完了効率λおよびＣＦεパラメータは、各々のヌクレオチド種の添加に関しては「Ｎ」（「Ａ」、「Ｇ」、「Ｃ」、または「Ｔ」）、およびフロー位置の関数「ｆ」（１、２、３、．．．）などのパラメータ形式を取ることができる：

$$\lambda_N(f) = {}^{0}\lambda_N * \exp(-\delta_N * f),$$

$$\varepsilon_N(f) = {}^{0}\varepsilon_N * \exp(-\beta_N * f).$$

ここで：

－　$\lambda_N(f)$は、「ｆ」番目のフローにおけるヌクレオチド種「Ｎ」の完了効率である

－　$\varepsilon_N(f)$は、「ｆ」番目のフローにおけるヌクレオチド種「Ｎ」のＣＦである

－　${}^{0}\lambda_N$および${}^{0}\varepsilon_N$は、初期値である

－　δ_Nおよびβ_Nは、減衰率である。

【００９３】

検索方法は、４つのパラメータ空間$\lambda_N(0)$、$\varepsilon_N(0)$、δ_N、およびβ_Nに適用して最適値を決定する。

【００９４】

さらに、当業者は、上記のＣＡＦＩＥメカニズムに関連しないその他の雑音源が存在し得ることも理解するであろう。こうした雑音源としては、電子的源、たとえば「暗電流」と呼ばれるもの、光源、生物学的起源、化学的起源、または先行技術で公知か、または将来発見され得るその他の源が挙げられるが、これらだけに限らない。本明細書に記載する発明のいくつかの実施態様は、その他の雑音源に対して様々なレベルの感受性を示す場合があり、こうした感受性は、多くのアプリケーションでは、実質的に一定であるか、およ

び／または予測可能なレベルである。たとえば、既知または未知の源に起因する予測可能および一定レベルの雑音は、概して補正が容易である。１つの補正方法は、雑音に関連する値（雑音が過剰信号を追加するか、または検出信号を減少させるかどうかによる）を数学的に、フローに関連するすべての信号値に加算するか、またはこうした検出信号から減算することである。

【００９５】

雑音のレベルが予測不可能ないくつかの実施態様の場合、少なくとも部分的に、雑音レベルの概算は、信号データに埋め込まれる情報から導くことができる。たとえば、配列位置に存在しないことが分かっているか、または予測されるヌクレオチド種の場合、実際の信号値はゼロに等しいはずであると予測される。したがって、どの検出信号も、システム内のすべての雑音源に起因し得る。本明細書の例では、本明細書に記載する発明は、雑音形式のＣＡＦＩＥメカニズムを概算するため、こうした雑音は、データから除去し、下にある雑音を明らかにすることができる。本明細書の例では、概算は、配列実行におけるすべての「ゼロｍｅｒ」配列位置を調査することによって改善することができる。この場合、２進コード化ｐ’の方程式（４）の「閾値」の値は、上記の実施態様に記載した固定値ではなく、その雑音レベルを表すように、各々の実行について動的に決定することができる。

【００９６】

さらに、本発明のいくつかの実施態様は、観察されたフローグラムに示される配列データの過剰補正を防止するため、「安全基準」と呼ぶことができるものを含み得る。上記のとおり、過剰補正は、上記のアルゴリズムが繰返し適用される時に導入される誤差の指数関数的な蓄積を生じる可能性がある。たとえば、上記のその他の雑音源は、信号データに適用するべき補正量を含む安全基準を決定し得る。たとえば、インプリメンテーションによっては、その他のＣＡＦＩＥ以外の源からの一定レベルの雑音を想定し、６０％補正（たとえば、１００％は完全な補正を意味する）と呼ばれる場合がある安全基準をデータに適用することができる。この概算は、計算されたクリーンなフローグラムｐを６０％、および観察されたダーティなフローグラムｑを４０％含む「ハイブリッド」フローグラム、「０．６ｐ　＋　０．４ｑ」を使用する。あるいは、ＣＡＦＩＥ以外の雑音が「低」レベルである場合、より高度、たとえば８０％の補正率を適用し得る。

【実施例】

【００９７】

実施例１

黄色ブドウ球菌ＣＯＬおよびマイコプラズマジェニタリウムのゲノムは、４５４　Ｌｉｆｅ　Ｓｃｉｅｎｃｅｓのゲノムシーケンサ上にショットガン配列した（Ｍａｒｇｕｌｉｅｓ等、２００５、上記で引用することにより本願に援用する）。図７は、ＩＥ補正のみの効果、ゲノム範囲に関するＣＡＦＩＥ補正、共通配列の正確さ、中間読込み長さ、並びに、１２５を超える配列位置の読込み長さのうち、１００％の精度を達成したウェルの割合の具体的な例を示す。これらの各々の測定基準では、ＣＡＦＩＥ補正は、ＩＥ補正単独より優れていた。ＩＥ補正単独では、補正を行わない場合に得られた結果より優れていた。対照配列を含むビードを別に調製し、実験サンプルと混合してからアレイを調製した。

【００９８】

上記の手順を使用することにより、６３サイクルの実行の平均読込み長さは、１１２配列位置から１４７配列位置に増加した。これは、６３サイクルの理論上のほぼ最大、または２５２回のフロー繰返しである（たとえば、各々のフローサイクルは、４回のヌクレオチド種フローの繰返しを含む）。理論上の最大は、フローサイクルの数、この場合は６３に、平均して４回のヌクレオチド添加サイクルごとに伸長される配列位置の数（２．５）を乗算して計算される：６３×２．５＝１５７．５（理論上の最大）。１４７配列位置の平均読込み長さは、フローサイクル全体で９５％の精度で、フローグラムを既知のゲノム配列にマッピングして決定した。

【００９９】

さらに、本明細書には、以下のとおり、上記の方法１を使用して、上記のデータ処理アプリケーションによってインプリメント可能な４つの例示的な擬似コードコンピュータプログラムを開示する：

（１）ｂｕｉｌｄＴｒａｎｓｉｔｉｏｎＭａｔｒｉｘＩＥＯｎｌｙ．ｃ

不完全な伸長に関する遷移マトリックスを構築する。

（２）ｂｕｉｌｄＴｒａｎｓｉｔｉｏｎＭａｔｒｉｘＣＦＯｎｌｙ．ｃ

繰越に関する遷移マトリックスを構築する。

（３）ｃａｆｉｅＣｏｒｒｅｃｔＯｎｅＮｕｋｅＴｒａｃｅＦａｓｔＴＭＣ２．ｃ

（１）で計算した遷移マトリックスを反転し、ＩＥ値を検索する。

（４）ｃａｆｉｅＣｏｒｒｅｃｔＯｎｅＮｕｋｅＴｒａｃｅＦａｓｔＣａｒｒｙＦｏｒｗａｒｄＯｎｌｙ．ｃ

（２）で計算した遷移マトリックスを反転し、ＣＦ値を検索する。

【０１００】

入力は、各々の読込みに対するダーティなフローグラムおよびフロー順序（ヌクレオチドの添加）であり；出力は、クリーンになったフローグラムおよび最適値（ε＊、λ＊）である。これらの擬似コードコンピュータプログラムは、単に具体的に示すためのものであり、様々な修正および変更は本発明の範囲内であることが理解されるであろう。

【０１０１】

したがって、核酸の配列決定の際に得られた配列データの誤差を補正する方法およびシステムが提供されることが分かる。本明細書では、特定の実施態様について詳細に開示したが、これは、具体的に示すためにのみ開示したのであって、以下の添付の請求の範囲を制限することを意図するものではない。特に、請求の範囲によって定義される本発明の精神および範囲を逸脱することなく、様々な置換、変更、および修正を加えることができることが意図されている。その他の態様、利点、および修正は、以下の請求の範囲に含まれると考えられる。提示される請求の範囲は、本明細書に開示する発明を代表するものである。請求項に記載されていないその多数の発明も予想される。こうした発明を後の請求の範囲で追及する権利は、本明細書により留保される。

【０１０２】

（コンピュータプログラムリスティング）

【０１０３】

【表２－１】

コンピュータプログラムリスティング **1: buildTransitionMatrixCFOnly**

```
/*****************************************************************************
**
** Function Name: buildTransitionMatrixCFOnly
**
*****************************************************************************
** Description:
**
**    This function generates a square matrix for a given nuke signal trace
**      and given extension completion efficiency and carry over factor.
**
*****************************************************************************
**
** Inputs:
**              numPositives    -       number of positive flows - dimesion of the matrix
**              eff                             -       extension completion efficiency
**              carryOver               -       carry over factor
**
** Outputs:
**              u                               -       N x N matrix, where N is the
number of nuke flows
**
** Returns:
**              none
**
```

【０１０４】

【表２－２】

```
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Entry Point
**
*****************************************************************************/
void buildTransitionMatrixCFOnly (struct listTracking *nukeTraceList,
                                  double carryOver,
                                  int *numPositives,
                                  int **positives,
                                  int *numCallables,
                                  int **callables,
                                  double ***outputMatrix)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
    static char functionName[] = "buildTransitionMatrixCFOnly";
    static int traceLevel      = TRACE_LEVEL_8;

    int listError;
    int i;
    int j;
    int index;
//  int numPositives;
    int numNukeFlows;
    int numSoFar;
    int unCondensedIndex;

    int *index1;
    int index1Num = 0;
    int *index2;
    int index2Num = 0;
    struct wellInfo **index2Flows;
```

【０１０５】

【表２－３】

```
        BOOL sequenceEnded;
        double eff = 1.0;
        double df;
        double maxMarginalValue;
        int maxMarginalIndex;
        int testMarginalIndex;
        struct wellInfo *maxMarginalFlow;
        double *fractionAtEachFlow;
#if USE_DELTA_BUFFER
        double *deltaFractionAtEachFlow;
#endif
        double **fM;
        double **condensedMatrix;

        struct wellInfo *fromNukeList;
        struct wellInfo *fromPosList;
        struct wellInfo *prevNukeInfo = NULL;
/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
        traceIn (functionName, traceLevel);

        /*
        **      Get the number of nuke flows
        */
        numNukeFlows = listGetCount (nukeTraceList, &listError);

        *positives = index1 = safeMalloc (sizeof(int) * numNukeFlows);
        *callables = index2 = safeMalloc (sizeof(int) * numNukeFlows);
        index2Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);

        /*
        **      Get the number of positive nuke flows
        */

        /*
        **      Allocate space for a rectangular matrix of dimension:
        **      numNukeFlows x numPositives
        */
        fM = safeMalloc (numNukeFlows * sizeof (double *));
        for (i = 0,sequenceEnded = FALSE;
                i < numNukeFlows;
                i++)
        {
                fM[i] = safeMalloc (numNukeFlows * sizeof (double));
                fromNukeList = listGetAt (nukeTraceList,i,&listError);
                if (fromNukeList->signal > TMC_LOWER_CUTOFF)
                {
#if 1
                        fM[i][i] = 1.0;
#endif
                        index1[index1Num] = i;
                        index1Num++;
                        if (fromNukeList->signal > TMC_UPPER_CUTOFF)
```

【０１０６】

【表２－４】

```
                {
                        index2[index2Num] = i;
                        index2Flows[index2Num] = fromNukeList;
                        index2Num++;
                }
        }
        /*
        ** Check to make sure this wouldn't be three negatives in a row
        ** which will totally hose up the correction.  Since we know we
        ** key pass don't worry about the boundry with no positives
        */
#if FORCE_CALLS
        if ((i - index1[index1Num-1] == 3) && !sequenceEnded)
        {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 positive
                */
                maxMarginalIndex = index1[index1Num-1]+1;
                fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                maxMarginalValue = fromNukeList->signal;
                for (testMarginalIndex = maxMarginalIndex+1;
                        testMarginalIndex <= i;
                        testMarginalIndex++)
                {
                        fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                        if (fromNukeList->signal > maxMarginalValue)
                        {
                                maxMarginalValue = fromNukeList->signal;
                                maxMarginalIndex = testMarginalIndex;
                        }
                }
#if 1
                fM[maxMarginalIndex][maxMarginalIndex] = 1.0;
#endif
                index1[index1Num] = maxMarginalIndex;
                index1Num++;
        }
        if ((i - index2[index2Num-1] == 3) && !sequenceEnded)
        {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.5 singlet
                */
                maxMarginalIndex = index2[index2Num-1]+1;
                fromNukeList = listGetAt
        (nukeTraceList,maxMarginalIndex,&listError);
                maxMarginalValue = fromNukeList->signal;
                maxMarginalFlow = fromNukeList;
                for (testMarginalIndex = maxMarginalIndex+1;
                        testMarginalIndex <= i;
                        testMarginalIndex++)
                {
```

【０１０７】

【表２－５】

```
                                fromNukeList = listGetAt
        (nukeTraceList,testMarginalIndex,&listError);
                                if (fromNukeList->signal > maxMarginalValue)
                                {
                                        maxMarginalValue = fromNukeList->signal;
                                        maxMarginalIndex = testMarginalIndex;
                                        maxMarginalFlow = fromNukeList;
                                }
                        }
                        if (maxMarginalValue > TMC_TOO_LOW_TO_FORCE)
                        {
                                index2[index2Num] = maxMarginalIndex;
                                index2Flows[index2Num] = maxMarginalFlow;
                                index2Num++;
                        }
                        else
                        {
                                sequenceEnded = TRUE;
                        }
                }
#endif
        }

        /*
        ** Return the actual numbers of positives and callables
        */
        *numCallables = index2Num;
        *numPositives = index1Num;

        condensedMatrix = safeMalloc      (sizeof(double *) * index1Num);
        *outputMatrix = condensedMatrix;
        for (i = 0; i < index1Num; i++)
        {
                condensedMatrix[i] = safeMalloc (index1Num * sizeof (double));
        }

#if DUMP_INFO
        for (i = 0; i < index1Num; i++)
        {
                fprintf   (stderr,"%s - index1 pos '%d'\t'%d'\n",functionName,i,index1[i]);
        }
        fprintf   (stderr,"\n");
        for (i = 0; i < index2Num; i++)
        {
                fprintf   (stderr,"%s - index2 pos '%d'\t'%d'\n",functionName,i,index2[i]);
        }
        fprintf   (stderr,"\n");

        fprintf   (stderr,"%s - Whole matrix before calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
```

【０１０８】

【表２－６】

```
        }
#endif


#if 0
        /*
        **      Set all matrix elements to zero
        */
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        u[i][j] = 0.0;
                }
        }
#endif

        /*
        **      Allocate space to hold incorporation info
        */
        fractionAtEachFlow = safeMalloc (index2Num * sizeof (double));
#if USE_DELTA_BUFFER
        deltaFractionAtEachFlow = safeMalloc (index2Num * sizeof (double));
#endif
        fractionAtEachFlow[0] = 1.0;

        for (i = 0; i < numNukeFlows; i++)
        {
                /*
                **      First pass of positiveTraceList calculates incomplete extention
                */
                fromNukeList = listGetAt (nukeTraceList, i, &listError);
#if USE_DELTA_BUFFER
                memset (deltaFractionAtEachFlow,0,sizeof(double)*index2Num);
#endif
                for (j = 0; j < index2Num; j++)
                {
                        fromPosList = index2Flows[j];
                        unCondensedIndex = index2[j];

                        if (fromNukeList->fluidFlowed == fromPosList->fluidFlowed)
                        {
                                df = fractionAtEachFlow[j] * eff;

#if USE_DELTA_BUFFER
                                deltaFractionAtEachFlow[j] -= df;
#else
                                fractionAtEachFlow[j] -= df;
#endif

                                fM[i][unCondensedIndex] = df;
                                if (j != (index2Num - 1))
                                {
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j+1] += df;
#else
```

【０１０９】

【表２－７】

```
                                        fractionAtEachFlow[j+1] += df;
#endif
                                }
                        }
                }
#if USE_DELTA_BUFFER
                for (j = 0; j < index2Num; j++)
                {
                        fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                }
#endif

                /*
                **      Second pass calculates carryforward (i > 0)
                */
                if (i > 0)
                {
#if USE_DELTA_BUFFER
                        memset (deltaFractionAtEachFlow,0,sizeof(double)*index2Num);
#endif
                        for (j = 0; j < index2Num; j++)
                        {
                                fromPosList = index2Flows[j];
                                unCondensedIndex = index2[j];

                                if (prevNukeInfo->fluidFlowed == fromPosList-
>fluidFlowed)
                                {
                                        df = fractionAtEachFlow[j] * carryOver;
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j] -= df;
#else
                                        fractionAtEachFlow[j] -= df;
#endif
                                        fM[i][unCondensedIndex] = df;

                                        if (j != (index2Num - 1))
                                        {
#if USE_DELTA_BUFFER
                                                deltaFractionAtEachFlow[j+1] += df;
#else
                                                fractionAtEachFlow[j+1] += df;
#endif

                                        }
                                }
                        }
#if USE_DELTA_BUFFER
                        for (j = 0; j < index2Num; j++)
                        {
                                fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                        }
#endif
                }

                prevNukeInfo = fromNukeList;
```

【０１１０】

【表２－８】

```
        }

        /*
        **      Truncate to condensed space
        */
#if DUMP_INFO

        fprintf   (stderr,"%s - Whole matrix after calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }

        fprintf   (stderr,"%s - EFF-'%lf', CF-'%lf'\n",functionName,eff,carryOver);
        for (i = 0; i < index1Num; i++)
        {
                fromPosList = listGetAt   (nukeTraceList,index1[i],&listError);
                fprintf   (stderr,"\t%.4lf",fromPosList->signal);
        }
        fprintf (stderr,"\n");
#endif
        index = 0;
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        condensedMatrix[i][j] = fM[index1[i]][index1[j]];
#if DUMP_INFO
                        fprintf   (stderr,"\t%.4lf",condensedMatrix[i][j]);
#endif
                }
#if DUMP_INFO
                fprintf   (stderr,"\n");
#endif
        }

        safeFree (fractionAtEachFlow);
#if USE_DELTA_BUFFER
        safeFree (deltaFractionAtEachFlow);
#endif
        safeFree (index2Flows);

        if (fM)
        {
                for (i = 0; i < numNukeFlows; i++)
                {
                        safeFree (fM[i]);
                }
                safeFree (fM);
        }
        traceOut (functionName, traceLevel);
        return;
}
```

【０１１１】

【表３－１】

コンピュータプログラムリスティング **2: buildTransitionMatrixIEOnly**

```
/*****************************************************************************
**
** Function Name: buildTransitionMatrixIEOnly

**
*****************************************************************************
** Description:
**
**    This function generates a square matrix for a given nuke signal trace
**       and given extension completion efficiency and carry over factor.
**
*****************************************************************************
**
** Inputs:
**              numPositives    -       number of positive flows - dimesion of the matrix
**              eff                             -       extension completion efficiency
**              carryOver               -       carry over factor
**
** Outputs:
**              u                               -       N x N matrix, where N is the
number of nuke flows
**
** Returns:
**              none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/

/*****************************************************************************
**
```

【０１１２】

【表３－２】

```
** Entry Point
**
*****************************************************************************/
void buildTransitionMatrix1EOnly (struct listTracking *nukeTraceList,
                                  double eff,
                                  int *numPositives,
                                  int **positives,
                                  int *numCallables,
                                  int **callables,
                                  double ***outputMatrix)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "buildTransitionMatrix1EOnly";
        static int traceLevel      = TRACE_LEVEL_8;

        int listError;
        int i;
        int j;
        int index;
//      int numPositives;
        int numNukeFlows;
        int numSoFar;
        int unCondensedIndex;

        double maxMarginalValue;
        int maxMarginalIndex;
        int testMarginalIndex;
        struct wellInfo *maxMarginalFlow;
        int *index1;
        int index1Num = 0;
        struct wellInfo **index1Flows;
        int *index2;
        int index2Num = 0;
        struct wellInfo **index2Flows;

        double df;
        double *fractionAtEachFlow;
        BOOL sequenceEnded;
#if USE_DELTA_BUFFER
        double *deltaFractionAtEachFlow;
#endif
        double **fM;
        double **condensedMatrix;

        struct wellInfo *fromNukeList;
        struct wellInfo *fromPosList;
        struct wellInfo *prevNukeInfo;
/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
```

【０１１３】

【表３－３】

```
traceIn (functionName, traceLevel);

/*
**      Get the number of nuke flows
*/
numNukeFlows = listGetCount (nukeTraceList, &listError);

*positives = index1 = safeMalloc (sizeof(int) * numNukeFlows);
*callables = index2 = safeMalloc (sizeof(int) * numNukeFlows);
index2Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);
index1Flows = safeMalloc (sizeof(struct wellInfo *) * numNukeFlows);

/*
**      Get the number of positive nuke flows
*/

/*
**      Allocate space for a rectangular matrix of dimension:
**      numNukeFlows x numPositives
*/
fM = safeMalloc (numNukeFlows * sizeof (double *));
for (i = 0,numSoFar = 0,sequenceEnded = FALSE;
        i < numNukeFlows;
        i++)
{
        fM[i] = safeMalloc (numNukeFlows * sizeof (double));
        fromNukeList = listGetAt (nukeTraceList,i,&listError);
        if (fromNukeList->signal > TMC_LOWER_CUTOFF && !sequenceEnded)
        {
#if 0
                fM[i][i] = 1.0;
#endif
                index1[index1Num] = i;
                index1Flows[index1Num] = fromNukeList;
                index1Num++;
                if (fromNukeList->signal > TMC_UPPER_CUTOFF)
                {
                        index2[index2Num] = i;
                        index2Flows[index2Num] = fromNukeList;
                        index2Num++;
                }
        }
#if FORCE_CALLS
        /*
        ** Check to make sure this wouldn't be three negatives in a row
        ** which will totally hose up the correction.  Since we know we
        ** key pass don't worry about the boundry with no positives
        */
        if ((i - index1[index1Num-1] == 3) && !sequenceEnded)
        {
                /*
                ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 singlet
                */
                maxMarginalIndex = index1[index1Num-1]+1;
```

【０１１４】

【表３－４】

```
                    fromNukeList = listGetAt
    (nukeTraceList,maxMarginalIndex,&listError);
                    maxMarginalValue = fromNukeList->signal;
                    maxMarginalFlow = fromNukeList;
                    for (testMarginalIndex = maxMarginalIndex+1;
                            testMarginalIndex <= i;
                            testMarginalIndex++)
                    {
                            fromNukeList = listGetAt
    (nukeTraceList,testMarginalIndex,&listError);
                            if (fromNukeList->signal > maxMarginalValue)
                            {
                                    maxMarginalValue = fromNukeList->signal;
                                    maxMarginalFlow = fromNukeList;
                                    maxMarginalIndex = testMarginalIndex;
                            }
                    }
#if 0
                    fM[maxMarginalIndex][maxMarginalIndex] = 1.0;
#endif
                    index1[index1Num] = maxMarginalIndex;
                    index1Flows[index1Num] = maxMarginalFlow;
                    index1Num++;
            }
            if ((i - index2[index2Num-1] == 3) && !sequenceEnded)
            {
                    /*
                    ** Promote the highest value from index1[index1Num-1]+1 to i to a
0.2 singlet
                    */
                    maxMarginalIndex = index2[index2Num-1]+1;
                    fromNukeList = listGetAt
    (nukeTraceList,maxMarginalIndex,&listError);
                    maxMarginalValue = fromNukeList->signal;
                    maxMarginalFlow = fromNukeList;
                    for (testMarginalIndex = maxMarginalIndex+1;
                            testMarginalIndex <= i;
                            testMarginalIndex++)
                    {
                            fromNukeList = listGetAt
    (nukeTraceList,testMarginalIndex,&listError);
                            if (fromNukeList->signal > maxMarginalValue)
                            {
                                    maxMarginalValue = fromNukeList->signal;
                                    maxMarginalIndex = testMarginalIndex;
                                    maxMarginalFlow = fromNukeList;
                            }
                    }
                    if (maxMarginalValue > TMC_TOO_LOW_TO_FORCE)
                    {
                            index2[index2Num] = maxMarginalIndex;
                            index2Flows[index2Num] = maxMarginalFlow;
                            index2Num++;
                    }
                    else
                    {
```

【０１１５】

【表３－５】

```
                                sequenceEnded = TRUE;
                        }
                }
#endif
        }

        /*
        ** Return the number of callables and positives
        */
        *numCallables = index2Num;
        *numPositives = index1Num;

        condensedMatrix = safeMalloc      (sizeof(double *) * index1Num);
        *outputMatrix = condensedMatrix;
        for (i = 0; i < index1Num; i++)
        {
                condensedMatrix[i] = safeMalloc (index1Num * sizeof (double));
        }

#if DUMP_INFO
        for (i = 0; i < index1Num; i++)
        {
                fprintf   (stderr,"%s - index1 pos '%d'\t'%d'\n",functionName,i,index1[i]);
        }
        fprintf   (stderr,"\n");
        for (i = 0; i < index2Num; i++)
        {
                fprintf   (stderr,"%s - index2 pos '%d'\t'%d'\n",functionName,i,index2[i]);
        }
        fprintf   (stderr,"\n");

        fprintf   (stderr,"%s - Whole matrix before calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%.4lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }
#endif


#if 0
        /*
        **      Set all matrix elements to zero
        */
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        u[i][j] = 0.0;
                }
        }
#endif
```

【０１１６】

【表３－６】

```
        /*
        **      Allocate space to hold incorporation info
        */
        fractionAtEachFlow = safeMalloc (index1Num * sizeof (double));
#if USE_DELTA_BUFFER
        deltaFractionAtEachFlow = safeMalloc (index1Num * sizeof (double));
#endif
        fractionAtEachFlow[0] = 1.0;

        for (i = 0; i < numNukeFlows; i++)
        {
                /*
                **      First pass of positiveTraceList calculates incomplete extention
                */
                fromNukeList = listGetAt (nukeTraceList, i, &listError);

#if USE_DELTA_BUFFER
                memset  (deltaFractionAtEachFlow,0,sizeof(double)*index1Num);
#endif
                for (j = 0; j < index1Num; j++)
                {
                        fromPosList = index1Flows[j];
                        unCondensedIndex = index1[j];

                        if (fromNukeList->fluidFlowed == fromPosList->fluidFlowed)
                        {
                                df = fractionAtEachFlow[j] * eff;
#if USE_DELTA_BUFFER
                                deltaFractionAtEachFlow[j] -= df;
#else
                                fractionAtEachFlow[j] -= df;
#endif
                                fM[i][unCondensedIndex] = df;
                                if (j != (index1Num - 1))
                                {
#if USE_DELTA_BUFFER
                                        deltaFractionAtEachFlow[j+1] += df;
#else
                                        fractionAtEachFlow[j+1] += df;
#endif
                                }
                        }
                }
#if USE_DELTA_BUFFER
                for (j = 0; j < index1Num; j++)
                {
                        fractionAtEachFlow[j] += deltaFractionAtEachFlow[j];
                }
#endif
        }

        /*
        **      Truncate to condensed space
        */
#if DUMP_INFO
```

【０１１７】

【表３－７】

```
        fprintf   (stderr,"%s - Whole matrix after calculation\n",functionName);
        for (i = 0; i < numNukeFlows; i++)
        {
                for (j = 0; j < numNukeFlows; j++)
                {
                        fprintf   (stderr,"\t%lf",fM[i][j]);
                }
                fprintf   (stderr,"\n");
        }

        fprintf   (stderr,"%s - EFF-'%lf', CF-'%lf'\n",functionName,eff,0.0);
        for (i = 0; i < index1Num; i++)
        {
                fromPosList = listGetAt   (nukeTraceList,index1[i],&listError);
                fprintf   (stderr,"\t%.4lf",fromPosList->signal);
        }
        fprintf (stderr,"\n");
#endif
        index = 0;
        for (i = 0; i < index1Num; i++)
        {
                for (j = 0; j < index1Num; j++)
                {
                        condensedMatrix[i][j] = fM[index1[i]][index1[j]];
#if DUMP_INFO
                        fprintf   (stderr,"\t%.4lf",condensedMatrix[i][j]);
#endif
                }
#if DUMP_INFO
                fprintf   (stderr,"\n");
#endif
        }


        safeFree (fractionAtEachFlow);
#if USE_DELTA_BUFFER
        safeFree (deltaFractionAtEachFlow);
#endif
        safeFree (index1Flows);
        safeFree (index2Flows);

        if (fM)
        {
                for (i = 0; i < numNukeFlows; i++)
                {
                        safeFree (fM[i]);
                }
                safeFree (fM);
        }

        traceOut (functionName, traceLevel);
        return;
}
```

【０１１８】

【表４－１】

コンピュータプログラムリスティング **3:**
cafieCorrectOneNukeTraceFastCarryForwardOnly

```
/*****************************************************************************
**
** Function Name:  cafieCorrectOneNukeTraceFastTMC2
**
**

**
*****************************************************************************
** Description:
**
**    This function reads in a nuke pyrogram information list and perform
**       CAFIE Correction to it.
**
*****************************************************************************
**
** Inputs:
**                 none
**
** Outputs:
**    error            - error value
**
** Input/Outputs:
**                 nukeTraceList   -   binary well trace list
**
** Returns:
**                 none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>

#define TEST 0
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/
extern FILE *errorOut;
/*****************************************************************************
**
** Entry Point
```

【０１１９】

【表４－２】

```
**
*****************************************************************************/
double cafieCorrectOneNukeTraceFastTMC2 (struct listTracking *nukeTraceList,
                                         int *error)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "cafieCorrectOneNukeTraceFastTMC2";
        static int traceLevel      = TRACE_LEVEL_8;

        int i;
        int j;
        int listCnt;
        int index;
        int listError;
        int numFlows;
        int numCallables;
        int numPositives = 0;
        int *indx = NULL;

        double effMax = 0.999;
        double effMin = 0.990;
        double newVal;
        double bestEff;
        double eff;
        double efficiencyUsed = 1.0;
        double sum;
        double carryOver;
        double lambda;
        double maxD;
        double maxQ;
        double numPositiveD;
        double d;

        double *nukeSignals = NULL;
        double *superDirtySignals = NULL;
        double *newSuperDirtySignals = NULL;
        double *tempDubP;
        double *x = NULL;
        double *b = NULL;
        double **condensedMatrix = NULL;
        double **nonLuCondensedMatrix = NULL;

        int *positives = NULL;
        int *callables = NULL;

        struct wellInfo *wellInfo;

/*****************************************************************************
**
** Begin Executable Code
**
*****************************************************************************/
```

【０１２０】

【表４－３】

```
traceIn (functionName, traceLevel);

/*
**      Assume we will be ok
*/
*error = ANALYSIS_OK;

/*
**      Build the transition matrix. Set the carryOver parameter to zero
**      and use the effMax
*/
buildTransitionMatrixIEOnly (nukeTraceList,
                effMax,
                &numPositives,
                &positives,
                &numCallables,
                &callables,
                &condensedMatrix);
numFlows = listGetCount (nukeTraceList, &listError);

/*
**      If there're not enough number (1/4 of total flows) of positives,
**      we won't do anything
*/
if (numCallables <= 0.25 * numFlows)
{
        goto bye;
}

/*
**      Populate it and hold a copy in an array
*/
nukeSignals = safeMalloc (numPositives * sizeof (double));
for (i = 0; i < numPositives; i++)
{
        wellInfo = listGetAt (nukeTraceList,positives[i], &listError);
        if (listError != LIST_OK)
        {
                fprintf (errorOut,
                        "%s - Error getting %d-th wellInfo from
nukeList.\n",
                        functionName,
                        i);
                *error = listError;
                goto bye;
        }
        nukeSignals[i] = wellInfo->signal;
}

/*
**      Allocate space for a copy of nuke signals
*/
b = safeMalloc (numPositives * sizeof (double));
superDirtySignals = safeMalloc (numPositives * sizeof (double));
newSuperDirtySignals = safeMalloc (numPositives * sizeof (double));
for (i = 0; i < numPositives; i++)
```

【０１２１】

【表４－４】

```
        {
                superDirtySignals[i] = b[i] = nukeSignals[i];
        }

        /*
        **      Allocate space for these arrays to be used for solving linear
        **      equations
        */
        indx = safeMalloc (numPositives * sizeof (int));
        x = safeMalloc (numPositives * sizeof (double));
        nonLuCondensedMatrix = safeMalloc (numPositives * sizeof (double *));
        for (i = 0; i < numPositives; i++)
        {
                nonLuCondensedMatrix[i] = safeMalloc (numPositives * sizeof (double));
                for (j = 0; j < numPositives; j++)
                {
                        nonLuCondensedMatrix[i][j] = condensedMatrix[i][j];
                }
        }

        /*
        **      We will solve a linear system of form: AX=B, where A will be u,
        **      X will be the solution vector, and B is initially nukeSignals,
        **      afterwards, it will store the solution vector.
        */
#if DO_LU_DECOMPOSITION

        if (!luDecomposition (condensedMatrix, numPositives, indx, &d))
        {
                *error = ERR_CAFIE_NON_INVERTABLE_MATRIX;
                goto bye;
        }
#endif

        /*
        **      Main trunk of the Condensed Matrix Algorithm below is a loop
        **      process to search for the best extension efficiency parameter
        */
        bestEff = 0.0;
        for (eff = effMax; eff >= effMin; eff -= 0.001)
        {
                /*
                ** Apply the matrix transform
                */
                luBackSubstitution (condensedMatrix, numPositives, indx, b);

                /*
                ** Also create the new, dirtier pyrogram
                */
                for (i = 0; i < numPositives; i++)
                {
                        newVal = 0.0;
                        for (j = 0; j < numPositives; j++)
                        {
                                newVal += nonLuCondensedMatrix[i][j] *
superDirtySignals[j];
```

【０１２２】

【表４－５】

```
        }
        newSuperDirtySignals[i] = newVal;
}

/*
** Swap the pointers
*/
tempDubP = newSuperDirtySignals;
newSuperDirtySignals = superDirtySignals;
superDirtySignals = tempDubP;

/*
** See if we have any negative values.  This is the indicator that we
** have corrected enough.
*/
numPositiveD = 0;
for (i = 0; i < numPositives; i++)
{
        if (b[i] < 0.0)
        {
                numPositiveD++;
                break;
        }
}

/*
** If we have a positive count then it is time to jump into the actual
** correction phase
*/
if (numPositiveD > 0)
{
        maxD = fabs (b[0] - nukeSignals[0]);
        for (i = 1; i < numPositives; i++)
        {
                if (maxD < fabs (b[i] - nukeSignals[i]))
                {
                        maxD = fabs (b[i] - nukeSignals[i]);
                }
        }

        maxQ = fabs (superDirtySignals[0] - nukeSignals[0]);
        for (i = 1; i < numPositives; i++)
        {
                newVal = fabs (superDirtySignals[i] - nukeSignals[i]);
                if (maxQ < newVal)
                {
                        maxQ = newVal;
                }
        }

        lambda = 0.8 * maxQ / maxD;
        if (lambda > 1.0)
        {
                lambda = 1.0;
        }
```

【０１２３】

【表４－６】

```
                        /*
                        **      Construct the new nuke signals from solution
                        */
                        for (i = 0; i < numPositives; i++)
                        {
                                x[i] = nukeSignals[i] + lambda * (b[i] - nukeSignals[i]);

                                /*
                                **      Corrected signals
                                */
                                nukeSignals[i] = x[i];
                        }

                        /*
                        **      We found the best efficiency, won't try more stages
                        */
                        efficiencyUsed = bestEff = eff;
                        break;
                }
        }

        /*
        **      If we didn't find the best efficiency, we won't alter nuke trace
        */
        if (bestEff == 0.0)
        {
                goto bye;
        }

        /*
        **      Now make the change official in the original nuke trace
        */
        for (i = 0; i < numPositives; i++)
        {
                wellInfo = listGetAt (nukeTraceList, positives[i], &listError);
                if (listError != LIST_OK)
                {
                        fprintf (errorOut,
                                        "%s - Error getting %d-th wellInfo from
positiveTraceList.\n",
                                        functionName,
                                        i);
                        *error = listError;
                        goto bye;
                }
#if DO_CORRECTION
                wellInfo->signal = nukeSignals[i];
                if (wellInfo->signal < 0.0)
                {
                        wellInfo->signal = 0.0;
                }
#endif
        }

bye:
        /*
```

【０１２４】

【表４－７】

```
        **      Free up space
        */
        if (positives)
        {
                safeFree (positives);
        }
        if (callables)
        {
                safeFree (callables);
        }
        if (nukeSignals)
        {
                safeFree (nukeSignals);
        }
        if (superDirtySignals)
        {
                safeFree (superDirtySignals);
        }
        if (newSuperDirtySignals)
        {
                safeFree (newSuperDirtySignals);
        }
        if (indx)
        {
                safeFree (indx);
        }
        if (b)
        {
                safeFree (b);
        }
        if (x)
        {
                safeFree (x);
        }

        for (i = 0; i < numPositives; i++)
        {
                if (condensedMatrix && condensedMatrix[i])
                {
                        safeFree (condensedMatrix[i]);
                }
                if (nonLuCondensedMatrix && nonLuCondensedMatrix[i])
                {
                        safeFree (nonLuCondensedMatrix[i]);
                }
        }

        if (condensedMatrix)
        {
                safeFree (condensedMatrix);
        }
        if (nonLuCondensedMatrix)
        {
                safeFree (nonLuCondensedMatrix);
        }
        traceOut (functionName, traceLevel);
        return (efficiencyUsed);
}
```

【０１２５】

【表５－１】

コンピュータプログラムリスティング **4: cafieCorrectOneNukeTraceFastTMC2**

```
/*****************************************************************************
**
** Function Name: cafieCorrectOneNukeTraceFastTMC2
**
**

**
*****************************************************************************
** Description:
**
**    This function reads in a nuke pyrogram information list and perform
**       CAFIE Correction to it.
**
*****************************************************************************
**
** Inputs:
**                 none
**
** Outputs:
**    error            - error value
**
** Input/Outputs:
**                 nukeTraceList   -   binary well trace list
**
** Returns:
**                 none
**
*****************************************************************************/

/*****************************************************************************
**
** Includes
**
*****************************************************************************/
#include <cafieCommons.h>

#define TEST 0
/*****************************************************************************
**
** Function Declarations
**
*****************************************************************************/

/*****************************************************************************
**
** Global Declarations
**
*****************************************************************************/
```

【０１２６】

【表５－２】

```
extern FILE *errorOut;
/*****************************************************************************
**
** Entry Point
**
*****************************************************************************/
double cafieCorrectOneNukeTraceFastTMC2 (struct listTracking *nukeTraceList,
                                                    int *error)
{
/*****************************************************************************
**
** Variable declarations
**
*****************************************************************************/
        static char functionName[] = "cafieCorrectOneNukeTraceFastTMC2";
        static int traceLevel     = TRACE_LEVEL_8;

        int i;
        int j;
        int listCnt;
        int index;
        int listError;
        int numFlows;
        int numCallables;
        int numPositives = 0;
        int *indx = NULL;

        double effMax = 0.999;
        double effMin = 0.990;
        double newVal;
        double bestEff;
        double eff;
        double efficiencyUsed = 1.0;
        double sum;
        double carryOver;
        double lambda;
        double maxD;
        double maxQ;
        double numPositiveD;
        double d;

        double *nukeSignals = NULL;
        double *superDirtySignals = NULL;
        double *newSuperDirtySignals = NULL;
        double *tempDubP;
        double *x = NULL;
        double *b = NULL;
        double **condensedMatrix = NULL;
        double **nonLuCondensedMatrix = NULL;

        int *positives = NULL;
        int *callables = NULL;

        struct wellInfo *wellInfo;

/*****************************************************************************
```

【０１２７】

【表５－３】

```
**
** Begin Executable Code
**
*****************************************************************************/
        traceIn (functionName, traceLevel);

        /*
        **      Assume we will be ok
        */
        *error = ANALYSIS_OK;

        /*
        **      Build the transition matrix. Set the carryOver parameter to zero
        **      and use the effMax
        */
        buildTransitionMatrixIEOnly (nukeTraceList,
                        effMax,
                        &numPositives,
                        &positives,
                        &numCallables,
                        &callables,
                        &condensedMatrix);
        numFlows = listGetCount (nukeTraceList, &listError);

        /*
        **      If there're not enough number (1/4 of total flows) of positives,
        **      we won't do anything
        */
        if (numCallables <= 0.25 * numFlows)
        {
                goto bye;
        }

        /*
        **      Populate it and hold a copy in an array
        */
        nukeSignals = safeMalloc (numPositives * sizeof (double));
        for (i = 0; i < numPositives; i++)
        {
                wellInfo = listGetAt (nukeTraceList,positives[i], &listError);
                if (listError != LIST_OK)
                {
                        fprintf (errorOut,
                                        "%s - Error getting %d-th wellInfo from
nukeList.\n",
                                        functionName,
                                        i);
                        *error = listError;
                        goto bye;
                }
                nukeSignals[i] = wellInfo->signal;
        }

        /*
        **      Allocate space for a copy of nuke signals
        */
```

【０１２８】

【表５－４】

```
        b = safeMalloc (numPositives * sizeof (double));
        superDirtySignals = safeMalloc (numPositives * sizeof (double));
        newSuperDirtySignals = safeMalloc (numPositives * sizeof (double));
        for (i = 0; i < numPositives; i++)
        {
                superDirtySignals[i] = b[i] = nukeSignals[i];
        }

        /*
        **      Allocate space for these arrays to be used for solving linear
        **      equations
        */
        indx = safeMalloc (numPositives * sizeof (int));
        x = safeMalloc (numPositives * sizeof (double));
        nonLuCondensedMatrix = safeMalloc (numPositives * sizeof (double *));
        for (i = 0; i < numPositives; i++)
        {
                nonLuCondensedMatrix[i] = safeMalloc (numPositives * sizeof (double));
                for (j = 0; j < numPositives; j++)
                {
                        nonLuCondensedMatrix[i][j] = condensedMatrix[i][j];
                }
        }

        /*
        **      We will solve a linear system of form: AX=B, where A will be u,
        **      X will be the solution vector, and B is initially nukeSignals,
        **      afterwards, it will store the solution vector.
        */
#if DO_LU_DECOMPOSITION

        if (!luDecomposition (condensedMatrix, numPositives, indx, &d))
        {
                *error = ERR_CAFIE_NON_INVERTABLE_MATRIX;
                goto bye;
        }
#endif

        /*
        **      Main trunk of the Condensed Matrix Algorithm below is a loop
        **      process to search for the best extension efficiency parameter
        */
        bestEff = 0.0;
        for (eff = effMax; eff >= effMin; eff -= 0.001)
        {
                /*
                ** Apply the matrix transform
                */
                luBackSubstitution (condensedMatrix, numPositives, indx, b);

                /*
                ** Also create the new, dirtier pyrogram
                */
                for (i = 0; i < numPositives; i++)
                {
                        newVal = 0.0;
```

【０１２９】

【表５－５】

```
                for (j = 0; j < numPositives; j++)
                {
                        newVal += nonLuCondensedMatrix[i][j] *
superDirtySignals[j];
                }
                newSuperDirtySignals[i] = newVal;
        }

        /*
        ** Swap the pointers
        */
        tempDubP = newSuperDirtySignals;
        newSuperDirtySignals = superDirtySignals;
        superDirtySignals = tempDubP;

        /*
        ** See if we have any negative values.  This is the indicator that we
        ** have corrected enough.
        */
        numPositiveD = 0;
        for (i = 0; i < numPositives; i++)
        {
                if (b[i] < 0.0)
                {
                        numPositiveD++;
                        break;
                }
        }

        /*
        ** If we have a positive count then it is time to jump into the actual
        ** correction phase
        */
        if (numPositiveD > 0)
        {
                maxD = fabs (b[0] - nukeSignals[0]);
                for (i = 1; i < numPositives; i++)
                {
                        if (maxD < fabs (b[i] - nukeSignals[i]))
                        {
                                maxD = fabs (b[i] - nukeSignals[i]);
                        }
                }

                maxQ = fabs (superDirtySignals[0] - nukeSignals[0]);
                for (i = 1; i < numPositives; i++)
                {
                        newVal = fabs (superDirtySignals[i] - nukeSignals[i]);
                        if (maxQ < newVal)
                        {
                                maxQ = newVal;
                        }
                }

                lambda = 0.8 * maxQ / maxD;
                if (lambda > 1.0)
```

【０１３０】

【表５－６】

```
                {
                        lambda = 1.0;
                }

                /*
                **      Construct the new nuke signals from solution
                */
                for (i = 0; i < numPositives; i++)
                {
                        x[i] = nukeSignals[i] + lambda * (b[i] - nukeSignals[i]);

                        /*
                        **      Corrected signals
                        */
                        nukeSignals[i] = x[i];
                }

                /*
                **      We found the best efficiency, won't try more stages
                */
                efficiencyUsed = bestEff = eff;
                break;
        }
    }

    /*
    **      If we didn't find the best efficiency, we won't alter nuke trace
    */
    if (bestEff == 0.0)
    {
        goto bye;
    }

    /*
    **      Now make the change official in the original nuke trace
    */
    for (i = 0; i < numPositives; i++)
    {
        wellInfo = listGetAt (nukeTraceList, positives[i], &listError);
        if (listError != LIST_OK)
        {
            fprintf (errorOut,
                     "%s - Error getting %d-th wellInfo from
positiveTraceList.\n",
                     functionName,
                     i);
            *error = listError;
            goto bye;
        }
#if DO_CORRECTION
        wellInfo->signal = nukeSignals[i];
        if (wellInfo->signal < 0.0)
        {
            wellInfo->signal = 0.0;
        }
#endif
```

【０１３１】

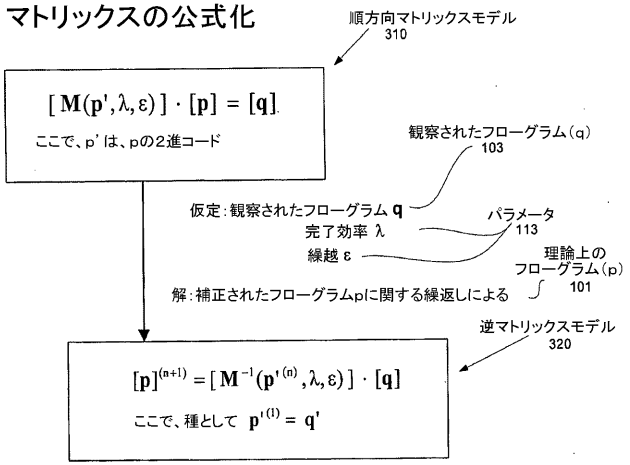
【表５－７】

```
        }

bye:
        /*
        **      Free up space
        */
        if (positives)
        {
                safeFree (positives);
        }
        if (callables)
        {
                safeFree (callables);
        }
        if (nukeSignals)
        {
                safeFree (nukeSignals);
        }
        if (superDirtySignals)
        {
                safeFree (superDirtySignals);
        }
        if (newSuperDirtySignals)
        {
                safeFree (newSuperDirtySignals);
        }
        if (indx)
        {
                safeFree (indx);
        }
        if (b)
        {
                safeFree (b);
        }
        if (x)
        {
                safeFree (x);
        }

        for (i = 0; i < numPositives; i++)
        {
                if (condensedMatrix && condensedMatrix[i])
                {
                        safeFree (condensedMatrix[i]);
                }
                if (nonLuCondensedMatrix && nonLuCondensedMatrix[i])
                {
                        safeFree (nonLuCondensedMatrix[i]);
                }
        }

        if (condensedMatrix)
        {
                safeFree (condensedMatrix);
        }
        if (nonLuCondensedMatrix)
        {
                safeFree (nonLuCondensedMatrix);
        }

        traceOut (functionName, traceLevel);
        return (efficiencyUsed);
}
```

【図３Ａ】

Figure 3A

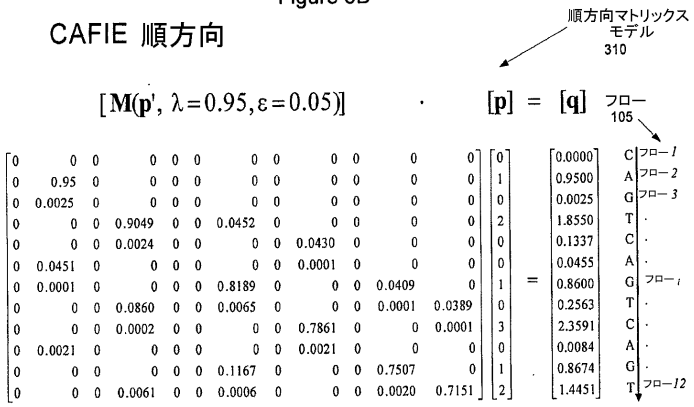
マトリックスの公式化



【図３Ｂ】

Figure 3B

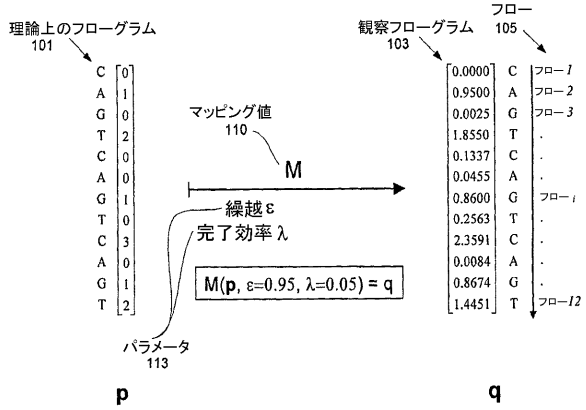
CAFIE 順方向



【図１】

Figure 1

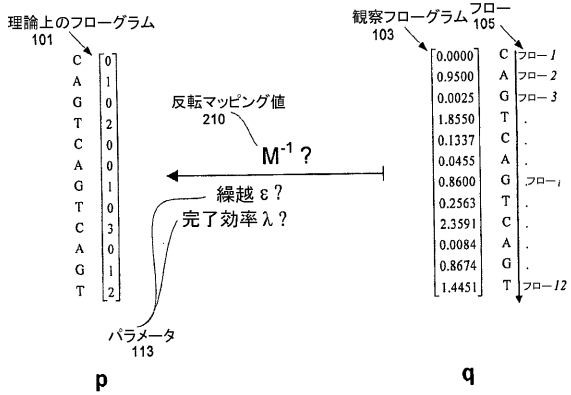
CAFIE モデル（順方向）



【図２】

Figure 2

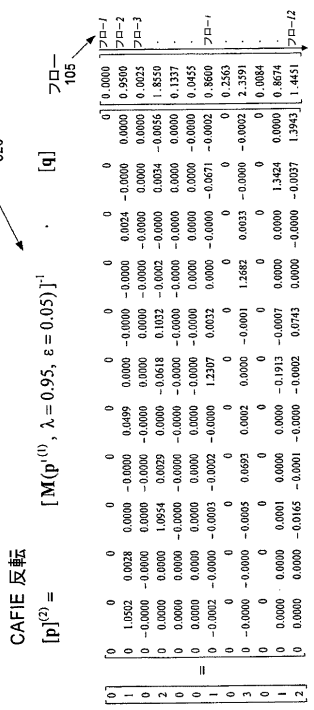
CAFIE 反転



(56)
JP 2013-211043 A 2013.10.10

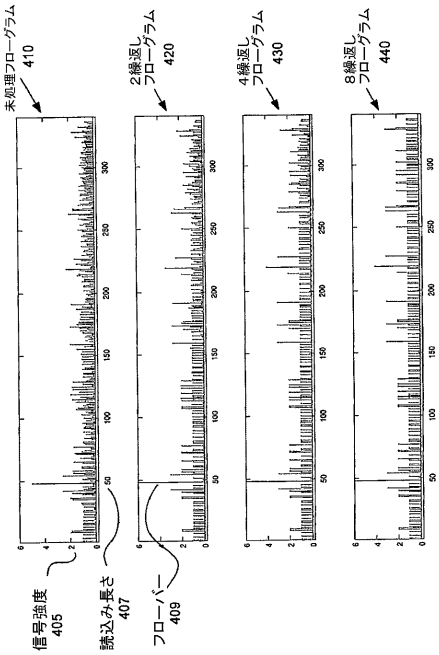
【図４Ａ】

Figure 4A



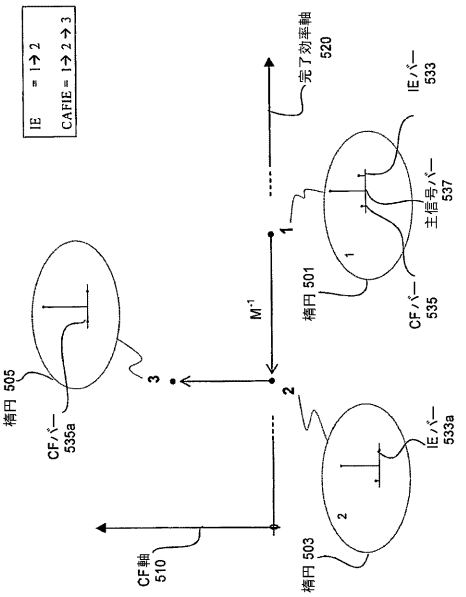
【図４Ｂ】

Figure 4B



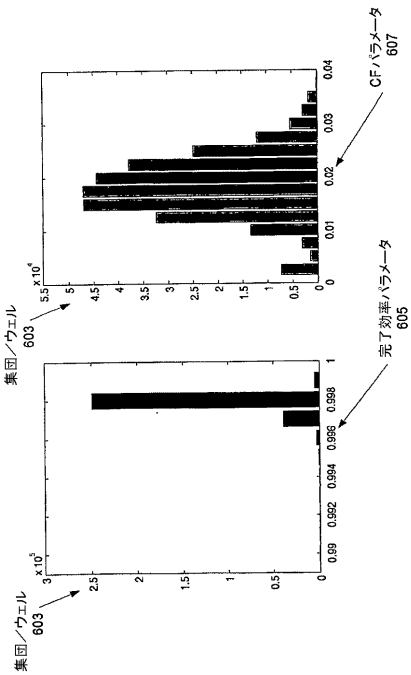
【図５】

Figure 5



【図６】

Figure 6



【図７】

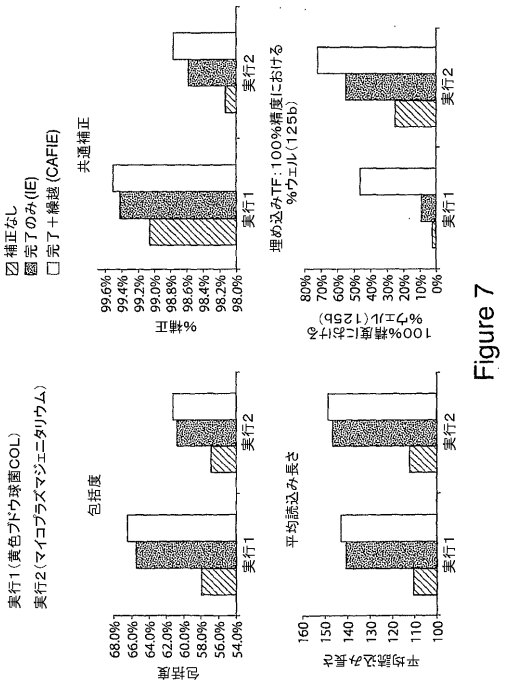

実行1（黄色ブドウ球菌COL）
実行2（マイコプラズマジェニタリウム）
補正なし
完了のみ(IE)
完了＋繰越 (CAFIE)
共通補正
包括度
%精正
包括度
実行1
実行2
99.6%
99.4%
99.2%
99.0%
98.8%
98.6%
98.4%
98.2%
98.0%
68.0%
66.0%
64.0%
62.0%
60.0%
58.0%
56.0%
54.0%
平均読込み長さ
160
150
140
130
120
110
100
埋め込みTF：100%精度における%ウェル（125b）
100%精度における%ウェル（125b）
80%
70%
60%
50%
40%
30%
20%
10%
0%

Figure 7

フロントページの続き

(72)発明者　キース　マクデード
アメリカ合衆国　コネチカット　０６４４１，　ヒガナム，　クリスチャン　ヒル　ロード　２０
(72)発明者　ジョン　シンプソン
アメリカ合衆国　コネチカット　０６４４３，　マディソン，　ウッドランド　ロード　２３，
スイート　ビー４

Ｆターム(参考) 4B024 AA11 AA19 CA02 HA11 HA19
4B029 AA07 BB02 BB20 CC03 FA15
4B063 QA13 QQ43 QR66 QS12 QS36 QS39 QX02

【外国語明細書】
2013211043000001.pdf