

Pensieve header: The “Speedy” engine.

Program

The “Speedy” Engine

Program

Internal Utilities

Program

Canonical Form:

Program

```
In[*]:= CCF[ $\mathcal{E}$ _] := PP_CCF@ExpandDenominator@ExpandNumerator@PP_Together@Together[PP_Exp[
    Expand[ $\mathcal{E}$ ] /. ex-ey->ex+y /. ex->eCCF[x]]];
CF[ $\mathcal{E}$ _List] := CF/@ $\mathcal{E}$ ;
CF[ $sd\_SeriesData$ ] := MapAt[CF,  $sd$ , 3];
CF[ $\mathcal{E}$ _] := PP_CF@Module[
    { $vs$  = Cases[ $\mathcal{E}$ , (y|b|t|a|x| $\eta$ |\beta|\tau|\alpha|\xi)_ ,  $\infty$ ] U {y, b, t, a, x,  $\eta$ ,  $\beta$ ,  $\tau$ ,  $\alpha$ ,  $\xi$ },
    Total[CoefficientRules[Expand[ $\mathcal{E}$ ],  $vs$ ] /. (ps_ -> c_) => CCF[c] (Times@@ $vs^{ps}$ )
    ]};
CF[ $\mathcal{E}\_E$ ] := CF/@ $\mathcal{E}$ ; CF[Esp___[ $\mathcal{E}S$ ___]] := CF/@Esp[ $\mathcal{E}S$ ];
```

Program

The Kronecker δ :

Program

```
In[*]:= K $\delta$  /: K $\delta$ i,j := If[i===j, 1, 0];
```

Program

Equality, multiplication, and degree-adjustment of perturbed Gaussians; $E[L, Q, P]$ stands for $e^{L+Q}P$:

Program

```
In[*]:= E /: E[L1_, Q1_, P1_] == E[L2_, Q2_, P2_] :=
    CF[L1 == L2] ^ CF[Q1 == Q2] ^ CF[Normal[P1 - P2] == 0];
E /: E[L1_, Q1_, P1_] E[L2_, Q2_, P2_] := E[L1 + L2, Q1 + Q2, P1 * P2];
E[L_, Q_, P_]$_k := E[L, Q, Series[Normal@P, { $\epsilon$ , 0, $k}]]];
```

Program

Zip and Bind

Program

Variables and their duals:

Program

```
In[*]:= { $t^*$ ,  $b^*$ ,  $y^*$ ,  $a^*$ ,  $x^*$ ,  $z^*$ } = { $\tau$ ,  $\beta$ ,  $\eta$ ,  $\alpha$ ,  $\xi$ ,  $\zeta$ };
{ $\tau^*$ ,  $\beta^*$ ,  $\eta^*$ ,  $\alpha^*$ ,  $\xi^*$ ,  $\zeta^*$ } = { $t$ ,  $b$ ,  $y$ ,  $a$ ,  $x$ ,  $z$ }; ( $u_{-i}$ )* := ( $u^*$ )i;
```

Program

Upper to lower and lower to Upper:

Program

```

U21 = { B_i^p_ := e^{-p h \gamma b_i}, B_p_ := e^{-p h \gamma b}, T_i^p_ := e^{p h t_i}, T_p_ := e^{p h t}, A_i^p_ := e^{p \gamma \alpha_i}, A_p_ := e^{p \gamma \alpha};
12U = { e^{c_-. b_i + d_-.} := B_i^{-c/(h \gamma)} e^d, e^{c_-. b + d_-.} := B^{-c/(h \gamma)} e^d,
e^{c_-. t_i + d_-.} := T_i^{c/h} e^d, e^{c_-. t + d_-.} := T^{c/h} e^d,
e^{c_-. \alpha_i + d_-.} := A_i^{c/\gamma} e^d, e^{c_-. \alpha + d_-.} := A^{c/\gamma} e^d,
e^\epsilon := e^{Expand@E}};

```

Program

Derivatives in the presence of exponentiated variables:

Program

```

In[*]:=
D_b[f_] := \partial_b f - h \gamma B \partial_B f; D_{b_i}[f_] := \partial_{b_i} f - h \gamma B_i \partial_{B_i} f;
D_t[f_] := \partial_t f + h T \partial_T f; D_{t_i}[f_] := \partial_{t_i} f + h T_i \partial_{T_i} f;
D_\alpha[f_] := \partial_\alpha f + \gamma A \partial_A f; D_{\alpha_i}[f_] := \partial_{\alpha_i} f + \gamma A_i \partial_{A_i} f;
D_v[f_] := \partial_v f; D_{\{v,0\}}[f_] := f; D_{\{v\}}[f_] := f; D_{\{v,n_Integer\}}[f_] := D_v[D_{\{v,n-1\}}[f]];
D_{\{L_List, Ls___\}}[f_] := D_{\{Ls\}}[D_L[f]];

```

Program

Finite Zips:

Program

```

In[*]:=
collect[sd_SeriesData, \xi_] := MapAt[collect[#, \xi] &, sd, 3];
collect[\epsilon, \xi_] := PPCollect@Collect[\epsilon, \xi];
Zip_{\{ \}}[P_] := P;
Zip_{\xi_s}[Ps_List] := Zip_{\xi_s} /@ Ps;
Zip_{\{\xi, \xi_s___\}}[P_] := PPZip[
(collect[P // Zip_{\xi_s}, \xi] /. f_ . \xi^d_ := (D_{\{\xi^*, d\}}[f])) /. \xi^* -> 0 /.
((\xi^* /. {b -> B, t -> T, \alpha -> A}) -> 1)]

```

Program

QZip implements the “Q-level zips” on $\mathbb{E}(L, Q, P) = e^{L+Q} P(\epsilon)$. Such zips regard the L variables as scalars.

$$\begin{aligned}
\left\langle P(z_i, \zeta^j) e^{c + \eta^i z_i + y_j \zeta^j + q^i z_i \zeta^j} \right\rangle &= |\tilde{q}| \left\langle P(z_i, \zeta^j) e^{c + \eta^i z_i} \Big|_{z_i \rightarrow \tilde{q}_i^k (z_k + y_k)} \right\rangle \\
&= |\tilde{q}| e^{c + \eta^i \tilde{q}_i^k y_k} \left\langle P(\tilde{q}_i^k (z_k + y_k), \zeta^j + \eta^i \tilde{q}_i^j) \right\rangle.
\end{aligned}$$

Program

```

QZip_{\xi_s List} @ \mathbb{E}[L_, Q_, P_] := PPQZip@Module[{ \xi, z, zs, c, ys, \eta s, qt, zrule, \xi rule, out},
zs = Table[\xi^*, {\xi, \xi_s}];
c = CF[Q /. Alternatives @@ (\xi_s \cup zs) -> 0];
ys = CF@Table[\partial_\xi (Q /. Alternatives @@ zs -> 0), {\xi, \xi_s}];
\eta s = CF@Table[\partial_z (Q /. Alternatives @@ \xi_s -> 0), {z, zs}];
qt = CF@Inverse@Table[K\delta_{z, \xi^*} - \partial_{z, \xi} Q, {\xi, \xi_s}, {z, zs}];
zrule = Thread[zs -> CF[qt . (zs + ys)]];
\xi rule = Thread[\xi_s -> \xi_s + \eta s . qt];
CF /@ \mathbb{E}[L, c + \eta s . qt . ys, Det[qt] Zip_{\xi_s}[P /. (zrule \cup \xi rule)]];

```

Program

LZip implements the “L-level zips” on $\mathbb{E}(L, Q, P) = P e^{L+Q}$. Such zips regard all of $P e^Q$ as a single “P”. Here

the z 's are b and α and the ζ 's are β and a .

Program

```

LZip $\zeta$ s_List@E[L_, Q_, P_] :=
  PPLZip@Module[{ $\zeta$ , z, zs, Zs, c, ys,  $\eta$ s, lt, zrule, Zrule,  $\zeta$ rule, Q1, EEQ, EQ},
    zs = Table[ $\zeta^*$ , { $\zeta$ ,  $\zeta$ s}];
    Zs = zs /. {b  $\rightarrow$  B, t  $\rightarrow$  T,  $\alpha$   $\rightarrow$  A};
    c = L /. Alternatives@@( $\zeta$ s  $\cup$  Zs)  $\rightarrow$  0 /. Alternatives@@Zs  $\rightarrow$  1;
    ys = Table[ $\partial_{\zeta}$ (L /. Alternatives@@Zs  $\rightarrow$  0), { $\zeta$ ,  $\zeta$ s}];
     $\eta$ s = Table[ $\partial_z$ (L /. Alternatives@@ $\zeta$ s  $\rightarrow$  0), {z, zs}];
    lt = Inverse@Table[K $\delta_{z, \zeta^*} - \partial_{z, \zeta} L$ , { $\zeta$ ,  $\zeta$ s}, {z, zs}];
    zrule = Thread[z  $\rightarrow$  lt.(zs + ys)];
    Zrule = Join[zrule,
      zrule /. r_Rule  $\Rightarrow$  ((U = r[[1]] /. {b  $\rightarrow$  B, t  $\rightarrow$  T,  $\alpha$   $\rightarrow$  A})  $\rightarrow$  (U /. U21 /. r // . l2U))];
     $\zeta$ rule = Thread[ $\zeta$ s  $\rightarrow$   $\zeta$ s +  $\eta$ s.lt];
    Q1 = Q /. (Zrule  $\cup$   $\zeta$ rule);
    EEQ[ps___] := EEQ[ps] = PPEEQ@
      (CF[e-Q1 DThread[{zs, {ps}}][eQ1]] /. {Alternatives@@Zs  $\rightarrow$  0, Alternatives@@Zs  $\rightarrow$  1});
    CF@E[c +  $\eta$ s.lt.yz, Q1 /. {Alternatives@@Zs  $\rightarrow$  0, Alternatives@@Zs  $\rightarrow$  1},
      Det[lt] (Zip $\zeta$ s[(EQ@@zs) (P /. (Zrule  $\cup$   $\zeta$ rule))] /.
        Derivative[ps___][EQ][___]  $\Rightarrow$  EEQ[ps] /. _EQ  $\rightarrow$  1) ]];
  
```

Program

In[]:=

```

B{}[L_, R_] := LR;
B{is___}[L_ E, R_ E] := PPB@Module[{n},
  Times[
    L /. Table[(v : b | B | t | T | a | x | y)i  $\rightarrow$  vnei, {i, {is}}],
    R /. Table[(v :  $\beta$  |  $\tau$  |  $\alpha$  | A |  $\xi$  |  $\eta$ )i  $\rightarrow$  vnei, {i, {is}}]
  ] // LZipJoin@Table[{ $\beta$ nei,  $\tau$ nei, anei}, {i, {is}}] // QZipJoin@Table[{ $\xi$ nei,  $\eta$ nei}, {i, {is}}] ];
Bis___[L_, R_] := B{is}[L, R];
  
```

Program

E morphisms with domain and range.

Program

```

Bis_List[Ed1  $\rightarrow$  r1[L1_, Q1_, P1_], Ed2  $\rightarrow$  r2[L2_, Q2_, P2_]] :=
  E(d1  $\cup$  Complement[d2, is])  $\rightarrow$  (r2  $\cup$  Complement[r1, is]) @@ Bis[E[L1, Q1, P1], E[L2, Q2, P2]];
Ed1  $\rightarrow$  r1[L1_, Q1_, P1_] // Ed2  $\rightarrow$  r2[L2_, Q2_, P2_] :=
  Br1  $\cap$  d2[Ed1  $\rightarrow$  r1[L1, Q1, P1], Ed2  $\rightarrow$  r2[L2, Q2, P2]];
Ed1  $\rightarrow$  r1[L1_, Q1_, P1_]  $\equiv$  Ed2  $\rightarrow$  r2[L2_, Q2_, P2_]  $\wedge$  :=
  (d1  $\equiv$  d2)  $\wedge$  (r1  $\equiv$  r2)  $\wedge$  (E[L1, Q1, P1]  $\equiv$  E[L2, Q2, P2]);
Ed1  $\rightarrow$  r1[L1_, Q1_, P1_] Ed2  $\rightarrow$  r2[L2_, Q2_, P2_]  $\wedge$  :=
  E(d1  $\cup$  d2)  $\rightarrow$  (r1  $\cup$  r2) @@ (E[L1, Q1, P1] E[L2, Q2, P2]);
Edr[L_, Q_, P_] $k_ := Edr@@E[L, Q, P] $k;
E[_E___][i_Integer] := {E}[i];
  
```

Program

$E[\wedge]$

Program

```
Edr_[A_] := CF@
Module[{L, Δθ = Limit[A, ε → 0]}, Edr[L = Δθ /. (η | y | ξ | x)_ → 0, Δθ - L, eA-Δθ]$k /. 12U]
```

Program

Exponentials as needed.

Program

Task. Define $\text{Exp}_{m,i,k}[P]$ to compute $e^{\mathcal{O}(P)}$ to ϵ^k in the using the $m_{i,j \rightarrow i}$ multiplication, where P is an ϵ -dependent near-docile element, giving the answer in \mathbf{E} -form.

Methodology. If $P_0 := P_{\epsilon=0}$ and $e^{\lambda \mathcal{O}(P)} = \mathcal{O}(e^{\lambda P_0} F(\lambda))$, then $F(\lambda = 0) = 1$ and we have:

$$\mathcal{O}(e^{\lambda P_0} (P_0 F(\lambda) + \partial_\lambda F)) = \mathcal{O}(\partial_\lambda e^{\lambda P_0} F(\lambda)) = \partial_\lambda \mathcal{O}(e^{\lambda P_0} F(\lambda)) = \partial_\lambda e^{\lambda \mathcal{O}(P)} = e^{\lambda \mathcal{O}(P)} \mathcal{O}(P) = \mathcal{O}(e^{\lambda P_0} F(\lambda)) \mathcal{O}(P).$$

This is a linear ODE for F . Setting inductively $F_k = F_{k-1} + \epsilon^k \varphi$ we find that $F_0 = 1$ and solve for φ .

Program

```
(* Bug: The first line is valid only if  $\mathcal{O}(e^{P_0}) = e^{\mathcal{O}(P_0)}$ . *)
Expm_,i_,0[P_] := Module[{LQ = Normal@P /. ε → 0},
E[LQ /. (x | y)i → 0, LQ /. (b | a | t)i → 0, 1 ];
```

Program

```
Expm_,i_,k_[P_] := Block[{$k = k},
Module[{P0, λ, φ, φs, F, j, rhs, eqn, pows, at0, atλ},
P0 = Normal@P /. ε → 0;
F = Normal@Last@Expm_,i_,k-1[λ P];
While[
rhs = mi,j→i[E{i}→{i}[λ P0 /. (x | y)i → 0, λ P0 /. (b | a | t)i → 0, F]k
si→j@E{i}→{i}[0, 0, P]k // Last // Normal;
eqn = CF[(∂λ F) + P0 F - rhs];
eqn != 0, (*do*)
pows = First /@ CoefficientRules[eqn, {yi, bi, ai, xi}];
F += Sum[εk φjs[λ] Times @@ {yi, bi, ai, xi}js, {js, pows}];
rhs = mi,j→i[E{i}→{i}[λ P0 /. (x | y)i → 0, λ P0 /. (b | a | t)i → 0, F]k
si→j@E{i}→{i}[0, 0, P]k // Last // Normal;
eqn = CF[(∂λ F) + P0 F - rhs];
φs = Table[φjs[λ], {js, pows}];
at0 = Table[φjs[0] == 0, {js, pows}];
atλ = (# == 0) & /@ (pows /. CoefficientRules[eqn, {yi, bi, ai, xi}]);
F = F /. DSolve[And@@ (at0 ∪ atλ), φs, λ] [1]
];
E{i}→{i}[P0 /. (x | y)i → 0, P0 /. (b | a | t)i → 0, F + 0[ε]k+1 /. λ → 1 ] ] ]
```

Program

“Define” Code

Program

Define[lhs = rhs, ...] defines the lhs to be rhs, except that rhs is computed only once for each value of \$k. Fancy Mathematica not for the faint of heart. Most readers should ignore.

Program

In[]:=

```

SetAttributes[Define, HoldAll];
Define[def_, defs__] := (Define[def]; Define[defs]);
Define[op_is__ =  $\varepsilon$ _] := Module[{SD, ii, jj, kk, isp, nis, nisp, sis}, Block[{i, j, k},
  ReleaseHold[Hold[
    SD[op_nisp, $k_Integer, PPBoot@Block[{i, j, k}, op_isp, $k =  $\varepsilon$ ; op_nis, $k]];
    SD[op_isp, op_{is}, $k]; SD[op_sis__, op_{sis}]];
  ] /. {SD -> SetDelayed,
    isp -> {is} /. {i -> i_, j -> j_, k -> k_},
    nis -> {is} /. {i -> ii, j -> jj, k -> kk},
    nisp -> {is} /. {i -> ii_, j -> jj_, k -> kk_}
  }} ]]
```