posed and discussed (see e.g. [3]). Many of the proposals are based on appropriate modifications of the user interface to the individual systems, but for the problems studied here, an alternative is to modify the mathematical setting so that special cases can be identified efficiently. We do this by formally replacing the notion of the 'row echelon reduction' with a factorization which preserves special-case information.

We also take the opportunity to write fraction-free Gaussian elimination (see e.g. [5]) as a factorization. This offers similar advantages in the case of parameters, but is more useful conceptually in that we feel this is a simpler presentation of[e fraction-fre e agmorihm, Bhic ae

ro echel

The following formula, which is based on a regularization procedure associated with the name of Tihonov [7], is more suited to symbolic computation (although as we will see it has severe limitations of its own):

$$A^{+} = \lim_{t \to 0} A^{*}(AA^{*} + tI)^{-1} . \tag{13}$$

This formula follows easily from (8), because the right hand side of (13) may be written

$$V\Sigma U^{*} \left(U(\Sigma^{2}\right.$$

```
MoorePenroseConditions :=
  proc( A::matrix, Ap::matrix )
  local herm;
  herm := m -> map(r->evalc(conjugate(r)),
                     linalg[transpose](m));
  print(map(normal,evalm( A &* Ap &* A - A )));
  print(map(normal,evalm( Ap&*A&*Ap - Ap)));
  print(map(normal@evalc,
             evalm( herm(A&*Ap) - A &* Ap)));
  print(map(normal@evalc,
             evalm( herm(Ap&*A) - Ap &* A)));
end:
```

The idea is that this procedure prints out four possibly differently-shaped zero matrices; if any of these matrices contains a nonzero entry, there may be a bug (but most probably just a weakness in zero-recognition). The following Maple session explores these routines.

```
> a := alpha[1] + I*alpha[2];
```

$$a := \alpha_1 + I\,\alpha_2$$

```
> b := beta[1] + I*beta[2];
```

$$b := \beta_1 + I\,\beta_2$$

```
> A := matrix(1,2,[a,b]);
```

$$A := \begin{bmatrix} \alpha_1 + I\,\alpha_2 & \beta_1 + I\,\beta_2 \end{bmatrix}$$

```
> M := MoorePenrose(A, 'prA');
```

$$M := \begin{bmatrix} -\dfrac{-\alpha_1 + I\,\alpha_2}{{\alpha_1}^2 + {\alpha_2}^2 + {\beta_1}^2 + {\beta_2}^2} \\ -\dfrac{-\beta_1 + I\,\beta_2}{{\alpha_1}^2 + {\alpha_2}^2 + {\beta_1}^2 + {\beta_2}^2} \end{bmatrix}$$

```
> MoorePenroseConditions(A, M);
```

$$\begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> B := matrix(2,2,[1,e,e,1]);
```

$$B := \begin{bmatrix} 1 & e \\ e & 1 \end{bmatrix}$$

```
> M := MoorePenrose(B,'prB');
```

$$M := \begin{bmatrix} -\dfrac{1}{-1+e^2} & \dfrac{e}{-1+e^2} \\ \dfrac{e}{-1+e^2} & -\dfrac{1}{-1+e^2} \end{bmatrix}$$

```
> prB;
```

$$1 - e^2$$

```
> MoorePenroseConditions(B,M);
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> C := subs(e=1,eval(B));
```

$$C := \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
> MC := MoorePenrose(C,'prC');
```

$$MC := \begin{bmatrix} \dfrac{1}{4} & \dfrac{1}{4} \\ \dfrac{1}{4} & \dfrac{1}{4} \end{bmatrix}$$

```
> prC;
```

$$1$$

```
> MoorePenroseConditions(C,MC);
```

This yields four 4 by 4 matrices containing only zeros.

4.3 Limitations

This routine uses symbolic inversion (of $AA^* + tI$) as a tool to compute the symbolic Moore-Penrose inverse. As is well-known, exact arithmetic solutions, and even more so exact symbolic solutions, lead very quickly to computationally intractable problems. Nonetheless if your problem contains only one or two parameters, and isn't of too high a dimension, then efficiency and insight can be gained by using a symbolic inverse or Moore-Penrose inverse. For more discussion and examples, see [2].

5 Fraction-free LU factoring

In this section we generalize the $PA = LU$ factorization to the fraction-free case, which appears not to have been done before. The paper [9] is based on a poster presented at ECCAD 97, in which they gave something they called (in quotes) a fraction-free 'factorization'; we modify this here to give a true factorization.

However, after writing down the true factorization (20 below), one realizes that the information in the extra factors F_1 and F_2 are duplicated in the U factor, and hence there is no need to form them explicitly except for conceptual understanding, and thus we see that while they do not give a true factorization, the treatment of [9] is complete and practical.

Nonetheless, while no new information is discovered this way, this true factorization approach has some advantages. First, it avoids Sylvester's Identity, and indeed avoids determinants altogether. It also, in our view, gives a clearer explanation of just why we can pull common integer factors out of certain submatrices, which is the key to the whole algorithm.

We first see a theorem and then work out an example in detail. The proof of the theorem follows the reasoning in the example and is thus omitted. Maple code for the fraction-free factorization, which works for matrices with entries from arbitrary integral domains, can be found in [2].

Theorem 4: *Fraction-Free Factorization.* Consider the rectangular matrix $A \in \mathbb{Z}^{n\times m}$. Then we may write

$$F_1 PA = LF_2 U \, , \tag{20}$$

where $F_1 = \mathrm{diag}(1, p_1, p_1p_2, \ldots, p_1p_2\cdots p_{n-1})$, P is a permutation matrix, $L \in \mathbb{Z}^{n\times n}$ is unit lower triangular, $F_2 = \mathrm{diag}(1, 1, p_1, p_1p_2, \ldots, p_1p_2\cdots p_{n-2})$, and $U \in \mathbb{Z}^{n\times m}$ is upper triangular. The p_i are the pivots that arise.

Remarks.

- This factorization (or rather, its construction by algorithm) gives a simple proof of divisibility of the submatrices by p_1, p_2, and so on. It becomes clear that since we put the factors in, with F_1, and we ought to be able to take them out again, with F_2.
- We may use either the one-step, or the two-step, fraction free algorithm (see [5] for details) to construct the factorization, which is the same in either case. Since the two-step method is asymptotically more efficient than the one-step method, we should use that. For clarity, we do not.
- Formation of the factors in F_i is not actually necessary. We may simply record the pivots p_1, p_2, and in fact even this is not necessary, since the pivots are already recorded in the diagonal entries of U.
- The determinants of each side are

$$p_1^{n-1}p_2^{n-2}\cdots p_{n-1}\det(A)$$

and

$$p_1^{n-2}p_2^{n-3}\cdots p_{n-2}\det(U)$$

respect[illegible] e reason[illegible]

5.1 Example

We use example 9.1 from [5], which has the augmented matrix

$$A = \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ 1 & -1 & 2 & 2 & 7 \\ 4 & -3 & 4 & -3 & 2 \\ -1 & 1 & 6 & -1 & 1 \end{bmatrix} . \tag{21}$$

In what follows we appear to temporarily allow divisions. This is a notational device only, for exposition, and it should be clear how to avoid ever forming any fractions even temporarily.

Applying one elementary matrix step of ordinary $PA = LU$ factorization to this matrix would give $A =$

$$\begin{bmatrix} 1 & & & \\ 1/3 & 1 & & \\ 4/3 & & 1 & \\ -1/3 & & & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ & -7/3 & 8/3 & 5/3 & \frac{23}{3} \\ & -\frac{25}{3} & \frac{20}{3} & -13/3 & 14/3 \\ & 7/3 & 16/3 & -2/3 & 1/3 \end{bmatrix} \tag{22}$$

To remove the fractions, we may rewrite the identity matrix as

$$I = \begin{bmatrix} 1 & & & \\ & 1/3 & & \\ & & 1/3 & \\ & & & 1/3 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 3 & & \\ & & 3 & \\ & & & 3 \end{bmatrix} \tag{23}$$

and insert these two factors in between the two factors of A we have found so far. Since multiplication by a diagonal matrix on the right multiplies columns, the 1's on the diagonal of the L factor all become 1/3. Once this happens, we may factor 1/3 out of each row, giving

$$A \;=\; \begin{bmatrix} 1 & & & \\ & 1/3 & & \\ & & 1/3 & \\ & & & 1/3 \end{bmatrix} \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ 4 & & 1 & \\ -1 & & & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ & -7 & 8 & 5 & 23 \\ & -25 & 20 & -13 & 14 \\ & 7 & 16 & -2 & 1 \end{bmatrix}$$

Of course, multiplying both sides by diag$(1, 3, 3, 3)$ will remove the fractions completely. So far, we have not captured the essence of the Bareiss-Jordan fraction-free algorithm; all we have done is cleared fractions in ordinary $PA = LU$ factorization. Indeed, this is just the beginning of what is called 'division-free Gaussian elimination' in [5]. [Apparently, 'division-free' is the accepted name for a slightly different algorithm, which is less clever than the 'fraction-free' algorithm. We will not have cause to refer to 'division-free' elimination again.] We need to do one more step before the idea of 'fraction-free factorization' becomes clear. Call the last factor in the above equation, $A^{(1)}$. We will work just with $A^{(1)}$, for easy typography.

We start as before by pretending to use ordinary rational LU factorization steps. We may write $A^{(1)} =$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & \frac{25}{7} & 1 & \\ & -1 & & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 & -2 & 1 & -2 \\ & -7 & 8 & 5 & 23 \\ & & -\frac{60}{7} & -\frac{216}{7} & -\frac{477}{7} \\ & & 24 & 3 & 24 \end{bmatrix} \tag{24}$$

and again we will wish to clear the fractions (accidentally, the last multiplier was also 7 and so the entries in the last row are integers, but in general this will not happen). In actual fact the pivot was -7, so we'll adjust the minus signs above, and use a similar rewriting ɒ iI 6 C o ahead of time that this divisibility will happen. Thus we may cheaply take advantage of it.

Writing this observation as a factorization, we have that $\operatorname{diag}(1,1,-7,-7)A^{(1)}$

$$= \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -25 & 1 & \\ & 7 & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 3 & \\ & & & 3 \end{bmatrix}$$

$$\times \left[\begin{matrix} 3 \\ \\ \\ \\ \end{matrix} \right.$$